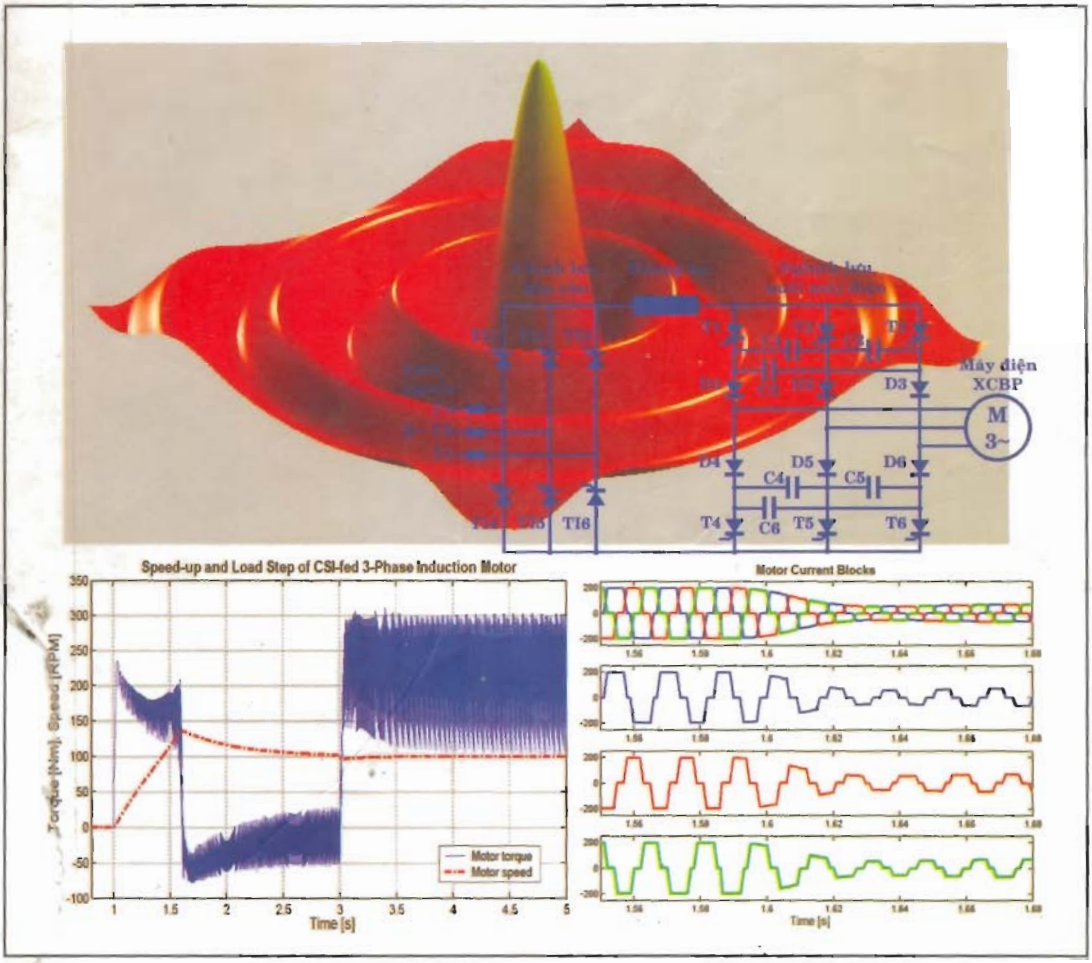
NGUYỄN PHÙNG QUANG

MATLAB & SIMULINK

DÀNH CHO KỸ SƯ ĐIỀU KHIỂN TỰ ĐỘNG



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

F973.2-018

Nguyễn Phùng Quang

MATLAB & Simulink dành cho kỹ sư điều khiển tự động

THƯ VIỆN HỌC VIỆN KỸ THUẬT QUÂN SỰ
Vtc 3898
20
PHỤ BẢN

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
HÀ NỘI - 2004

Lời nói đầu

Đối với kỹ sư điều khiển - tự động hóa nói riêng và những người nghiên cứu khoa học – kỹ thuật nói chung, mô phỏng là công cụ quan trọng cho phép khảo sát các đối tượng, hệ thống hay quá trình kỹ thuật – vật lý, mà không nhất thiết phải có đối tượng hay hệ thống thực. Được trang bị một *công cụ mô phỏng* mạnh và có hiểu biết về các *phương pháp mô hình hóa*, người kỹ sư sẽ có khả năng rút ngắn thời gian và giảm chi phí nghiên cứu – phát triển sản phẩm một cách đáng kể. Điều này đặc biệt có ý nghĩa khi sản phẩm là các hệ thống thiết bị kỹ thuật phức hợp với giá trị kinh tế lớn. Các khái niệm mô phỏng Off-Line, Software-in-the-Loop, Hardware-in-the-Loop (hay Real Time Simulation: mô phỏng thời gian thực) và Prototyping, đã thể hiện rõ nét các bước của quá trình phát triển sản phẩm với sự hỗ trợ của máy tính.

Trong nhiều năm qua, người biên soạn sách này đã sử dụng phần mềm mô phỏng MATLAB & Simulink trong hoạt động nghiên cứu – phát triển thực tiễn và giảng dậy của mình. Đặc biệt, khi điều kiện thí nghiệm tại trường đại học nước ta còn rất thiếu thốn, phần mềm trên đã góp phần hữu ích trong quá trình đào tạo các kỹ sư điều khiển - tự động hóa. Tuy nhiên, việc thiếu một tài liệu tham khảo thích hợp đã hạn chế hiệu quả đào tạo. Với MATLAB & Simulink ta có trong tay một *công cụ mô phỏng* mạnh kèm theo tài liệu dưới dạng files .pdf vô cùng chi tiết và phong phú. Nhược điểm chính của các tài liệu đó là: 1. Chúng không hề truyền đạt cho người đọc kiến thức về *phương pháp mô hình hóa*; 2. Chúng quá chi tiết, đến mức tản mạn, và vì vậy đã làm mất không ít thời gian của người học. Nhu cầu về một tài liệu bằng tiếng Việt, khắc phục được hai nhược điểm đó, là nhu cầu bức xúc của quá trình đào tạo, khiến tác giả đặt quyết tâm biên soạn cuốn sách này.

Cuốn sách bao gồm ba phần chính:

1. Phần *cơ sở về MATLAB* và một vài *Toolbox* quan trọng. Phần này gồm các chương 1 – 5.
2. Phần giới thiệu *Simulink* và các *thư viện* đi kèm. Phần này gồm các chương 6 – 9.
3. Phần giới thiệu *Motion Control Blockset*, công cụ khảo sát các hệ thống điều khiển chuyển động. Phần này gồm các chương 10 – 12.

Phần 1 và 2 được biên soạn chủ yếu dựa trên nền giáo trình về MATLAB (tài liệu [1]) của đại học München (CHLB Đức). Phần 3 bao gồm các kết quả nghiên cứu của bản thân tác giả, của đồng nghiệp (tài liệu [9]) và của sinh viên tại đại học Bách Khoa Hà Nội. Nội dung dự kiến ban đầu của sách còn bao gồm hai Toolbox quan trọng là Real-Time Workshop, Stateflow Blockset. Tuy nhiên, để giới hạn kích cỡ sách và thời gian biên soạn (trong khi sinh viên đang trông đợi tài liệu học) nên người viết tạm ngừng lại ở khối lượng 12 chương. Các nội dung còn thiếu sẽ được bổ sung ở lần tái bản sau này.

Cần phải khẳng định rằng MATLAB không phải là một ngôn ngữ lập trình, mặc dù phần mềm này cũng có các khả năng của một ngôn ngữ lập trình bậc cao: MATLAB & Simulink trước hết là một công cụ toán số, với thế mạnh tính

toán và mô phỏng hệ thống. Đặt mục tiêu hạn chế vào đối tượng bạn đọc chính là sinh viên và kỹ sư ngành điều khiển - tự động hóa, cuốn sách thể hiện rõ thế mạnh trên của bộ phần mềm. Khi biên soạn, người viết cố gắng trình bầy các vấn đề một cách dễ hiểu và từng bước đều có minh họa bởi ví dụ đi kèm. Để bảo đảm tính chính xác, người viết tự tay mình thực hiện lại tất cả các ví dụ (sử dụng MATLAB phiên bản 6.0 / rel. 12, Simulink phiên bản 4.0).

Tác giả cảm ơn Dipl.-Ing. (Uni.) J. Meyer (Dresden, CHLB Đức) đã cho phép sử dụng các mô hình của mình. Xin cảm ơn TS. Trần Thị Thu Hương, bạn đời của tác giả, người đã luôn động viên và tạo điều kiện để tác giả hoàn thành sách. Cảm ơn chị Nguyễn Thị Ngọc Khuê, Nhà xuất bản Khoa học và Kỹ thuật, đã hợp tác và tạo điều kiện để sách nhanh chóng ra mắt bạn đọc.

Xin nhấn mạnh lần cuối đặc điểm biên soạn của cuốn sách: Để nhanh chóng có sách phục vụ cho sinh viên (đối tượng bạn đọc chính), tác giả đã biên soạn chủ yếu dựa trên các tài liệu của nhiều tác giả xuất bản ở nước ngoài. Trừ một số kết quả của các nghiên cứu ứng dụng cụ thể, cuốn sách nói chung không mang đặc điểm công trình khoa học của cá nhân người biên soạn. Tuy nhiên, hy vọng đã cung cấp cho bạn đọc, cho người học một tài liệu tham khảo có ích.

Dù đã được viết khá cẩn thận, sách khó tránh khỏi còn những sai sót. Mọi lời góp ý, nhận xét hay đề xuất bổ sung nhằm hoàn thiện sách, xin bạn đọc gửi về Nhà xuất bản Khoa học và Kỹ thuật, hay về Phòng thí nghiệm trọng điểm về Tự động hóa – Trường đại học Bách khoa Hà Nội, xin chân thành cảm ơn.

Hà Nội, hè Quý Mùi 2003
Tác giả

Mục lục

		Trang
	Lời nói đầu	
PHẦN A	**MATLAB VÀ CÁC TOOLBOX**	
1	**Cơ sở về MATLAB**	1
1.1	Những bước đi đầu tiên với MATLAB	1
1.1.1	Màn hình MATLAB	1
1.1.2	Tiện ích trợ giúp (Help) của MATLAB	3
1.1.3	Các biến	4
1.1.4	Các hàm toán	5
1.2	Vector và ma trận	6
1.2.1	Tính toán với vector và ma trận	8
1.3	Cấu trúc và trường	10
1.3.1	Cấu trúc	10
1.3.2	Trường	12
1.4	Quản lý biến	13
1.5	Các phép so sánh và phép tính logic	14
1.6	Rẽ nhánh và vòng lặp	16
1.6.1	Lệnh rẽ nhánh `if` và `switch`	16
1.6.2	Vòng lặp sử dụng `for` và `while`	16
1.6.3	Gián đoạn bằng `continue` và `break`	17
1.7	Các Scripts và các hàm của MATLAB	18
1.7.1	Các Scripts của MATLAB	18
1.7.2	Các hàm của MATLAB	19
1.8	Tóm tắt nội dung chương 1	20
2	**Xuất và nhập dữ liệu trên màn hình**	23
2.1	Điều khiển xuất ra màn hình	23
2.2	Đối thoại của MATLAB với người sử dụng	24
2.2.1	Văn bản (Text) trong MATLAB	25
2.2.2	Đối thoại khi nhập văn bản	25
2.2.3	Xuất theo định dạng	26
2.3	Nhập và xuất dữ liệu	27
2.3.1	Cất vào hoặc gọi dữ liệu từ File	27
2.3.2	Cất có định dạng vào File văn bản	28
2.4	Hệ điều hành và quản lý File	29

2.5	Biểu diễn bằng đồ họa	30
2.5.1	Cửa sổ Figure – cơ sở của đồ họa MATLAB	30
2.5.2	Trục và điền ký tự cho trục	32
2.6	Đồ họa 2 chiều (2-D Graphics)	34
2.6.1	Các lệnh vẽ (Plot Commands)	34
2.6.2	Ví dụ: Khâu quán tính bậc nhất PT_1, và khâu tỷ lệ – vi phân PD	36
2.7	Đồ họa 3 chiều (3-D Graphics)	38
2.7.1	Các lệnh Plots	38
2.7.2	Phối cảnh (Perspective) trong đồ họa 3-D	39
2.7.3	Ví dụ về đồ họa 3-D có phối cảnh	39
2.8	Nhập, xuất và in đồ họa	41
2.9	Giao diện đồ họa	43
2.9.1	Layout (diện mạo) của GUI	43
2.9.2	Nhập và xuất ký tự, số liệu ra GUI	44
2.9.3	Nhập số liệu từ thanh trượt (Slider)	47
2.9.4	Nhập dữ liệu tùy chọn (Popup Menu, List Box, Radio Button and Check Box)	49
2.9.5	Các phương pháp tạo GUI	53
2.10	Tóm tắt nội dung chương 2	62
3	**Control System Toolbox: Công cụ khảo sát - thiết kế hệ thống điều khiển**	65
3.1	Mô hình hóa các hệ tuyến tính – dừng (hệ LTI)	65
3.1.1	Mô hình truyền đạt	66
3.1.2	Mô hình điểm không - điểm cực	68
3.1.3	Mô hình trạng thái	71
3.1.4	Mô hình dữ liệu đặc tính tần số	72
3.1.5	Mô hình gián đoạn theo thời gian	74
3.1.6	Thời gian trễ trong các hệ tuyến tính – dừng (hệ LTI)	76
3.2	Nguyên tắc sử dụng mô hình LTI	79
3.2.1	Đặc điểm của mô hình LTI	79
3.2.2	Truy cập nhanh dữ liệu của mô hình	83
3.2.3	Trình tự ưu tiên của mô hình LTI	83
3.2.4	Tính kế thừa của mô hình LTI	84
3.2.5	Đảo loại cho mô hình LTI	85
3.2.6	Các phép tính số học	85
3.2.7	Lựa chọn, thay đổi và ghép nối mô hình LTI	87
3.2.8	Chuyển đổi giữa hai hệ liên tục và gián đoạn về thời gian	92
3.3	Khảo sát mô hình LTI	96
3.3.1	Các đặc điểm tổng quát	96
3.3.2	Khảo sát động học của mô hình	98
3.3.3	Đáp ứng của hệ trên miền thời gian	104
3.3.4	Đáp ứng của hệ trên miền tần số	109

3.3.5	Mô hình giảm bậc	119
3.3.6	Các phương pháp mô tả trên không gian trạng thái	122
3.4	Thiết kế vòng điều chỉnh	126
3.4.1	Thiết kế theo phương pháp quỹ đạo điểm cực	127
3.4.2	Thiết kế theo phương thức đối thoại giữa người và PC	131
3.4.3	Điều khiển và quan sát trạng thái	132
3.4.4	Thiết kế theo phương pháp gán cực	134
3.4.5	Thiết kế theo tiêu chuẩn tích phân tối ưu	140
3.5	Các vấn đề khi tính toán số	147
3.5.1	Khái niệm lỗi	147
3.5.2	Khái niệm điều hòa mô hình	148
3.5.3	Tính ổn định số học	149
3.5.4	Đánh giá mô hình LTI theo quan điểm tính số	150
3.6	Tóm tắt nội dung chương 3	150
4	**Optimization Toolbox: Công cụ tính toán tìm tối ưu**	153
4.1	Inline Objects	153
4.2	Điều khiển thuật toán	155
4.3	Tìm điểm không	157
4.3.1	Hàm scalar	157
4.3.2	Hàm vector và hệ phương trình	161
4.4	Tìm cực tiểu cho hàm phi tuyến	166
4.5	Tìm cực tiểu khi có điều kiện phụ	170
4.6	Phương pháp bình phương sai phân bé nhất	177
4.7	Tìm bộ tham số tối ưu cho mô hình SIMULINK	185
4.8	Tóm tắt nội dung chương 4	189
5	**Signal Processing Toolbox: Công cụ xử lý tín hiệu**	191
5.1	Phương pháp nội suy (Interpolation)	191
5.2	Biến đổi Fourier gián đoạn	194
5.2.1	Phương pháp Averaging	196
5.2.2	Phương pháp tạo cửa sổ số liệu	198
5.3	Hàm tương quan (Correlation Functions)	200
5.4	Lọc số (Digital Filter)	205
5.4.1	Bộ lọc FIR và hàm cửa sổ	206
5.4.2	Bộ lọc IIR	209
5.5	Lọc tương tự (Analog Filter)	211
5.6	Tóm tắt nội dung chương 5	212
PHẦN B	**SIMULINK VÀ CÁC THƯ VIỆN**	
6	**Cơ sở về SIMULINK**	213
6.1	Khởi động SIMULINK	213
6.2	Tạo mới và soạn thảo lưu đồ tín hiệu	216

6.3	Tín hiệu và các loại dữ liệu	217
6.3.1	Làm việc với tín hiệu	217
6.3.2	Làm việc với các loại số liệu	220
6.4	Thư viện SOURCES và SINKS	220
6.4.1	Thư viện SOURCES	220
6.4.2	Thư viện SINKS	224
6.5	Thư viện Math	230
6.6	Chuẩn bị mô phỏng: Khai báo tham số và phương pháp tích phân	233
6.6.1	Khởi động và ngừng mô phỏng	241
6.6.2	Xử lý lỗi	242
6.6.3	Tập hợp các tham số trong Script của MATLAB	243
6.6.4	Ví dụ	243
6.6.5	In mô hình SIMULINK	246
6.7	Hệ thống con (Subsystem)	246
6.7.1	Tạo hệ thống con	246
6.7.2	Thư viện Signals & Subsystems	247
6.7.3	Kích hoạt có điều kiện các hệ thống con	251
6.7.4	Đánh dấu các hệ con (Mask Subsystems)	254
6.8	Tóm tắt nội dung chương 6	256
7	**Các hệ thống tuyến tính và phi tuyến**	257
7.1	Thư viện Continuous	257
7.2	Tuyến tính hóa	261
7.3	Xác định điểm cân bằng	263
7.4	Thư viện Nonlinear	265
7.5	Thư viện Function & Tables	268
7.6	Vòng quẩn đại số	271
7.7	Hàm S (S-Functions)	272
7.8	Tóm tắt nội dung chương 7	281
8	**Các hệ thống trích mẫu (hệ gián đoạn)**	283
8.1	Các khái niệm tổng quan	283
8.2	Tham số mô phỏng	284
8.3	Thư viện Discrete	286
8.4	Hệ có chu kỳ trích mẫu hỗn hợp và hệ lai	287
8.4.1	Hệ có chu kỳ hỗn hợp	287
8.4.2	Hệ lai	288
8.5	Tóm tắt nội dung chương 8	290
9	**Phân tích và tổng hợp vòng điều chỉnh**	291
9.1	Động cơ một chiều kích thích độc lập	291
9.1.1	Khai báo lập trị ban đầu	292
9.1.2	Mô hình SIMULINK	293

9.2	Khảo sát động học của đối tượng	294
9.2.1	Khảo sát bằng SIMULINK	294
9.2.2	Khảo sát bằng MATLAB sử dụng mô hình tuyến tính hóa	295
9.2.3	Khảo sát mô hình LTI qua đối thoại với LTI-Viewer	298
9.3	Điều chỉnh với nhiều vòng phân cấp (Cascade Control)	300
9.3.1	Điều chỉnh dòng phần ứng	301
9.3.2	Điều chỉnh tốc độ quay	303
9.4	Quan sát trạng thái	306
9.4.1	Khâu quan sát Luenberger	308
9.4.2	Khâu quan sát nhiễu (QS phụ tải)	310
9.5	Điều khiển trạng thái sử dụng khâu quan sát trạng thái	313
9.6	Tóm tắt nội dung chương 9	318

PHẦN C MOTION CONTROL BLOCKSET: CÔNG CỤ KHẢO SÁT CÁC HỆ THỐNG ĐIỀU KHIỂN CHUYỂN ĐỘNG

10	**Thư viện mô hình máy điện quay**	319
10.1	Khái quát về Motion Control Blockset	319
10.2	Máy điện một chiều kích thích độc lập	321
10.3	Máy điện không đồng bộ xoay chiều ba pha	326
10.4	Máy điện đồng bộ ba pha kích thích vĩnh cửu	343
10.5	Máy điện đồng bộ ba pha kích thích độc lập	351
10.6	Máy điện không đồng bộ nguồn kép	361
10.7	Máy điện từ kháng kiểu đóng ngắt	365
10.8	Tóm tắt nội dung chương 10	374
11	**Thư viện mô hình thiết bị biến đổi (điện tử công suất)**	375
11.1	Mô hình cầu chỉnh lưu 6 xung có điều khiển cắt pha	376
11.2	Mô hình nghịch lưu băm xung nguồn áp	379
11.3	Mô hình nghịch lưu nguồn dòng nuôi MĐDB	389
11.4	Mô hình thiết bị biến đổi trực tiếp nuôi MĐDB và MĐĐB	397
11.5	Tóm tắt nội dung chương 11	399
12	**Thư viện mô hình phần cơ và ví dụ ứng dụng của Motion Control Blockset**	401
12.1	Thư viện mô hình các phần tử truyền động cơ học	401
12.2	Vài ví dụ ứng dụng	404
12.2.1	Mô phỏng một trục chuyển động của tay máy sử dụng MĐDB	404
12.2.2	Mô phỏng hệ thống phát điện chạy sức gió sử dụng MĐDB-RDQ	409
12.2.3	Mô phỏng hệ truyền động dị bộ theo phương pháp "*Direct Torque Control*" sử dụng Logic mờ	419
12.3	Tóm tắt nội dung chương 12	467

11 Tài liệu tham khảo 469
11.1 Tài liệu tham khảo chung 469
11.2 Các tài liệu dưới dạng .pdf-Files 470
11.3 Các luận văn, đồ án tốt nghiệp tại ĐHBK Hà Nội đã góp phần xây dựng Motion Control Blockset 470

Danh mục từ tra cứu 473

1 Cơ sở về MATLAB

MATLAB là một bộ chương trình phần mềm lớn của lĩnh vực toán số. Tên bộ chương trình chính là chữ viết tắt từ **MAT**rix **LAB**oratory, thể hiện định hướng chính của chương trình là các phép tính vector và ma trận. Phần cốt lõi của chương trình bao gồm một số hàm toán, các chức năng nhập/xuất cũng như các khả năng điều khiển chu trình mà nhờ đó ta có thể dựng nên các *Scripts*.

Thêm vào phần cốt lõi, có thể mua bổ sung các *Toolbox*[1] (bộ công cụ) với phạm vi chức năng chuyên dụng mà người sử dụng cần. Tài liệu này sẽ chỉ giới thiệu hạn chế một số Toolbox liên quan tới *Điều khiển – Tự động hóa* như: *Control System Toolbox*, *Signal Processing Toolbox*, *Optimization Toolbox*, *Stateflow Blockset*, *Power System Blockset*, *Real-Time Workshop* và *SIMULINK*. SIMULINK là một Toolbox có vai trò đặc biệt quan trọng: Vai trò của một công cụ mạnh phục vụ *mô hình hóa và mô phỏng các hệ thống Kỹ thuật – Vật lý* trên cơ sở sơ đồ cấu trúc dạng khối. Cùng với SIMULINK, Stateflow Blockset tạo cho ta khả năng mô hình hóa và mô phỏng các automat trạng thái hữu hạn.

Cuối từng chương, mục sẽ có phần tổng kết lại tất cả các lệnh quan trọng nhất, giúp bạn đọc hệ thống hóa lại nội dung của chương, mục đó. Cũng cần phải quy ước trước: Các biến sẽ luôn được *viết nghiêng* và các tham số được viết trong ngoặc vuông [ ]. Phần mã của MATLAB cũng như các số liệu nhập/xuất được phân biệt nhờ `dạng chữ đánh máy`.

1.1 Những bước đi đầu tiên với MATLAB

1.1.1 Màn hình MATLAB

Sau khi khởi động *matlab* ta thu được màn hình MATLAB, môi trường tích hợp với những cửa sổ chính như hình 1.1. Để có màn hình đó, ta lần lượt chọn menu: *View*, *Desktop Layout*, *Five Panel*.

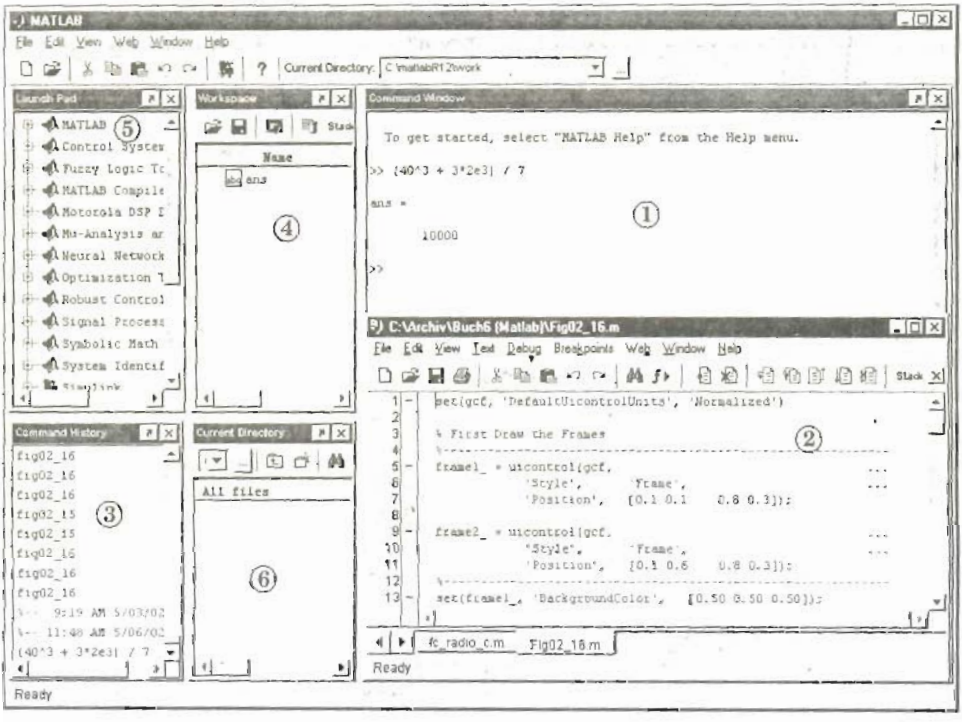
Cửa sổ lệnh **Command Windows** ①: Đây là cửa sổ chính của MATLAB. Tại đây ta thực hiện toàn bộ việc nhập dữ liệu và xuất kết quả tính toán. Dấu nhấp nháy >> báo hiệu chương trình sẵn sàng hoạt động:

---

[1] Chú ý: Người biên soạn sách chủ trương sử dụng nguyên từ gốc. Nghĩa dịch trong ngoặc chỉ có giá trị tham khảo.

```
>> (40^3 + 3*2e3) / 7
ans =
        1000
```

Mỗi lần nhập dữ liệu được kết thúc bằng động tác nhấn phím ENTER. Nguyên tắc "nhân, chia thực hiện trước cộng, trừ" và thứ tự ưu tiên của dấu ngoặc vẫn như bình thường. Số có giá trị lớn thường được nhập với hàm e mũ (có thể viết E). Có thể kết thúc chương trình bằng cách đóng màn hình MATLAB, hoặc gọi lệnh `quit`, `exit`, hoặc nhấn tổ hợp phím `Ctrl+q`.



Hình 1.1 Màn hình MATLAB

Cửa sổ soạn thảo **Editor** ②: Nhờ chương trình soạn thảo của MATLAB ta có thể viết mới, hay xử lý sửa đổi các Scripts và các hàm. Bên cạnh chức năng soạn thảo, còn có các chức năng thông thường khác mà một môi trường soạn thảo chương trình cần phải có, phục vụ xử lý từng bước nội dung chương trình, hay để phát hiện lỗi.

Cửa sổ quá khứ **Command History** ③: Tất cả các lệnh đã sử dụng trong Command Windows ① được lưu giữ và hiển thị tại đây. Có thể lặp lại lệnh cũ bằng cách nháy chuột kép vào lệnh đó. Cũng có thể cắt, sao hoặc xóa cả nhóm lệnh hoặc từng lệnh riêng rẽ.

Cửa sổ môi trường công tác **Workspace Browser** ④: Tất cả các biến, các hàm tồn tại trong môi trường công tác đều được hiển thị tại cửa sổ này với đầy

đủ thông tin như: tên, loại biến/hàm, kích cỡ tính theo Bytes và loại dữ liệu. Ngoài ra còn có thể cất vào bộ nhớ các dữ liệu đó, hoặc sử dụng chức năng Array Editors (soạn thảo mảng) để thay đổi các biến.

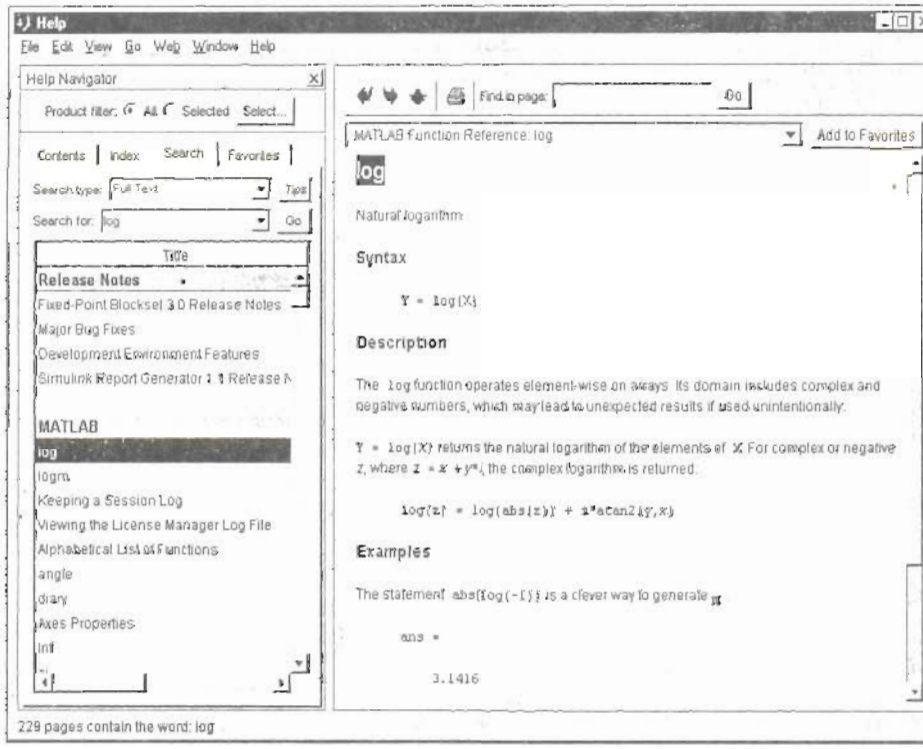
Cửa sổ **Launch Pad** ⑤: Cửa sổ này cho phép người sử dụng truy cập nhanh các công cụ của MATLAB, phần Help (trợ giúp) hoặc Online Documents (tài liệu trực tuyến), mở Demos (chương trình trình diễn).

Cửa sổ thư mục hiện tại **Current Directory Browser** ⑥: Nhờ cửa sổ này, người sử dụng có thể nhanh chóng nhận biết, chuyển đổi thư mục hiện tại của môi trường công tác, mở File, tạo thư mục mới.

Trên đây chỉ thuần túy là ví dụ về một khả năng tạo màn hình công tác. Nhờ menu *View*, người sử dụng có thể thay đổi linh hoạt màn hình MATLAB cho phù hợp với thói quen và nhu cầu sử dụng của bản thân.

1.1.2 Tiện ích trợ giúp (Help) của MATLAB

Tiện ích Help của MATLAB là vô cùng phong phú. Tùy theo nhu cầu, hoặc gọi `help` [*command*] để xem nội dung hỗ trợ của lệnh *command* trực tiếp trên Command Windows, hoặc sử dụng công cụ truy cập Help (hình 1.2).



Hình 1.2 Cửa sổ truy cập Help của MATLAB (gọi doc log)

Có thể gọi cửa sổ ở hình 1.2 bằng cách gọi trên menu, gọi lệnh `helpwin` hay `doc` trực tiếp trên cửa sổ Command Windows. Bằng lệnh `lookfor` *searchstring* ta có thể tìm chuỗi ký tự *searchstring* trong dòng đầu của mọi MATLAB Files trong thư mục MATLAB:

```
>> help log

 LOG    Natural logarithm.
    LOG(X) is the natural logarithm of the elements of X.
    Complex results are produced if X is not positive.

    See also LOG2, LOG10, EXP, LOGM.

 Overloaded methods
    help sym/log.m

>>
```

Các lệnh liên quan tới tiện ích Help được tập hợp trong bảng sau:

	Help
`help` [*command*]	Tiện ích Help trực tuyến của MATLAB trong cửa sổ lệnh Command Workspace
`helpwin` [*command*]	Tiện ích Help trực tuyến của MATLAB trong cửa sổ truy cập Help
`doc` [*command*]	Tư liệu trực tuyến của MATLAB trong cửa sổ truy cập Help
`lookfor` *searchstring*	Tìm chuỗi ký tự *searchstring* trong dòng đầu tiên của mọi MATLAB Files trong thư mục MATLAB

1.1.3 Các biến

Thông thường, kết quả của các biến được gán cho `ans`. Sử dụng dấu = ta có thể định nghĩa một biến, đồng thời gán giá trị cho biến đó. Khi nhập tên của một biến mà không gán giá trị, ta thu được giá trị hiện tại của biến. Tất cả các biến đều là biến *global* trong *Workspace*. Tên của biến có thể chứa tới 32 chữ cái, gạch ngang thấp (_) cũng như chữ số. Chữ viết hoa to và chữ viết nhỏ đều được phân biệt.

Việc nhập giá trị có thể được thực hiện thành một chuỗi lệnh trong cùng một dòng, chỉ cách nhau bởi dấu (;). Nếu sử dụng dấu phẩy (,) để tách các lệnh, khi ấy các giá trị sẽ được xuất ra màn hình:

```
>> variable_1 = 25; variable_2 = 10;
>> variable_1
variable_1 =
    25

>> a = variable_1 + variable_2, A = variable_1 / variable_2
a =
    35
A =
    2.5000
```

Một số tên biến như: `pi`, `i`, `j` và `inf` đã được MATLAB dùng để chỉ các hằng số hay ký hiệu, vậy ta phải tránh sử dụng chúng. Đối với các phép tính bất định (ví dụ: 0/0), trên màn hình sẽ hiện kết quả `NaN` (*Not a Number*). `eps` cho ta biết cấp chính xác tương đối khi biểu diễn số với dấu phẩy động (ví dụ: `eps` = 2.2204e-016):

```
>> 1 / 0                    % Inf: infinite (vô cùng)
Warning: Divide by zero.
ans =
   Inf

>> 0 / 0                    % NaN: not-defined (bất định)
Warning: Divide by zero.
ans =
   NaN
```

	Các ký hiệu
=	Gán giá trị cho biến
+ - * / ^	Các phép tính
;	Nhập giá trị (còn giữ vai trò dấu cách khi nhập nhiều giá trị trong cùng một dòng)
,	Dấu cách khi xuất nhiều giá trị trong cùng một dòng
eps	Cấp chính xác tương đối khi sử dụng giá trị dấu phẩy động
i j	Toán tử ảo
inf	Vô cùng (∞)
NaN	Not a Number
Pi	Hằng số π

1.1.4 Các hàm toán

Chương trình MATLAB có sẵn rất nhiều hàm toán tập hợp trong bảng sau đây. Để xem kỹ hơn, bạn đọc có thể sử dụng các lệnh `help elfun` hoặc `help datafun`. Tất cả các hàm trong bảng đều có khả năng sử dụng tính với vector.

Các hàm toán			
`sqrt(x)`	Căn bậc hai	`rem(x,y)`	Số dư của phép chia *x/y*
`exp(x)`	Hàm mũ cơ số e	`round(x)`	Làm tròn số
`log(x)`	Logarithm tự nhiên	`ceil(x)`	Làm tròn lên
`log10(x)`	Logarithm cơ số thập phân	`floor(x)`	Làm tròn xuống
`abs(x)`	Giá trị tuyệt đối	`sum(v)`	Tổng các phần tử vector
`sign(x)`	Hàm dấu	`prod(v)`	Tích các phần tử vector
`real(x)`	Phần thực	`min(v)`	Phần tử vector bé nhất
`imag(x)`	Phần ảo	`max(v)`	Phần tử vector lớn nhất
`phase(x)`	Góc pha của số phức	`mean(v)`	Giá trị trung bình cộng
Các hàm lượng giác			
`sin(x)`	Hàm sin	`atan(x)`	Hàm arctg $\pm 90^0$
`cos(x)`	Hàm cos	`atan2(x,y)`	Hàm arctg $\pm 180^0$
`tan(x)`	Hàm tg	`sinc(x)`	Hàm $\sin(\pi x)/(\pi x)$

1.2 Vector và ma trận

MATLAB có một số lệnh đặc biệt để khai báo hoặc xử lý vector và ma trận. Cách đơn giản nhất để khai báo, tạo nên vector hoặc ma trận là nhập trực tiếp. Khi nhập trực tiếp, các phần tử của một hàng được cách bởi dấu phẩy hoặc vị trí cách bỏ trống[1], các hàng được cách bởi dấu (;) hoặc ngắt dòng.

```
>> my_vector = [1 2 3]
my_vector =
     1     2     3

>> my_matrix = [my_vector; 5 6 7]
my_matrix =
     1     2     3
     5     6     7
```

Vector có các phần tử tiếp diễn với một bước nhất định, có thể được nhập một cách đơn giản nhờ toán tử (:) như sau (*start*: *increment*: *destination*)[2]. Nếu chỉ nhập *start* và *destination*, MATLAB sẽ tự động đặt *increment* là +1.

Cũng có thể nhập các vector tuyến tính cũng như vector có phân hạng logarithm bằng cách dùng lệnh `linspace`(*start*, *destination*, *number*[3]) và `logspace`. Đối với `logspace`, *start* và *destination* được nhập bởi số mũ thập phân, ví dụ: thay vì nhập 100 (=10^2) ta chỉ cần nhập 2:

[1] Trong mọi trường hợp khác MATLAB sẽ bỏ qua vị trí cách bỏ trống

[2] (*xuất phát*:[*bước*:]*đích*)

[3] Số lượng phần tử của vector

```
>> long = 1:8
long =
     1   2   3   4   5   6   7   8

>> deep = 10:-2:0
deep =
    10   8   6   4   2   0

>> longer = linspace(1,19,10)
longer =
    1   3   5   7   9   11   13   15   17   19

>> increase = logspace(1,2,5)
increase =
   10.0000   17.7828   31.6228   56.2341  100.0000
```

Bằng các hàm ones(*line*, *column*) và zeros(*line*, *column*) ta tạo các ma trận có phần tử là 1 hoặc 0. Hàm eye(*line*) tạo ma trận đơn vị, ma trận toàn phương với các phần tử 1 thuộc đường chéo, tất cả các phần tử còn lại là 0. Kích cỡ của ma trận hoàn toàn phụ thuộc người nhập:

```
>> two_three = ones(2, 3)
two_three =
     1    1    1
     1    1    1
```

Việc truy cập từng phần tử của vector hoặc ma trận được thực hiện bằng cách khai báo chỉ số của phần tử, trong đó cần lưu ý rằng: Chỉ số bé nhất là 1 chứ không phải là 0. Đặc biệt, khi cần xuất từng hàng hay từng cột, có thể sử dụng toán tử (:) một cách rất lợi hại. Nếu dấu (:) đứng một mình, điều ấy có nghĩa là: Phải xuất mọi phần tử thuộc hàng hay cột:

```
>> my_matrix(2,3)
ans =
     7

>> my_matrix(2,:)
ans =
     5    6    7
```

MATLAB có một lệnh rất hữu ích, phục vụ tạo ma trận với chức năng tín hiệu thử, đó là rand(*m*, *n*). Khi gọi, ta thu được ma trận *m* hàng, *n* cột với phần tử mang các giá trị ngẫu nhiên:

```
>> chance = rand(2,3)
chance =
    0.9501    0.6068    0.8913
    0.2311    0.4860    0.7621
```

Khai báo vector và ma trận	
[*x*1 *x*2 ... ; *x*3 *x*4 ...]	Nhập giá trị cho vector và ma trận
start:increment:destination	Toán tử (:)
`linspace` (*start*, *destination*, *number*)	Khai báo tuyến tính cho vector
`logspace` (*start*, *destination*, *number*)	Khai báo logarithm cho vector
`eye` (*line*)	Khai báo ma trận đơn vị
`ones` (*line*, *column*)	Khai báo ma trận với các phần tử 1
`zeros` (*line*, *column*)	Khai báo ma trận với các phần tử 0
`rand` (*line*, *column*)	Khai báo ma trận với các phần tử nhận giá trị ngẫu nhiên

1.2.1 Tính toán với vector và ma trận

Nhiều phép tính có thể được áp dụng cho vector và ma trận. Ví dụ: Phép nhân với ký hiệu (*) được dùng để tính tích của vector hoặc ma trận. Việc chuyển vị của vector và ma trận được thực hiện nhờ lệnh `transpose` hoặc (′). Nếu vector, ma trận là phức, ta dùng thêm lệnh `ctranspose` hoặc (′) để tìm giá trị phức liên hợp. Đối với các giá trị thực, hai lệnh trên cho ra kết quả như nhau:

```
>> two_three * my_matrix'
ans =
     6    18
     6    18
```

Nếu như một trong các phép tính * / ^ cần được thực hiện cho từng phần tử của vector hoặc ma trận, ta sẽ phải đặt thêm vào trước ký hiệu của phép tính đó ký hiệu (.). Phép tính đối với các biến vô hướng luôn được thực hiện cho từng phần tử một:

```
>> two_three ./ my_matrix
ans =
    1.0000    0.5000    0.3333
    0.2000    0.1667    0.1429
```

Phép tính trên cũng có hiệu lực cả khi ma trận có các phần tử phức:

```
>> complex = [1+i 1-i; 2 3]
complex =
```

```
   1.0000 + 1.0000i   1.0000 - 1.0000i
   2.0000             3.0000

>> complex * complex
ans =
   2.0000             5.0000 - 3.0000i
   8.0000 + 2.0000i  11.0000 - 2.0000i

>> complex .* complex
ans =
        0 + 2.0000i        0 - 2.0000i
   4.0000             9.0000
```

Lệnh `diff(`*vector* [*n*]) tính vector sai phân. Bằng lệnh `conv(`*vector*_1, *vector*_2) ta chập hai vector *vector*_1 và *vector*_2. Nếu hai vector cần chập có phần tử là các hệ số của hai đa thức, kết quả thu được sẽ ứng với các hệ số sau khi nhân hai đa thức đó với nhau:

```
>> diff(my_vector)
ans =
     1    1
```

Hai lệnh `inv` và `det` dùng để nghịch đảo ma trận toàn phương và tính định thức của ma trận. Giá trị riêng của ma trận *matrix* được tính bởi lệnh `eig(`*matrix*) và hạng bởi `rank(`*matrix*):

```
>> matrix = [1 2; 4 9]
matrix =
     1    2
     4    9

>> rank(matrix)
ans =
     2

>> eig(matrix)
ans =
    0.1010
    9.8990

>> det(matrix)
ans =
     1
```

Tính toán với vector và ma trận	
`.*  ./  .^`	Các phép tính với từng phần tử
`transpose` (*matrix*) hoặc *matrix*.'	Chuyển vị ma trận *matrix*
`ctranspose` (*matrix*) hoặc *matrix*'	Chuyển vị ma trận *matrix* có phần tử phức liên hợp
`inv` (*matrix*)	Đảo ma trận
`det` (*matrix*)	Tính định thức của ma trận
`eig` (*matrix*)	Tính các giá trị riêng của ma trận
`rank` (*matrix*)	Xác định hạng của ma trận
`diff` (*vector* [*n*])	Tính vector sai phân
`conv` (*vector_1*, *vector_2*)	Chập vector (nhân đa thức)

1.3 Cấu trúc và trường

1.3.1 Cấu trúc

Để thuận tiện cho việc quản lý và sử dụng, ta có thể tập hợp nhiều biến lại trong một cấu trúc. Trong đó mỗi mảng có một tên riêng (một chuỗi ký tự *string*[1]) đặt giữa hai dấu (' ') có kèm theo giá trị. Một cấu trúc được tạo nên bởi lệnh `struct('name_1', value_1, 'name_2', value_2, ...)`:

```
>> my_structure = struct('data', my_matrix, 'size', [2 3]);
```

Việc truy cập vào dữ liệu được thực hiện với dấu cách (.):

```
>> my_structure(2).data = my_matrix.^(-1);

>> my_structure(2).data(1,:)
ans =
    1.0000    0.5000    0.3333
```

Cấu trúc móc vòng

Các cấu trúc đương nhiên cũng có thể được tạo nên móc vòng với nhau. Ví dụ sau đây minh họa khả năng đó: Ta khai báo một cấu trúc có tên là `componist` với mảng đầu tiên có tên là `name`, được gán giá trị là chuỗi ký tự `'Johann Sebastian Bach'`. Một cấu trúc thứ 2 có tên `datum` với 3 mảng `Day`, `Month` và `Year` để cất giữ ngày, tháng và năm sinh. Sau đó ta gán cấu trúc `datum` vào mảng `born` của cấu trúc `componist`:

[1] Xem thêm mục 2.2 về vấn đề *strings*

```
>> componist = struct('name','Johann Sebastian Bach')
componist =
    name: 'Johann Sebastian Bach'

>> datum.Day = 21;
>> datum.Month = 'March';
>> datum.Year = 1685;
>> componist.born = datum;

>> componist
componist =
    name: 'Johann Sebastian Bach'
    born: [1x1 struct]
```

Ta gán cho mảng `name` của cấu trúc `componist` giá trị mới là chuỗi ký tự `'Wolgang Amadeus Mozart'`. Các giá trị của mảng `born` được gán trực tiếp:

```
>> componist(2).name = 'Wolfgang Amadeus Mozart';
>> componist(2).born.Day = 27;
>> componist(2).born.Month = 'January';
>> componist(2).born.Year = 1756;

>> componist(2)
ans =
    name: 'Wolfgang Amadeus Mozart'
    born: [1x1 struct]

>> componist(2).born
ans =
      Day: 27
    Month: 'January'
     Year: 1756
```

Cấu trúc `componist` lúc này mang đặc điểm cấu trúc của các vector, và vì vậy có thể xử lý các phần tử của cấu trúc đó như là các vector. Trong ví dụ vừa nêu, các vector đó chính là hai mảng `name` và `born`:

```
>> componist
componist =
1x2 struct array with fields:
    name
    born
```

1.3.2 Trường

Tổng quát ở một mức cao hơn cấu trúc là *trường* (*Cell Array*). Đó chính là các *Array* (mảng nhiều chiều), chứa *Cell* (tế bào) với dữ liệu thuộc các loại và kích cỡ khác nhau. Ta có thể tạo ra *Cell Array* bằng lệnh `cell`, hoặc đơn giản bằng cách ghép các phần tử bên trong dấu ngoặc { }. Từng phần tử của *Cell Array* có thể được truy cập như các vector, ma trận thông thường hoặc như các *Array* nhiều chiều, chỉ cần lưu ý rằng: Thay vì sử dụng dấu ngoặc tròn ( ) ta sử dụng dấu ngoặc móc { }.

Giả sử ta tạo một *Cell Array* rỗng có tên `my_cell` như sau:

```
>> my_cell = cell(2,3)
my_cell =
     []      []      []
     []      []      []
```

Bây giờ ta lần lượt gán cho từng mảng của `my_cell` các giá trị sau đây, trong đó có cả các phần tử cấu trúc `componist(1)` và `componist(2)` ở mục 1.3.1:

```
>> my_cell{1,1} = 'The first Cell (1,1) contains a text';
>> my_cell{1,2} = 10;
>> my_cell{1,3} = [1 2; 3 4];
>> my_cell{2,1} = componist(1);
>> my_cell{2,2} = componist(2);
>> my_cell{2,3} = date;
```

Khi nhập tên của *Cell Array* trên màn hình hiện lên đầu đủ cấu trúc của nó. Có thể biết nội dung (hay giá trị) của một hay nhiều *Cell* khi ta nhập các chỉ số của *Cell*:

```
>> my_cell
my_cell =
    [1x36 char  ]    [        10]     [2x2 double]
    [1x1  struct]    [1x1 struct]    '12-Apr-2002'

>> my_cell
my_cell =
    [1x36 char  ]    [        10]     [2x2 double]
    [1x1  struct]    [1x1 struct]    '12-Apr-2002'

>> my_cell{2,3}
ans =
12-Apr-2002
```

```
>> my_cell{2,1:2}
ans =
    name: 'Johann Sebastian Bach'
    born: [1x1 struct]
ans =
    name: 'Wolfgang Amadeus Mozart'
    born: [1x1 struct]

>> my_cell{2,2}.born.Month
ans =
January
```

Cấu trúc (Structure) và trường (Cell Array)	
`struct('n1', 'v1', 'n2', 'v2', ... )`	Khai báo cấu trúc
structure.name	Truy cập vào phần tử *name*
`my_cell = {}`	Tạo *Cell Array* rỗng
`cell(n)`	Tạo $n \times n$ *Cell Array*
`cell(m, n)`	Tạo $m \times n$ *Cell Array*

1.4 Quản lý biến

Kích cỡ của vector hay ma trận được xác định bởi lệnh `size(variable)`. Đối với vector còn có thể dùng lệnh `length(variable)`, và khi sử dụng lệnh đó cho ma trận ta sẽ thu được giá trị của vector mang kích cỡ lớn nhất. Ngoài ra, một biến có thể có kích cỡ là 0, nếu nó đã được tạo nên bởi lệnh `variable = []`.

```
>> length(my_matrix)
ans =
     3

>> size(my_matrix)
ans =
     2     3
```

Bằng lệnh `who` ta có thể kiểm tra được mọi biến đang tồn tại trong *Workspace* nhờ danh mục hiện trên màn hình. Bằng `whos` ta còn biết thêm các thông tin về kích cỡ và nhu cầu bộ nhớ của biến. Bằng lệnh `clear` [*variable_1 variable_2* ...] ta có thể xóa có chủ đích một số biến nhất định, nếu chỉ gọi `clear` ta sẽ xóa toàn bộ biến trong *Workspace*.

```
>> whos
  Name            Size              Bytes  Class

  ans             1x2                  16  double array
  complex         2x2                  64  double array (complex)
  componist       1x2                1320  struct array
  datum           1x1                 398  struct array
  deep            1x6                  48  double array
  increase        1x5                  40  double array
  long            1x8                  64  double array
  longer          1x10                 80  double array
  my_cell         2x3                2070  cell array
  my_matrix       2x3                  48  double array
  my_structure    1x2                 456  struct array
  my_vector       1x3                  24  double array
  two_three       2x3                  48  double array

Grand total is 276 elements using 4676 bytes
```

Có thể sử dụng cửa sổ *Workspace Browser* (hình 1.1, gọi qua menu *Show Workspace* hoặc bằng lệnh `workspace`) để xem thông tin về các biến, cũng có thể xem bằng lệnh `help elmat`.

Quản lý biến	
`size` (*variable*)	Kích cỡ của biến
`length` (*variable*)	Chiều dài của vector, kích cỡ lớn nhất trong ma trận
`clear`	Xóa tất cả biến
`clear` [*variable_1 variable_2*]	Chỉ xóa các biến trong ngoặc [ ]
`Who`	Danh mục các biến trong Workspace
`Whos`	Danh mục chi tiết các biến trong Workspace, có kèm theo tên, kích cỡ, dung lượng choán bộ nhớ và loại dữ liệu

1.5 Các phép so sánh và phép tính logic

Các phép tính logic có thể được sử dụng cho tất cả các số. Khi tính, các giá trị khác 0 ứng với logic *true* và giá trị 0 ứng với logic *false*. Khi xuất giá trị lên màn hình ta sẽ chỉ thu được các số 0 hoặc 1.

MATLAB cung cấp đầy đủ các phép so sánh và tính logic VÀ (AND), HOẶC (OR), PHỦ ĐỊNH (NOT) và HOẶC LOẠI TRỪ (exclusive OR), thông thường biểu diễn bằng một ký hiệu hay dấu ngoặc đơn. Ví dụ phép *a* VÀ *b*: *a* & *b*, hoặc and(*a*, *b*). Các phép tính được thực hiện theo trình tự: Trước hết là các biểu

thức toán, tiếp theo là biểu thức logic. Tuy nhiên, khi có cảm giác không chắc chắn, bạn đọc nên dùng cách viết với dấu ngoặc đơn.

```
>> test = 1;
>> test ~= 1
ans =
     0

>> (4>5-2 & 4<=5) | ~test
ans =
     1
```

Ví dụ tiếp theo cung cấp cho ta bảng kết quả khi sử dụng các phép tính logic AND, OR, exclusive OR, NOT, NAND và NOR đối với hai số nhị phân *a* và *b*.

```
>> a = [0 0 1 1]';
>> b = [0 1 0 1]';

>> logtab = [a b a&b a|b xor(a,b) ~a ~(a&b) ~(a|b)]
logtab =
     0   0   0   0   0   1   1   1
     0   1   0   1   1   1   1   0
     1   0   0   1   1   0   1   0
     1   1   1   1   0   0   0   0
```

Một lệnh rất hữu ích là `exist` (*variable*) giúp kiểm tra xem trong *Workspace* có tồn tại biến hay hàm nào tên là *variable* hay không: Nếu không, ta thu được kết quả là số 0, nếu kết quả là số khác 0, đó chính là số nói lên bản chất của *variable*. Ví dụ: 1 nói rằng *variable* là biến trong *Workspace*, 2 nói rằng *variable* là một MATLAB File trong thư mục MATLAB, 7 nói rằng *variable* là một thư mục[1] vv... Có thể xem danh mục các lệnh nhờ `help ops`.

Phép so sánh			Phép tính logic		
==	eq(a,b)	bằng	~	not(*a*,*b*)	Negation (NOT)
~=	ne(a,b)	khác	&	and(*a*,*b*)	AND
<	lt(a,b)	bé hơn	|	or(*a*,*b*)	OR
<=	le(a,b)	bé hơn hoặc bằng		xor(*a*,*b*)	exclusive OR
>	gt(a,b)	lớn hơn		**Sự tồn tại ?**	
>=	ge(a,b)	lớn hơn hoặc bằng	exist('*x*')	Tìm sự tồn tại của *x*	

[1] Có thể xem danh mục chi tiết ở phần Help của `exist`

1.6 Rẽ nhánh và vòng lặp

1.6.1 Lệnh rẽ nhánh `if` và `switch`

Bằng các phép so sánh và logic ở mục trước, ta có thể đưa ra được các quyết định, phân biệt các trường hợp. Để làm điều đó, MATLAB có các lệnh sau đây:

- `if` *term command* [`elseif` *term command* ...] [`else` *command*] `end`
- `switch` *term* `case` *term command* [...] [`otherwise` *command*] `end`

```
>> test = 5;
>> if test<=2; a=2, elseif test<=5; a=5, else a=10, end;
a =
     5
>> switch test case 2; a=2, case {3 4 5}; a=5, otherwise
a=10, end;
a =
     5
```

Trong cả hai trường hợp trên, các lệnh con được ngăn cách bởi dấu (;) và dấu (,). Trong các *Scripts*, thường ta hay viết mỗi lệnh con trong một dòng riêng. Ngoài ra, ta cũng có thể viết nhiều cấu trúc `if` và `switch` móc vòng, đan xen lẫn nhau.

1.6.2 Vòng lặp sử dụng `for` và `while`

Bằng vòng lặp ta có thể thực hiện lặp lại nhiều lần một số lệnh nhất định:

- `for` *variable* = *term command* `end`
- `while` *term command* `end`

Trong cả hai trường hợp, lệnh `break` đều có tác dụng kết thúc vòng lặp.

```
>> for k=1:0, k^2, end;
>> n = 1;
>> while 1, n=n+1; m=n^2, if m>10, break; end; end
m =
     4
m =
     9
m =
    16
```

Trong ví dụ trên vòng lặp `for` đã không hề được thực hiện vì phạm vi 1:0 của `k` là phạm vi rỗng và điều kiện ngừng được kiểm tra trước. Ngược lại, cũng trong ví dụ đó vòng lặp `while` đã được thực hiện ít nhất 1 lần, vì điều kiện ngừng chỉ được kiểm tra sau cùng. Ngoài ra, vòng `while` cần hai lệnh `end` để kết thúc.

1.6.3 Gián đoạn bằng `continue` và `break`

Hai lệnh hữu ích rất hay được sử dụng để điều khiển chu trình tính toán là `continue` và `break`. Trong vòng lặp `for` hay `while`, khi gọi `continue` ngay lập tức chu trình tính chuyển sang bước lặp (*iteration step*) kế tiếp, mọi lệnh chưa thực hiện của vòng lặp (thuộc về bước lặp hiện tại) sẽ bị bỏ qua.

Lệnh `break` còn mạnh hơn: Ngừng vòng lặp đang tính. Lệnh `break` có tác dụng cả trong các cấu trúc rẽ nhánh dùng `if`, `switch`. Nếu `break` được sử dụng ngoài vòng `for`, `while` trong phạm vi của một *script file* hay *function* của MATLAB, khi ấy *script file* và *function* sẽ bị ngừng tại vị trí của `break`.

Ví dụ sau sẽ minh họa các nội dung vừa đề cập đến trong mục này với nội dung: Cần kiểm tra xem trong các số nguyên thuộc khoảng 3 – 7, số nào là số nguyên tố. Việc kiểm tra được thực hiện ở mạch vòng bên ngoài.

```
>> for m = 3:1:7,
       for n = 2:1:m-1,
           if mod(m,n) ~= 0, continue, end
           fprintf(' %2d is not a prime number!\n', m)
           break
       end % n
       if n == m-1,
           fprintf('!! %2d is a prime number!\n', m)
       end % if
   end % m
```

Mạch vòng trong có nhiệm vụ: Lần lượt chia số cần kiểm tra m cho tất cả các số trong khoảng từ 2 đến (m-1), sau đó kiểm tra xem số dư `mod`(m, n) của phép chia có khác 0 hay không. Nếu số dư bằng 0, khi ấy m chia hết cho n và lệnh `continue` không được gọi, lệnh `fprintf`[1] xuất thông báo lên màn hình. Nếu số dư khác 0, khi ấy m không chia hết cho n và lệnh `continue` có hiệu lực, lệnh `fprintf` và `break` bị bỏ qua để chuyển sang kiểm tra vòng lặp mới với n lớn hơn. Nếu m không chia hết cho các số trong khoảng từ 2 đến (m-1), mà chỉ chia hết cho 1 và bản thân m, khi ấy m là số nguyên tố. Việc kiểm tra n == m-1 là cần thiết, vì nếu m không phải là số nguyên tố, và vì vậy vòng lặp phía trong đã được rời bỏ bởi lệnh `break` để tiếp tục các lệnh thuộc vòng lặp phía ngoài.

[1] Lệnh fprintf được giới thiệu kỹ hơn ở mục 2.3

THƯ VIỆN HỌC VIỆN KỸ THUẬT QUÂN SỰ
Vc 3898 (3898)
20
PHỤ BẢN

MATLAB đưa ra kết quả trên màn hình như sau:

```
!!  3 is a prime number!
    4 is not a prime number!
!!  5 is a prime number!
    6 is not a prime number!
!!  7 is a prime number!
```

Để xem tất cả các lệnh tạo khả năng điều khiển chu trình tính toán, bạn đọc hãy gọi `help lang`.

Rẽ nhánh, vòng lặp và điều khiển chu trình tính	
`if` ... , [`elseif` ...] [`else` ...] , `end`	Lệnh `if`
`switch` ... , `case` ... [`otherwise` ...] , `end`	Lệnh `switch`
`for` *variable* = *term* , *command* , `end`	Vòng lặp `for`
`while` *term command* `end`	Vòng lặp `while`
`continue`	Lập tức chuyển sang bước tiếp theo của vòng lặp
`break`	Lập tức kết thúc vòng lặp

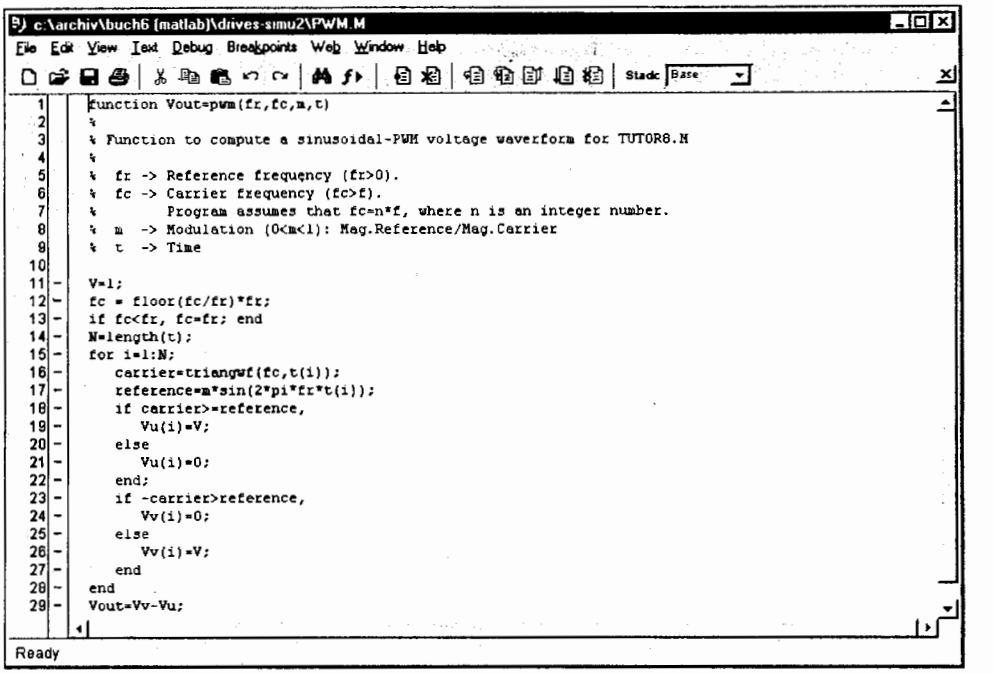
1.7 Các Scripts và các hàm của MATLAB

1.7.1 Các Scripts của MATLAB

Bên cạnh khả năng nhập lệnh trực tiếp, ta có thể viết và cất nhiều chuỗi lệnh trong các *scripts* của MATLAB dưới dạng *file* với ký tự ASCII (*m-file*). Một *script* được khai báo tên không có đuôi *.m*. Để soạn thảo các *file* đó ta có thể sử dụng trình soạn thảo của MATLAB bằng cách gọi menu *File/New/M-File* hoặc *File/Open*. Cũng có thể gọi trực tiếp nhờ nút nhấn trên cửa sổ MATLAB. Giả sử bạn đọc chưa cài đặt trình soạn thảo đó, có thể sử dụng bất kỳ trình soạn thảo ASCII nào khác cũng được.

Hình 1.3 minh họa ví dụ *file pwm.m*, có nhiệm vụ tính toán điều chế điện áp hình sin. File *pwm.m* giữ vai trò của một hàm chức năng, được một *m-file* khác có tên là *tutor8.m* gọi. Trong ví dụ trên ta có thể thấy rõ phần thuyết minh (mô tả bằng lời) các dòng lệnh đều được cách ly bởi dấu %, nghĩa là: Mọi ký tự nằm bên phải dấu % cho đến hết dòng đều bị MATLAB bỏ qua.

Vì một dòng lệnh có thể trở nên quá dài, người sử dụng có thể xuống dòng (chưa kết thúc) bằng dấu ...



Hình 1.3 Trình soạn thảo của MATLAB với ví dụ file PWM.M

1.7.2 Các hàm của MATLAB

Một dạng đặc biệt của *m-files* là các hàm của MATLAB (các `function`). Khi gọi một `function` ta có thể chuyển giao dữ liệu cho `function` hay nhận dữ liệu do `function` đó trả lại. Ngoài ra, một `function` cũng có thể được các `function` khác hay *script* gọi, và một *script* cũng có thể được các *scripts* gọi.

Các biến trong phạm vi một `function` là biến *local* (cục bộ). Các biến *global* (toàn cục, có giá trị sử dụng chung) được định nghĩa bởi lệnh `global` *variable*... Lệnh định nghĩa đó phải được gọi trực tiếp từ *Command Windows* của MATLAB, hay từ một *script*, và cũng có thể định nghĩa trong phạm vi một `function`.

Trong phạm vi `function` ta có thể sử dụng hai biến `nargin` và `nargout` để xác định số lượng dữ liệu được chuyển giao hay nhận trở lại.

Hàm trong ví dụ sau được dùng để tính giá trị trung bình, cất trong *m-file* nào đó và có dạng như sau:

```
function [arithm, geom] = average_value (x)
arithm = mean(x);                % arithmetic average value
geom = prod(x)^(1/length(x));    % geometric average value
```

Bằng việc gọi `average_value(test)` ta sẽ chỉ nhận được dữ liệu thứ nhất, vì vậy việc gọi để nhận đầy đủ dữ liệu cần sẽ được thực hiện như sau:

```
>> test = 1:2:7;
>> [A, G] = average_value(test)
A =
     4
G =
     3.2011
```

Nếu một MATLAB *script* hay một MATLAB `function` lần đầu tiên được gọi, MATLAB sẽ dịch ra mã ảo (*Pseudo-Code*[1]), là mã sẽ được kích hoạt để thực hiện nhiệm vụ đặt ra cho *script* hay của `function`. Nếu về sau không có sự thay đổi gì trong *m-file*, quá trình dịch sẽ không xảy ra lần thứ hai. Bằng lệnh `clear functions` ta có thể xóa cưỡng bức các hàm đã dịch, đồng thời giữ nguyên các *m-files*.

Các Scripts và các hàm	
`%`	Dấu cách phần thuyết minh với lệnh
`function` [*out*, ...] = *name* (*in*, ...)	Định nghĩa hàm của MATLAB
`nargin`	Xác định số lượng dữ liệu vào
`nargout`	Xác định số lượng dữ liệu ra
`clear function`	Xóa các hàm đã dịch

1.8 Tóm tắt nội dung chương 1

Nhằm giúp bạn đọc hệ thống hóa được nội dung, sau mỗi chương sẽ có phần tóm tắt các vấn đề đã được giải quyết tại chương đó. Các vấn đề đã được trả lời ở chương 1 là:

1. Làm thế nào để khởi động MATLAB ?
2. *Workspace* là gì ?
3. Làm thế nào để định nghĩa một biến ?
4. Các quy định phải tuân theo khi đặt tên biến ?
5. Ý nghĩa của biến *ans* ?
6. Làm thế nào để thực hiện các phép tính cộng, trừ, nhân, chia và lũy thừa ?
7. Làm thế nào để khai báo một số phức ?
8. Làm thế nào để khai báo một vector ?
9. Làm thế nào để khai báo một ma trận ?
10. Làm thế nào để truy cập tới tận từng phần tử của vector hay ma trận ?

[1] Có thể tạo *Pseudo-Code* bằng lệnh `pcode`, sau đó cất dưới dạng *p-file*

11. Những lệnh nào có tác dụng tạo ra: Ma trận với mọi phần tử là 1 hoặc 0, ma trận đơn vị, ma trận ngẫu nhiên ?
12. Dấu (.) đứng trước các dấu của phép toán có tác dụng gì khi tính với vector hoặc với ma trận ?
13. Các đặc điểm của cấu trúc `struct` là gì ?
14. Làm thế nào để khai báo một `struct` ?
15. Làm thế nào để truy cập vào một `struct` ?
16. Làm thế nào để hiển thị các biến trong *Workspace* ?
17. Làm thế nào để xóa các biến ?
18. Có những phép so sánh gì trong MATLAB ?
19. Có những phép tính logic nào trong MATLAB ?
20. Có những khả năng rẽ nhánh nào trong MATLAB ?
21. Có những dạng vòng lặp nào trong MATLAB ?
22. Sự khác nhau giữa *script* và hàm `function` trong MATLAB ?
23. Làm thế nào để tạo ra và sử dụng một `function` ?
24. Sử dụng tiện ích trợ giúp *Help* khi cần tra cứu một lệnh như thế nào ?

2 Xuất và nhập dữ liệu trên màn hình

Tất cả các lệnh giới thiệu trong chương này có tác dụng điều khiển màn *hình*, phục vụ xuất và nhập dữ liệu theo phương thức đối thoại với người sử dụng, cũng như xuất và nhập khẩu dữ liệu. Nội dung quan trọng thứ hai là vấn đề biểu diễn bằng đồ thị, cũng như việc xuất và nhập khẩu đồ họa thuộc các chuẩn (*format*) khác nhau.

2.1 Điều khiển xuất ra màn hình

Về mặt cú pháp, các lệnh điều khiển xuất ra màn hình MATLAB khá giống với các lệnh tương tự của UNIX. Chúng trước hết được sử dụng vào mục đích kiểm tra, phát hiện lỗi chương trình, hay để tìm hiểu một số mối liên quan nhất định.

Nếu cần thực hiện việc *xuất ra màn hình theo từng trang* một, ta có lệnh `more`. Bằng `more on` ta đóng và `more off` ta ngắt chế độ xuất ra theo từng trang, `more` thay đổi trạng thái và `more(n)` hiển thị *n* dòng của một trang. Việc điều khiển giống như trong hệ điều hành UNIX: Nếu lượng xuất ra dài hơn một trang, khi ấy phím *Return* có tác dụng nhích lên từng dòng, phím trống hiển thị trang kế tiếp và phím chữ *Q* sẽ ngừng việc xuất.

Bằng `echo` ta có thể *hiển thị tất cả các lệnh* đã được thực hiện sau khi gọi một *script* hay một `function`. Lệnh `echo on` có tác dụng đóng, `echo off` có tác dụng ngắt chế độ hiển thị lệnh của *script*. Lệnh `echo` *function* `on` sẽ hiển thị các lệnh (xem ví dụ sau đây: sử dụng hàm `average_value(x)` ở mục 1.7.2) do hàm có tên *function* gọi, `echo` hoặc `echo` *function* sẽ chuyển màn hình sang trạng thái khác.

```
>> echo average_value on
>> [A, G] = average_value(1:2:7)
arithm = mean(x);                 % arithmetic average value
geom = prod(x)^(1/length(x));  % geometric average value
A  =
     4
G =
     3.2011
```

Đôi khi, để phục vụ công tác lưu trữ và xử lý sau này ta rất cần *cất dữ liệu đã nhập hoặc xuất* tại *Command Windows* dưới dạng *File*. Để ghi tất cả động

thái nhập / xuất (không kể đồ họa) qua bàn phím vào *File* với định dạng ASCII ta sử dụng lệnh `diary('file')`[1]. Nếu đã tồn tại *File*, các động thái mới sẽ được ghi nối tiếp thêm vào đuôi. Lệnh `diary on` kích hoạt ghi, `diary off` ngừng và `diary` sẽ chuyển đổi giữa hai chế độ.

Có thể *tạm ngừng chế độ xuất màn hình* bằng lệnh `pause`, chế độ tạm ngừng đó sẽ bị hủy bỏ nếu xảy ra động tác nhấn phím mới bất kỳ. Lệnh `pause(n)` có tác dụng tạm ngừng trong *n* giây, `pause off` vô hiệu hóa tất cả các lệnh `pause` tiếp theo và `pause on` lại cho phép `pause` có tác dụng trở lại.

Trong ví dụ sau đây, cứ sau mỗi vòng lặp bộ đếm lại được xuất ra màn hình và đợi khoảng thời gian (tính bằng giây) ứng với nội dung bộ đếm. Cuối cùng ta thoát khỏi trạng thái tạm ngừng bằng một động thái bàn phím.

```
>> for i=1:2:6, disp(i), pause(i),end ,disp('Ende'),pause
     1
     3
     5
Ende
>>
```

Ngoài ra còn có một lệnh rất hữu ích là `clc` (*Clear Command Windows*) có tác dụng xóa tất cả các nhập / xuất trên *Command Windows* và đưa *Cursor* về dòng đầu tiên của cửa sổ.

Điều khiển xuất ra màn hình	
`more`	Xuất từng trang một ra màn hình
`echo`	Hiển thị các lệnh đã gọi trong *Scripts* (*m-File*) và *Functions*
`diary('file')`	Cất các thao tác xuất ra màn hình trong *File* có tên *'file'*
`pause`	Tạm ngừng xuất ra màn hình
`clc`	Xóa mọi lưu giữ về thao tác nhập / xuất ra màn hình

2.2 Đối thoại của MATLAB với người sử dụng

Các đối thoại với MATLAB, cần thiết khi nhập / xuất văn bản (*Text*) và dữ liệu (*Data*), được thực hiện nhờ các lệnh mô tả trong mục này. Trước hết ta hãy xem xét việc xử lý văn bản trong MATLAB.

[1] Tên của *File* chuẩn là `'diary'`. Bằng lệnh `get(0, 'DiaryFile')` ta có thể hỏi xem *File* ghi hiện tại có tên gì

2.2.1 Văn bản (Text) trong MATLAB

Các đoạn văn bản (các chuỗi ký tự, *Strings*) thường được bao trong phạm vi hai dấu móc trên cao và có thể được gán cho các biến theo kiểu: *string* = *'text'*. Các biến *String* được cất dưới dạng vector và có thể tập hợp lại bằng cách viết [*'text*1','*text*2']. Nếu một *Script* có chuỗi ký tự với nội dung kéo dài nhiều dòng, chuỗi ký tự đó phải được kết thúc sau mỗi dòng, nhằm loại trừ khả năng dấu ngắt dòng ... được MATLAB coi là một phần tử của chuỗi ký tự đó.

```
>> text = ['This is',' ','a Text']
text =
This is a Text

>> whos text
  Name        Size            Bytes  Class
  text        1x14               28  char array
Grand total is 14 elements using 28 bytes
```

2.2.2 Đối thoại khi nhập văn bản

Việc hỏi tìm số liệu được thực hiện bởi lệnh *variable* = `input (`*string*`)`. Chuỗi ký tự *'string'*được xuất ra màn hình và đồng thời gán cho biến có tên *'variable'*. Nếu trong dữ liệu hỏi ta không tìm số liệu mà tìm chuỗi ký tự, khi ấy lệnh có dạng *string* = `input (`*string*`,'s')`. Trong khi thực hiện lệnh, có thể sử dụng một số ký hiệu đặc biệt như dấu ngắt xuống dòng `\n`, dấu móc trên cao `''`[1] và gạch nghiêng chéo `\\`[2].

Sau đây là ví dụ về một *Script* có khúc đối thoại nhập văn bản:

```
price = input (['What does the exchange rate \n', ...
                'look like today ?        ']);
currency = input ('DEM\\EUR ?                ',  's');
```

Sau khi gọi *Script* và nhập số liệu của năm 2001, ví dụ ta có thể nhận kết quả như sau.

```
What does the exchange rate
look like today ?              1.93
DEM\EUR ?                      DEM
```

[1] Khi thực hiện ta thu được trên màn hình duy nhất một dấu '

[2] Khi thực hiện ta thu được trên màn hình duy nhất một dấu \

2.2.3 Xuất theo định dạng

Khi xuất số liệu hay chuỗi ký tự ra màn hình ta đều có thể đặt định dạng cho việc xuất. Lệnh `disp(`*string*`)` có tác dụng xuất chuỗi ký tự. Lệnh này có thể sử dụng cho chuỗi ký tự chứa văn bản động, trong đó các chữ số phải được biến thành chuỗi ký tự nhờ lệnh `num2str(`*variable*`[,`*format*`])`.

Đối với số liệu vector ta có thể lập định dạng xuất nhờ *string* = `sprintf(`*string*`,`*variable*`)`, sau đó xuất bằng lệnh `disp`. Khi sử dụng hai lệnh `num2str` và `sprintf`, cú pháp lập định dạng về cơ bản cũng giống như ngôn ngữ lập trình C (xem thêm thông tin bằng lệnh `help sprintf`). Tất cả các biến phải được khai báo chung trong một ma trận duy nhất, và mỗi cột là một bộ tham số xuất. Ta hãy tiếp tục ví dụ ở mục trước:

```
disp(['However ',num2str(price, '%0.2f'), '', currency, ...
      ' is already very expensive ! ']);
disp(' ');                                        % Dòng bỏ trống
pause                                             % Tạm ngừng
display = sprintf('Two Euros cost %2.2f %s. ', ...
                  price*2, currency);
disp(display);
once_more = sprintf('%4d Euros cost %2.2f. \n', ...
                    [3:5; (3:5)*price]);
disp(once_more);
```

Kết quả thu được là:

```
However 1.93 DEM is already very expensive !

Two Euros cost 3.86 DEM.
  3 Euros cost 5.79 DEM.
  4 Euros cost 7.72 DEM.
  5 Euros cost 9.65 DEM.
```

Bên cạnh các khả năng vừa được giới thiệu, MATLAB còn tạo điều kiện xây dựng các giao diện với người sử dụng bằng đồ họa (GUI[1]), mô tả ở mục 2.9.

Đối thoại khi nhập văn bản			
variable	=	`input(`*string*`)`	Hỏi tìm số liệu của biến
string	=	`input(`*string*`, 's')`	Hỏi tìm chuỗi ký tự
		`num2str(`*variable*`[,`*format*`])`	Biến một số thành chuỗi ký tự
		`disp(`*string*`)`	Xuất chuỗi ký tự ra màn hình
string	=	`sprintf(`*string*`,` *variable*`)`	Tạo định dạng cho chuỗi ký tự

[1] GUI: Graphical User Interface

Ký hiệu đặc biệt	
\n	Ngắt xuống dòng
\\	Gạch nghiêng trái
' '	Dấu móc đơn trên cao

Định dạng	
%2d	Số nguyên (ví dụ: 21)
%6.3f	Số dấu phẩy động (ví dụ: 65,321)
%s	Chuỗi ký tự

2.3 Nhập và xuất dữ liệu

2.3.1 Cất vào hoặc gọi dữ liệu từ File

Tùy sự lựa chọn, có thể thông qua *File* để nhập hoặc xuất dữ liệu dưới dạng mã ASCII hoặc mã nhị phân. Để cất hoặc gọi dữ liệu ta dùng một trong hai lệnh `load` *file_name* [*variable_1 variable_2 ...*] và `save` *file_name* [*variable_1 variable_2 ...*]. Giả sử tên *File* đã được cất trong biến ký tự *string*, khi ấy ta chỉ cần viết ngắn gọn `load` (*string*) và `save` (*string*, [*variable_1, variable_2 ...*]).

Nếu sau `load` ta viết tên một *File* không có đuôi (phần sau dấu chấm .), khi ấy MATLAB sẽ chọn một *File* với định dạng dữ liệu theo mã nhị phân (các *File* có đuôi MAT). Một *File* MAT bao giờ cũng cất đầy đủ các giá trị và tên của biến. Để gọi số liệu ở dạng mã ASCII, ta phải viết tên đầy đủ của *File*, tức là phải có cả phần đuôi. Các số liệu thuộc cùng một dòng phải được viết cách nhau nhờ phím trống hoặc phím Tab (không dùng dấu phẩy). Mỗi dòng phải chứa nhiều phần tử (được phép chèn cả thuyết minh). Khi gọi (đọc dữ liệu), các giá trị sẽ được gán cho biến có tên trùng với tên của *File*.

```
>> test_vector = [0:0.1:10]';      % Column vector
>> test_matrix = [test_vector cos(test_vector)];   % Matrix
>> save test                       % Save into file test.mat
>> clear                           % Clear Workspace
>> load test                       % Load from file test.mat
>> who                             % Display Workspace

Your variables are:
test_matrix  test_vector
```

Khi cần cất ở dạng mã ASCII ta sẽ phải bổ sung vào dòng lệnh `save` thêm khóa `-ascii`. Trong trường hợp này, tên của các biến sẽ không được cất kèm theo và giá trị của các biến được ghi tuần tự nối đuôi nhau vào *File*. Do không cất tên của các biến, sau này khi gọi ra, MATLAB sẽ không có khả năng phân loại các giá trị theo biến nữa.

```
>> save test.txt -ascii test_matrix % Save into file test.txt
>> clear                           % Clear Workspace
>> load test.txt                   % Load from file test.txt
>> who                             % Display Workspace

Your variables are:
test
```

Nếu khi gọi hoặc cất ta không viết cụ thể tên các biến, khi ấy MATLAB sẽ gọi tất cả các biến đã được cất trong MAT-*File*, hoặc cất vào MAT-*File* mọi biến đang tồn tại trong *Workspace*.

2.3.2 Cất có định dạng vào File văn bản

Khi xuất một *File* văn bản ta cũng có thể sử dụng lệnh `fprintf` với cú pháp giống như của `sprintf`. Tuy vậy, bằng `fprintf` ta chỉ có thể xử lý số thực, để xử lý số phức, ta phải chuyển số sang thành phần thực và ảo riêng rẽ. Để mở và đóng một *File* xuất, ta có các lệnh *id* = `fopen`(*file.extension*, `'w'`) và `fclose`(*id*), trong đó *id* phục vụ nhận dạng *File* đã mở. Giả sử *File* ví dụ ở hai mục 2.2.2 và 2.2.3 có tên *exchange.txt* (với nội dung về tỷ giá quy đổi) , việc xuất *File* đó sẽ như sau:

```
>> exchange_id = fopen('exchange.txt','w');
>> fprintf (exchange_id, '%4d Euros cost %2.2f.\n', ...
                         [3:5; (3:5)*price]);
>> fclose (exchange_id);
```

Để xem thông tin chi tiết về các lệnh phục vụ việc truy cập *File* ta sử dụng lệnh `help iofun`.

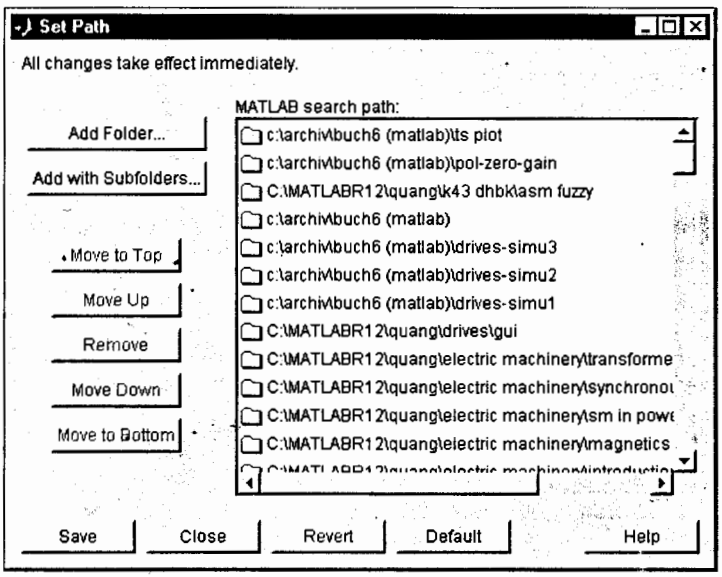
Nhập và xuất khẩu dữ liệu		
	load *file* [*variable*...]	Đọc từ MAT-*File*
	save *file* [*variable*...]	Cất vào MAT-*File*
	load *file.extension* [*variable*...]	Đọc từ ASCII-*File*
	save *file.extension* -ascii [variable...]	Cất vào ASCII-*File*
	fprintf (*f_id*,*string*,*variable*)	Ghi vào ASCII-*File*
f_id =	fopen (*file.extension*,'w')	Mở File
	fclose (*f_id*)	Đóng File

2.4 Hệ điều hành và quản lý File

Thông thường, để MATLAB tìm được các *Scripts* hay dữ liệu, bắt buộc các *Files* liên quan phải nằm tại thư mục (*Folder*, *Directory*) hiện tại. Sau đây bạn đọc sẽ làm quen với một số lệnh dựa trên cơ sở các hệ điều hành phổ biến, cho phép tìm thấy File trong khu rừng dầy đặc các thư mục. Nếu ta đặt dấu chấm than ! ở đầu của một dòng, khi ấy MATLAB sẽ chuyển giao toàn bộ phần còn lại của dòng cho hệ điều hành dưới dạng các lệnh.

Một lệnh tương đối mạnh (nhưng cũng khá rắc rối khi sử dụng) là lệnh `eval(`*string*`)`. `eval` có tác dụng kích hoạt các lệnh mà MATLAB đã trao cho hệ điều hành. Ví dụ sau đây tạo nên thư mục *test_directory*, gán cho biến cấu trúc (biến loại `struct`) *directory* tên của thư mục và sau đó lại xóa thư mục đó đi.

```
>> mkdir ('test_directory')
>> directory = dir ('test_direc*')
directory =
      name: 'test_directory'
      date: '20-Apr-2002 14:36:16'
     bytes: 0
     isdir: 1
>> eval (['!rmdir ',directory.name])
```



Hình 2.1 *Cửa sổ truy cập thư mục (Path Browser) của MATLAB*

Ngoài ra, bạn đọc có thể đi theo menu *View / SetPath* hay trực tiếp gọi `pathtool` từ *Command Windows* để tới được cửa sổ truy cập thư mục *Path*

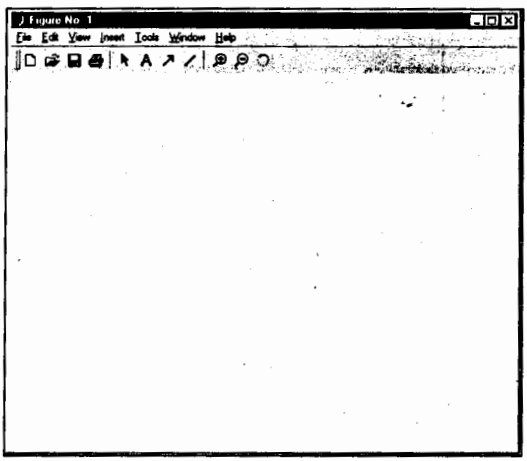
Browser (hình 2.1). Trong cửa sổ đó, có thể nhấn nút *Add Folder* để tạo thêm thư mục mới. Để xem chi tiết bạn đọc hãy gọi `help general`.

Gọi từ hệ điều hành và quản lý File	
`pwd`	Hiển thị thư mục hiện tại
`dir [...]`	Hiển thị nội dung của thư mục [...]
`ls [...]`	Hiển thị nội dung của thư mục [...]
`cd` *directory*	Chuyển thư mục
`mkdir` *directory*	Tạo thư mục mới
`copyfile` *source destination*	Sao chép (copy) file
`delete` *file*	Xóa file
`!` *commando*	Gọi lệnh từ hệ điều hành

2.5 Biểu diễn bằng đồ họa

Khi khối lượng dữ liệu lớn, việc xuất ra màn hình dưới dạng số sẽ ít có ý nghĩa. Việc biểu diễn chúng dưới dạng đồ họa trở nên quan trọng và là đối tượng của mục này. Trước hết ta đề cập đến các lệnh liên quan với gần như mọi quá trình xuất ra màn hình dưới dạng đồ họa, như các lệnh tạo nên, truy cập vào môi trường và vào từng phần tử của đồ họa, ví dụ: tới trục hay việc đặt tên và thứ nguyên cho trục. Trong hai mục tiếp theo 2.6, 2.7 bạn đọc sẽ làm quen với các khả năng tạo đồ thị 2 - 3 chiều từ tập các dữ liệu đã có, minh họa qua ví dụ cụ thể. Mục 2.8 giới thiệu khả năng nhập / xuất và in các loại đồ họa.

2.5.1 Cửa sổ Figure - cơ sở của đồ họa MATLAB



Hình 2.2 *Cửa sổ Figure của MATLAB*

Khuôn khổ của mọi thao tác xuất đồ họa trên nền MATLAB là *Figure*[1]. Có thể tạo ra cửa sổ như hình 2.2 bằng cách gọi lệnh `figure` và mỗi *Figure* sẽ tự động được đánh số.

```
>> figure
>> gcf
ans =
     1
```

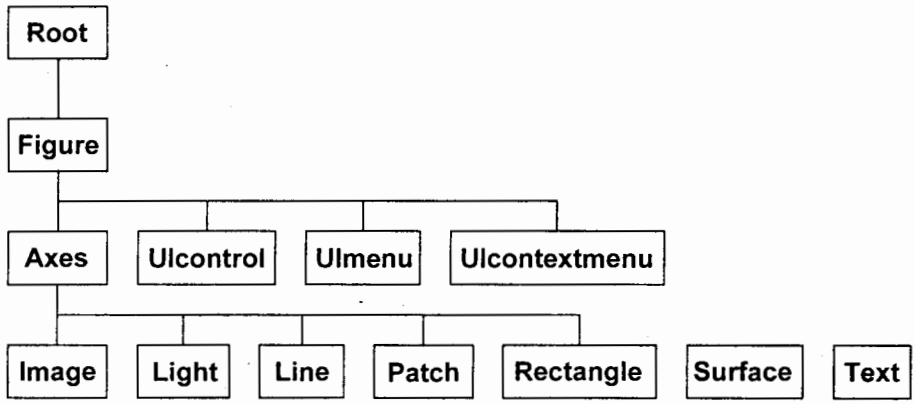
[1] Figure: có nghĩa là "hình"

Có thể gọi một *Figure* đã có số bằng lệnh `figure` (*number*). Số của *Figure* (thực chất là *handle* chứ không phải *number*) sẽ hiển thị nếu ta gọi `gcf` (*Get handle to Current Figure*).

Trong các nút nhấn nằm dưới thanh menu ta cần biết chức năng của các nút sau: nút mở trình *Property Editor* (soạn thảo đặc tính, hình 2.4), nút cho phép ta viết *Text* vào hình, nút và cho phép ta vẽ thêm các nét có hoặc không có mũi tên, và kích hoạt chức năng *Zoom* (dãn, co hình), còn sẽ giúp ta xoay đồ họa, đặc biệt là đồ họa 3 chiều.

Bằng lệnh `subplot(`*row*, *column*, *counter*`)`, có thể chia đều một *Figure* thành nhiều *Subplots* (đồ họa con) được *counter* (bộ đếm) đánh số ở phía trên bên trái. Nếu việc đánh số chỉ cần một chữ số, chúng sẽ được viết tuần tự không cần dấu phẩy hay dấu cách (hình 2.5).

Có thể xóa nội dung của một *Figure* bằng lệnh `clf` (*clear current figure*), và lệnh `delete figure(`*number*`)` sẽ xóa chính *Figure*. Tương tự, lệnh `close(`*number*`)` sẽ đóng *Figure* mang số *number* còn lệnh `close all` sẽ đóng tất cả các *Figures* đang mở.



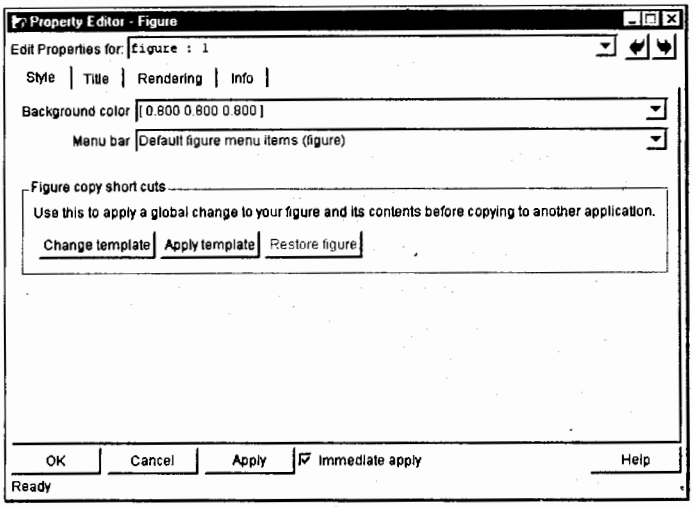
Hình 2.3 *Cấu trúc phân cấp của một đối tượng đồ họa*

Một đối tượng đồ họa có cấu trúc phân cấp được mô tả ở hình 2.3. Có thể xem đặc điểm của một *Figure* (`handle` *FigureHandle*) bằng lệnh `get`, với lệnh `set`[1] ta lại có thể lập đặc điểm cho *Figure*.

`get(`*FigureHandle*`,'`*FigureProperty*`')`
`set(`*FigureHandle*`,'`*FigureProperty*`',`*PropertyValue*`)`

Việc lập trình đồ họa trong MATLAB luôn phụ thuộc vào đối tượng cụ thể và vô cùng phong phú. Một công cụ làm nhẹ bớt công việc là trình soạn thảo đặc tính đồ họa (*Property Editor*) với cửa sổ giới thiệu ở hình 2.4.

[1] Lập (xác định) đặc điểm cho *Figure*: `set(0,'DefaultFigure`*Property*`',`*PropertyValue*`)`



Hình 2.4 *Cửa sổ Property Editor của MATLAB*

Vì khả năng phục vụ xử lý đồ họa của *Property Editor* khá phong phú, vượt khuôn khổ của mục này, đề nghị bạn đọc tra cứu chi tiết tính năng và ý nghĩa của từng mục, từng nút nhấn ở phần trợ giúp trực tuyến (*Online Help*).

Tổng quan về đồ họa	
`figure[(`*number*`)]`	Tạo mới (hay mở) *Figure*
`subplot(`*row*`,`*column*`,`*counter*`)`	Tạo *subplot*
`gcf`	Hỏi số của *Figure* hiện tại
`clf`	Xóa nội dung của *Figure*
`get(`*Handle*`,'`*Property*`')`	Xem đặc tính của *Figure*
`set(`*Handle*`,'`*Property*`',`*Value*`)`	Lập đặc tính cho *Figure*
`delete(figurer(`*number*`))`	Xóa *Figure* mang số *number*
`close(`*number*`)`	Đóng *Figure* mang số *number*
`close all`	Đóng tất cả *Figure*

2.5.2 Trục và điền ký tự cho trục

Việc **phân chia thang bậc của trục** thường được MATLAB tự động thực hiện. Tuy nhiên, ta có thể phân chia thủ công trong trường hợp hai chiều (2-D) bằng lệnh `axis([`*x_min*`,`*x_max*`,`*y_min*`,`*y_max*`])` và trong trường hợp ba chiều (3-D) bằng lệnh `axis([`*x_min*`,`*x_max*`,`*y_min*`,`*y_max*`,`*z_min*`,`*z_max*`])`. Lệnh `axis('auto')` sẽ trao quyền chia trục lại cho MATLAB. Lệnh `grid on` sẽ tạo ra một **lưới tọa độ** ứng với cách chia trục đã xác định (mục 2.6.2 giới thiệu thêm về cách thay đổi lưới tọa độ). Đối với đồ họa 3-D ta có thêm lệnh `box on` để tạo khung bao cho 3-D-*Plot*.

Để điền ký tự vào một đồ họa ta có nhiều khả năng khác nhau: Dùng `xlabel(string)`, `ylabel(string)`, `zlabel(string)` để điền tên cho trục; dùng `title(string)` để điền tên cho *Figure*. Ngoài ra ta còn có thể viết các ký tự lên cao, tụt thấp hay các ký tự Hy Lạp theo cách viết[1] LaTeX. Ví dụ ta viết lệnh: `ylabel('\alpha_a_u_t_o [m/s^2]')`, ta sẽ thu được $\alpha_{auto}[m/s^2]$.

Bằng lệnh `legend(string_1,string_2,...,[position])` ta có thể điền thêm một số **lời ghi chú** vào đồ họa. Vị trí của các lời ghi chú được xác định bởi số ghi trong `[position]`, với ý nghĩa: 1...4 sẽ đặt lời ghi chú vào 4 góc, 0 đặt tự động và -1 đặt vào bên phải, cạnh đồ họa. Lệnh `text(x_value,y_value,string)` cho phép ta **điền một đoạn văn bản** với nội dung *string* vào tọa độ bất kỳ *x_value*, *y_value* trong đồ họa.

Để có thể quan sát kỹ hơn một mảng nào đó trong đồ họa, ta sử dụng lệnh `zoom on` để dùng chuột **cắt và co dãn mảng** đó (hoặc nhấn vào nút tương ứng trong cửa sổ *Figure*, xem hình 2.5). Ngoài ra, cửa sổ *Figure* còn có một vài nút cho phép dùng chuột điền đoạn văn bản, vẽ thêm nét hoặc mũi tên, và mở *Property Editor*.

Có thể xem thêm thông tin chi tiết về xuất đồ họa ra màn hình bằng cách gọi `help graph2d`, `help graph3d` và `help specgraph`.

Đồ họa: Các trục	
`axis([x_min,x_max,y_min,y_max])`	Chia trục 2-D
`axis([x_min,x_max,y_min,y_max,z_min,z_max])`	Chia trục 3-D
`axis('auto')`	Tự động chia trục
`grid [on | off]`	Tạo lưới tọa độ
`zoom [on | off]`	Cắt và co dãn mảng
Đồ họa: Điền ký tự	
`xlabel(string)`	Điền tên trục *x*
`ylabel(string)`	Điền tên trục *y*
`zlabel(string)`	Điền tên trục *z*
`title(string)`	Điền tên *Figure*
`text(x_value,y_value,string)`	Điền văn bản
`legend(string_1,string_2,...,[position])`	Điền lời ghi chú

[1] Cách viết các ký hiệu đặc biệt \n, '' hay \\ (mô tả ở mục 2.2) không có giá trị tại đây

2.6 Đồ họa 2 chiều (2-D Graphics)

2.6.1 Các lệnh vẽ (Plot Commands)

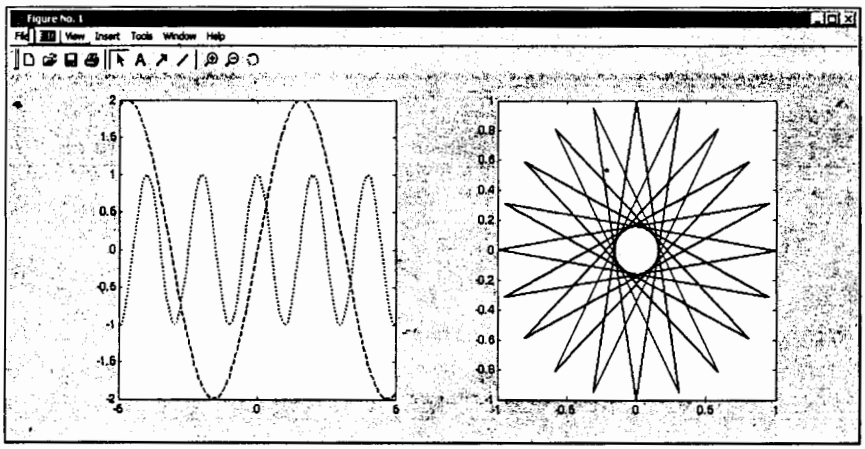
Lệnh `plot(`*x_value*`,`*y_value*`...[,`*plotstyle*`])` vẽ đồ thị nối các điểm cho bởi cặp giá trị *x_value*, *y_value*. Thông thường các điểm đó được nối bởi một nét liền. Nếu ta nạp luân phiên nhiều vector *x*/*y*, ta sẽ thu được nhiều nét nối độc lập với nhau. Nếu thiếu *x_value*, khi ấy các giá trị của *y_value* sẽ được vẽ theo thứ tự chỉ số của chúng. Nếu *y_value* là các giá trị phức, khi ấy đồ thị vẽ với hai trục ảo và trục thực. Lệnh `stars` cũng được viết với cú pháp tương tự nhưng sẽ tạo ra đồ thị có dạng bậc thang (xem các tín hiệu trích mẫu, ví dụ ở mục 8.4).

Chuỗi ký tự *plotstyle* cấu tạo bởi hai thành phần: Thành phần thứ nhất là một chữ cái để chọn mầu và thành phần thứ hai là chuỗi ký hiệu đặc trưng cho dạng chấm / gạch nối tạo nên nét đồ thị. Ví dụ: `'r--'` sẽ tạo nên đồ thị dùng nét đứt và có mầu đỏ (dùng lệnh `help plot` để xem thêm chi tiết).

Mầu			
k	Đen	r	Đỏ
b	Xanh lam	m	Đỏ sẫm
c	Xám	y	Vàng
g	Xanh lá cây	w	Trắng

Nét và điểm			
-	Nét liền	o	Chấm tròn
--	Nét đứt	*	Chấm sao
:	Nét gạch chấm	+	Dấu cộng
.	Nét chấm	×	Dấu nhân

Mỗi lần gọi mới lệnh `plot`, các đồ thị đã có trong *Figure* (hoặc trong *Subplot*) hiện tại sẽ bị xóa. Có thể ngăn chặn điều đó bằng cách gọi lệnh `hold on` sau lệnh `plot` đầu tiên. Đồ họa ở hình 2.5 được tạo nên bởi ví dụ kế tiếp.

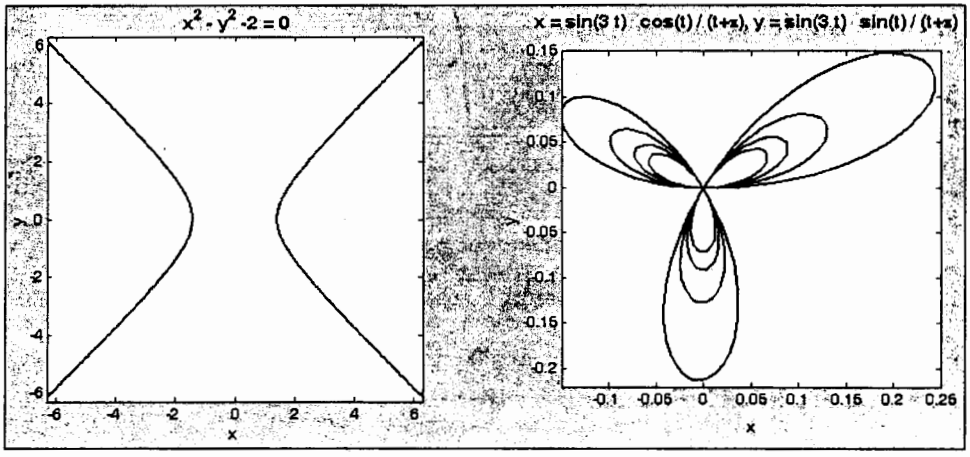


Hình 2.5 *Đồ họa MATLAB có chứa hai Subplots*

```
>> figure;
>> subplot (121);
>>    plot ([-5:0.1:5], cos((-5:0.1:5)*pi), 'k:');
>>    hold on;
>>    fplot ('2*sin(x)', [-5 5], 'k--');
>> subplot (122);
>> t = (0:20)*0.9*pi;
>> plot (cos(t), sin(t));
```

Lệnh `fplot` (*function*, *range*) trong ví dụ trên minh họa khả năng vẽ trực tiếp các hàm tường minh (*explicit*). Ngoài ra, MATLAB còn tạo điều kiện vẽ các hàm không tường minh (*implicit*) một cách dễ dàng nhờ lệnh `ezplot` (*function_1*, [*function_2*,] *range*).

```
>> subplot (121);
>>    ezplot ('x^2 - y^2 -2');
>> subplot (122);
>> axis([-0.15,0.25,-0.22,0.15]);
>>    ezplot ('sin(3*t) * cos(t) / (t+pi)', ...
             'sin(3*t) * sin(t) / (t+pi)', [0, 4*pi]);
```



Hình 2.6 *Hàm implicit (bên trái) và hàm explicit (bên phải: đồ thị có tham số) vẽ bằng lệnh* `ezplot`

Hai lệnh `semilogx` và `semilogy` cũng có cú pháp giống như `plot` với điểm khác duy nhất: Hai trục *x* và *y* được chia thang *logarithm*. Lệnh `loglog` có tác dụng chia đồng thời cả hai trục *x* và *y* theo thang *logarithm*.

Đồ họa: Các lệnh vẽ đồ họa 2-D	
`plot([x_value,]y_value...[,plotstyle])`	Vẽ đồ thị trơn
`stairs([x_value,]y_value...[,plotstyle])`	Vẽ đồ thị bậc thang
`loglog(x_value,y_value...[,plotstyle])`	Vẽ hai trục *logarithm*
`semilogx(x_value,y_value...[,plotstyle])`	Vẽ trục *x logarithm*
`semilogy(x_value,y_value...[,plotstyle])`	Vẽ trục *y logarithm*
`fplot(function, range)`	Vẽ hàm *explicit*
`ezplot(function(x,y)[,range])`	Vẽ hàm *implicit*
`ezplot(function_1,[function_2,][,range])`	Vẽ hàm *implicit* có tham số
`hold [on | off]`	Bảo vệ các đồ thị đã vẽ

2.6.2 Ví dụ: Khâu quán tính bậc nhất PT_1 và khâu tỷ lệ-vi phân PD

Sau đây là một ví dụ sẽ tạo nên đồ thị ở hình 2.7. Ví dụ tính hàm đặc tính tần số của khâu quán tính bậc nhất PT_1 và khâu tỷ lệ-vi phân PD với hằng số thời gian T trên miền ảnh *Laplace*, sau đó vẽ đồ thị của hàm vừa tính được.

$$PT_1: \quad H(s)=\frac{1}{1+sT}; \qquad PD: \quad H(s)=1+sT$$

Trước khi theo dõi ví dụ, xin lưu ý bạn đọc: Bằng lệnh `set` ta thay đổi thang chia của hai trục. Lệnh `gca` (*Get handle to Current Axis*) truy cập trực tiếp tới trục của từng *Subplots* để thay đổi tham số *ytick* và *yticklabel*.

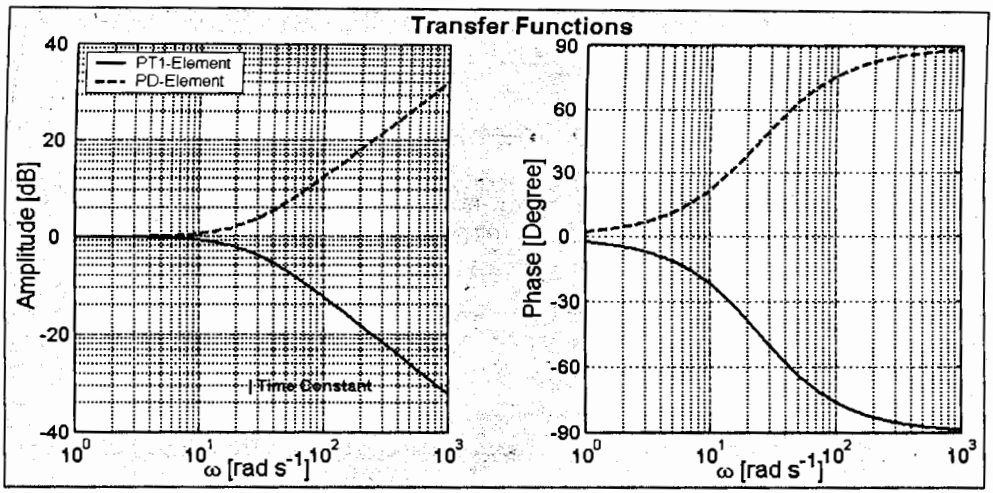
Toàn văn đoạn lệnh sau đây được cất vào *File* có tên là *Fig02_07.m*, sau khi gọi trong *Command Windows* ta thu được đồ thị ở hình 2.7.

```
T       = 0.04;                          % Time constant PT1 [s]
omeg    = logspace (0,3,100);            % Vector frequency[rad/s]
frequency_response =(T*j*omega+1).^(-1); % Frequency response PT1
amplitude_response =abs(frequency_response); % Amplitude response
phase_response = angle(frequency_response);  % Phase response

figure (3);                               % Make figure 3
clf;                                      % Clear all old plots
subplot (121);                            % Subplot: Amplitude
    loglog (omega, amplitude_response, 'b-');
    hold on;
    loglog (omega, amplitude_response.^(-1), 'r--'); %PD: invers
    grid on;
    title ('Transfer Functions');
    xlabel ('\omega [rad s^-^1]');
    ylabel ('Amplitude [dB]');
    legend ('PT1-Element', 'PD-Element', 2);
    text (1/T, 0.03, '| Time Constant');
    axis ([1 1e3 0.01 100]);              % Visible range
    skala = -40:20:40                     % Desired axis scale
```

```
    set (gca, 'ytick', 10.^(skala/20));   % Set axis scale (dB)
    set (gca, 'yticklabel', skala);       % Set label

subplot (122);                            % Subplot: Phase
    semilogx (omega, phase_response*180/pi, 'b-');
    hold on;
    semilogx (omega, -phase_response*180/pi, 'r--'); % PD: invers
    grid on;
    xlabel ('\omega [rad s^-^1]');
    ylabel ('Phase [Degree]');
    axis ([1 1e3 -90 90]);                % Visible range
    skala = -90:30:90;                    % Desired axis scale
    set (gca, 'ytick', skala);            % Set axis.scale(Degree)
    set (gca, 'yticklabel', skala);       % Set label
```

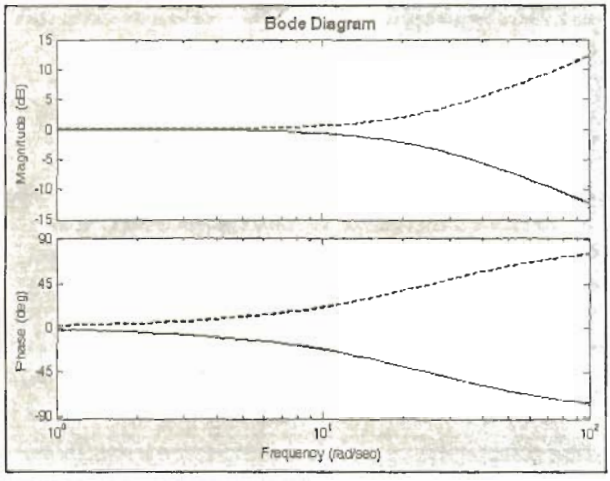


Hình 2.7 *Ví dụ vẽ đặc tính tần số của khâu PT1 (nét liền) và khâu PD (nét đứt)*

Bạn đọc cần biết: Hàm đặc tính tần số còn có thể được tính và biểu diễn nhờ hai lệnh `bode` và `tf` (chương 3).

```
>> pt1 = tf ([1], [0.04 1]);
>> pd = tf ([0.04 1], [1]);
>> bode (pt1,'b-',pd,'r--')
```

Nhờ lệnh `tf` ta có thể tạo nên hàm truyền đạt dưới dạng phân thức (có đa thức mẫu và tử số). Sau đó, lệnh `bode` giúp ta vẽ đồ thị BODE (hình 2.8: nửa trên là đồ thị biên, nửa dưới là đồ thị pha logarithm).

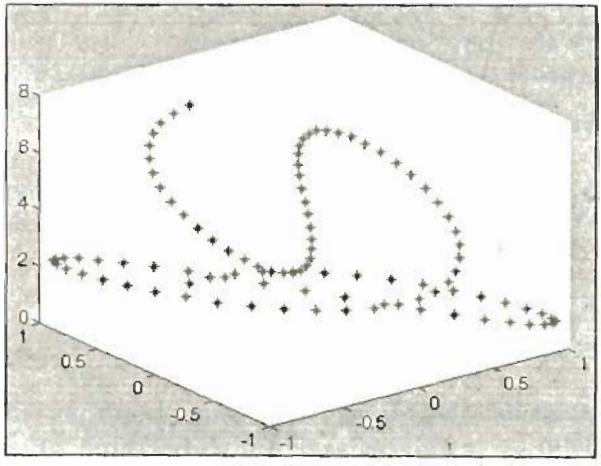


Hình 2.8 *Đồ thị BODE vẽ bằng lệnh bode*

2.7 Đồ họa 3 chiều (3-D Graphics)

Để biểu diễn, minh họa các quan hệ nhiều chiều ta thường sử dụng đồ họa 3-D. Mục này mô tả một số lệnh cơ bản phục vụ mục đích đó. Riêng các lệnh điền ký tự và chia thang cho trục hoàn toàn giống với đồ họa 2-D.

2.7.1 Các lệnh Plots



Hình 2.9 *Đồ họa 3-D vẽ bằng lệnh plot3*

Lệnh `plot3` có tác dụng tương tự như lệnh `plot`, điểm khác duy nhất là `plot3` có thêm vector số liệu thứ ba dành cho trục *z*. Ví dụ ta có thể tạo được đồ họa như ở hình 2.9 khi gọi chuỗi lệnh sau đây:

```
>> phi = (0:100) / 100*2*pi;
>> plot3(sin(2*phi), cos(3*phi), phi, 'b*');
```

Để biểu diễn các hàm 2 chiều dưới dạng mặt trong không gian ta sử dụng lệnh `surf(x_value,y_value,z_value... [, color])`. Nếu *x_value*, *y_value*, *z_value* là các ma trận có số hàng và số cột giống nhau, khi ấy các điểm của đồ họa sẽ được vẽ và nối liền thành mặt.

Nếu các điểm có một khoảng cách đều đặn về phía hai trục *x* và *y*, khi ấy *x_value* và *y_value* có thể chỉ là vector. Trong trường hợp này, các giá trị *x_value* được chuẩn theo cột và *y_value* chuẩn theo hàng của ma trận *z_value*.

Hai lệnh `mesh` và `waterfall` có cú pháp giống như `surf`, nhưng lại tạo ra mặt lưới không điền đầy và đồ họa kiểu thác nước. Ngược lại, `contour` lại vẽ nên các đường "*đẳng mức*" (đường nối các điểm có cùng *z_value*).

Ngoài ra ta còn có thể thêm một ma trận *color* để xác định mầu cho đồ họa. Mỗi phần tử của *color* ứng với một phần tử của *z_value*. Các giá trị mầu sẽ được sử dụng trong một bảng mầu, và ta có thể thay đổi bảng đó nhờ lệnh `colormap(name)`. Nếu không khai báo ma trận mầu, MATLAB sẽ tự động gán *color* = *z_value*. Dải mầu có thể được co dãn thang nhờ lệnh `caxis(color_min, color_max)`. Bạn đọc có thể tìm hiểu kỹ hơn về các bảng mầu có sẵn của MATLAB bằng cách gọi `help graph3d`.

2.7.2 Phối cảnh (Perspective) trong đồ họa 3-D

Có thể dùng lệnh `view(horizontal, vertical)` để phối cảnh cho đồ họa 3 chiều bằng cách khai các góc theo phương nằm ngang (*horizontal*) và phương thẳng đứng (*vertical*) tính bằng độ (°, *Degree*). Góc chuẩn cho trước là (-37.5°, 30°). Ta có thể quan sát các khả năng biểu diễn khác nhau minh họa ở hình 2.10. Ngoài ra, cũng có thể tạo dựng hay thay đổi phối cảnh bằng cách nháy và kéo thả chuột, sau khi đã gọi lệnh `rotate3d`.

2.7.3 Ví dụ về đồ họa 3-D có phối cảnh

Trong ví dụ sau đây, lệnh `meshgrid` có tác dụng tạo nên hai ma trận *X* và *Y* từ hai vector *x* và *y*, có hàng và cột ứng với *x* và *y*. Từ đó MATLAB tính được ma trận *Z*. Toàn bộ ví dụ bao gồm chuỗi lệnh (cất dưới dạng *File* có tên *Fig02_10.m*) sau đây (hình 2.10):

```
x = 0:0.05:2;
y = -1:0.2:1;

[X,Y] = meshgrid(x, y); %Produce matrix over the range of x, y

Z = (Y+1).* cos(2*pi*X.^2) + (Y-1).* sin(2*pi*X.*2) / 5;

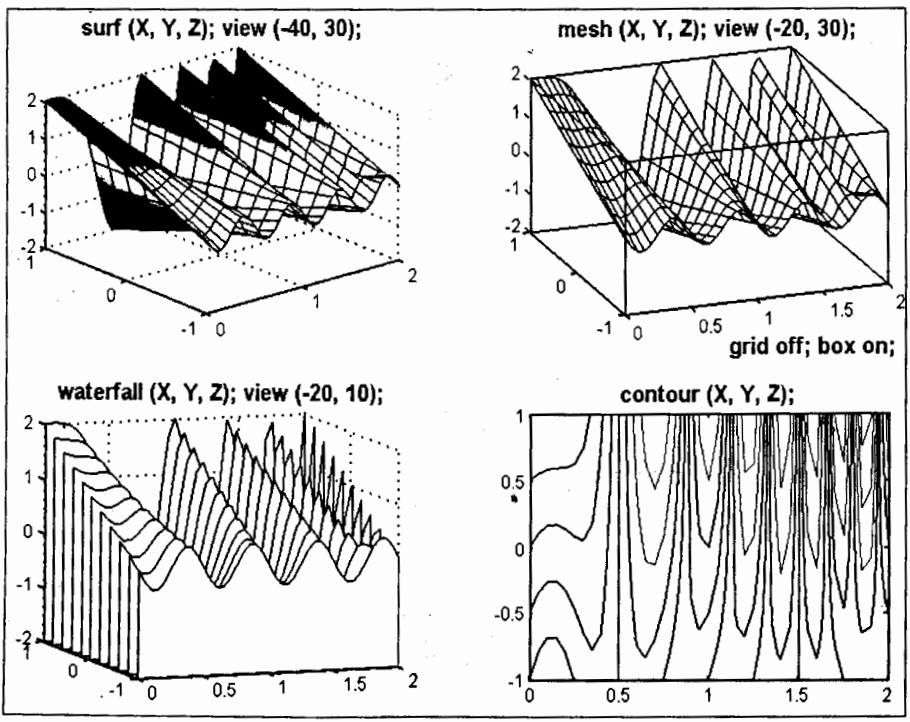
figure
```

```
subplot (221);
    surf (X, Y, Z);
    view (-40, 30);
    title ('surf (X, Y, Z); view (-40, 30);');

subplot (222);
    mesh (X, Y, Z);
    view (-20, 30);
    title ('mesh (X, Y, Z); view (-20, 30);');
    xlabel ('grid off; box on;');
    grid off;
    box on;

subplot (223);
    waterfall (X, Y, Z);
    view (-20, 10);
    title ('waterfall (X, Y, Z); view (-20, 10);');

subplot (224);
    contour (X, Y, Z);
    title ('contour (X, Y, Z);');
```

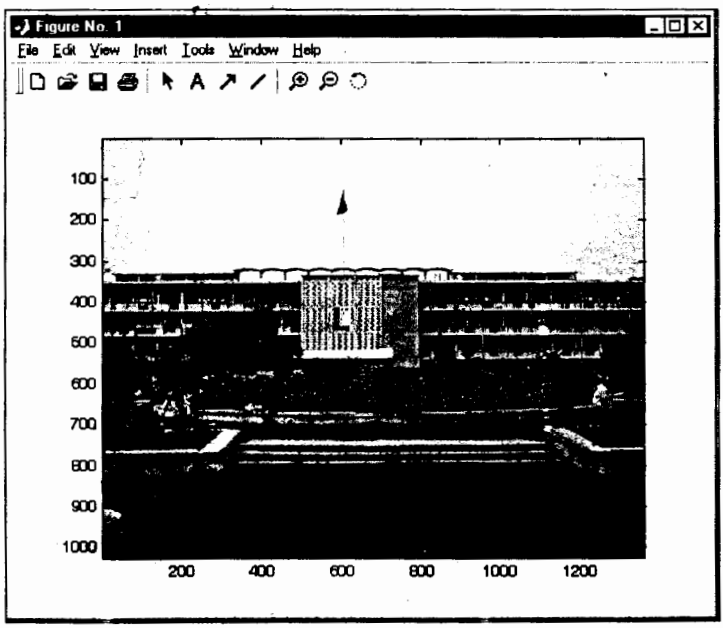


Hình 2.10 *Ví dụ về các phương pháp biểu diễn khác nhau đối với dữ liệu 3-D*

Đồ họa 3-D	
`[X,Y] = meshgrid (x_vector,y_vector)`	Ma trận tọa độ
`plot3(x_value,y_value,z_value...[,plotstyle])`	Vẽ 3-D (điểm / nét)
`surf(x_value,y_value,z_value...[, color])`	Vẽ 3-D (mặt)
`mesh(x_value,y_value,z_value...[, color])`	Vẽ 3-D (lưới)
`waterfall(x_value,y_value,z_value...[...])`	Vẽ 3-D (thác nước)
`contour(x_value,y_value,z_value...[...])`	Vẽ 2-D (nét đẳng mức)
`box [on | off]`	Đóng / ngắt box
`rotate3d [on | off]`	Tạo phối cảnh nhờ Mouse
`view(horizontal,vertical)`	Đặt góc phối cảnh
`zlabel (string)`	Điền ký tự cho trục *z*
Đặt mầu	
`colormap(name)`	Chọn mầu
`caxis(color_min, color_max)`	Co dãn thang mầu

2.8 Nhập, xuất và in đồ họa

Nếu cần phải gán một *File* đồ họa có sẵn (ví dụ: *File* ảnh) vào khuôn hình của *Figure*, ta có thể sử dụng hai lệnh *variable* = `imread(file,fmt)` và `image(variable)`.



Hình 2.11 File ảnh được gán vào khuôn hình của Figure

Bằng lệnh `imread` ta gán *File* đồ họa với định dạng *fmt* cho biến *variable*. Nếu *variable* nhận hình ảnh chỉ bao gồm gam mầu xám, *variable* sẽ là một biến 2 chiều. Nếu đó là ảnh mầu RGB, *variable* sẽ là một mảng (*Array*) 3 chiều. Định dạng của đồ họa được khai báo bởi *fmt*[1]. Lệnh `image` (*variable*) sẽ xuất đồ họa mới gán cho *variable* ra màn hình *Figure*. Theo cách vừa mô tả, những dòng lệnh sau đây sẽ tạo ra một *Figure* (hình 2.11) có chứa ảnh với định dạng *jpg* của Trường đại học Bách Khoa Hà Nội *hanoi_uni.jpg*.

```
>> example = imread('hanoi_uni.jpg','jpeg');
>> image(example)
```

Đồ họa *Figure* của MATLAB cũng có thể được xuất sang các định dạng khác. Lệnh `print -f`*number* sẽ in *Figure* mang số *number* ra máy in. Lệnh `print -f`*number option file* sẽ xuất *Figure* thành *File* với các định dạng đồ họa khác. Ví dụ: `bmp` (Windows bitmap), `emf` (Enhanced metafile), `eps` (EPS level 1), `jpg` (JPEG image), `pcx` (Paintbrush 24-bit) hay `tif` (TIFF image, compressed). Để biết chi tiết bạn đọc hãy gọi lệnh `help print`.

Để minh họa, sau khi đã tạo ra được đồ họa *Figure*1 như hình 2.11, lúc này có thể viết `-f`*number* = `-f1`, bạn đọc hãy lần lượt thử chuỗi lệnh sau đây:

```
>> print -f1;                        % Print to standard printer
>> print -f1 -dmeta 'picture';       % Convert to picture.emf
>> print -f1 -depsc 'picture';       % Convert to picture.eps
```

Nếu cần phải lưu lại để sau này xử lý, bạn đọc có thể cất các đồ họa đã thu được thành *File* với định dạng `fig` của MATLAB. Để cất, hoặc ta đi theo menu *File* / *Save as*, hoặc gọi lệnh `saveas` (*handle*, *'file'* `[`,*format*`]`). Lệnh `saveas` cất *handle* (*Figure* hiện tại, có thể dùng `gcf` để hỏi) thành tệp có tên *file* với một trong các định dạng: 'fig' (*File* nhị phân), 'm' (gồm một *File* `fig` và một *File Script*, là *File* sẽ gọi *File* `fig`). Ngoài ra, có thể dùng lệnh `print` để cất với các định dạng khác như đã mô tả ở trên.

Đồ họa: Nhập, xuất và in	
`print -f`*number*	In đồ họa ra máy in
`print -f`*number* `-d`*fmt file*	Cất đồ họa ra *file* với định dạng *fmt*
`saveas` (*handle*, *'file'*, `'fig'`)	Cất đồ họa dưới định dạng MATLAB
variable = `imread` (*file*, *fmt*)	Gán cho *variable* đồ họa cất dưới định dạng *fmt* trong *file*
`image` (*variable*)	Xuất đồ họa thuộc *variable* ra màn hình

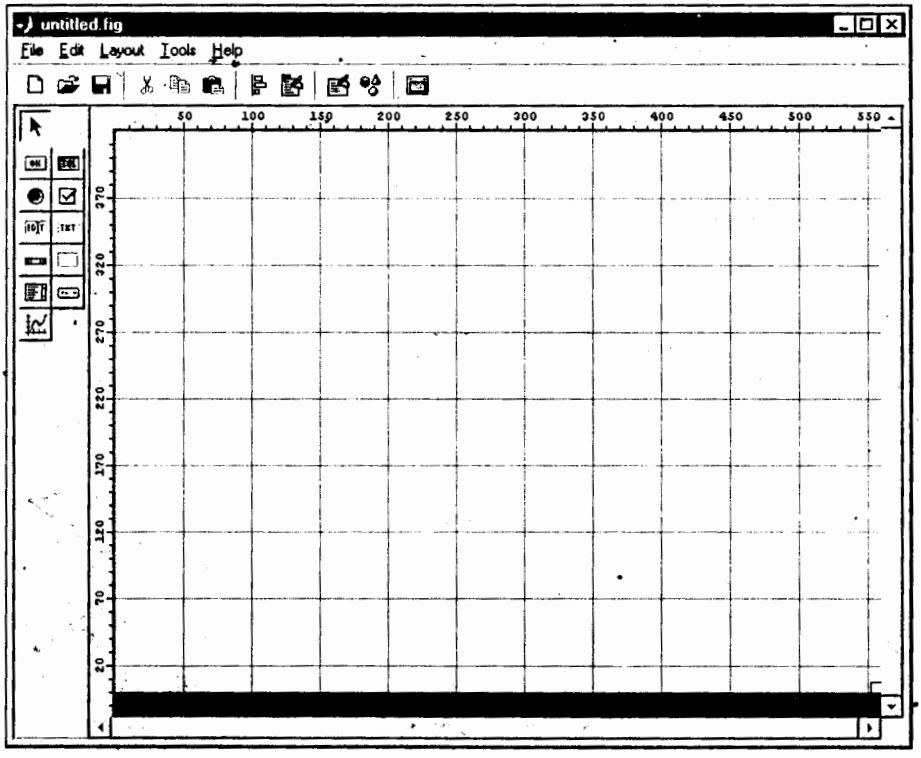
[1] Các định dạng: 'bmp' Windows Bitmap, 'ico' Windows Icon Resources, 'jpg' hoặc 'jpeg' Joint Photographic Expert Group, 'pcx' Windows Paintbrush, 'tif' hoặc 'tiff' Tagged Image File Format.

2.9 Giao diện đồ họa

Để tiện dụng và thoải mái, ta có thể tạo nên một giao diện đồ họa[1] (GUI) giữa người sử dụng và MATLAB. Giao diện đó là một đồ họa giao lưu tích cực (*interactive*), được xây dựng tương tự như một hộp đối thoại (*Diaglog Box*). Trong giao diện đó, ta có thể xuất dữ liệu dưới cả hai dạng: Văn bản và đồ họa. Để tạo GUI, hoặc ta tạo theo phương thức thủ công, hoặc ta tận dụng các công cụ đồ họa do MATLAB cung cấp. Mục 2.9 sẽ giới thiệu cả hai phương thức. Tuy nhiên, phần trợ giúp tốt nhất cho chủ đề tạo GUI vẫn là cuốn sổ tay với nội dung vô cùng phong phú và chi tiết, được cấp kèm theo đĩa CD cài MATLAB.

2.9.1 Layout (diện mạo) của GUI

Để tạo nên hay xử lý *Layout* của GUI, ta vào lệnh `guide` để gọi trình soạn thảo GUIDE và thu được một *Layout* rỗng như hình 2.12.



***Hình 2.12** Trình GUIDE mở cửa sổ soạn thảo GUI, bắt đầu bằng một Layout rỗng*

[1] Graphical User Interface

Tuy nhiên, ta sẽ quay trở lại với `guide` sau, vì chỉ khi đó ta mới thực sự hiểu rõ: `guide` đã làm thay cho ta những việc phiền toái như thế nào. Cho đến nay ta đã làm quen với một số lệnh như `input`, `disp` hay `fprintf`, phục vụ việc nhập / xuất ký tự và số liệu lên MATLAB *Command Windows*. Việc tạo GUI chính là nhằm tạo nên một công cụ đồ họa, phục vụ nhập / xuất một cách trực giác, rất thuận tiện. Hơn thế nữa, ta còn có thể dùng GUI để giám sát các quá trình, hiển thị các đặc tính của đối tượng.

2.9.2 Nhập và xuất ký tự, số liệu ra GUI

Ta hãy thử gọi chuỗi các lệnh sau đây với tác dụng nhập giá trị nhiệt độ đo bằng độ *Fahrenheit*, sau đó quy đổi tương đương sang độ C (*Celsius*).

```
>> f = input('Enter temperature (degrees K): ');
>> c = (f - 32)*5/9;
>> fprintf(1, 'Temperature (degrees C) is: %g\n', c);
```

Ba dòng lệnh trên đây đã thực hiện các thao tác

- Nhập giá trị đầu vào
- Thực hiện tính quy đổi tương đương
- Xuất giá trị quy đổi ra màn hình.

Vậy ta hãy thử tìm cách cài các dòng lệnh trên, sao cho chúng thực hiện trong khuôn khổ của một khung đồ họa nào đó.

Tạo khung hình đầu tiên

Để tạo ra hai khung hình chữ nhật trong cửa sổ *Figure* hiện tại[1] với màu nền xám, bạn đọc hãy lần lượt vào chuỗi lệnh sau đây.

```
set(gcf, 'DefaultUicontrolUnit', 'Normalized')

%------------------------------------------------------------
frame1_ = uicontrol(gcf,                                    ...
            'Style',      'Frame',                          ...
            'Position',   [0.1 0.1    0.8 0.3]);
frame2_ = uicontrol(gcf,                                    ...
            'Style',      'Frame',                          ...
            'Position',   [0.1 0.6    0.8 0.3]);
%------------------------------------------------------------

set(frame1_, 'BackgroundColor',   [0.50 0.50 0.50]);
set(frame2_, 'BackgroundColor',   [0.50 0.50 0.50]);
```

[1] Nếu hiện tại không có *Figure* nào đang được mở, MATLAB sẽ tự động tạo *Figure* mới

Hai khung (Frames) đó có góc trái phía dưới đặt ở hai tọa độ (0.1 0.1) và (0.1 0.6), cùng có chiều cao 0.3 đơn vị và bề rộng là 0.8 đơn vị. Tại đây cần phải lưu ý bạn đọc: Đơn vị được tính bằng % của kích cỡ ngoài của Figure. Vậy ta có thể diễn đạt lại cho dễ hiểu như sau:

- Khung thứ nhất có góc trái phía dưới tại điểm có tọa độ ứng với 10% chiều ngang và 10% chiều cao của khung ngoài *Figure*.
- Khung thứ hai có góc trái phía dưới tại điểm có tọa độ ứng với 10% chiều ngang và 60% chiều cao của khung ngoài *Figure*.
- Cả hai khung có chiều cao bằng 30% chiều cao, và bề ngang bằng 80% bề ngang của khung ngoài *Figure*.

Dùng lệnh `edit` *và* `text` *để nhập / xuất ký tự và số liệu*

Trên đây ta đã sử dụng lệnh `uicontrol` để tạo và xác định vị trí của hai khung hình. Đoạn lệnh sau đây sử dụng `uicontrol` để viết chuỗi ký tự "*Fahrenheit*" (chuỗi `Text`) lên khung bên trên.

```
text_f_ = uicontrol(gcf,                                    ...
            'Style',      'Text',                           ...
            'String',     'Fahrenheit: ',                   ...
            'Position',   [0.3 0.7    0.2 0.05],            ...
            'HorizontalAlignment', 'Left');
```

Chuỗi ký tự "*Fahrenheit*" được đặt vào đúng vị trí dồn trái của ô có Position ghi trong đoạn chương trình trên. Đoạn sau đây sử dụng `Edit` để viết chuỗi ký tự "68.0" vào vị trí bên cạnh của "*Fahrenheit*". Chuỗi ký tự đó có vị trí dồn phải trong ô (*Position Box*).

```
edit_f_ = uicontrol(gcf,                                    ...
            'Style',      'Edit',                           ...
            'String',     '68.0',                           ...
            'Position',   [0.6 0.7    0.1 0.05],            ...
            'HorizontalAlignment', 'Right',                 ...
                                                            ...
            'Callback', 'fc_calc' );
```

Do sử dụng `Edit`, chuỗi ký tự "68.0" là chuỗi có thể viết lại được (*editable*) trực tiếp trên GUI. Sau khi nhấn nút [*Enter*], giá trị mới viết lại được tiếp nhận và MATLAB sẽ gọi lệnh viết trong phần `Callback`: `fc_calc`.

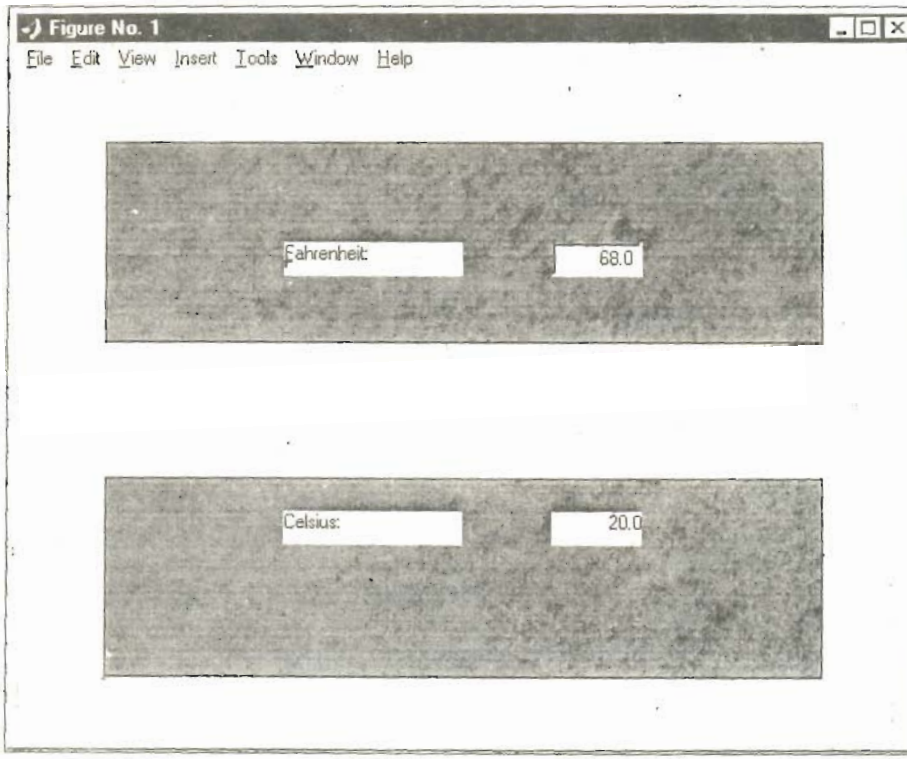
Cuối cùng, ta còn phải dùng `uicontrol` để tạo ra hai chuỗi `Text`, hiển thị ký tự "*Celsius*" và "20.0" trong khung bên dưới.

```
text_c1_ = uicontrol(gcf,                                   ...
            'Style',      'Text',                           ...
            'String',     'Celsius: ',                      ...
```

```
            'Position',    [0.3 0.3    0.2 0.05],        ...
            'HorizontalAlignment', 'Left');

text_c2_ = uicontrol(gcf,                                ...
            'Style',       'Text',                       ...
            'String',      '20.0',                       ...
            'Position',    [0.6 0.3    0.1 0.05],        ...
            'HorizontalAlignment', 'Right');
```

Sau khi đã lần lượt vào tất cả các chuỗi lệnh trên ta sẽ thu được kết quả như hình 2.13. Để thuận tiện, bạn đọc nên viết các lệnh đó trong một *Script* (*m-File*), ví dụ: *Fig*02_13.*m*.



Hình 2.13 Ví dụ GUI phục vụ tính quy đổi từ Fahrenheit (°F) sang Celsius (°C)

Tự động cập nhật giá trị lên GUI

Để hoàn thiện ví dụ GUI ở hình 2.13, ta còn phải thực hiện chương trình với nhiệm vụ tính quy đổi từ °K sang °C và tự động điền kết quả tính vào ô bên cạnh ô mang ký tự "*Celsius*". Đoạn chương trình phục vụ vào mục đích "*Callback*" (hoàn trả giá trị) đó được cất dưới tên *fc_calc.m*..

```
f = get(edit_f_, 'String');
f = str2num(f);
c = (f - 32)*5/9;
c = num2str(c);
set(text_c2_, 'String', c);
```

Đoạn *Script* trên đã nhận giá trị do lệnh `uicontrol 'Edit'` đọc vào dưới dạng *String* và sau đó lần lượt:

- biến đổi từ *String* sang dạng số (lệnh `str2num`),
- tính quy đổi từ nhiệt độ *Fahrenheit* sang *Celsius*,
- biến đổi từ số sang *String* (lệnh `num2str`) và
- xuất kết quả dưới dạng *String* ra GUI nhờ `text_c2_`.

2.9.3 Nhập số liệu từ thanh trượt (Slider)

Ngoài cách nhập số liệu từ bàn phím, ta có thể nhập số liệu từ một thanh trượt. Bạn đọc hãy hình dung ra vai trò của một biến trở, thường được sử dụng để (ví dụ) thay đổi giá trị đặt (giá trị chủ đạo) của một vòng điều chỉnh tương tự.

Chú ý: MATLAB chỉ hỗ trợ việc xây dựng các *Slider* theo phương nằm ngang (*horizontal*) chứ không hỗ trợ tạo *Slider* theo phương thẳng đứng (*vertical*).

Để tạo *Slider* (hình 2.14), bạn đọc hãy nhập chuỗi lệnh:

```
slider_f_ = uicontrol(gcf,                                    ...
            'Style',      'Slider',                           ...
            'Min',         32.0,                              ...
            'Max',        212.0,                              ...
            'Value',       68.0,                              ...
            'Position',   [0.6 0.8    0.2 0.05],             ...
                                                             ...
            'Callback', 'fc_slider_f; fc_calc');
```

Ví dụ trên cho thấy: `Callback` có thể gọi một chuỗi các lệnh MATLAB, viết cách bởi dấu chấm than hoặc dấu phẩy. Chuỗi `Callback` gọi *fc_slider_f.m*:

```
f = get(slider_f_, 'Value');
f = num2str(f);
set(edit_f_, 'String', f);
```

với tác dụng nhập giá trị nhiệt độ giữ tại `'Value'` của `slider_f_f`, vào vị trí bên cạnh ô ký tự "*Fahrenheit*". Sau đó, `Callback` gọi tiếp *fc_calc.m* để tính quy đổi giá trị nhiệt độ và gán vào vị trí cạnh ô ký tự "*Celsius*".

Tuy nhiên, để có thể cập nhật được giá trị mới, do người sử dụng trượt Slider gây nên, ta phải thay đổi lại chuỗi lệnh `Callback` của `Edit uicontrol` như sau:

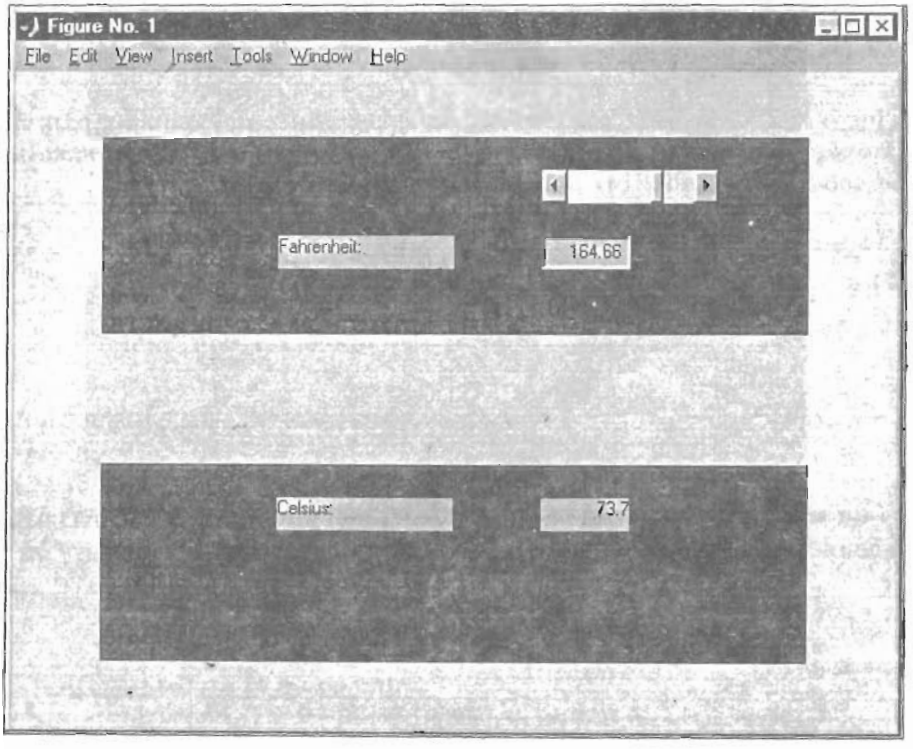
```
set(edit_f_, ...
    'Callback', 'fc_edit_f, fc_calc');
```

Trong đó, *fc_edit_f.m*:

```
f = get(edit_f_, 'String');
f = str2num(f);
set(slider_f_, 'Value', f);
```

có nhiệm vụ cập nhật giá trị giữ tại `'Value'` của `slider_f_f`, để rồi sau đó fc_calc.m làm nốt phần việc còn lại: Tính quy đổi và gán vào vị trí cạnh ô ký tự "*Celsius*".

Từ đây trở đi, cả hai khung (*Frames*) của GUI hoạt động hoàn toàn đồng bộ với nhau.



Hình 2.14 Quy đổi từ Fahrenheit (°F) sang Celsius (°C), nhập giá trị bằng thanh trượt (Slider)

2.9.4 Nhập dữ liệu tùy chọn (Popup Menu, List Box, Radio Button and Check Box)

Ngoài khả năng nhập dữ liệu cố định theo kiểu ký tự (*String*) hay kiểu số, ta hoàn toàn có thể thực hiện việc nhập tùy chọn từ một danh mục nào đó. Để minh họa, ta tạm thời rời bỏ GUI để quay về với *Command Windows* và thực hiện chuỗi lệnh sau đây:

```
f = input('Enter temperature (degrees F): ');

r = f + 459.7;
c = (f - 32) * 5 / 9;
k = c + 273.15;

choise = input(['Enter 1 for Rankine,',      ...
                '  2 for Celsius,',       ...
                '  3 for Kelvin:  '       ]);
if choise == 1
   fprintf(1, ' Temperature (degrees R) is: %g\n', r);
elseif choise == 2
   fprintf(1, ' Temperature (degrees C) is: %g\n', c);
elseif choise == 3
   fprintf(1, ' Temperature (degrees K) is: %g\n', k);
end
```

Giả sử bạn đọc cất chuỗi lệnh đó trong một *File* có tên là *test.m* với giá trị nhập của nhiệt độ có đơn vị là °F (độ *Fahrenheit*), giá trị xuất lên màn hình có thể được biểu diễn với đơn vị °R (độ *Rankine*), °C (độ *Celsius*) hoặc °K (độ *Kelvin*). Khi gọi *test.m* ta thu được kết quả:

```
>> test
Enter temperature (degrees F): 26
Enter 1 for Rankine,  2 for Celsius,  3 for Kelvin:  1
Temperature (degrees R) is: 485.7
```

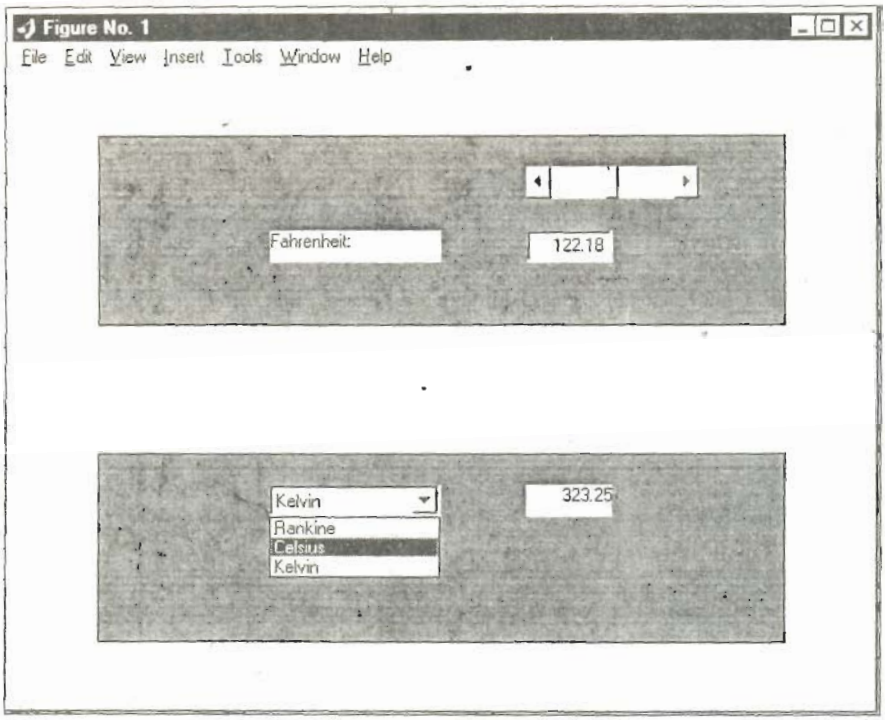
Trong ví dụ trên: Khi được hỏi ta đã chọn nhiệt độ đầu vào là 26°F, và chọn đích quy đổi là °R. Tuy nhiên, công cụ GUI của MATLAB cho phép ta thực hiện sự lựa chọn cách nhập dữ liệu vào một cách thuận lợi hơn thế. Tiếp tục ví dụ tính quy đổi và xuất giá trị nhiệt độ, mục này giới thiệu với bạn đọc bốn phương án nhập tùy chọn sau đây:

- Phương án menu (*Popup Menu*).
- Phương án hộp danh mục (*List Box*).
- Phương án nút chọn (*Radio Button*).
- Phương án hộp hỏi đáp (*Check Box*).

Phương án Popup Menu

Ta hãy xóa bỏ ký tự "*Celsius*" trong lệnh `text_c1_` và thay vào đó khả năng lựa chọn theo *Popup Menu* (hình 2.15) như sau:

```
delete(text_c1_);
popup_c_ = uicontrol(gcf,                                    ...
            'Style',      'Popupmenu',                       ...
            'String',     'Rankine|Celsius|Kelvin',          ...
            'Value',      2,                                 ...
            'Position',   [0.3 0.3    0.2 0.05],             ...
                                                             ...
            'Callback', 'fc_popup_c; fc_calc2');
```



***Hình 2.15** Quy đổi từ Fahrenheit (°F) sang đơn vị tùy chọn: Phương án Popup Menu*

Hình 2.15 là kết quả thu được sau khi ta bổ xung chuỗi lệnh trên. Khi kích chuột vào *Popup Menu*, có ba khả năng lựa chọn sẽ xuất hiện, tiếp tục nháy chuột vào một trong ba khả năng đó, *Popup Menu* sẽ biến mất chỉ còn lại đơn vị được chọn. Khi dùng chuột kéo thanh trượt ở Frame phía trên, ta sẽ có giá trị quy đổi sang đơn vị được chọn hiển thị ở phía dưới.

Trong đoạn chương trình trên bạn đọc hãy lưu ý: Then chốt là `'Value'` với giá trị đặt sẵn là 2. Khi `Callback` gọi *fc_popup_c.m*:

```
choise = get(popup_c_, 'Value');
```

giá trị của biến choise được đưa tới 'Value'. Sau đó Callback gọi tiếp *fc_calc2.m* để xem kết quả chọn được cất trong choise.

```
f = get(edit_f_, 'String');
f = str2num(f);

r = f + 459.7;
c = (f - 32) * 5 / 9;
k = c + 273.15;

if     choise == 1,   t = r;
elseif choise == 2,   t = c;
elseif choise == 3,   t = k;
end

t = num2str(t);
set(text_c2_, 'String', t);
```

Cần lưu ý rằng, để đạt được kết quả như hình 2.15, ta còn phải báo cho Edit và Slider uicontrols biết để sử dụng *fc_calc2.m* (trước kia hai lệnh đó dùng *fc_calc.m*) bằng cách bổ sung thêm các dòng sau:

```
set(edit_f_, ...
    'Callback', 'fc_edit_f; fc_calc2');
set(slider_f_, ...
    'Callback', 'fc_slider_f; fc_calc2');
```

Chú ý: Bằng cách thay 'Popupmenu' uicontrol bởi 'Listbox' uicontrol bạn đọc sẽ có được phương án *List Box* (hộp danh mục). Điểm khác duy nhất là: Nếu sau khi chọn, *Popup Menu* chỉ chứa một phần tử, thì *List Box* có thể chứa đồng thời trong danh mục nhiều phần tử hơn, duy nhất phụ thuộc vào chiều cao tại vị trí ('Position' *property*).

Phương án Radio Button (nút chọn)

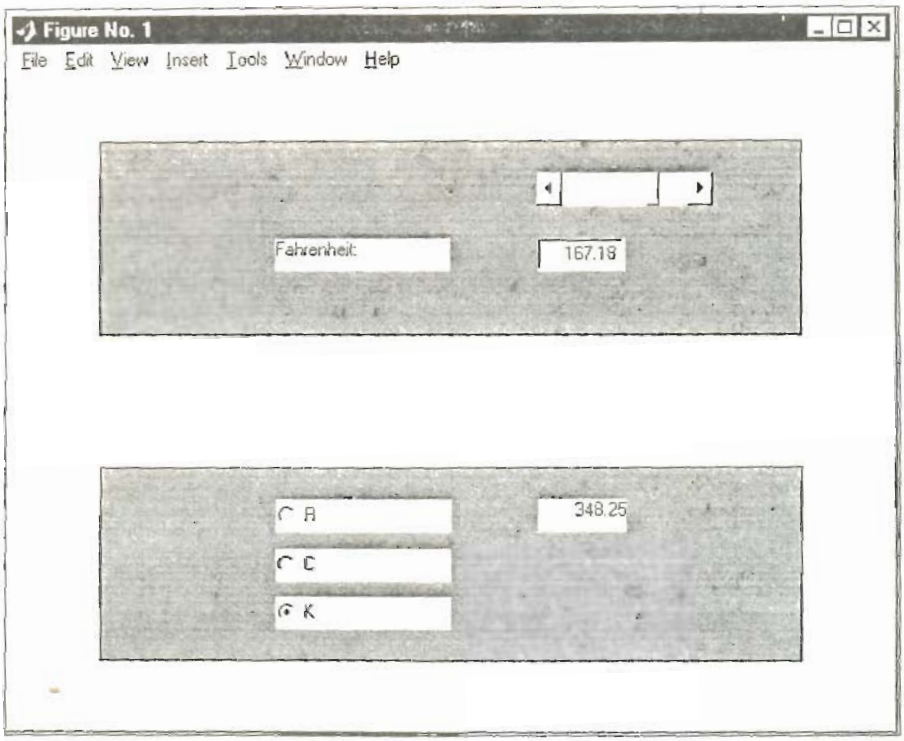
Ta hãy dùng lệnh delete(popup_c_) để xóa uicontrol tạo *Popup Menu* "*Rankine/Celsius/Kelvin*" và thêm vào đó ba uicontrol tạo ba nút chọn thứ nguyên nhiệt độ sau đây:

```
delete(popup_c_);

strings = ['Rankine'; 'Celsius'; 'Kelvine'];
show    = [   0     ;     1    ;    0     ];
ys      = [   3     ;     2    ;    1     ]*0.075 + 0.075;
```

```
for i=1:3
    radio_c_(i) = uicontrol(gcf,                                    ...
            'Style',        'Radiobutton',                          ...
            'String',       strings(i),                             ...
            'Value',        show(i),                                ...
                                                                    ...
            'Position',     [0.3 ys(i)  0.2 0.05],                  ...
                                                                    ...
            'Callback', 'fc_radio_c; fc_calc2');
end
```

Kết quả thu được sẽ như hình 2.16 sau đây.



Hình 2.16 Quy đổi từ Fahrenheit (°F) sang đơn vị tùy chọn: Phương án Radio Button

Nút được chọn ban đầu là "*Celsius*" (`'Value'` nhận giá trị ban đầu là 1). Nếu ta dùng chuột nháy vào một trong ba nút, MATLAB sẽ điền giá trị vào nút đó, sau đó gọi chuỗi `Callback` với các lệnh đã định. Cần lưu ý là ta chỉ sử dụng một chuỗi `Callback` chung cho cả ba nút. Trước hết `Callback` gọi *fc_radio_c.m* để nhận biết xem nút nào (1, 2 hay 3) được chọn, sau đó cất giá trị được chọn vào biến `choice`.

```
for i=1:3
    if gcbo == radio_c_(i)
        choice = i;
        set(radio_c_(i), 'Value', 1);
    else
        set(radio_c_(i), 'Value', 0);
    end;
end;
```

Đoạn *Script* trên là một vòng lặp, so sánh số (*handle*) `Callback` do thu được (giá trị do hàm `gcbo` trả lại) với số (*handle*) của mỗi nút. Nút nào có số trùng sẽ được đóng (*turned on*, `'Value'` = 1), nút nào khác số sẽ bị ngắt (*turned off*, `'Value'` = 0). Cuối cùng, `Callback` gọi *fc_calc2.m* để thực hiện việc tính quy đổi đã được chọn và hiển thị kết quả.

2.9.5 Các phương pháp tạo GUI

Thông qua một ví dụ cụ thể xuyên suốt các mục 2.9.1 – 2.9.4, bạn đọc đã làm quen với cách tạo GUI, đã thấy được khả năng phong phú của các công cụ do MATLAB cung cấp. Tuy nhiên, ví dụ đó mới cho thấy cách tạo GUI theo phương pháp thủ công. Còn có thể tạo GUI một cách đơn giản hơn nữa bằng các công cụ đồ họa. Trên cơ sở ví dụ tính đổi tiền "*Exchange of Funds*" (được trích từ tài liệu tham khảo), mục này hệ thống hóa hai phương pháp, giúp bạn đọc có một cách nhìn so sánh, thuận tiện khi cần phải lựa chọn.

a) Phương pháp tạo GUI bằng công cụ đồ họa

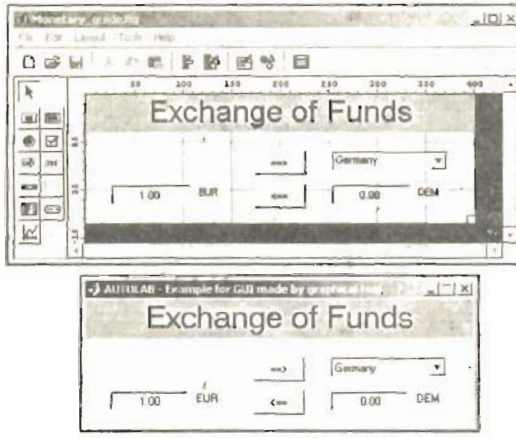
Ta đã biết: Khi vào lệnh `guide` ta sẽ gọi trình đồ họa GUIDE để soạn thảo *Layout*, kết quả đầu tiên là *Layout* rỗng ở hình 2.12. Việc đầu tiên phải làm là mở Menue *Tools/Application Options* để xác định:

- Liệu ta chỉ muốn tạo GUI dưới dạng *file* có định dạng `.fig`,
- hay đồng thời tạo cả *script* (*m-file*). Trong ví dụ sau đây ta sẽ chọn *Option* này. Sau khi đã hoàn thiện, để sử dụng GUIDE ta chỉ việc nhập tên của *m-file* đó ở cửa sổ Command Windows, hoặc kích hoạt ở Menue *Tools/Activate Figure*.

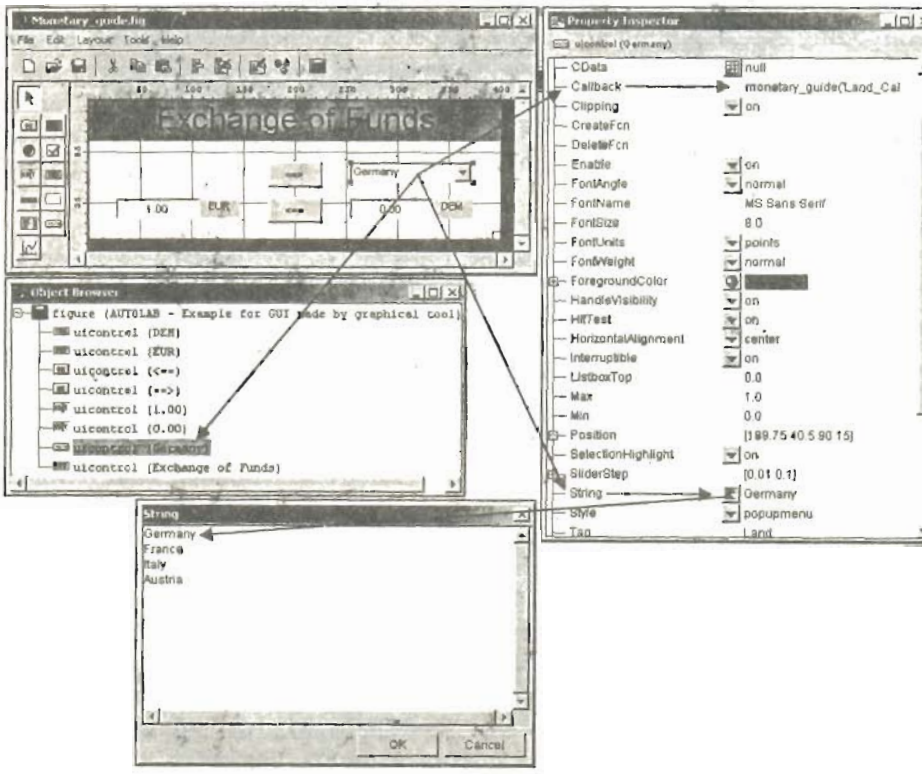
Tiếp theo, ta lần lượt sử dụng chuột để gắp các phần tử của GUI từ thư viện nằm phía bên trái của *Layout* rỗng (hình 2.12) và nhả vào các vị trí ta muốn. Có thể sử dụng chuột để co dãn, thay đổi kích thước của *Layout* và của các phần tử thuộc *Layout*. Các kích thước đã xác định trên *Layout* cũng sẽ là kích thước xuất hiện trên GUI sau này (hình 2.17).

Các phần tử của *Layout* được quản lý theo một *cấu trúc có phân cấp*, trong đó đồ họa của GUI đứng ở cấp cao nhất. Có thể kiểm tra cấu trúc đó bằng cách mở cửa sổ *Tools/Object Browser* (hình 2.18: hình ở giữa, bên trái).

Mỗi phần tử của cửa sổ *Layout* đều cần phải có những đặc điểm (tham số) nhất định. Để soạn thảo các tham số ấy, ta sử dụng *Property Inspector* (hình 2.18: hình phải), gọi bằng một trong các cách: Nháy chuột kép vào phần tử, dùng lệnh `inspect`, hay qua Menue *Tools/Property Inspector*.



Hình 2.17 Trên: Chọn và xác định kích thước của các phần tử trên cửa sổ Layout. Dưới: Kết quả thu được của GUI



Hình 2.18 Sử dụng Property Inspector để soạn thảo tham số cho các phần tử của GUI

Hình 2.18 minh họa khả năng soạn thảo tham số cho phần tử Popup Menue dùng để chọn nước (Đức, Pháp, Ý hay Áo): Chọn đặc điểm `'String'` của Popup Menue để mở cửa sổ soạn thảo, sau đó viết tên các nước vào cửa sổ (hình 2.18: hình dưới). Tuy nhiên, `'String'` chỉ là một trong nhiều đặc điểm cần được xác định của phần tử Popup Menue. Bảng sau đây tập hợp một số đặc điểm (tham số) quan trọng của phần tử và các giá trị đặc trưng đặt sẵn.

Các tham số xác định đặc điểm của GUI

Parent	*handle*	Phần tử cấp trên
Position	[*left*, *below*, *width*, *height*]	Vị trí (khoảng cách, kích cỡ)
Units	`'points'`	Đơn vị xác định vị trí
BackgroundColor	[*red*, *green*, *blue*]	Mầu nền
Style	`'pushbutton'`	Diện tích chọn
	`'togglebutton'`	Diện tích chọn
	`'checkbox'`	Vùng của hộp hỏi đáp
	`'radiobutton'`	Vùng của nút chọn
	`'popupmenue'`	Danh mục chọn
	`'text'`	Vùng của ký tự (Text)
	`'edit'`	Vùng nhập ký tự
	`'slider'`	Thanh trượt
	`'frame'`	Khung có mầu
String	*'string'*	Ký tự được hiển thị
	[*'string_1'*; *'string_2'*]	Các phần tử của danh mục (`'popupmenue'`)
Value	*number*	Tích cực = 1, phần tử được chọn từ danh mục
Tag	*string*	Nhãn để nhận dạng phần tử
Callback	*'string'*	Gọi hàm

Mọi phần tử đều có thể nhận nội dung hay giá trị cho sẵn trước, và trong quá trình sử dụng do gọi hàm (Callback) nội dung hay giá trị đó có thể bị thay đổi. Vậy là chúng đã có sẵn chức năng xuất dữ liệu. Một vùng `'edit'` có cả hai chức năng nhập và xuất (nhập và hiển thị ký tự).

Từng phần tử – tùy theo thứ tự gắp/thả chúng – có thể bị xếp chồng lên nhau. Đặc biệt các phần tử `'frame'` cần được tạo đầu tiên để tránh chúng che lấp các phần tử khác. Sau này, ta có thể thay trình tự đó bằng cách đổi thứ tự của các chuỗi lệnh tương ứng trong *m-file*. Ngoài ra, để thu được một *Layout* đẹp mắt, ta có thể sử dụng công cụ Menue *Layout/Align Objects* để gióng thẳng hàng các phần tử của GUI.

Tham số `Value` chứa giá trị hiện tại của phần tử:

- Phần tử 'pushbutton': Giá trị đó cho biết, liệu diện tích của nút nhấn có được chọn (=1) hay không (=0). Hoặc đối tượng nào trong danh mục (đánh số thứ tự 1, 2, 3, ...) được chọn.
- Phần tử 'edit' (nhập ký tự): Cho biết giá trị của số được nhập. Nếu dữ liệu được nhập là chuỗi ký tự, giá trị sẽ là một ma trận rỗng.
- Phần tử 'popupmenue': Do đặc điểm chọn mang tính duy nhất nên không tồn tại các giá trị khác nhau.
- Các phần tử còn lại: Giá trị được cung cấp nhờ thao tác gọi hàm.

Ví dụ "*Exchange of Funds*" ở hình 2.17 minh họa kết quả tạo GUI, giao diện phục vụ tính quy đổi tiền của các nước Đức (DEM), Pháp (FRF), Ý (ITL) và Áo (ATS) sang đồng tiền chung EUR của châu Âu và ngược lại. Ví dụ được biên soạn lại dựa trên mẫu waehrung_guide.m và waehrung_guide.fig của trang Web www.eat.ei.tum.de/lehre/ssm. Sau khi biên soạn, hai Files mới có tên monetary_guide.m và monetary_guide.fig.

Chức năng của GUI

Chức năng thật sự của phần tử thuộc GUI nằm ẩn trong các tham số của Callback. Mỗi khi kích hoạt phần tử (nháy chuột hay nhập dữ liệu), các lệnh sẽ được chuyển cho MATLAB dưới dạng chuỗi ký tự và được thực hiện. Nếu ta sử dụng *m-file* (gọi là *Application M-File*, viết tắt: AMF), trên cửa sổ *Property Inspector*, tại vị trí nhập dòng lệnh của Callback ta có:

```
monetary_guide('Value_Land_Callback',gcbo,[],guidata(gcbo))
```

Nếu ta viết một giá trị nào đó vào vùng nhập giành cho đồng tiền riêng của nước được chọn, sau đó nhấn nút *Enter*, khi ấy hàm monetary_guide (tên của hàm AMF) sẽ được kích hoạt. Trong cấu trúc của monetary_guide, hàm con Value_Land_Callback đã được định nghĩa như sau:

```
function varargout = Value_Land_Callback(h, eventdata, handles, varargin)
set(handles.Value_Land,'Value',str2num(get(handles.Value_Land,'String')))
```

Trong hàm trên, bằng lệnh set, giá trị lấy từ vùng nhập ký tự được gán cho handles.Value_Land. Trước đó, giá trị ấy đã được đọc từ vùng 'String' của handles.Value_Land và đã được MATLAB dùng hàm str2num đảo thành một giá trị số học. Nói chung, mỗi đồ họa GUI đều được MATLAB nhận biết nhờ một chỉ số, gán cho biến handles ở đầu của hàm AMF. Khi cất *Layout* của GUI, MATLAB sẽ tự động tạo ra phần đầu (phần định nghĩa) của AMF. Ta sẽ chỉ phải điền bằng tay phần nội dung của hàm . Trong ví dụ trên, phần đó như sau:

```
function varargout = monetary_guide(varargin)
  % MONETARY_GUIDE Application M-file for monetary_guide.fig
```

```
% FIG = MONETARY_GUIDE launch monetary_guide GUI.
% MONETARY_GUIDE('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 21-Jan-2003 18:25:51

if nargin == 0  % LAUNCH GUI

   fig = openfig(mfilename,'reuse');

   % Use system color scheme for figure:
   set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

   % Generate a structure of handles to pass to callbacks, and
   % store it.
   handles = guihandles(fig);
   guidata(fig, handles);

   if nargout > 0
      varargout{1} = fig;
   end

   global Factor;                          % Factor global define
   Factor = 1.95583 ;                      % Default for Factor
   set(handles.Monetary,'String','DEM');   % Default for Monetary

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

   try
      [varargout{1:nargout}] = feval(varargin{:});
                                           % FEVAL switchyard
   catch
      disp(lasterr);
   end
end
```

Phần định nghĩa nằm ở đầu của hàm AMF đó được tạo nên hoàn toàn tự động. Người thiết kế GUI duy nhất chỉ phải điền bằng tay ba dòng sau:

```
   global Factor;                          % Factor global define
   Factor = 1.95583 ;                      % Default for Factor
   set(handles.Monetary,'String','DEM');   % Default for Monetary
```

Khi gọi GUI, ba dòng lệnh đó có tác dụng lập trị cho biến `Factor` và xác định đó là biến có tác dụng `global` trong phạm vi của `monetary_guide`, cho phép các hàm con sử dụng `Factor` một cách dễ dàng. Sau đó, các giá trị của hệ số quy đổi giữa đồng tiền chung Euro và đồng tiền riêng của quốc gia được đặt

vào Mark của CHLB Đức: `1.95583` và `'DEM'`. Trước đó, vùng danh mục chọn tương ứng `'Germany'` đã được đặt giá trị là `'1'` tại dòng `'Value'` của cửa sổ *Property Inspector*. Tuy nhiên, ta cũng có thể thực hiện thủ công thao tác đó bằng cách viết thêm dòng lệnh `set(handles.Land,'Value',1)`.

Phần đầu của Application M-File (AMF)

Qua ví dụ trên ta thấy, do lệnh `if`, phần đầu của hàm `monetary_guide` đã bị chia thành hai phần như sau:

- Phần thứ nhất sẽ được thực hiện, nếu không có trao tham số khi gọi `monetary_guide`. Đó chính là lần gọi GUI đầu tiên: Đồ họa được mở ra, chỉ số `handles` được cất vào biến `fig`, mầu của đồ họa được lập, GUI được gán cho biến `handles` và các thông số của GUI được xác lập.
- Phần thứ hai được thực hiện khi GUI đã được kích hoạt và chỉ được gọi: Hoặc khi nút chọn bị nhấn, hoặc khi danh mục chọn hoạt động, hoặc khi gọi một hàm con xác định ở vùng `CallBack`.

Các hàm con của Application M-File

Phần hàm con có chứa các hàm cho phép thực hiện các thao tác theo yêu cầu. Ví dụ: Thay đổi ký tự nhập (Style `'edit'`), thay đổi danh mục chọn (Style `'popupmenue'`), hay gọi hàm con riêng trong hàm `monetary_guide` như `Land_Callback` của danh mục để chọn nước.

```
% --------------------------------------------------------------------
function varargout = Land_Callback(h, eventdata, handles, varargin)
global Factor
switch get(handles.Land,'Value') ,
   case 1, Factor = 1.95583; set(handles.Monetary,'String','DEM');
   case 2, Factor = 6.55957; set(handles.Monetary,'String','FRF');
   case 3, Factor = 1936.27; set(handles.Monetary,'String','ITL');
   case 4, Factor = 13.7603; set(handles.Monetary,'String','ATS');
   end;

% --------------------------------------------------------------------
function varargout = Value_Land_Callback(h, eventdata, handles, varargin)
set(handles.Value_Land,'Value',str2num(get(handles.Value_Land,'String')))

% -------- ------------------------------------------------------------
function varargout = Value_Euro_Callback(h, eventdata, handles, varargin)
set(handles.Value_Euro,'Value',str2num(get(handles.Value_Euro,'String')))

% --------------------------------------------------------------------
function varargout = Euro2Land_Callback(h, eventdata, handles, varargin)
global Factor
set(handles.Value_Land,'Value',get(handles.Value_Euro,'Value')*Factor)
```

```
set(handles.Value_Land,'String',num2str(get(handles.Value_Land,'Value')))

% --------------------------------------------------------------------
function varargout = Land2Euro_Callback(h, eventdata, handles, varargin)
global Factor
set(handles.Value_Euro,'Value',get(handles.Value_Land,'Value')/Factor)
set(handles.Value_Euro,'String',num2str(get(handles.Value_Euro,'Value')))
```

Mỗi lần, khi thêm một phần tử (có khả năng thực hiện một thao tác nào đó) vào *Layout* của GUI, MATLAB sẽ tự động bổ sung thêm một hàm con mang tên *TagName*_Callback. Trong đó *TagName* là ký hiệu giành cho thao tác, viết ở dòng *Tag* của cửa sổ Property Inspector. Thông thường, MATLAB đặt tên cho các hàm con đó như: checkbox1_Callback, popupmenu3_Callback vv... và lần lượt đánh số (nếu có nhiều hàm con cùng loại) chúng theo thứ tự tăng dần. Ví dụ:

```
function varargout = checkbox1_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.checkbox1.
disp('checkbox1 Callback not implemented yet.')
```

Nếu người thiết kế GUI muốn đặt tên khác, người đó sẽ phải chủ động đặt tên trên dòng *Tag* trước khi cất (*save*) GUI. Nếu đã lỡ cất rồi, ta không chỉ phải thay đổi *Tag* trong *Property Inspector*, mà còn phải đồng thời thay thủ công tên hàm con trong AMF.

b) Phương pháp thủ công tạo GUI

Để có thể truy cập vào tham số của mỗi phần tử (*object*) của GUI, phần tử đó phải có khả năng được MATLAB nhận biết một cách chính xác. Có thể bảo đảm khả năng nhận biết nhờ đặc điểm *Tag*. Lệnh findobj('Tag',*string*) sẽ hiển thị chỉ số nhận biết (gọi là handles) của phần tử có tên *string*.

Việc truy cập đọc nội dung được thực hiện bởi lệnh variable = get(handle,'Parameter'). Tương tự, có thể ghi bằng lệnh set(handle,'Parameter'). Để ghi / đọc số liệu vào các vùng 'text', 'edit', ta phải dùng các lệnh scanf, sprintf hoặc num2str.

Khi định tạo GUI theo phương pháp thủ công, người thiết kế nên xác định trước phạm vi tính năng dự định cũng như vẽ phác bố cục của *Layout*. Thao tác mang tính chuẩn bị này sẽ làm đơn giản công việc sau này rất nhiều.

Bằng lệnh *handle* = figure ('*paramater*',*value*, ...) ta tạo một cửa sổ rỗng. Sau này, giá trị *handle* sẽ được dùng để sắp xếp trật tự các phần tử con (gọi là *Childs*) của *Layout*.

```
h0 = figure('Units','points', ...
    'Position',[100 100 300 100];
```

Các phần tử con được tạo bởi *handle* = uicontrol (*'paramater',value*, ...). Lệnh đó cho phép ta quy định đặc điểm của phần tử con, và mỗi đặc điểm sẽ được khai báo trên một dòng riêng, xuống dòng bởi dấu ... Trình tự khai báo đặc điểm không có ý nghĩa quan trọng. Ví dụ: Đoạn mã MATLAB của ô bên trái hình 2.17 có thể có dạng sau:

```
h1 = uicontrol('Parent',h0, ...     % Vùng nhập ký tự
    'Units','points', ...
    'Position',[20 13 60 15], ...
    'BackgroundColor',[1 1 1], ...
    'String','    1.00', ...
    'Tag','Value_Euro', ...         % Nhận biết phần tử
    'Style','edit');
```

Các tham số Callback sẽ được nhập theo cách tương tự. Bởi vì các lệnh của MATLAB đều được viết dưới dạng ký tự, để dễ phân biệt: Mọi ký tự trong phạm vi một lệnh đều phải được gói gọn bên trong dấu ngoặc kép '' giống như trong ví dụ sau đây:

```
h1 = uicontrol('Parent',h0, ...
    'Callback',['Land=get(findobj(''Tag'',''Land''),''Value'');';
```

Bảng sau đây tổng hợp các lệnh do MATLAB cung cấp để tạo GUI thủ công.

Các lệnh tạo GUI

handle =	figure (*'parameter',value*, ...)	Tạo cửa sổ rỗng
handle =	uicontrol (*'parameter',value*, ...)	Tạo phần tử GUI
handle =	uimenu (*'parameter',value*, ...)	Tạo menu của GUI
handle =	findobj (*'Tag',string*)	Tìm chỉ số *handle*
variable =	get (*handle,'parameter'*)	Đọc tham số
	set (*handle,'parameter',value*)	Ghi tham số
string =	num2str (*variable*[,*format*])	Đảo số → Ký tự
string =	sprintf (*string,variable*)	Xuất ký tự
variable =	sscanf (*string,format*)	Đọc ký tự

Sau đây là mã MATLAB của ví dụ "giao diện tính quy đổi tiền", thiết kế theo phương pháp thủ công. Phần mã in ra đây chỉ bao gồm những tham số cần thiết nhất.

```
h0 = figure('Units','points', ...  % Tham số của Layout
    'Position',[100 100 300 100], ...
    'NumberTitle','off', ...       % Không có "Figure" ở dòng tên
    'Name','AUTOLAB - Example for manual made GUI', ...
                                   % Dòng tên của Layout
    'MenuBar','none');             % Không cần Menu ?
```

```
h1 = uicontrol('Parent',h0, ...     % Vùng Text
    'Units','points', ...
    'Position',[0 70 300 30], ...
    'FontSize',24, ...
    'String','Exchange of Funds', ...
    'Style','text');

% Chọn nước
Factor   = 1.95583;                 % Đặt trước
Monetary = 'DEM';                   % Đặt trước

h1 = uicontrol('Parent',h0, ...     % Tạo Popup Menu ?
    'Units','points', ...
    'Position',[190 40 90 15], ...
    'BackgroundColor',[1 1 1], ...
    'String',['Germany';'France ';'Italy  ';'Austria'], ...
    'Callback',['Land=get(findobj(''Tag'',''Land''),''Value'');',...
      'switch Land case 1,Factor = 1.95583; Monetary = ''DEM'';',...
      '            case 2,Factor = 6.55957; Monetary = ''FRF'';',...
      '            case 3,Factor = 1936.27; Monetary = ''ITL'';',...
      '            case 4,Factor = 13.7603; Monetary = ''ATS'';end;',...
      'set (findobj(''Tag'',''Monetary''), ''String'', Monetary);'], ...
    'Tag','Land', ...               % Nhận biết phần tử
    'Style','popupmenu', ...
    'Value',1);                     % Đặt trước: Germany

% Giá trị đã tính quy đổi sang Euro

h1 = uicontrol('Parent',h0, ...     % Vùng nhập dữ liệu
    'Units','points', ...
    'Position',[20 13 60 15], ...
    'BackgroundColor',[1 1 1], ...
    'String','    1.00', ...
    'Tag','Value_Euro', ...         % Nhận biết phần tử
    'Style','edit');

h1 = uicontrol('Parent',h0, ...     % Vùng nhập Text
    'Units','points', ...
    'Position',[80 13 30 15], ...
    'String','EUR', ...
    'Style','text');

% Giá trị nhập bằng đồng tiền quốc gia

h1 = uicontrol('Parent',h0, ...     % Vùng nhập dữ liệu
    'Units','points', ...
    'Position',[190 13 60 15], ...
```

```
    'BackgroundColor',[1 1 1], ...
    'String',' ` 0.00', ...          % Đặt trước
    'Tag','Value_Land', ...          % Nhận biết phần tử
    'Style','edit');

h1 = uicontrol('Parent',h0, ...      % Vùng Text (Xuất)
    'Units','points', ...
    'Position',[250 13 30 15], ...
    'String','DEM', ...              % Đặt trước
    'Tag','Monetary', ...            % Nhận biết phần tử
    'Style','text');

% Tính quy đổi

h1 = uicontrol('Parent',h0, ...      % Vùng chứa ký hiệu ===>
    'Units','points', ...
    'Position',[130 36 40 20], ...
 'Callback',['Value_Euro=sscanf(get(findobj(''Tag'',''Value_Euro''),',...
   '         ''String''), ''%f'');', ...
   'Value_Land = sprintf(''%8.2f'', Value_Euro*Factor);', ...
'set(findobj(''Tag'',''Value_Land''),''String'',Value_Land);',...
'set(findobj(''Tag'',''Monetary''),''String'',Monetary);'], ...
   'Style','pushbutton', ...
   'String','===>');

h1 = uicontrol('Parent',h0, ...      % Vùng chứa ký hiệu <===
    'Units','points', ...
   'Position',[130 10 40 20], ...
 'Callback',['Value_Land=sscanf(get(findobj(''Tag'',''Value_Land''), ...
              ''String''), ''%f'');', ...
   'Value_Euro = sprintf(''%8.2f'', Value_Land / Factor);', ...
'set(findobj(''Tag'',''Value_Euro''),''String'',Value_Euro);',...
'set(findobj(''Tag'',''Monetary''), ''String'', Monetary);'], ...
   'Style','pushbutton', ...
   'String','<===');
```

2.10 Tóm tắt nội dung chương 2

Chương 2 nhằm giúp bạn đọc trả lời được các câu hỏi sau đây:

1. Làm thế nào để định nghĩa một biến có dạng đoạn văn bản (chuỗi ký tự, *Strings*) ?
2. Làm thế nào để xuất số liệu hay chuỗi ký tự ra màn hình ?
3. Những lệnh nào giúp ta xuất chuỗi ký tự theo định dạng ?
4. Có những ký hiệu đặc biệt hay phần tử định dạng nào khi xuất ?
5. Làm thế nào để cất văn bản hay dữ liệu dưới dạng File ?

6. Làm thế nào để cất dữ liệu dưới dạng mã nhị phân hay mã ASCII ?
7. Sử dụng các lệnh của hệ điều hành như thế nào ?
8. Cấu trúc của đồ họa MATLAB ?
9. Cú pháp tổng quát của một lệnh vẽ đồ thị trong MATLAB ?
10. Các khả năng điền ký tự và chia trục của đồ thị cần vẽ ?
11. Làm thế nào để vẽ đồ thị 2 chiều (2-D) ?
12. Làm thế nào để vẽ đồ thị 2 chiều (2-D) logarith ?
13. Làm thế nào để vẽ mặt cong 3 chiều (3-D) ?
14. Đồ họa 2-D và 3-D khác nhau ở đặc điểm gì ?
15. Các khả năng điền ký tự và chia trục của đồ họa 3-D ?
16. Ý nghĩa của *x_value*, *y_value* và *z_value* trong lệnh `plot3(`*x_value*, *y_value*, *z_value*`)` ?
17. Tác dụng của lệnh `[X,Y] = meshgrid(`*x_vector*, *y_vector*`)` ?
18. Làm thế nào để in hay cất vào bộ nhớ các đồ thị, đồ họa đã tạo được ?

3 Control System Toolbox: Công cụ khảo sát - thiết kế hệ thống điều khiển

Control System Toolbox (CST) là một bộ công cụ cực kỳ có ý nghĩa và tiện lợi đối với kỹ sư điều khiển tự động hay những người nghiên cứu lý thuyết hệ thống. Với CST ta có thể thực hiện tất cả các bước cần thiết để khảo sát – thiết kế hệ thống, đặc biệt là các hệ thống điều khiển (*Control System*):

- Mô tả các hệ tuyến tính – dừng (hệ có tham số hằng) dưới dạng liên tục hay gián đoạn (hàm truyền đạt, sơ đồ phân bố điểm không - điểm cực, mô hình trạng thái, mô hình đặc tính tần số; mục 3.1).
- Chuyển đổi hoặc xử lý hệ (mục 3.2), phân tích đặc tính hệ thống (động học, đáp ứng bước nhảy, giảm bậc; mục 3.3).
- Thiết kế và tính tối ưu các khâu điều chỉnh (khâu ĐC: quỹ đạo điểm cực, gán cực, tối ưu LQ[1]; mục 3.4).

Ngoài ra, CST còn cung cấp một số thuật toán cho phép đánh giá độ tin cậy của các phép toán số (mục 3.5) sử dụng khi khảo sát hệ thống. Phần lớn các thuật toán số được cất dưới dạng *m-Files* (các *Scripts*) và do đó hoàn toàn ngỏ, cho phép người sử dụng sửa đổi, bổ sung theo ý mình.

3.1 Mô hình hóa các hệ tuyến tính - dừng (hệ LTI[2])

Thường bao giờ ta cũng bắt đầu công việc nghiên cứu hệ thống điều khiển bằng việc mô hình hóa. Xuất phát từ các đặc điểm vật lý, ta tìm cách mô tả hệ bằng toán, phản ánh các quan hệ nhân quả giữa các tín hiệu vào **u** và các tín hiệu ra **y**. MATLAB cung cấp cho các hệ LTI (liên tục hoặc gián đoạn) có dạng:

- hệ một vào / một ra (hệ SISO: *single-input / single-output*) hay
- hệ nhiều vào / nhiều ra (hệ MIMO: *multiple-input / multiple-output*)

bốn phương thức mô tả sau đây:

- Hàm truyền đạt (*Transfer Function*: TF)
- Mô hình điểm không - điểm cực (*Zero Pole Gain*: ZPK)
- Mô hình trên không gian trạng thái (*State Space*: SS)

---

[1] Tối ưu LQ: linear – quadratic optimal control

[2] Hệ LTI: linear time-invariant systems

- Mô hình dữ liệu đặc tính tần số (*Frequency Response Data*: FRD)

Trong bốn phương thức, các mô hình TF, ZPK và SS đều là mô hình tham số, còn mô hình FRD phục vụ mô tả và phân tích đặc tính của hệ với bộ dữ liệu thu được qua đo đạc hay mô phỏng. Chính vì vậy, việc phân tích sử dụng mô hình FRD chỉ hạn chế vào các phương pháp đặc tính tần số.

Bạn đọc có thể xem chi tiết về mô hình bằng cách gọi lệnh `ltimodels`.

3.1.1 Mô hình truyền đạt

Hàm truyền đạt là một phân thức hữu tỷ của s với đa thức tử số num (*numerator*) và đa thức mẫu số den (*denominator*), mô tả đặc tính truyền đạt của hệ trên miền ảnh Laplace.

$$h(s) = \frac{num(s)}{den(s)} = \frac{a_m s^m + a_{m-1} s^{m-1} + \cdots + a_1 s^1 + a_0}{b_n s^n + b_{n-1} s^{n-1} + \cdots + b_1 s^1 + b_0} \tag{3.1}$$

Khai báo TF cho hệ SISO

Hàm truyền đạt của hệ SISO có thể được khai báo theo hai cách:

1. Sử dụng lệnh `tf(num,den)`: Nhập đa thức tử số num và đa thức mẫu số den dưới dạng vector tham số của s theo trình tự số mũ của s bé dần.

```
>> h = tf ([2 -3], [1 1])
Transfer function:
2 s - 3
-------
 s + 1
```

2. Khai báo dưới dạng hàm hữu tỷ của s: Trước hết ta phải khai báo s là biến của mô hình TF, sau đó nhập hàm truyền đạt dưới dạng hàm hữu tỷ của s.

```
>> s = tf ('s')
Transfer function:
s
>> h = (s+2)/(s^2+5*s+4)
Transfer function:
    s + 2
-------------
s^2 + 5 s + 4
```

Chú ý: Khi s đã được khai là biến của mô hình TF, mọi mô hình tiếp theo dưới dạng hàm hữu tỷ của s đều được coi là mô hình TF, cho tới khi s được khai báo mới.

Khai báo TF cho hệ MIMO

Hàm truyền đạt của hệ MIMO được mô tả dưới dạng ma trận 2 chiều **H**, với h_{ij} là phần tử truyền đạt từ đầu vào thứ j tới đầu ra thứ i. Tức là: Các hàng ứng với các biến ra và các cột ứng với các biến vào.

$$\mathbf{H}=\begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}=\begin{bmatrix} \dfrac{num_{11}}{den_{11}} & \dfrac{num_{12}}{den_{12}} \\ \dfrac{num_{21}}{den_{21}} & \dfrac{num_{22}}{den_{22}} \end{bmatrix}=\begin{bmatrix} \dfrac{2s-3}{s+1} & \dfrac{s^2+s-6}{s+5} \\ \dfrac{s+2}{s^2+5s+4} & \dfrac{-1}{2s^2+6s+10} \end{bmatrix} \tag{3.2}$$

Hàm truyền đạt (3.2) có thể được khai báo theo hai cách:

1. Khai báo TF cho từng phần tử của ma trận truyền đạt **H**:

```
>> h11 = tf ([2 -3], [1 1]);
>> h12 = tf ([1 1 -6], [1 5]);
>> h21 = tf ([1 2], [1 5 4]);
>> h22 = tf (-1, [2 6 10]);
```

hoặc dưới dạng hàm hữu tỷ của *s*:

```
>> s = tf ('s')
>> h11 = (2*s-3)/(s+1);
>> h12 = (s^2+s-6)/(s+5);
>> h21 = (s+2)/(s^2+5*s+4);
>> h22 = -1/(2*s^2+6*s+10);
```

Sau đó ghép lại các phần tử lại thành ma trận **H**:

```
>> H = [h11 h12 ; h21 h22];
```

2. Khai báo hai trường (*Cell Arrays*) với kích cỡ *Ny* × *Nu* (*Ny*: số biến ra, *Nu*: số biến vào): Trường **NUM** có chứa các đa thức tử số và trường **DEN** chứa các đa thức mẫu số.

$$\mathbf{NUM}=\begin{bmatrix} num_{11} & num_{12} \\ num_{21} & num_{22} \end{bmatrix};\quad \mathbf{DEN}=\begin{bmatrix} den_{11} & den_{12} \\ den_{21} & den_{22} \end{bmatrix} \tag{3.3}$$

Đến đây ta có thể nhập các vector tham số của *s* theo trình tự số mũ giảm dần như sau:

```
>> NUM = {[2 -3] [1 1 -6] ; [1 2] -1};
>> DEN = {[1 1] [1 -5] ; [1 5 4] [2 6 10]};
```

Ma trận **H** được khai báo bằng cách gọi `tf` với các tham số **NUM** và **DEN**:

```
>> H = tf(NUM, DEN);
```

Có thể kiểm tra kết quả bằng cách gọi **H**:

```
>> H
 Transfer function from input 1 to output...
      2 s - 3
 #1:  -------
       s + 1
           s + 2
 #2:  -------------
      s^2 + 5 s + 4

 Transfer function from input 2 to output...
      s^2 + s - 6
 #1:  -----------
         s - 5
              -1
 #2:  ----------------
      2 s^2 + 6 s + 10
```

Khi hệ MIMO chỉ chứa các phần tử tỷ lệ không có quán tính

Nếu hệ MIMO chỉ chứa duy nhất các phần tử tỷ lệ không có quán tính (các hệ số khuếch đại), khi ấy ta chỉ cần gọi như sau, để khai báo mô hình TF:

```
>> G = tf ([1 2 ; 3 0])
Transfer function from input 1 to output...
 #1:  1
 #2:  3

Transfer function from input 2 to output...
 #1:  2
 #2:  0
```

3.1.2 Mô hình điểm không - điểm cực

Dạng mô tả đặc tính truyền đạt tương tự như hàm truyền đạt là mô hình điểm không - điểm cực:

$$h(s) = k\frac{(s-z_1)\cdots(s-z_{m-1})(s-z_m)}{(s-p_1)\cdots(s-p_{n-1})(s-p_n)} \tag{3.4}$$

Trong (3.4) k là hệ số khuếch đại với giá trị thực, $z_1 \ldots z_m$ và $p_1 \ldots p_n$ là các giá trị thực và / hoặc các cặp giá trị phức liên hợp của điểm không, điểm cực[1]. Chúng chính là nghiệm của đa thức tử và mẫu số.

[1] Zero, pole

Khai báo ZPK cho hệ SISO

Giống như mô hình TF, ta có hai khả năng để khai báo một mô hình ZPK:

1. Sử dụng lệnh `zpk(z,p,k)`: Nhập vector điểm không `z ([-6 1 1])`, vector điểm cực `p ([-5 1])` và hệ số khuếch đại `k (3)`.

```
>> h = zpk ([-6 1 1],[-5 1],3)

Zero/pole/gain:
3 (s+6) (s-1)^2
---------------
  (s+5) (s-1)
```

2. Khai báo dưới dạng hàm hữu tỷ của *s*: Trước hết ta phải khai báo *s* là biến của mô hình ZPK, sau đó nhập hàm truyền đạt dưới dạng hàm hữu tỷ của *s*.

```
>> s = zpk('s')

Zero/pole/gain:
s

>> h = 2*1/((s-1)*(s+2))

Zero/pole/gain:
      2
-----------
(s-1) (s+2)
```

Cần phải lưu ý: Khi ta đã khai báo *s* là biến của mô hình ZPK, tất cả các mô hình biểu diễn dưới dạng hàm hữu tỷ của *s* sẽ được coi là mô hình ZPK, tới khi nào ta khai báo lại biến s hay biến mô hình ZPK sang mô hình TF.

Khai báo ZPK cho hệ MIMO

Ma trận truyền đạt **H** với kích cỡ $Ny \times Nu$ và các phần tử h_{ij}, đặc trưng cho hàm truyền đạt từ đầu vào thứ j (cột) tới đầu ra thứ i (hàng) ví dụ có dạng sau:

$$\mathbf{H}=\begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}=\begin{bmatrix} k_{11}\dfrac{z_{11}}{p_{11}} & k_{12}\dfrac{z_{12}}{p_{12}} \\ k_{21}\dfrac{z_{21}}{p_{21}} & k_{22}\dfrac{z_{22}}{p_{22}} \end{bmatrix}=\begin{bmatrix} 2\dfrac{1}{(s-1)(s+2)} & 3\dfrac{(s-1)^2(s+6)}{(s-1)(s+5)} \\ 1\dfrac{(s-1)(s-2)}{(s+4)(s+1)} & -1\dfrac{s+1}{(s^2+2s+2)(s+1)} \end{bmatrix} \tag{3.5}$$

Có thể khai báo **H** theo 2 cách mà ta đã biết ở trên:

1. Dùng lệnh `zpk` để khai báo từng hàm truyền đạt ZPK-SISO riêng rẽ:

```
>> h11 = zpk ([],[-2 1],2);
>> h12 = zpk ([-6 1 1],[-5 1],3);
>> h21 = zpk ([1 2],[-4 -1],1);
>> h22 = zpk (-1,[-1 -1+i -1-i],-1);
```

hoặc dưới dạng hàm hữu tỷ của *s*

```
>> s = zpk('s');
>> h11 = 2*1/((s-1)*(s+2));
>> h12 = 3*(s^2-2*s+1)*(s+6)/(s^2+4*s-5);
>> h21 = (s^2-3*s+2)/(s^2+5*s+4);
>> h22 = -1*(s+1)/((s^2+2*s+2)*(s+1));
```

và sau đó ghép các phần tử thành ma trận **H**:

```
>> H = [h11 h12 ; h21 h22];
```

2. Khai báo hai trường (*Cell Arrays*, kích cỡ $Ny \times Nu$, có các đầu vào $i = 1 \ldots Ny$, các đầu ra $j = 1 \ldots Nu$) cho các đa thức điểm không (**Z**), các đa thức điểm cực (**P**) và ma trận (**K**) kích cỡ $Ny \times Nu$ cho các hệ số khuếch đại.

$$\mathbf{Z} = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix}; \mathbf{P} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}; \mathbf{K} = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} \tag{3.6}$$

```
>> Z = {[] [-6 1 1];[1 2] -1};
>> P = {[-2 1] [-5 1];[-4 -1] [-1 -1+i -1-i]};
>> K = [2 3 ; 1 -1];
```

Sau đó gọi `zpk` với các tham số **Z**, **P** và **K**:

```
>> H = zpk (Z,P,K)

Zero/pole/gain from input 1 to output...
           2
 #1:  -----------
      (s+2) (s-1)
       (s-1) (s-2)
 #2:  -----------
      (s+4) (s+1)
Zero/pole/gain from input 2 to output...
      3 (s+6) (s-1)^2
 #1:  ---------------
        (s+5) (s-1)
            - (s+1)
 #2:  --------------------
      (s+1) (s^2 + 2s + 2)
```

3.1.3 Mô hình trạng thái

Trong phương pháp mô tả bằng mô hình trạng thái ta không quan tâm tới quan hệ truyền đạt giữa các biến vào và biến ra, mà chỉ quan tâm tới số *các phần tử tích lũy năng lượng độc lập* tồn tại trong hệ: Ta phải tìm cho mỗi phần tử tích lũy (ứng với một khâu tích phân, Intergrator) *một phương trình vi phân bậc 1*. Chính vì vậy, một hệ có n khâu tích lũy độc lập sẽ không được mô tả bởi một hàm truyền đạt bậc n, mà bởi một *hệ n phương trình vi phân bậc 1*. Hệ MIMO thường được mô tả bởi mô hình trạng thái tổng quát, viết dưới dạng ma trận như sau:

- Phương trình trạng thái:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{3.7}$$

- Phương trình đầu ra:

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{3.8}$$

với:

x:	Vector trạng thái	($Nx \times 1$)	**A**:	Ma trận trạng thái	($Nx \times Nx$)
u:	Vector biến vào	($Nu \times 1$)	**B**:	Ma trận đầu vào	($Nx \times Nu$)
y:	Vector biến ra	($Ny \times 1$)	**C**:	Ma trận đầu ra	($Ny \times Nx$)
			D:	Ma trận liên thông	($Ny \times Nu$)

Trong hệ trên, Nx là bậc của hệ (ứng với số phần tử tích lũy năng lượng độc lập trong hệ), Nu là số biến vào và Ny là số biến ra. Khi đối tượng là hệ SISO thường gặp, chỉ có 1 biến vào và 1 biến ra, khi ấy hai phương trình (3.7),(3.8) sẽ trở nên đơn giản và ta chỉ việc thay thế **B** bởi **b** ($Nx \times 1$), **C** bởi $\mathbf{c}^T$ ($Nx \times 1$) và **D** bởi d (**b**, **c** là hai vector, d là đại lượng vô hướng):

- Phương trình trạng thái:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{b}u \tag{3.9}$$

- Phương trình đầu ra:

$$y = \mathbf{c}^T\mathbf{x} + du \tag{3.10}$$

Vì trong thực tiễn, hầu hết các đối tượng điều khiển đều có đặc điểm quán tính (đặc điểm lọc thông thấp) nên không có thành phần liên thông, tức là $\mathbf{D} = \mathbf{0}$ hoặc $d = 0$.

Việc khai báo mô hình trạng thái trong MATLAB được thực hiện một cách rất đơn giản nhờ lệnh `ss(A,B,C,D)`, trong đó **A**, **B**, **C** và **D** là các ma trận hay vector tương ứng.

Chú ý:

- *Khi đối tượng là khâu quán tính*: Khi ấy chỉ cần gán cho ma trận liên thông **D** giá trị vô hướng 0 là đủ, mà không cần quan tâm tới kích cỡ của **u** và **y**.

```
>> D = 0;
```

- Hệ SISO: Nếu vector **c** vốn là vector cột, ta phải chuyển **c** thành vector hàng $\mathbf{c}^T$.

```
>> ss(A,B,C',D)
```

Để minh họa ta theo dõi ví dụ về đối tượng MIMO có 2 trạng thái, 2 đầu vào và 3 đầu ra sau đây.

$$\mathbf{A}=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix};\mathbf{B}=\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix};\mathbf{C}=\begin{bmatrix}0 & 1\\ 1 & 2\\ 3 & 1\end{bmatrix};\mathbf{D}=\begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}$$

Khai báo với MATLAB:

```
>> SYS = ss([1 2 ; 3 4],[1 1 ; 0 1],[0 1 ; 1 2 ; 3 1],0)

a =
                    x1          x2
        x1           1           2
        x2           3           4

b =
                    u1          u2
        x1           1           1
        x2           0           1

c =
                    x1          x2
        y1           0           1
        y2           1           2
        y3           3           1

d =
                    u1          u2
        y1           0           0
        y2           0           0
        y3           0           0
```

3.1.4 Mô hình dữ liệu đặc tính tần số

Ngược lại với ba loại mô hình đã giới thiệu, mô hình dữ liệu đặc tính tần số FRD không dựa trên cơ sở mô tả toán học của hệ thống, của đối tượng, FRD dựa trên một bộ dữ liệu đặc tính tần số thu thập được qua đo đạc hay mô phỏng. Có

thể thu thập được bộ dữ liệu đó thông qua đo biên độ và góc pha của tín hiệu ra khi kích thích hệ ở đầu vào bởi một tín hiệu hình sin với các tần số khác nhau.

Tín hiệu ra thường có dạng như sau:

$$y(t)=|Y(\omega)|\sin(\omega t+\varphi(\omega))=|F(j\omega)|\,\mathrm{Im}\left\{e^{j(\omega t+\varphi(\omega))}\right\} \tag{3.11}$$

Quan sát thuần túy toán, hàm đặc tính tần số chính là đáp ứng phức của hệ khi kích thích ở đầu vào bởi một tín hiệu hình sin:

$$F(j\omega)=|F(j\omega)|e^{j\varphi(\omega)}=\frac{a_m(j\omega)^m+\cdots+a_1(j\omega)+a_0}{b_n(j\omega)^n+\cdots+b_1(j\omega)+b_0} \tag{3.12}$$

$$|F(j\omega)|=\sqrt{\mathrm{Re}\{F(j\omega)\}^2+\mathrm{Im}\{F(j\omega)\}^2} \tag{3.13}$$

$$\varphi(\omega)=\mathrm{arctg}\frac{\mathrm{Im}\{F(j\omega)\}}{\mathrm{Re}\{F(j\omega)\}} \tag{3.14}$$

Khai báo mô hình FRD cho hệ SISO

Việc khai báo được thực hiện bởi lệnh `frd(answer, freq, unit)`. Trong đó, *answer* là vector chứa các đáp ứng phức, ứng với các tần số cất trong vector *freq*. *unit* là tham số tùy chọn, cho biết đơn vị của tần số là `'rad/s'` (đơn vị chuẩn mặc định) hay là `'Hz'`.

Để minh họa ta hãy khai báo khâu quán tính bậc nhất PT_1 sau đây:

$$F=\frac{1}{1+j\omega T}=\frac{1-j\omega T}{1+(\omega T)^2}=\frac{1}{\sqrt{1+(\omega T)^2}}e^{-j\,\mathrm{arctg}(\omega T)} \tag{3.15}$$

```
>> freq = [0.01 0.1 1 10 100 1000 10000];     % Tần số
>> answer = (1-j*freq) ./(1+freq.^2);         % Tạo PT1
>> sysfrd = frd(answer,freq,'Units','rad/s')

From input 1 to:

  Frequency(rad/s)           output 1
  ----------------           --------
             0.01      9.99900e-001-0.009999i
             0.10      9.90099e-001-0.099010i
             1.00      5.00000e-001-0.500000i
            10.00      9.90099e-003-0.099010i
           100.00      9.99900e-005-0.009999i
          1000.00      9.99999e-007-0.001000i
         10000.00      1.00000e-008-0.000100i

Continuous-time frequency response data model.
```

Khai báo mô hình FRD cho hệ MIMO

Mô hình FRD của hệ MIMO được khai báo cũng theo phương thức tương tự, điểm khác là: Đáp ứng phức *answer* không còn là vector mà là một *tensor*[1] có kích cỡ $Ny \times Nu \times Nf$, với Nu biến vào, Ny biến ra, và vector tần số *freq* có độ dài Nf.

3.1.5 Mô hình gián đoạn theo thời gian

Đối với mô hình gián đoạn theo thời gian ta không dùng ảnh Laplace mà dùng phương pháp ảnh z để mô tả hệ, và thay thế các phương trình vi phân sẽ là các phương trình sai phân. Các mô hình gián đoạn được khai báo tương tự như mô hình thời gian liên tục, thêm vào đó là *chu kỳ trích mẫu* T_s dưới dạng tham số `Ts`. Nếu ta chưa xác định chu kỳ trích mẫu, `Ts` sẽ tự động nhận giá trị –1. Để khai báo ta sử dụng các lệnh MATLAB sau đây:

systf = `tf` (*num,den,Ts*)
syszpk = `zpk` (*z,p,k,Ts*)
sysss = `ss` (*a,b,c,d,Ts*)
sysfrd = `frd` (*answer,freq,Ts*)

Mô hình gián đoạn TF và ZPK

Hàm truyền đạt gián đoạn được định nghĩa là hàm hữu tỷ của toán tử z, mô tả dưới dạng hàm truyền đạt (mô hình TF) hay biểu đồ điểm không - điểm cực (mô hình ZPK).

$$h_{TF} = \frac{z-0,5}{z^2+z-2} \quad \triangleq \quad h_{ZPK} = \frac{z-0,5}{(z+2)(z-1)}$$

Ta có thể khai báo theo 2 cách:

1. Dùng lệnh `tf(num,den,Ts)` hoặc lệnh `zpk(z,p,k,Ts)`: Khai báo các tham số hệ thống giống như mục 3.1.1 và 3.1.2, bổ xung thêm chu kỳ trích mẫu T_s.

```
>> h = tf ([1 -0.5],[1 1 -2],0.01)

Transfer function:
  z - 0.5
-----------
z^2 + z - 2

Sampling time: 0.01
```

Mô hình ZPK:

[1] Tensor: Ma trận nhiều chiều

```
>> h = zpk (0.5,[-2 1],1,0.01)

Zero/pole/gain:
  (z-0.5)
-----------
(z+2) (z-1)
Sampling time: 0.01
```

2. Khai báo hàm hữu tỷ của z: Trước hết ta phải khai báo z là biến của mô hình TF hoặc ZPK, sau đó nhập hàm truyền đạt dưới dạng hàm hữu tỷ của z.

Mô hình TF

```
>> z = tf ('z',0.01)

Transfer function:
z
Sampling time: 0.01

>> h = (z-0.5)/(z^2+z-2)

Transfer function:
  z - 0.5
-----------
z^2 + z - 2

Sampling time: 0.01
```

Mô hình ZPK

```
>> z = zpk ('z',0.01)

Zero/pole/gain:
z
Sampling time: 0.01

>> h = (z-0.5)/((z+2)*(z-1))

Zero/pole/gain:
  (z-0.5)
-----------
(z+2) (z-1)

Sampling time: 0.01
```

Mô hình gián đoạn TF theo định dạng DSP[1]

Trong lĩnh vực điều khiển tự động, cần đến xử lý tín hiệu số, các hàm truyền đạt thường được cho dưới dạng *hàm hữu tỷ của biến* z^{-1}, viết *theo thứ tự số mũ tăng dần*. Điều này mâu thuẫn với cách viết thông thường, các đa thức của z với bậc khác nhau ở tử và mẫu số được viết theo thứ tự số mũ giảm dần. Giải quyết mâu thuẫn này, MATLAB cung cấp lệnh `filt(num,den,Ts)`, cho phép ta khai báo mô hình TF theo định dạng DSP.

```
>> h = filt ([1 -0.5],[1 1 -2],0.01)

Transfer function:
  1 - 0.5 z^-1
-----------------
1 + z^-1 - 2 z^-2

Sampling time: 0.01
```

[1] DSP: Digital Signal Processing (xử lý tín hiệu số)

Mô hình gián đoạn SS

Mô hình gián đoạn SS (mô hình gián đoạn trên không gian trạng thái) có dạng tổng quát:

$$\mathbf{x}(k+1) = \mathbf{A}\,\mathbf{x}(k) + \mathbf{B}\,\mathbf{u}(k) \tag{3.16}$$

$$\mathbf{y}(k) = \mathbf{C}\,\mathbf{x}(k) + \mathbf{D}\,\mathbf{u}(k) \tag{3.17}$$

Trong hai phương trình trên ta cần hiểu: $\mathbf{a}(k) = \mathbf{a}(kT_s)$. Việc khai báo hoàn toàn tương tự mục 3.1.3: Thêm tham số `Ts`.

```
>> sys = ss([ 3 ],[ 1 3 ],[ 0.5 ],0,-1);
```

Các phương thức khai báo mô hình gián đoạn cho đối tượng tuyến tính – dừng (hệ LTI) được tập hợp lại trong dưới đây.

Khai báo mô hình gián đoạn của hệ LTI

`tf` (*num,den,Ts*)	**Hàm truyền đạt:** Vector các hệ số của đa thức tử số *num*, vector các hệ số của đa thức mẫu số *den*
`zpk` (*z,p,k,Ts*)	**Biểu đồ điểm không - điểm cực:** Vector các điểm không *z*, vector các điểm cực *p*, vector các hệ số khuếch đại *k*
`ss` (*A,B,C,D,Ts*)	**Mô hình trạng thái:** Ma trận hệ thống **A**, ma trận đầu vào **B**, ma trận đầu ra **C**, ma trận liên thông **D**
`frd` (*answer,freq,unit,Ts*)	**Mô hình dữ liệu đặc tính tần số:** Đáp ứng tần số *answer*, vector tần số *freq*, *unit* là đơn vị (thứ nguyên) của tần số `rad/s` (mặc định) hoặc `Hz` (`unit='Units','rad/s'`)
T_s	**Chu kỳ trích mẫu** của hệ gián đoạn không khai T_s: Mô hình *liên tục về thời gian* T_s = -1: Mô hình *gián đoạn về thời gian*, chu kỳ trích mẫu chưa xác định

3.1.6 Thời gian trễ trong các hệ tuyến tính - dừng (hệ LTI)

Trong các hệ thống kỹ thuật thường tồn tại hiện tượng trễ thời gian. Nghĩa là, chỉ sau một khoảng thời gian nhất định, tín hiệu kích thích mới có hiệu lực ở đầu vào, hoặc ta mới quan sát được tín hiệu đáp ứng ở đầu ra, hoặc nội tại của hệ thống có các hiệu ứng trễ nào đó. Để mô hình hóa hiện tượng trễ, *Control System Toolbox* cung cấp cho ta các lệnh sẽ được mô tả dưới đây.

Thời gian trễ giữa đầu vào - đầu ra

Trên miền ảnh Laplace, đặc điểm thời gian trễ T_d thể hiện qua việc nhân thêm $e^{-T_d s}$ vào hàm truyền đạt. Trong MATLAB, đặc điểm trễ thời gian giữa đầu vào thứ j và đầu ra thứ i được khai báo thông qua tính chất `IODelay` của mô hình LTI, thể hiện bởi ma trận `Td` chứa thời gian trễ giữa từng cặp vào/ra. Ma trận `Td` phải có kích cỡ đúng như kích cỡ của mô hình LTI. Lệnh có dạng:

```
set(sys,'IODelay',Td)
```

Để minh họa, ta sử dụng ví dụ khai báo mô hình TF cho hệ MIMO có 2 biến vào và 2 biến ra ở cuối mục 3.1.1. Ta lần lượt khai báo ma trận `NUM` của các đa thức tử số, ma trận `DEN` của các đa thức mẫu số, định nghĩa ma trận truyền đạt `H`. Dùng lệnh `set` để gán cho quan hệ giữa đầu vào 2 và đầu ra 1 thời gian trễ là 0,4s. Cuối cùng ta gọi `H(1,2)` để kiểm tra phần tử của `H` ở hàng 1, cột 2.

```
>> NUM = {[2 -3] [1 1 -6] ; [1 2] -1};        % Tử số
>> DEN = {[1 1] [1 -5] ; [1 5 4] [2 6 10]};   % Mẫu số
>> H = tf(NUM, DEN);                          % Khai báo H
>> H                                       % Kiểm tra kết quả

Transfer function from input 1 to output...
      2 s - 3
 #1:  -------
       s + 1

          s + 2
 #2:  -------------
      s^2 + 5 s + 4

Transfer function from input 2 to output...
      s^2 + s - 6
 #1:  -----------
         s - 5

             -1
 #2:  ----------------
      2 s^2 + 6 s + 10

>> set(H,'IODelay',[0 0.4 ; 0 0]);  % Trễ: input 2/output 1
>> H(1,2)                    % Kiểm tra TF: input 2/output 1

Transfer function:
                s^2 + s - 6
exp(-0.4*s) * -----------
                  s - 5
```

Thời gian trễ xuất hiện trực tiếp ở đầu vào hay đầu ra

Nếu thời gian trễ xuất hiện trực tiếp ở đầu vào hay đầu ra, ta có thể khai báo chúng bằng cách bổ sung đặc điểm `'InputDelay'` hoặc `'OutputDelay'` vào mô hình SS của đối tượng LTI. Trên miền thời gian, khi có trễ T_{di} giữa vector biến vào **u** và vector trạng thái **x**, hoặc trễ T_{do} giữa vector trạng thái **x** và vector biến ra **y**, mô hình trạng thái được viết dưới dạng:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t - T_{di}) \tag{3.18}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t - T_{do}) + \mathbf{D}\mathbf{u}(t - (T_{di} + T_{do})) \tag{3.19}$$

Theo mô hình trên, việc khai báo thời gian trễ trở nên đơn giản hơn nhiều so với trường hợp trễ giữa đầu vào - đầu ra. Lý do: Thay vì phải khai báo trễ giống nhau (có giá trị `IODelay`) riêng rẽ cho từng phần tử của ma trận, ta chỉ cần nhập một vector thời gian trễ đầu vào `InputDelay` hoặc đầu ra `OutputDelay`.

Thời gian trễ trong các hệ gián đoạn theo thời gian

Đối với các hệ gián đoạn, việc khai báo thời gian trễ được thực hiện tương tự như các hệ liên tục về thời gian, chỉ cần bổ sung thêm bội số n lần của chu kỳ trích mẫu T_s. Trên miền ảnh z, việc bổ sung ứng với việc nhân thêm thừa số z^{-n} vào hàm truyền đạt gián đoạn, đồng nghĩa với việc gán thêm cho hệ điểm cực lặp lại n lần tại $z = 0$ (tại gốc tọa độ). Nếu xuất hiện nhu cầu biến đổi thời gian trễ thành điểm cực như trên, ta có thể dùng lệnh `sysp = delay2z (sys)`.

Ví dụ sau đây minh họa việc khai báo thời gian trễ đầu vào ứng với 4 chu kỳ trích mẫu.

```
>> h = tf ([1 -0.5],[1 1 -2],0.01,'InputDelay';4)

Transfer function:
           z - 0.5
z^(-4) * -----------
         z^2 + z - 2

Sampling time: 0.01
```

Phép xấp xỉ Pade[1] cho các hệ thời gian liên tục

Bằng phép xấp xỉ Pade ta có thể thay thế gần đúng thành phần trễ $e^{-T_d s}$ của hệ liên tục bởi một hàm hữu tỷ.

```
pade (sys,n)                [num,den] = pade (sys,ni,no,nio)
pade (sys,ni,no,nio)        sysx = pade (sys,ni,no,nio)
```

Trong lệnh `pade`, n chính là bậc của hàm xấp xỉ cần tìm.

[1] Padé-Approximation

Với đối tượng MIMO, có thể khai báo bậc riêng rẽ cho từng đầu vào trong vector *ni*, cho từng đầu ra trong vector *no*, và thời gian trễ giữa các đầu vào/ra trong ma trận *nio*. Các biến xuất *num* và *den* chứa đa thức tử số và mẫu số của hàm hữu tỷ xấp xỉ, thu được dưới dạng mô hình TF. Biến *sysx* sẽ lưu lại dạng của mô hình gốc *sys*. Áp dụng xấp xỉ Pade cho hàm truyền đạt con H(1,2) ở ví dụ trên ta sẽ thu được hàm truyền đạt (hữu tỷ) xấp xỉ sau đây:

```
>> pade(H(1,2))

Transfer function:
-s^3 + 4 s^2 + 11 s - 30
------------------------
        s^2 - 25
```

Khai báo đặc điểm của thời gian trễ	
`InputDelay`	Thời gian trễ trực tiếp ở đầu vào
`OutputDelay`	Thời gian trễ trực tiếp ở đầu ra
`IODelay`	Thời gian trễ giữa đầu vào - đầu ra
Lệnh liên quan đến thời gian trễ	
`totaldelay(sys)`	Xác định mọi thời gian trễ của toàn hệ
`delay2z(sys)`	Thay thời gian trễ bằng điểm cực tại $z = 0$
`pade(sys,n)`	Xấp xỉ Pade với bậc *n*

3.2 Nguyên tắc sử dụng mô hình LTI

3.2.1 Đặc điểm của mô hình LTI

Đối tượng LTI

Hoàn toàn không phụ thuộc vào phương thức mô tả hệ (đối tượng), MATLAB luôn cất mọi dữ liệu về đối tượng LTI (*LTI object*) trong một biến có đặc điểm cấu trúc (*Struct*, mục 1.3) hoặc trường (*Cell Arrays*, mục 1.3). Đặc điểm của đối tượng vậy là sẽ được thể hiện qua *nhiều mảng* khác nhau với tên cho trước (*EigName*), có chứa *các giá trị* được gán (*EigValue*).

Việc truy cập vào từng mảng cụ thể (vào đặc điểm) được thực hiện bằng lệnh "`.`" hoặc các lệnh *get* (hỏi đặc điểm) hoặc *set* (thay đổi đặc điểm). Cần lưu ý rằng, MATLAB không phân biệt chữ viết to hay nhỏ đối với tên của đặc điểm. Có thể tra cứu thêm về đặc điểm của các đối tượng LTI bằng cách gọi `ltiprops`.

Xác lập và thay đổi đặc điểm của mô hình
Có ba khả năng gán giá trị *EigValue* cho đặc điểm *EigName*:

1. Sử dụng các lệnh `tf`, `zpk`, `ss` hoặc `frd`: Khi khai báo đối tượng LTI có thể kết hợp gán đặc điểm bằng cách khai luôn tên *EigName* kèm theo giá trị *EigValue*.

   `sys = tf (`*num,den,Ts,'EigName',EigValue*`)`

2. Sử dụng lệnh `set`: Bằng lệnh này có thể xác lập hay thay đổi đồng thời nhiều đặc điểm của mô hình.

   `set(`*sys,'EigName1',EigValue1,'EigName2',EigValue2*`)`

   Lệnh `set(`*sys*`)` không kèm theo tham số có tác dụng hiển thị tất cả các đặc điểm của đối tượng LTI.

3. Lệnh "`.`" cho phép thay đổi đặc điểm (có dạng cấu trúc) của đối tượng một cách rất đơn giản. Ví dụ, bằng cách gọi:

   sys`.`*EigName* = *EigValue*

   ta đã gán giá trị *EigValue* cho đặc điểm *EigName* của mô hình *sys*.

Hỏi đặc điểm của mô hình
Có thể hỏi đặc điểm của mô hình bằng một trong hai cách:

1. Sử dụng lệnh `get(`*sys,'EigName'*`)`: Lệnh `get` có tác dụng ngược với lệnh `set` và cho biết giá trị hiện tại của các đặc điểm. Nếu chỉ gọi `get(`*sys*`)`, toàn bộ các đặc điểm của *sys* sẽ được hiển thị hoặc gán cho một biến nào đó.

2. Sử dụng lệnh "`.`": Bằng cách gọi *sys*`.`*EigName* ta có thể hiển thị giá trị *EigValue* của đặc điểm *EigName* của hệ *sys*, hay gán cho các biến khác.

Ví dụ
Trước hết ta sử dụng lệnh `tf` để khai báo hàm truyền đạt của hệ *sys*. Đặc điểm `'Notes'` của *sys* nhận giá trị là chuỗi ký tự `'LTI-Object'`. Sau đó ta dùng lệnh `set` để xác lập giá trị `0.05` cho chu kỳ trích mẫu `Ts` (mà trước đó chưa khai báo bằng `tf`), đồng thời gán cho đặc điểm `'InputName'` chuỗi ký tự `'InputVariable'`. Khi ta gọi `sys.OutputName = 'OutputVariable'`, đặc điểm `'OutputName'` sẽ nhận giá trị là chuỗi ký tự `'OutputVariable'`. Ngoài ra, ta gán cho biến `samplingtime` chu kỳ trích mẫu `Ts`, gán cho biến `notiz` giá trị `Notes`. Cuối cùng ta hiển thị toàn bộ đặc điểm của *sys* bằng lệnh `get`.

```
>> sys = tf ([1 -0.5],[1 1 -2],-1,'Notes','LTI-Object')

Transfer function:
  z - 0.5
-----------
z^2 + z - 2

Sampling time: unspecified

>> set(sys,'Ts',0.05,'InputName','InputVariable')
>> sys.OutputName = 'OutputVariable'

Transfer function from input "InputVariable" to output
"OutputVariable":
  z - 0.5
-----------
z^2 + z - 2

Sampling time: 0.05

>> samplingtime = sys.Ts
samplingtime =
    0.0500

>> notiz = get(sys,'Notes')
notiz =
    'LTI-Object'

>> get(sys)
            num: {[0 1 -0.5]}
            den: {[1 1 -2]}
       Variable: 'z'
             Ts: 0.05
        ioDelay: 0
     InputDelay: 0
    OutputDelay: 0
      InputName: {'InputVariable'}
     OutputName: {'OutputVariable'}
     InputGroup: {0x2 cell}
    OutputGroup: {0x2 cell}
          Notes: {'LTI-Object'}
       UserData: []
```

Các đặc điểm chung và riêng của mô hình

Các đối tượng LTI mang một số đặc điểm tổng quát chung (*generic properties*), nhưng đồng thời lại có một số đặc điểm riêng (*model-specific properties*) tùy theo loại của mô hình đối tượng (loại TF, ZPK, SS hay FRD).

Trong hai bảng sau đây, cột thứ nhất chứa tên của các đặc điểm, cột thứ hai chứa đoạn mô tả ngắn về đặc điểm đó và cột thứ ba cho biết dạng của giá trị mà đặc điểm có thể nhận.

Các đặc điểm chung của mô hình LTI

`Ts`	Chu kỳ trích mẫu (tính bằng giây)	Vô hướng (*Scalar*)
`InputDelay`	Thời gian trễ tại đầu vào	Vector
`OutputDelay`	Thời gian trễ tại đầu ra	Vector
`ioDelay`	Thời gian trễ giữa hai đầu vào-ra	Ma trận
`InputName`	Tên của đầu vào	Vector Cell của Strings
`OutputName`	Tên của đầu ra	Vector Cell của Strings
`InputGroup`	Tên của nhóm đầu vào	Trường (*Cell Arrays*)
`OutputGroup`	Tên của nhóm đầu ra	Trường (*Cell Arrays*)
`Notes`	Ghi chú	Text
`Userdata`	Dữ liệu phụ	Tùy ý

Các đặc điểm riêng của mô hình TF

`num`	Tử số	Trường (*Cell Arrays*) các vector hàng thực
`den`	Mẫu số	Trường (*Cell Arrays*) các vector hàng thực
`Variable`	Biến TF	`'s','p','z','q'` hoặc `'z^-1'`

Các đặc điểm riêng của mô hình ZPK

`z`	Điểm không	Trường (*Cell Arrays*) các vector cột thực
`p`	Điểm cực	Trường (*Cell Arrays*) các vector cột thực
`k`	Hệ số khuếch đại	Ma trận thực 2 chiều
`Variable`	Biến ZPK	`'s','p','z','q'` hoặc `'z^-1'`

Các đặc điểm riêng của mô hình SS

`a`	Ma trận trạng thái	Ma trận thực 2 chiều
`b`	Ma trận đầu vào	Ma trận thực 2 chiều
`c`	Ma trận đầu ra	Ma trận thực 2 chiều
`d`	Ma trận liên thông	Ma trận thực 2 chiều
`e`	Ma trận Descriptor	Ma trận thực 2 chiều
`StateName`	Tên các trạng thái	Vector Cell của Strings

Các đặc điểm riêng của mô hình FRD

`Frequency`	Các giá trị tần số	Vector thực
`ResponseData`	Đáp ứng tần số	Ma trận phức nhiều chiều
`Units`	Thứ nguyên (đơn vị) của tần số	`'rad/s'` hoặc `'Hz'`

3.2.2 Truy cập nhanh dữ liệu của mô hình

Vì MATLAB cất mọi dữ liệu của mô hình trong *LTI Object* (xem mục 3.2.1), ta có thể nhanh chóng tìm được các dữ liệu quan trọng nhất của mô hình nhờ các lệnh truy cập sau:

```
[num,den,Ts]     = tfdata (sys)
[z,p,k,Ts]       = zpkdata (sys)
[a,b,c,d,Ts]     = ssdata (sys)
[answer,freq,Ts] = frddata (sysfrd)
```

Trong các lệnh trên, *sys* là mô hình TF, ZPK hay SS, còn *sysfrd* là mô hình FRD. Ví dụ, sử dụng `tfdata` với hàm truyền đạt `h = tf ([2 -3],[1 1])` ở mục 3.1.1 sẽ cho ta kết quả sau:

```
>> [num,den,Ts] = tfdata (h)
num =
    [1x2 double]
den =
    [1x2 double]
Ts =
     0

>> num{1,1}                          % Truy cập Cell Arrays
ans =
     2    -3
```

Bởi vì `num` và `den` của `tf`, cũng như `p` và `z` của `zpk` (ngay cả đối với hệ SISO) đều thuộc dạng trường (*Cell Arrays*), ta có thể sử dụng tham số `'v'` để buộc MATLAB phải cất chúng dưới dạng *Vector* hàng chứ không phải *Cell Arrays* nữa. Khi ấy, việc truy cập *Vector* còn trở nên nhanh hơn nữa.

```
>> [num,den,Ts] = tfdata (h,'v');
>> num                               % Truy cập Vector
num =
     2    -3
```

3.2.3 Trình tự ưu tiên của mô hình LTI

Khi sử dụng toán tử đối với các mô hình LTI (thuộc các loại khác nhau), hay khi đầu vào của lệnh là các loại mô hình khác nhau, khi ấy loại của mô hình kết quả sẽ được quyết định theo trình tự ưu tiên sau (khi viết A > B: có nghĩa là A được ưu tiên hơn B):

FRD > SS > ZPK > TF

Theo trình tự nói trên, trước hết MATLAB sẽ tự động tìm ra loại mô hình có mức ưu tiên cao nhất, tiếp theo đảo tất cả các mô hình loại khác sang loại có mức cao nhất đó, và cuối cùng mới thực hiện phép toán được yêu cầu. Nếu muốn vượt qua nguyên tắc này, để buộc kết quả theo một loại mô hình mong muốn nào đó cụ thể, ta có hai cách đi (mô hình TF: `systf`, mô hình SS: `sysss`):

- **Đảo loại mô hình từ trước**: Trước hết ta đảo tất cả các mô hình sang loại mà ta cần thiết, sau đó mới tính toán.

```
sys = systf + tf(sysss)    % Đảo loại trước, sau mới tính
```

- **Đảo loại mô hình sau**: Trước hết ta thực hiện phép tính, sau đó đảo kết quả sang loại mô hình mong muốn.

```
sys = tf(systf + sysss)    % Tính trước, đảo loại sau
```

Chú ý: Hai cách tiến hành trên không đồng nhất về mặt số. Do trình tự đảo khác nhau có thể dẫn đến khó khăn khi hiển thị số. Mục 3.5 sẽ quay lại đề cập đến vấn đề này.

3.2.4 Tính kế thừa của mô hình LTI

Một điều quan trọng cần được quan tâm là: Các phép tính ảnh hưởng như thế nào tới các đặc điểm của mô hình LTI? Câu hỏi tương tự cũng nẩy sinh khi ta kết hợp nhiều mô hình có loại khác nhau. Mặc dù điều này còn tùy thuộc vào phép tính, vẫn có thể đưa ra được một số nguyên tắc cơ bản sau:

- **Mô hình LTI gián đoạn**: Khi sử dụng một phép tính để kết hợp các mô hình LTI gián đoạn, mọi mô hình bắt buộc phải có chu kỳ trích mẫu giống nhau. Chu kỳ trích mẫu của kết quả sẽ là chu kỳ của từng mô hình.
- `Notes` **và** `UserData`: Không được kế thừa đối với đa số các phép tính.
- **Kết hợp hai hệ**: Nếu hai hệ được kết hợp bởi các toán tử `+`, `*`, `[,]`, `[;]`, `append` hay `feedback`, khi ấy tên của đầu vào, đầu ra (`InputName`, `OutputName`) hay tên của nhóm đầu vào, đầu ra (`InputGroup`, `OutputGroup`) sẽ được kế thừa, nếu chúng giống nhau ở cả hai hệ. Nếu không giống, các đặc điểm đó của hệ kết quả sẽ bị bỏ trống (không nhận giá trị cụ thể).
- **Các đại lượng biến thiên**: Đối với hai loại mô hình TF và ZPK, trình tự ưu tiên như sau:
  - Mô hình liên tục: `p > s`
  - Mô hình gián đoạn: `z^-1 > q > z`

3.2.5 Đảo loại cho mô hình LTI

Để sử dụng đồng thời mô hình LTI có loại khác nhau, ví dụ: Để ghép nối mô hình, hay để sử dụng các phép tính (phép tính chỉ có thể thực hiện với một loại mô hình nhất định) đối với mọi mô hình, ta bắt buộc phải có khả năng đảo mô hình từ loại này sang loại khác.

Đảo tường minh (explicite)

Khi cần đảo mô hình LTI có sẵn sang một loại nhất định, ta sử dụng lệnh vẫn dùng để khai báo loại đó, với tham số là mô hình cần được đảo *sys*.

```
sys     = tf (sys)              % Đảo sang mô hình TF
sys     = zpk (sys)             % Đảo sang mô hình ZPK
sys     = ss (sys)              % Đảo sang mô hình SS
sysfrd  = frd (sys,freq)        % Đảo sang mô hình FRD
```

Việc đảo giữa ba loại TF, ZPK và SS là hoàn toàn không có vấn đề gì. Khi đảo từ một loại khác sang mô hình FRD ta phải khai báo thêm vector tần số `freq`. *Việc đảo từ mô hình FRD sang loại khác là không thể được.*

Ngoài ra, MATLAB còn cung cấp thêm một số công cụ đảo sau đây:

		TF →	ZPK →	SS →
TF:	`[num,den]=`		`zp2tf(z,p,k)`	`ss2tf(A,B,C,D,iu)`
ZPK:	`[z,p,k] =`	`tf2zp(num,den)`		`ss2zp(A,B,C,D,i)`
SS:	`[A,B,C,D]=`	`tf2ss(num,den)`	`zp2ss(z,p,k)`	

Đảo tự động

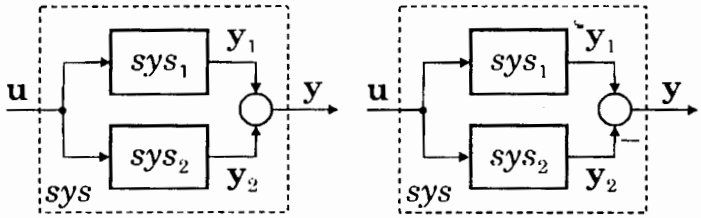
Có một số phép tính (một số lệnh) chỉ hợp với vài loại mô hình nhất định. Nếu ta trao cho lệnh đó những tham số là mô hình không hợp loại, MATLAB sẽ tự động đảo loại trước khi thực hiện lệnh. Ví dụ: Lệnh `step` có tác dụng tính đáp ứng bước nhảy trên cơ sở mô hình trạng thái SS của hệ. Vì vậy, nếu hệ được trao cho lệnh dưới dạng mô hình khác loại, mô hình sẽ tự động được đảo sang mô hình SS. Cũng tại đây ta phải thận trọng với các vấn đề khi biểu diễn bằng số (mục 3.5).

3.2.6 Các phép tính số học

Vì mô hình LTI là mô hình toán, ta có thể áp dụng hầu hết các phép tính (cả phép tính ma trận) đối với chúng.

Phép cộng và phép trừ

Phép cộng hay trừ hai hệ ứng với việc mắc song song hai hệ đó. Nghĩa là, cả hai hệ nhận tín hiệu đầu vào như nhau, ta chỉ thực hiện cộng hay trừ hai tín hiệu đầu ra.



Hình 3.1 *Phép cộng và trừ đối với mô hình LTI*

Phép cộng
$$\mathbf{y} = \mathbf{y}_1 + \mathbf{y}_2 = \mathbf{G}_1\mathbf{u} + \mathbf{G}_2\mathbf{u}$$

Phép trừ
$$\mathbf{y} = \mathbf{y}_1 - \mathbf{y}_2 = \mathbf{G}_1\mathbf{u} - \mathbf{G}_2\mathbf{u} \quad (3.20)$$

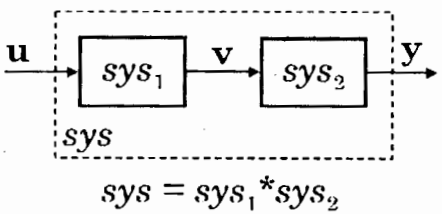
Khi sử dụng mô hình trên không gian trạng thái ta có:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 \end{bmatrix}; \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix}; \mathbf{C} = \begin{bmatrix} \mathbf{C}_1 & (-)\mathbf{C}_2 \end{bmatrix}; \mathbf{D} = \mathbf{D}_1 \pm \mathbf{D}_2 \quad (3.21)$$

Phép nhân

Phép nhân hai hệ LTI ứng với việc mắc nối tiếp hai hệ đó. Khi mắc nối tiếp hai hệ có cấu trúc ma trận cần phải chú ý đến trình tự của hai hệ.

$$\mathbf{y} = \mathbf{G}_1\mathbf{v} = \mathbf{G}_1(\mathbf{G}_2\mathbf{u}) = (\mathbf{G}_1 \times \mathbf{G}_2)\mathbf{u} \quad (3.22)$$



Hình 3.2 *Phép nhân hai mô hình LTI*

Đối với mô hình trên không gian trạng thái ta có:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{B}_1\mathbf{C}_2 \\ \mathbf{0} & \mathbf{A}_2 \end{bmatrix}; \mathbf{B} = \begin{bmatrix} \mathbf{B}_1\mathbf{D}_2 \\ \mathbf{B}_2 \end{bmatrix}; \mathbf{C} = \begin{bmatrix} \mathbf{C}_1 & \mathbf{D}_1\mathbf{C}_2 \end{bmatrix}; \mathbf{D} = \mathbf{D}_1\mathbf{D}_2 \quad (3.23)$$

Phép đảo, phép chia ma trận từ phía trái và phía phải

Các phép tính này chỉ được định nghĩa cho ma trận toàn phương (ma trận vuông), tức là cho các đối tượng có số đầu vào và đầu ra như nhau, điều đó đồng nghĩa với việc **D** cũng là toàn phương.

Phép đảo: `sys = inv(sys)` $\triangleq \mathbf{u} = \mathbf{G}^{-1}\mathbf{y}$ (3.24)

Phép chia từ phía trái: `sys1 \ sys2` $\triangleq \mathbf{G}_1^{-1}\mathbf{G}_2$ (3.25)

Phép chia từ phía phải: `sys1 / sys2` $\triangleq \mathbf{G}_1\mathbf{G}_2^{-1}$ (3.26)

Đối với mô hình trên không gian trạng thái ta sẽ phải thực hiện đảo ma trận **D** và sau đó tính như sau:

$$\mathbf{A} = \mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C};\ \mathbf{B} = \mathbf{B}\mathbf{D}^{-1};\ \mathbf{C} = -\mathbf{D}^{-1}\mathbf{C};\ \mathbf{D} = \mathbf{D}^{-1} \quad (3.27)$$

Các phép tính số học

+	Cộng	*	Nhân	inv	Đảo ma trận
-	Trừ	^	Lũy thừa	/	Chia từ phía phải
		.'	Đảo cột thành hàng	\	Chia từ phía trái

3.2.7 Lựa chọn, thay đổi và ghép nối mô hình LTI

Lựa chọn và thay đổi

Vì các mô hình LTI có cấu trúc theo kiểu ma trận, ta có thể xử lý, biến đổi chúng khá dễ dàng: Có thể tách hệ thống con ra khỏi hệ thống mẹ hay thêm bớt đầu vào, đầu ra một cách đơn giản.

Ví dụ sau đây minh họa các khả năng đó: Trước hết ta khai báo một đối tượng MIMO với hai đầu vào, hai đầu ra theo mô hình TF có tên là *systf*. Sau đó ta tách hệ thống con *partsys* (hàm truyền đạt từ đầu vào 2 tới đầu ra 1) ra khỏi hệ thống mẹ *systf*. Tiếp theo ta gán cho *partsys* một hàm truyền đạt có đặc tính I (đặc tính tích phân) và trả lại cho hệ thống mẹ *systf*.

```
>> s = tf('s');
>> h11 = (2*s+1)/(s+0.1);
>> h21 = 0;
>> h12 = 1/(s+0.5);
>> h22 = 1/(s-1);
>> systf = tf ([h11 h12;h21 h22])   % Khai báo hệ thống mẹ

Transfer function from input 1 to output...
      2 s + 1
 #1:  -------
      s + 0.1
 #2:  0

Transfer function from input 2 to output...
         1
 #1:  -------                      % h21 trước khi biến đổi
      s + 0.5
```

```
         1
 #2:  -----
      s - 1

>> partsys = systf (1,2)      % Tách hệ thống con partsys
Transfer function:
   1
-------
s + 0.5

>> partsys = tf (1,[1 0])     % Gán đặc tính I cho partsys
Transfer function:
1
-
s

>> systf (1,2) = partsys      % Trả partsys lại cho systf

Transfer function from input 1 to output...
      2 s + 1
 #1:  -------
      s + 0.1
 #2:  0

Transfer function from input 2 to output...
      1
 #1:  -                       % Kết quả h21 sau khi biến đổi
      s
        1
 #2:  -----
      s - 1
```

Bây giờ ta xóa cột thứ nhất (ứng với: xóa đầu vào 1), thay vào đó một cột mới (ứng với: bổ sung đầu vào mới). Khi thực hiện ta phải chú ý hai vấn đề: Nếu ta gọi `sys(i,j) = new`, mô hình sẽ giữ nguyên loại gốc, nếu ta gọi `sys = [sys, new]` mô hình sẽ bị đảo sang loại của *new*.

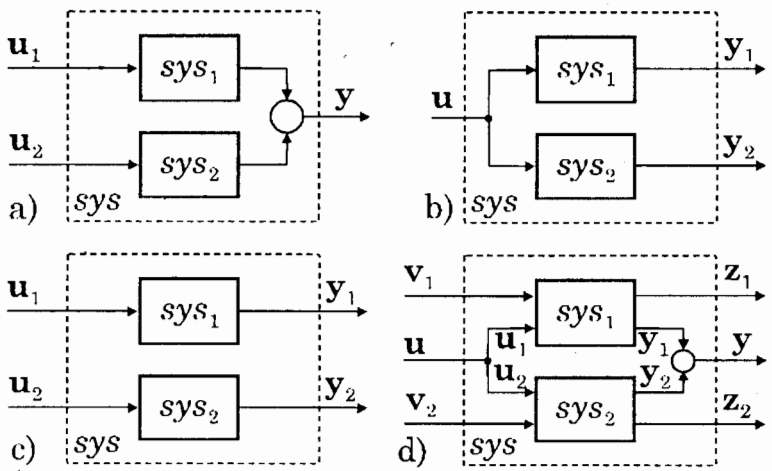
```
>> systf (:,1) = []                    % Xóa đầu vào 1
Transfer function from input to output...
      1
 #1:  -
      s
        1
 #2:  -----
      s - 1

>> systf (:,2) = [tf(1,[1 -1]) ; tf(1,1)]  % Thêm đầu vào
```

Khi tách hệ thống con khỏi hệ thống mẹ có dạng mô hình SS ta sẽ phải gọi `partsys = sys(i,j)` để tạo hệ con với các ma trận `a`, `b(:,j)`, `c(i,:)` và `d(i,j)`.

Ghép nối mô hình LTI

Để ghép nối, tạo nên hệ thống từ nhiều hệ thống con ta có thể sử dụng một số khả năng sau đây.



Hình 3.3 *Các khả năng ghép mô hình LTI:* ***a)*** *Ghép theo hàng,* ***b)*** *ghép theo cột,* ***c)*** *ghép theo đường chéo, và* ***d)*** *ghép song song*

- **Ghép theo hàng** (hình 3.3a): Ghép theo hàng (khi quan sát trên phương trình của mô hình) có nghĩa là ghép đầu ra của các hệ thống con có đầu vào khác nhau. *Ma trận đầu ra* $\mathbf{H}$ *thu được nhờ ghép theo hàng hai ma trận đầu vào* $\mathbf{H}_1$, $\mathbf{H}_2$ của hai hệ thống con.

$$sys = [sys1, sys2]$$
$$y = [\mathbf{H}_1, \mathbf{H}_2]\begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{bmatrix} = \mathbf{H}_1\mathbf{u}_1 + \mathbf{H}_2\mathbf{u}_2 \tag{3.28}$$

- **Ghép theo cột** (hình 3.3b): Ghép theo cột (khi quan sát trên phương trình của mô hình) có nghĩa là ghép các hệ thống con có chung một đầu vào. Vector biến ra bao gồm hai vector biến ra của các hệ thống con. *Ma trận đầu ra* $\mathbf{H}$ *thu được nhờ ghép theo cột hai ma trận đầu vào* $\mathbf{H}_1$, $\mathbf{H}_2$ của hai hệ thống con.

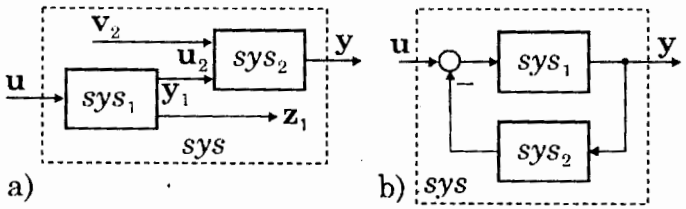
$$sys = [sys1, sys2]; \quad \mathbf{y} = \begin{bmatrix}\mathbf{y}_1\\ \mathbf{y}_2\end{bmatrix} = \begin{bmatrix}\mathbf{H}_1\\ \mathbf{H}_2\end{bmatrix}\mathbf{u} \tag{3.29}$$

- **Ghép theo đường chéo** (hình 3.3c): Khi ghép theo đường chéo ta sẽ thu được một hệ thống mới bảo đảm cách ly hoàn toàn giữa hai hệ thống con ban đầu.

$$sys = \text{append}(sys1, sys2); \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{H}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_2 \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix} \tag{3.30}$$

- **Ghép song song** (hình 3.3d): Ghép song song là dạng tổng quát của phép cộng các hệ. Khi ghép ta có thể khai báo ghép cả đầu vào và đầu ra với nhau: Hai vector chỉ số (*Index Vector*) *inp*1 và *inp*2 cho ta biết, những đầu vào nào của $\mathbf{u}_1$ (vector biến vào của hệ *sys*1) và của $\mathbf{u}_2$ (vector biến vào của hệ *sys*2) cần được nối với nhau. Hai vector chỉ số *out*1 và *out*2 cho biết, những đầu ra nào của $\mathbf{y}_1$ (vector biến ra của hệ *sys*1) và của $\mathbf{y}_2$ (vector biến ra của hệ *sys*2) cần được cộng với nhau.

$$sys = \text{parallel}(sys1, sys2, inp1, inp2, out1, out2) \tag{3.31}$$



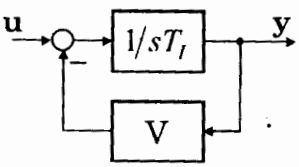
Hình 3.4 *Các khả năng ghép mô hình LTI: a) Ghép tuần tự, b) ghép có phản hồi*

- **Ghép tuần tự** (hình 3.4a): Ghép tuần tự là dạng tổng quát của phép nhân hệ. Để ghép ta cần phải khai báo: Những đầu ra *outputs*1 nào của *sys*1 cần được nối với những đầu vào *inputs*2 nào của *sys*2. Hệ mới xuất hiện sau khi ghép sẽ có *Dim*(**u**) biến vào và *Dim*(y) biến ra.

$$sys = \text{series}(sys1, sys2, outputs1, inputs2) \tag{3.32}$$

- **Ghép có phản hồi** (hình 3.4b): Đặc biệt lợi hại là khả năng ghép có phản hồi nhờ lệnh `feedback`. Nhờ lệnh đó ta có thể tạo ra một vòng điều chỉnh một cách dễ dàng.

$$sys = \text{feedback}(sys1, sys2); \quad (\mathbf{E} + sys1 \cdot sys2)^{-1} sys1 \tag{3.33}$$



Hình 3.5 *Vòng điều chỉnh: xuất hiện bằng cách phản hồi tín hiệu ra.*

Ví dụ đơn giản ở hình 3.5 sẽ minh họa tác dụng của lệnh `feedback`, Theo đó, đầu ra của một khâu tích phân được phản hồi qua một khâu tỷ lệ V.

Hằng số thời gian tích phân: $T_I = 3s$
Hệ số khuếch đại: $V = 2$
Hàm truyền đạt: $G(s) = \dfrac{1/sT_I}{1+V(1/sT_I)}$

```
>> integrator = tf(1,[3 0])     % Khai báo khâu tích phân I

Transfer function:
 1
---
3 s

>> proportional = tf (2,1)      % Khai báo khâu tỷ lệ V

Transfer function:
2

>> feedback (integrator,proportional)   % Tạo vòng phản hồi

Transfer function:
   1
-------
3 s + 2
```

Tự tạo mô hình LTI

Control System Toolbox tạo điều kiện cho người sử dụng tự tạo một số dạng mô hình LTI nhất định. Ví dụ: Việc tạo nên các mô hình ngẫu nhiên liên tục hay gián đoạn, việc tạo nên các hệ bậc 2.

- **Tạo các hệ ngẫu nhiên**: Các lệnh `rss` hay `drss` tạo nên các mô hình LTI ổn định (liên tục hay gián đoạn) trên không gian trạng thái. Đó là mô hình có ma trận chứa các phần tử mang giá trị ngẫu nhiên.

```
sys = rss (n[,p,m])
sys = drss (n[,p,m])
```

Bằng các lệnh `rmodell` hay `drmodell` ta có thể tạo ra mô hình TF bậc n ngẫu nhiên (liên tục hay gián đoạn) và mô hình SS bậc n với m đầu vào và p đầu ra.

```
[num,den] = rmodell (n)
[num,den] = drmodell (n)
[A,B,C,D] = rmodell (n[,p,m])
[A,B,C,D] = drmodell (n[,p,m])
```

- **Tạo các hệ bậc 2**: Một hệ bậc 2 được mô tả bởi hàm truyền đạt sau:

$$G(s) = \frac{1}{s^2 + 2\omega_0 D s + \omega_0^2} \tag{3.34}$$

Trong đó ω_0 là tần số riêng và D là hệ số tắt dần của hệ. Có thể tạo mô hình TF hay SS cho hệ bằng các lệnh sau:

```
[num,den] = ord2 (ω0,D)
[A,B,C,D] = ord2 (ω0,D)
```

Ghép mô hình LTI	
`[,]`	Ghép theo hàng
`[;]`	Ghép theo cột
`append`	Ghép theo đường chéo
`parallel`	Ghép song song
`series`	Ghép tuần tự
`feedback`	Ghép có phản hồi
Tạo mới mô hình LTI	
`drmodel, drss`	Tạo mô hình ngẫu nhiên, ổn định và gián đoạn về thời gian
`rmodel, rss`	Tạo mô hình ngẫu nhiên, ổn định và liên tục về thời gian
`ord2(ω0,D)`	Tạo mô hình bậc 2

3.2.8 Chuyển đổi giữa hai hệ liên tục và gián đoạn về thời gian

Trong nhiều trường hợp ta có nhu cầu biến đổi từ mô hình liên tục của đối tượng sang mô hình gián đoạn hay ngược lại. Ví dụ: Khi xây dựng mô hình để tổng hợp các hệ trích mẫu (điều khiển có sử dụng vi xử lý), hay đơn giản chỉ nhằm so sánh các kết quả (liên tục) thu được từ mô phỏng với các giá trị (gián đoạn) đo được.

Để chuyển đổi các mô hình TF, ZPK hay SS ta có thể sử dụng hai lệnh sau đây của MATLAB:

```
sysd = c2d (sysc,Ts [,method])
sysc = d2c (sysd [,method])
```

Lệnh `c2d` chuyển đổi một hệ liên tục *sysc* sang hệ gián đoạn *sysd*, còn lệnh `d2c` có tác dụng ngược lại. Khi gián đoạn hóa mô hình liên tục, ta bắt buộc phải khai báo chu kỳ trích mẫu *Ts*. Tham số tùy chọn *method* quyết định phương pháp gián đoạn hóa được sử dụng khi chuyển đổi: Sử dụng khâu giữ chậm bậc 0 (ZOH: *Zero Order Hold*), giữ chậm bậc 1 (FOH: *First Order Hold*), xấp xỉ Tustin (xấp xỉ kiểu hình thang). Phương pháp mặc định được chọn là sử dụng khâu giữ chậm bậc 0.

Khâu giữ chậm bậc 0

Phương pháp tính khi sử dụng khâu ZOH là rất đơn giản: Giá trị trích mẫu được giữ nguyên đến thời điểm trích mẫu mới (xấp xỉ kiểu hình chữ nhật).

$$u(t) = u[k] \qquad kT_s \le t \le (k+1)T_s \tag{3.35}$$

Nếu chuyển một mô hình gián đoạn dùng ZOH sang mô hình liên tục, ta cần chú ý các hạn chế sau đây:

- Không được phép tồn tại điểm cực tại gốc tọa độ $z = 0$.
- Các điểm cực thực âm trên miền ảnh z sẽ chuyển thành các cặp điểm cực phức liên hợp trên miền ảnh Laplace s, nghĩa là: bậc của mô hình bị nâng lên.

Khâu giữ chậm bậc 1

Khâu FOH nội suy tuyến tính giữa hai giá trị trích mẫu và vì vậy (về nguyên tắc) đối với mô hình liên tục sẽ chính xác hơn ZOH, tuy nhiên chỉ thích hợp với mô hình gián đoạn.

$$u(t) = u[k] + \frac{t - kT_s}{T_s}\left(u[k+1] - u[k]\right) \qquad kT_s \le t \le (k+1)T_s \quad (3.36)$$

Xấp xỉ Tustin

Để tính $u[k]$, xấp xỉ Tustin sử dụng giá trị trung bình của đạo hàm tại biên trái và biên phải của khoảng thời gian được xét. Phép xấp xỉ không được định nghĩa cho các đối tượng có điểm cực tại $z = -1$, và rất khó sử dụng khi hệ có điểm cực ở lân cận $z = -1$. Giữa ảnh Laplace s và ảnh z có quan hệ sau:

$$z = e^{sT_s} = \frac{1 + s\frac{T_s}{2}}{1 - s\frac{T_s}{2}} \qquad s = \frac{2}{\Delta t}\frac{z-1}{z+1} \quad (3.37)$$

Trong ví dụ sau đây, trước hết ta khai báo hàm truyền đạt liên tục của một khâu PT_2 (khâu tỷ lệ có quán tính bậc hai), sau đó gián đoạn hóa khâu PT_2 đó bằng các lệnh MATLAB khác nhau `zoh`, `foh` và `tustin`. Cuối cùng ta in các kết quả thu được để so sánh.

Khâu PT_2 có công thức sau: $F(s) = \dfrac{1}{s^2 + 0{,}7s + 1}$

```
>> sysc = tf ([1],[1 .7 1])          % Khai báo khâu PT2

Transfer function:
       1
---------------
s^2 + 0.7 s + 1

>> sysd = c2d (sysc,1.5)             % Gián đoạn bằng ZOH

Transfer function:
   0.6844 z + 0.4704
-----------------------
z^2 - 0.1951 z + 0.3499
```

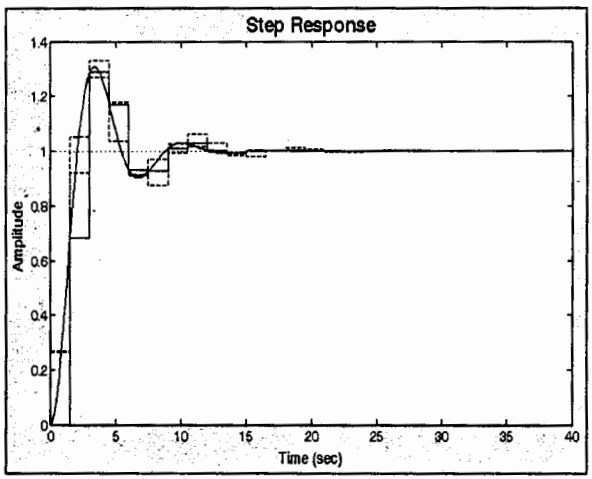
```
Sampling time: 1.5
>> sysd1 = c2d (sysc,1.5,'foh')      % Gián đoạn bằng ZOH

Transfer function:
0.2654 z^2 + 0.7353 z + 0.1542
------------------------------
   z^2 - 0.1951 z + 0.3499

Sampling time: 1.5
>> sysd2 = c2d (sysc,1.5,'tustin')  % Xấp xỉ Tustin

Transfer function:
0.2695 z^2 + 0.5389 z + 0.2695
------------------------------
    z^2 - 0.4192 z + 0.497

Sampling time: 1.5                  % Vẽ đáp ứng bước nhẩy
>> step (sysc,'r-',sysd,'c-',sysd1,'g--',sysd2,'y--')
```



Hình 3.6 *Đáp ứng bước nhẩy của khâu PT_2*

Thay đổi chu kỳ trích mẫu

Nếu cần thay đổi chu kỳ trích mẫu T_s cho một đối tượng, đã được khai báo dưới dạng mô hình TF, ZPK hay SS gián đoạn, ta sử dụng chuỗi lệnh `c2d(d2c(sys),Ts)`.

Control System Toolbox sử dụng lệnh `d2c` là lệnh đợi ở đầu vào một mô hình ZOH. Chu kỳ trích mẫu mới không bắt buộc phải là bội số nguyên lần chu kỳ cũ, trừ khi: *Các điểm cực trên miền ảnh z là điểm cực phức*, khi ấy *chu kỳ trích mẫu mới phải là bội số nguyên chẵn của chu kỳ trích mẫu cũ*. Ta hãy quan sát ví dụ khâu quán tính bậc nhất PT_1 sau đây:

```
>> sysc = tf(1,[1 1])                % Khai báo khâu PT1
```

```
 Transfer function:
  1
-----
s + 1

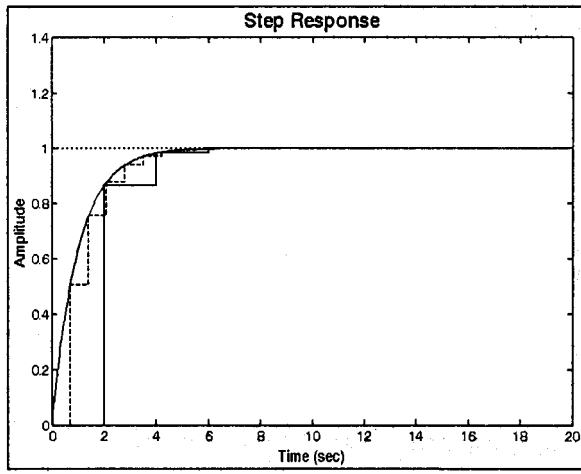
>> sysd = c2d (sysc,2)              % Gián đoạn bằng ZOH

Transfer function:
  0.8647
----------
z - 0.1353

Sampling time: 2
>> sysdd =d2d (sysd,0.7)            % Đổi chu kỳ trích mẫu

Transfer function:
  0.5034
----------
z - 0.4966

Sampling time: 0.7
>> step (sysc,'r-',sysd,'c-',sysdd,'g--')     % Vẽ đáp ứng
```



Hình 3.7 *Đáp ứng bước nhảy sau khi thay đổi chu kỳ trích mẫu khâu PT_1*

Chuyển đổi giữa hai hệ liên tục và gián đoạn

c2d (*sysc*, *Ts*, *method*)	Chuyển hệ liên tục thành hệ gián đoạn
d2c (*sysd*, *method*)	Chuyển hệ gián đoạn thành hệ liên tục
d2d (*sys*, *Ts*)	Thay đổi chu kỳ trích mẫu
method	Phương pháp gián đoạn hóa: 'zoh', 'foh', 'tustin', 'prewarp', 'matched'

3.3 Khảo sát mô hình LTI

Sau khi ta đã mô hình hóa một hệ thống điều khiển, bước tiếp theo sẽ là khảo sát đặc tính của hệ. Để thu được các kết luận về đặc tính của hệ, có thể sử dụng rất nhiều công cụ của *Control System Toolbox*, phục vụ khảo sát từ các đặc điểm tổng quát, tới đặc tính động học của mô hình trên miền thời gian hay trên miền tần số, kể cả việc kiểm tra tính điều khiển được.

3.3.1 Các đặc điểm tổng quát

Đặc điểm tổng quát là các tính chất rất cần khi viết các *scripts* hoặc hàm (*functions*) cần có đặc điểm tổng quát. Ví dụ: Khi viết một *script* xử lý số liệu đo, *script* sẽ phải có khả năng tiếp nhận dữ liệu từ những mô hình thuộc các loại khác nhau, để rồi sau đó gọi các hàm phân tích phù hợp và chọn đúng phương thức biểu diễn dữ liệu loại đó.

Một trong số các lệnh thuộc loại kể trên là `class(object)`, cho ta biết *object* thuộc loại nào, `cell` hay `double` hay `char`. Có tác dụng ngược lại là lệnh `isa(object,'classname')`, lệnh đó kiểm tra xem liệu *object* có thuộc loại *classname* hay không.

```
>> sys = ss([ 1 2 ; 3 4 ],[1 1 ; 0 1],[ 3 1 ],0);
>> class (sys)
ans =
ss
>> isa (sys,'cell')
ans =
     0
```

Bằng hai lệnh `isct(sys)` và `isdt(sys)` có thể kiểm tra xem *sys* là mô hình liên tục hay mô hình gián đoạn, và lệnh `issiso(sys)` sẽ cho đáp án là mức logic 1 nếu *sys* là một hệ SISO.

```
>> isct (sys)
ans =
     1
>> issiso (sys)
ans =
     0
```

Dùng lệnh `isempty(sys)` có thể xác định các đầu vào, đầu ra còn thiếu.

```
>> sys = tf ([1 3],[2 1 5]);
>> isempty (sys)
ans =
     0
```

```
>> hasdelay (sys)
ans =
     0
>> isproper (sys)
ans =
     1
```

Một lệnh quan trọng trong việc khảo sát đặc điểm tổng quát của mô hình là lệnh `size`. Phụ thuộc tham số tùy chọn được sử dụng, lệnh sẽ cho biết số đầu vào, đầu ra, kích cỡ của trường LTI, bậc của mô hình TF, ZPK và SS, cũng như số lượng tần số của mô hình FRD.

`size(`*sys*`)`:	Cho biết số đầu vào, đầu ra đối với mô hình LTI; đối với trường LTI ta sẽ biết thêm kích cỡ của mô hình
d = `size(`*sys*`)`:	Vector hàng *d* = [*Ny Nu*] có phần tử là số đầu vào, đầu ra đối với mô hình LTI, hoặc *d* = [*Ny Nu S*1 *S*2 ... *Sp*] đối với trường LTI dạng *S*1 × *S*2 × ... × *Sp* có *Nu* đầu vào và *Ny* đầu ra
Ny = `size(`*sys*`,1)`:	Số lượng đầu ra
Nu = `size(`*sys*`,2)`:	Số lượng đầu vào
Ns = `size(`*sys*`,'order')`:	Bậc và số trạng thái của mô hình SS
Nf = `size(`*sys*`,'frequency')`:	Số lượng tần số của mô hình FRD

```
>> sys = tf ([1 3],[2 1 5]);
>> size (sys)
Transfer function with 1 output and 1 input.
>> d = size (sys)
d =
     1     1
>> Ns = size (sys,'order')
Ns =
     2
>> sys = [ tf([1 3],[2 1 5]) , zpk([1 -1],[2 2],2) ]

Zero/pole/gain from input 1 to output:
    0.5 (s+3)
------------------
(s^2 + 0.5s + 2.5)

Zero/pole/gain from input 2 to output:
2 (s-1) (s+1)
-------------
   (s-2)^2

>> d = size (sys)
d =
     1     2
```

Khảo sát đặc điểm tổng quát	
`class (sys)`	Cho biết loại của mô hình *sys*
`isa (sys,'classename')`	Kiểm tra xem *classname* có phải là loại của *sys*
`hasdelay (sys)`	Kiểm tra xem *sys* có trễ hay không
`isct (sys)`	Kiểm tra xem *sys* có phải là mô hình liên tục
`isdt (sys)`	Kiểm tra xem *sys* có phải là mô hình gián đoạn
`isempty (sys)`	Kiểm tra xem *sys* có các đầu vào/ra hay không
`isproper (sys)`	Kiểm tra xem liệu Rang(Tử số) > Rang(Mẫu số)
`issiso (sys)`	Kiểm tra xem liệu *sys* có phải là hệ SISO
`size (sys)`	Cho biết kích cỡ của *sys*

3.3.2 Khảo sát động học của mô hình

Không nhất thiết phải có đồ thị thời gian hay đáp ứng đầu ra khi kích thích ở đầu vào của hệ các tín hiệu thử, ta vẫn có thể trên cơ sở các thông số khác nhau để rút ra nhận xét về đặc điểm động học của hệ thống, ví dụ: Hệ số khuếch đại tĩnh và đặc điểm tắt dần, hay các tần số cộng hưởng và tính ổn định của hệ.

Hệ số khuếch đại tĩnh

Ta có thể tìm được hệ số khuếch đại tĩnh (khuếch đại một chiều) của hệ *sys* khi $s = 0$ (mô hình liên tục) hay $z = 1$ (mô hình gián đoạn) bằng lệnh `dcgain(sys)`. Các hệ với đặc tính tích phân áp đảo sẽ có hệ số khuếch đại tĩnh bằng ∞.

```
>> sys = [ tf(zpk([-1],[1 0],2)),tf([2 1 -6],[2 1 1 -4]) ]

Transfer function from input 1 to output:
2 s + 2
-------
s^2 - s

Transfer function from input 2 to output:
   2 s^2 + s - 6
------------------
2 s^3 + s^2 + s - 4

>> dcgain(sys)
ans =
   Inf   1.50000000000000
```

Tần số riêng và hệ số tắt dần

Tần số riêng ω_n và hệ số tắt dần D được xác định bởi cặp điểm cực phức liên hợp $\alpha \pm j\beta$ với:

$$\omega_n = \sqrt{\alpha^2 + \beta^2} \qquad D = -\frac{\alpha}{\sqrt{\alpha^2 + \beta^2}} \tag{3.38}$$

Khi các giá trị riêng (*Eigen Value*) đều là thực, hệ số tắt dần còn được nhân với `sign(EigenValue)`. Đối với mô hình gián đoạn, có thể thông qua $z = e^{sT_s}$ để tính các điểm cực s trên miền ảnh Laplace, ứng với các điểm cực trên miền ảnh z.

```
damp (sys)
[ω,d] = damp (sys)
[ω,d,p] = damp (sys)
```

Trong đó, ω là vector tần số riêng, d là vector hệ số tắt dần và p là vector điểm cực.

```
>> sys = tf(1,[2 -3 1 2])

Transfer function:
          1
---------------------
2 s^3 - 3 s^2 + s + 2

>> damp(sys)

      Eigenvalue              Damping      Freq. (rad/s)

 -5.83e-001                  1.00e+000      5.83e-001
  1.04e+000 + 7.94e-001i    -7.95e-001      1.31e+000
  1.04e+000 - 7.94e-001i    -7.95e-001      1.31e+000
```

Điểm không và điểm cực

Có thể tìm điểm cực của hệ bằng lệnh `pole(sys)`. Khi hệ có dạng mô hình SS, điểm cực chính là các giá trị riêng của ma trận hệ thống **A**. Đối với mô hình dạng TF hay ZPK, điểm cực là nghiệm của đa thức mẫu số, có thể tính được nhờ lệnh `roots`. Gọi p là vector các điểm cực của đối tượng *sys* và r là vector nghiệm của đa thức c, ta có:

```
p = pole (sys)
r = roots (c)
```

Lệnh `zero(sys)` cho ta biết các giá trị điểm không. Nếu ta gán kết quả của `zero(sys)` cho vector `[z,k]`, z sẽ chứa vector các điểm không, còn k là hệ số khuếch đại của hệ.

```
z = zero(sys)
[z,k] = zero(sys)
```

Trong ví dụ sau đây ta sẽ trước hết tính các điểm cực của mô hình ZPK, sau đó tìm các đa thức tử số và đa thức mẫu số của mô hình TF tương ứng, và cuối cùng tìm nghiệm của đa thức mẫu số (đương nhiên phải trùng với giá trị các

điểm cực). Kết thúc, ta tính vector điểm không và hệ số khuếch đại của mô hình ZPK.

```
>> sys = zpk([-0.2 0.6],[-0.5 -0.7-0.8*i -0.7+0.8*i 0.3],2)
                                  % Khai báo mô hình ZPK
Zero/pole/gain:
         2 (s+0.2) (s-0.6)
-----------------------------------
(s+0.5) (s-0.3) (s^2 + 1.4s + 1.13)

>> pole (sys)                     % Điểm cực của mô hình ZPK
ans =
  -0.5000
  -0.7000 - 0.8000i
  -0.7000 + 0.8000i
   0.3000

>> [num,den] = tfdata(tf(sys),'v');
                % Chuyển sys sang TF và gán đa thức num, den
>> num
num =
         0          0    2.0000   -0.8000   -0.2400
>> den
den =
    1.0000    1.6000    1.2600    0.0160   -0.1695

>> roots (den)
ans =
  -0.7000 + 0.8000i
  -0.7000 - 0.8000i
  -0.5000
   0.3000

>> [z,k] = zero (sys)
z =
   -0.2000
    0.6000
k =
     2
```

Liên quan tới các vector điểm không, điểm cực, MATLAB còn cung cấp hai công cụ hữu ích, phục vụ phân loại. Đó là các hàm `esort` và `dsort`.

`s = esort (p)`	`s = dsort (p)`
`[s,ndx] = esort (p)`	`[s,ndx] = dsort (p)`

Lệnh `esort` sắp xếp các điểm cực trong vector *p* theo thứ tự: Trước hết là các điểm không ổn định và sau đó là các điểm ổn định, theo thứ tự phần thực giảm dần. *ndx* mô tả chỉ số của điểm cực trong vector *p*.

```
>> pc = pole (sys);
>> [sc,ndxc] = esort (pc);
>> [ pc , sc , ndxc ]
ans =
  -0.5000              0.3000              4.0000
  -0.7000 - 0.8000i   -0.5000              1.0000
  -0.7000 + 0.8000i   -0.7000 + 0.8000i    3.0000
   0.3000             -0.7000 - 0.8000i    2.0000
```

Lệnh `dsort` sắp xếp các điểm cực của mô hình gián đoạn trong vector p theo thứ tự: Trước hết là các điểm không ổn định và sau đó là các điểm ổn định, theo thứ tự phần thực giảm dần.

```
>> pd = pole (c2d(sys,1));      % Chuyển sys từ liên tục
                                % sang gián đoạn với Ts = 1
>> [sd,ndxd] = dsort (pd);
>> [ pd , sd , ndxd ]
ans =
   0.3460 + 0.3562i   1.3499              4.0000
   0.3460 - 0.3562i   0.6065              3.0000
   0.6065             0.3460 + 0.3562i    1.0000
   1.3499             0.3460 - 0.3562i    2.0000
```

Biểu đồ điểm không - điểm cực

Chức năng của hai lệnh `pole` và `zeros` được thống nhất trong lệnh `pzmap`. Với `pzmap` ta có thể biểu diễn phân bố điểm không - điểm cực bằng đồ thị hoặc cất chúng dưới dạng hai vector điểm không z và điểm cực p.

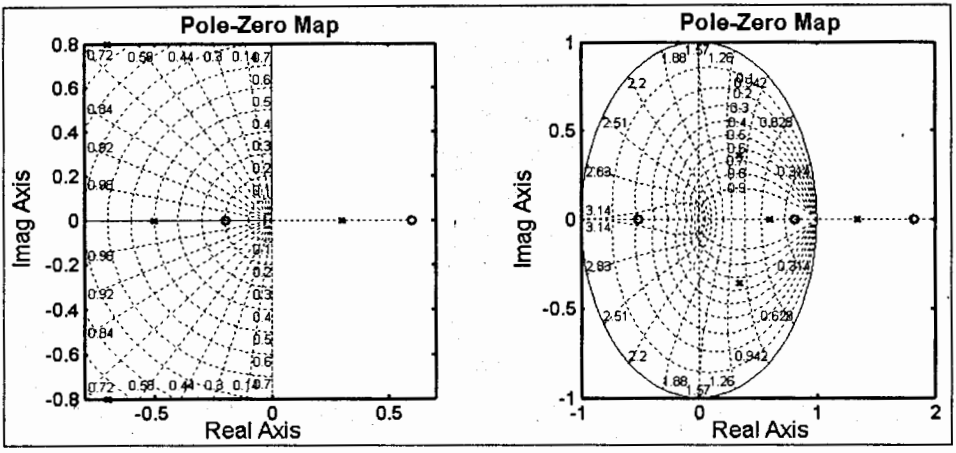
```
pzmap (sys)
[p,z] = pzmap (sys)
```

Để chia trục của đồ thị có thể sử dụng hai lệnh `sgrid` và `zgrid`.

```
sgrid                    zgrid
sgrid (z,wn)             zgrid (z,wn)
```

Đối với mô hình liên tục, lệnh `sgrid` sẽ tạo trên miền ảnh s (ảnh Laplace) mạng lưới ứng với các giá trị hằng của hệ số tắt dần (trong dải 0 – 1 với bước là 0,1) và của tần số riêng (trong dải 0 – 10 rad/s với bước là 1 rad/s). Đối với mô hình gián đoạn, lệnh `zgrid` sẽ tạo trên miền ảnh z mạng lưới ứng với các giá trị hằng của hệ số tắt dần (trong dải 0 – 1 với bước là 0,1) và của tần số riêng (trong dải 0 – π với bước là π/10).

Hình 3.8 minh họa biểu đồ điểm không - điểm cực (thu được nhờ `pzmap`) của mô hình ZPK trong hai trường hợp: Trên miền ảnh Laplace s (trái: mô hình liên tục) và trên miền ảnh z (phải: mô hình gián đoạn). Các điểm không được ký hiệu bởi o và các điểm cực bởi ×.



Hình 3.8 *Vẽ biểu đồ điểm không - điểm cực bằng* `pzmap`*: miền ảnh Laplace s (trái) và miền ảnh z (phải)*

```
>> sys = zpk([-0.2 0.6],[-0.5 -0.7-0.8*i -0.7+0.8*i 0.3],2)

Zero/pole/gain:
         2 (s+0.2) (s-0.6)
-----------------------------------
(s+0.5) (s-0.3) (s^2 + 1.4s + 1.13)

>> sysd = c2d(sys,1)        % Chuyển mô hình ZPK liên tục
                            % sang ZPK gián đoạn với Ts = 1
Zero/pole/gain:
  0.47509 (z-1.827) (z-0.8187) (z+0.5171)
-----------------------------------------------
(z-0.6065) (z-1.35) (z^2 - 0.6919z + 0.2466)

Sampling time: 1
>> subplot(121), pzmap (sys) , sgrid        % Hình 3.8 trái
>> subplot(122), pzmap (sysd) , zgrid       % Hình 3.8 phải
```

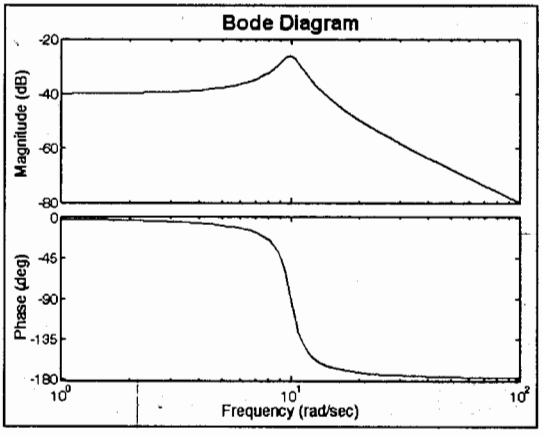
Chuẩn của mô hình LTI (H_2 và H_∞)

Bằng chuẩn của mô hình ta có thể khảo sát ổn định của hệ thống một cách khá dễ dàng. *Control System Toolbox* hỗ trợ hai loại chuẩn: Chuẩn H_2 và H_∞. Nếu như chuẩn H_2 không phù hợp với mô hình FRD, thì chuẩn H_∞ lại có thể xử lý không loại trừ mô hình nào. Cú pháp các lệnh dùng chuẩn như sau:

```
norm (sys)
norm (sys,2)
norm (sys,inf[,tol])
[ninf,fpeak] = norm (sys,inf)
```

Chuẩn mặc định của MATLAB là chuẩn H_2, có hiệu lực khi gọi `norm (sys)` hoặc `norm(sys,2)`. Chuẩn H_∞ sẽ có hiệu lực khi ta gọi `norm(sys,inf[,tol])` với tham số mặc định *tol* dùng để đặt độ chính xác khi tính. Lệnh `[ninf,fpeak] = norm(sys,inf)` cho ta biết hệ số khuếch đại tối đa *ninf* và tần số *fpeak* tại điểm đó. Ngoài ra, còn có thể dùng chuẩn H_∞ để tính dữ trữ pha (mục 3.3.4) của hệ.

Trong ví dụ sau đây, MATLAB tính chuẩn H_∞ của hàm truyền đạt và sau đó vẽ đặc tính tần số của hàm ở hình 3.9.



Hình 3.9 *Đồ thị BODE của hàm truyền đạt ở ví dụ dưới*

```
>> sys = tf(1,[1 2 100])      % Khai báo mô hình TF của sys

Transfer function:
       1
---------------
s^2 + 2 s + 100

>> [ninf,fpeak] = norm(sys,inf)
ninf =
    0.0503                    % Giá trị khuếch đại k = max
fpeak =
    9.8995                    % Tần số tại điểm k = max
>> bode (sys)                 % Vẽ đồ thị BODE của sys
```

Tìm phương sai khi kích thích bằng ồn trắng

Sử dụng lệnh `covar(sys,W)` ta có thể xác định được hàm tự tương quan (*Auto Correlation*) đầu ra của hệ *sys* khi kích thích bằng ồn trắng với biên độ *W* ở đầu vào của hệ. Đối với mô hình SS còn có thể thu được tự tương quan trạng thái.

`[P,Q] = covar (sys,W)`

Trong lệnh trên, *P* là hàm tự tương quan Φ_{yy} của đầu ra và Q là hàm Φ_{xx} của các trạng thái

```
>> sys = tf(1,[1 2 100]);                    % Ví dụ 1
>> P = covar(sys,1)
P =
    0.0025
>> sys = tf(1,[eps 1]);                      % Ví dụ 2
>> P = covar(sys,1)
P =
  2.2518e+015
```

Khảo sát động học của mô hình	
`covar` (*sys*, *W*)	Tìm phương sai với ồn trắng
`damp` (*sys*)	Tìm tần số riêng và hệ số tắt dần
`dcgain` (*sys*)	Tìm hệ số khuếch đại tĩnh
`norm` (*sys*)	Chuẩn của mô hình LTI (H_2 và H_∞)
`pole` (*sys*)	Tìm điểm cực của mô hình LTI
`dsort` (*sys*)	Xếp thứ tự điểm cực của mô hình LTI gián đoạn
`esort` (*sys*)	Xếp thứ tự điểm cực của mô hình LTI liên tục
`pzmap` (*sys*)	Biểu đồ điểm không - điểm cực
`eig` (*A*)	Tìm giá trị và vector riêng của ma trận **A**
`roots` (*c*)	Tìm nghiệm của đa thức *c*
`zero` (*sys*)	Tìm điểm không của mô hình LTI
`sgrid`	Tạo lưới cho biểu đồ điểm không - điểm cực (qũy đạo nghiệm số) trên miền ảnh Laplace *s*
`zgrid`	Tạo lưới cho biểu đồ điểm không - điểm cực (qũy đạo nghiệm số) trên miền ảnh *z*

3.3.3 Đáp ứng của hệ trên miền thời gian

Đối với kỹ sư điều khiển tự động, hiểu biết đặc tính của hệ trên miền thời gian giữ một vai trò rất quan trọng. Ta biết rằng, tín hiệu đầu ra trên miền thời gian (liên tục và gián đoạn) của một hệ có dạng:

$$\mathbf{y}(t) = \mathbf{C}e^{\mathbf{A}t}\mathbf{x}_0 + \int_{\tau=0}^{t} \left[\mathbf{C}e^{\mathbf{A}(t-\tau)}\mathbf{B} + \mathbf{D}\right]\mathbf{u}(\tau)d\tau \tag{3.39}$$

$$\mathbf{y}[k] = \mathbf{C}\mathbf{A}^k\mathbf{x}_0 + \sum_{i=0}^{k-1}\left[\mathbf{C}\mathbf{A}^{k-i-1}\mathbf{B} + \mathbf{D}\right]\mathbf{u}[i] \tag{3.40}$$

với vector trạng thái ban đầu $\mathbf{x}_0$ và vector tín hiệu vào **u**. Về nguyên tắc, ta thường quan tâm tới đáp ứng của hệ đối với một số tín hiệu thử dạng chuẩn như đáp ứng xung Dirac (hàm trọng lượng) hay đáp ứng bước nhẩy (hàm bước nhẩy). *Control System Toolbox* có sẵn các hàm: Tìm đáp ứng giá trị ban đầu `initial`, tìm đáp ứng xung Dirac `impulse` và tìm đáp ứng bước nhẩy `step`. Lệnh `gensig` cho phép tự tạo tín hiệu thử và `lsim` sẽ tính đáp ứng ra.

Đáp ứng giá trị ban đầu

Đáp ứng giá trị ban đầu mô tả phản ứng của hệ khi không có kích thích đầu vào nhưng tồn tại các giá trị ban đầu của vector trạng thái $\mathbf{x}_0$, phản ứng đó thường được gọi là *chuyển động tự do* của hệ. Đáp ứng được tính bằng lệnh `initial`.

```
initial(sys,x0,[,t])
[y,t,x] = initial(sys,x0,[,t])
```

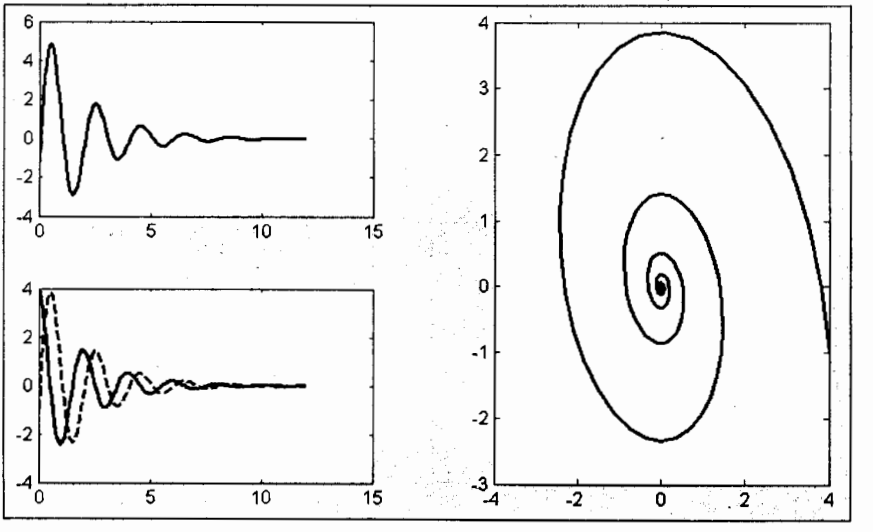
Trong lệnh thứ nhất, *sys* là một đối tượng MIMO biểu diễn theo mô hình SS với $\mathbf{x}_0$ là vector trạng thái ban đầu. Tham số tùy chọn *t* cho phép xác định khoảng thời gian tính, khi khai báo *t* là giá trị đơn, ta chỉ xác định khoảng thời gian tính. Nếu viết *t* = 0:*dt*:*Tf*, thời gian sẽ là một vector bao gồm các giá trị từ 0 tới *Tf* với bước *dt*. Nếu *sys* là mô hình gián đoạn, *dt* nên đồng thời là chu kỳ trích mẫu T_s. Nếu *sys* là mô hình liên tục ta phải chọn *dt* đủ nhỏ, vì khi tính đáp ứng MATLAB sẽ phải gián đoạn hóa mô hình theo phương pháp ZOH hoặc FOH và trích mẫu tín hiệu với chu kỳ *dt*.

Nếu sử dụng lệnh thứ hai, ta có *y* là vector biến ra, *t* là vector thời gian và *x* là vector trạng thái của các trạng thái. Trong trường hợp *sys* là đối tượng SIMO, đáp ứng *y* sẽ có số hàng đúng như số phần tử của *t* và số cột bằng số biến ra của *sys*, tức là có kích cỡ `length(t)` × *Ny*. Khi *sys* là đối tượng MIMO, không gian của *y* sẽ thêm một chiều thứ ba: Số lượng các biến vào *Nu*.

SIMO: Dim(y) = `length(t)` × *Ny*

MIMO: Dim(y) = `length(t)` × *Ny* × *Nu*

Tại đây ta cần chú ý: Những điều mô tả trên đây về vector thời gian *t* hay không gian của *y* là đúng với tất cả 4 lệnh `initial`, `impulse`, `step` và `lsim`. Đoạn ví dụ dưới đây đã được sử dụng để tạo nên các đáp ứng ở hình 3.10.



***Hình 3.10** Đáp ứng giá trị ban đầu: Đáp ứng thời gian (trái) và qũy đạo trạng thái (phải)*

```
>> systf = tf (5,[1 1 10])

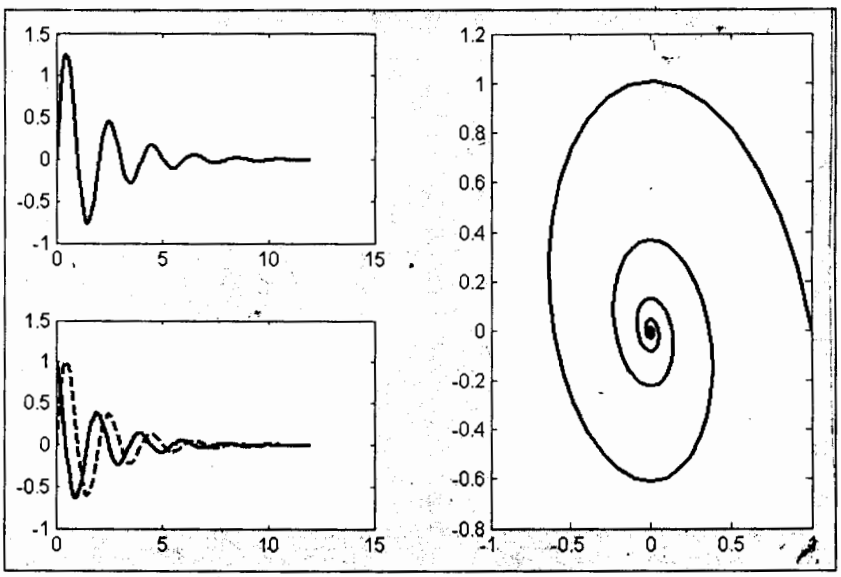
Transfer function:
     5
------------
s^2 + s + 10

>> sys = ss (systf);
>> [y,t,x] = initial (sys,'r-',[4 -1],12);
>> subplot (221)                    % Vẽ hình trên, bên trái
>> plot(t,y)
>> subplot (223)                    % Vẽ hình dưới, bên trái
>> plot(t,x(:,1),'r-',t,x(:,2),'g--')
>> subplot (122)                    % Vẽ hình bên phải:
>> plot(x(:,1),x(:,2))              % Qũy đạo trạng thái
```

Đáp ứng xung Dirac (hàm trọng lượng)

Ta thu được đáp ứng xung khi kích thích ở đầu vào của hệ một tín hiệu có dạng xung Dirac $\delta(t)$, hay xung đơn vị $\delta[k]$ khi đối tượng là hệ gián đoạn. Lệnh `impulse` có thể được sử dụng theo các phương án sau:

```
impulse(sys[,t])
y = impulse(sys[,t])
[y,t] = impulse(sys[,t])
[y,t,x] = impulse(sys[,t])
```



Hình 3.11 *Đáp ứng xung Dirac (thu được từ ví dụ của hình 3.9): Đáp ứng thời gian (trái) và qũy đạo trạng thái (phải)*

Khi dùng `impulse`, *sys* có thể là một mô hình LTI bất kỳ. Những thông tin ở trên (khi sử dụng `initial`) về vector thời gian t và về đối tượng MIMO là đúng kể cả khi sử dụng `impulse`. Việc tính vector trạng thái x chỉ có giá trị đối với mô hình SS, khi ấy vector trạng thái ban đầu được đặt bằng không.

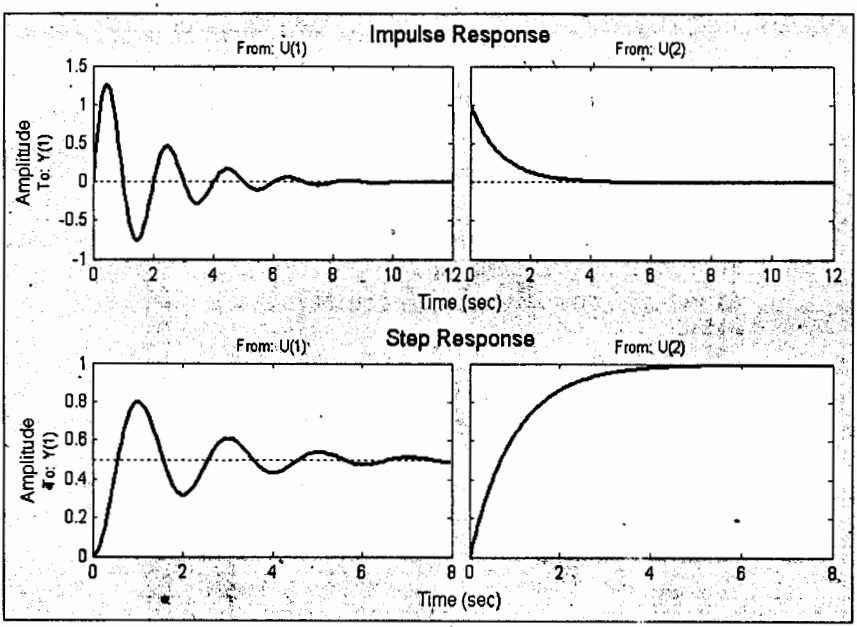
Đáp ứng bước nhẩy

Đáp ứng bước nhẩy (hàm bước nhẩy) thu được khi ta kích thích ở đầu vào của hệ một tín hiệu có dạng bước nhẩy đơn vị $\sigma(t)$, còn gọi là hàm $1(t)$. Các phương án lệnh cũng tương tự như khi tìm đáp ứng xung Dirac.

```
step(sys[,t])
y = step(sys[,t])
[y,t] = step(sys[,t])
[y,t,x] = step(sys[,t])
```

Các thông tin đã trình bầy ở trên đối với lệnh `impulse` cũng đúng đối với `step`. Hình 3.12 minh họa hiệu lực của `step`, mô hình ví dụ là đối tượng SIMO với hai hàm truyền đạt là các khâu quán tính bậc nhất PT_1 và bậc hai PT_2.

```
>> sys2in = [ tf(5,[1 1 10]) , tf(1,[1 1]) ];
>> subplot(211)
>> impulse (sys2in);
>> subplot(212)
>> step (sys2in);
```



Hình 3.12 *Đáp ứng xung Dirac (hàng trên) và đáp ứng bước nhẩy (hàng dưới) của khâu PT_2 (cột trái) và khâu PT_1 (cột phải)*

Đáp ứng đối với tín hiệu vào bất kỳ

Một hàm khác rất hữu ích là hàm `lsim(sys,u,t)`: Có thể sử dụng `lsim` để tìm đáp ứng của *sys* khi kích thích *sys* bởi một tín hiệu vào *u* gần như bất kỳ. Cú pháp của lệnh như sau:

`lsim(sys,u,t,[,x0])`
`[y,t] = lsim(sys,u,t)`
`[y,t,x] = lsim(sys,u,t,[,x0])`

Đối tượng *sys* là một hệ LTI (nếu ở dạng SS có thể khai báo vector trạng thái ban đầu x_0), *u* là ma trận tín hiệu vào bất kỳ, có số hàng bằng số phần tử của vector thời gian *t* và số cột bằng số biến vào của *sys*. Vậy kích cỡ của *u* là `length(t)` × *Nu*. Khi khai báo vector *t*, nếu ta chọn sai *dt* (ví dụ: quá lớn), `lsim` sẽ tự động nhận biết và thay vào đó bằng một giá trị *dt* phù hợp hơn.

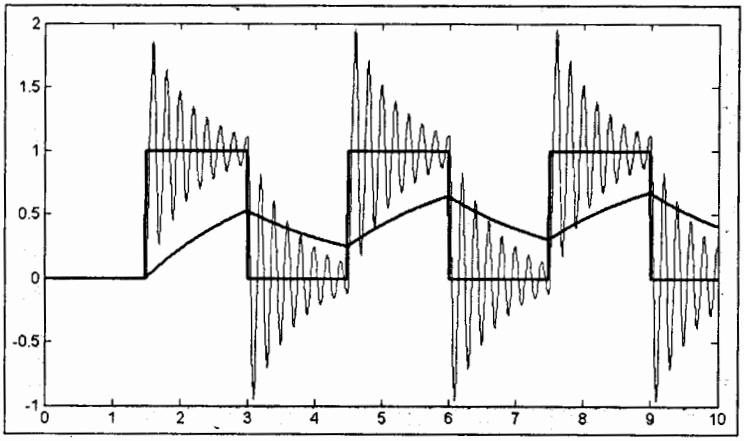
Tạo tín hiệu thử

Để tạo điều kiện cho người sử dụng không phải lập trình phức tạp khi cần một tín hiệu thử mới, *Control System Toolbox* cung cấp công cụ `gensig` để tạo tín hiệu thử *u* kèm theo vector thời gian *t*.

`[u,t] = gensig(typ,tau[,Tf,Ts])`

Bằng tham số *typ* ta có thể chọn một trong số các tín hiệu: Hình sin (`'sin'`), hoặc chuỗi xung vuông (`'square'`), hoặc hàm xung có chu kỳ (`'pulse'`). Chu kỳ của tín hiệu được khai báo nhờ tham số *tau*, *Tf* xác định khoảng thời gian đặt tín hiệu và *Ts* là chu kỳ trích mẫu. Hình 3.13 minh họa đáp ứng của hai khâu PT_1, PT_2 dưới đây khi bị kích thích bởi tín hiệu do `gensig` tạo nên.

```
>> sys2in = [ tf(1000,[1 3 1000]) ; tf(1,[2 1]) ];
>> [u,t] = gensig ('square',3,10,0.01); % Tạo tín hiệu u
>> [y,t] = lsim(sys2in,'r-',u,t);       % Đáp ứng đối với u
>> plot(t,y(:,1),t,y(:,2),t,u)
```



***Hình 3.13** Đáp ứng đối với tín hiệu thử tự tạo (dùng* `gensig` *và* `lsim`*)*

Đáp ứng của hệ trên miền thời gian	
`gensig(typ,tau)`	Tạo tín hiệu thử mới
`impulse(sys[,t])`	Tìm đáp ứng xung Dirac (hàm trọng lượng)
`initial(sys,x0,[,t])`	Tìm đáp ứng giá trị ban đầu
`lsim(sys,u,t,[,x0])`	Tìm đáp ứng đối với tín hiệu bất kỳ
`step(sys[,t])`	Tìm đáp ứng bước nhảy (hàm bước nhảy)

3.3.4 Đáp ứng của hệ trên miền tần số

Đối với kỹ sư điều khiển tự động, các hiểu biết về đặc tính của hệ trên miền tần số giữ một vai trò vô cùng quan trọng: Các tín hiệu thử thường có dạng sin, thêm vào đó là các tham số thiếu chính xác. Bằng các phương pháp khảo sát đặc tính tần số ta có thể rút ra được rất nhiều thông tin về các đặc điểm phụ thuộc tần số của hệ: Dự trữ module (biên độ) và dự trữ pha, bề rộng băng tần, tính ổn định của vòng điều chỉnh.

Trước hết ta quan tâm đến *hàm đặc tính tần* $F(j\omega)$, được định nghĩa với vai trò của hàm truyền đạt của một hệ LTI-SISO trên trục phức. Để tìm được $F(j\omega)$, điều kiện cần có là hệ phải ổn định tiệm cận: Phần thực của các giá trị riêng phải là âm. Ta chỉ quan tâm đến trạng thái xác lập, tức là: Toàn bộ các phần thực σ_i đều đã tắt tiệm cận hoàn toàn.

Tìm đáp ứng tần số

Để tính đáp ứng ta sử dụng hai lệnh `evalfr` và `freqresp`.

frsq = `evalfr(sys,f)`

H = `freqresp(sys,w)`

Nếu chỉ cần biết đáp ứng của *sys* đối với một tần số cụ thể, ta dùng `evalfr`. Cần biết rằng *f* cũng có thể là tần số phức. Khi tìm đáp ứng của *sys* đối với nhiều tần số, ta dùng `freqresp` với vector tần số *w*. Trong mọi trường hợp, biến đáp ứng *H* luôn là một trường 3 chiều với kích cỡ *Ny* × *Nu* × `length(w)`. Bằng cách viết *H*(*i*,*j*,*k*) ta luôn có khả năng truy cập tới từng phần tử của *H*: Đáp ứng thứ *k* của hàm truyền đạt giữa đầu vào thứ *j* và đầu ra thứ *i*.

```
>> sys = tf ([3 -1],[-2 1 1])

Transfer function:
  -3 s + 1
-------------
2 s^2 - s - 1

>> w = [-j , -2-j , -1+j]
w =
  Columns 1 through 2
                0 - 1.0000i -2.0000 - 1.0000i
```

```
  Column 3
                -1.0000 + 1.0000i
>> [evalfr(sys,w(1)) ; evalfr(sys,w(2)) ; evalfr(sys,w(3))]
ans =
        0 - 1.0000i
   0.5846 - 0.3231i
   0.6000 + 0.8000i
>> H = freqresp(sys,w)
H(:,:,1) =
        0 - 1.0000i
H(:,:,2) =
   0.5846 - 0.3231i
H(:,:,3) =
   0.6000 + 0.8000i
```

Đồ thị BODE

Đồ thị BODE biểu diễn đặc tính tần số của $F(j\omega)$ thành hai đồ thị riêng rẽ biên và pha với trục tần số được chia theo thang logarithm. Đồ thị thứ nhất có tên là đặc tính biên tần logarithm, được chia thang với thứ nguyên là decibel (dB, với $1dB = 20\log_{10}\left|F(j\omega)\right|$). Đồ thị thứ hai được gọi là đặc tính pha tần logarithm chia thang theo độ (°).

Cách biểu diễn này có ưu điểm, vì đặc tính tần số $F(j\omega)$ có cấu trúc mắc nối tiếp của nhiều hệ LTI con (tích của từng đặc tính tần số con $F_i(j\omega)$)

$$F(j\omega) = \prod_i^n \left|F_i(j\omega)\right| e^{j\sum_i^n \varphi_i(\omega)} \tag{3.41}$$

và vì vậy có biên độ logarithm là tổng của các biên độ logarithm thành phần $|F_i(j\omega)|$ và góc pha là tổng của các góc pha thành phần $\varphi_i(j\omega)$.

- Đặc tính biên tần logarithm:

$$\log_{10}\left|F(j\omega)\right| = \log_{10}\left|F_1(j\omega)\right| + \log_{10}\left|F_2(j\omega)\right| + \cdots + \log_{10}\left|F_n(j\omega)\right| \tag{3.42}$$

- Đặc tính pha tần logarithm:

$$\varphi(\omega) = \varphi_1(\omega) + \varphi_2(\omega) + \cdots + \varphi_n(\omega) \tag{3.43}$$

Trong MATLAB ta có thể vẽ đồ thị BODE của hệ *sys* một cách rất đơn giản nhờ các lệnh sau:

```
bode(sys)
bode(sys,w)
[mag, phase,w] = bode(sys)
```

Có thể khai báo vector tần số w một cách đơn giản chỉ bằng giới hạn dưới *wmin* và giới hạn trên *wmax* (tính bằng `rad/s`, viết trong ngoặc móc), hay dùng lệnh `logspace` (mục 1.2) để khai báo vector tần số với khoảng cách logarithm giữa các điểm tần số.

w = *wmin*:*increment*:*wmax*
w = { *wmin* , *wmax* }
w = logspace (*start*, *destination*, *number*)

Nếu ta sử dụng phương án lệnh thứ 3, khi ấy bode sẽ cất đặc tính biên trong *mag*, đặc tính pha trong *phase* và vector tần số kèm theo trong *w*. Cần chú ý rằng *mag* và *phase* là các trường 3 chiều với kích cỡ *Ny* × *Nu* × *k* với *k* = length(*w*). Trong trường hợp *sys* là hệ MIMO, mag(*i*,*j*,*k*) và phase(*i*,*j*,*k*) chính là giá trị biên tần và pha tần ứng với nhánh truyền đạt từ đầu vào thứ *j* tới đầu ra thứ *i*. Khi *sys* là hệ SISO, ta sẽ phải truy cập giá trị biên/pha tần cho giá trị ω_k cụ thể bằng lệnh mag(1,1,*k*) và phase(1,1,*k*). Đối với hệ liên tục, bode chỉ tính với các giá trị tần số ω dương trên trục ảo. Đối với hệ gián đoạn, bode chỉ tính hàm truyền đạt ứng với nửa trên của đường tròn đơn vị (miền ảnh *z*) theo:

$$z = e^{j\omega T_s} \qquad 0 \le \omega \le \omega_N = \frac{\pi}{T_s} \tag{3.44}$$

Trong (3.44), ω_N là tần số Nyquist. Nửa dưới của đường tròn đơn vị trùng với nửa trên do tính tuần hoàn với chu kỳ $2\omega_N$ của hàm truyền đạt *z*.

Chú ý: Nếu ta cần in đặc tính tần số của nhiều hệ trên cùng một đồ thị BODE hay NYQUIST, ta có thể gọi các lệnh bode và nyquist như sau:

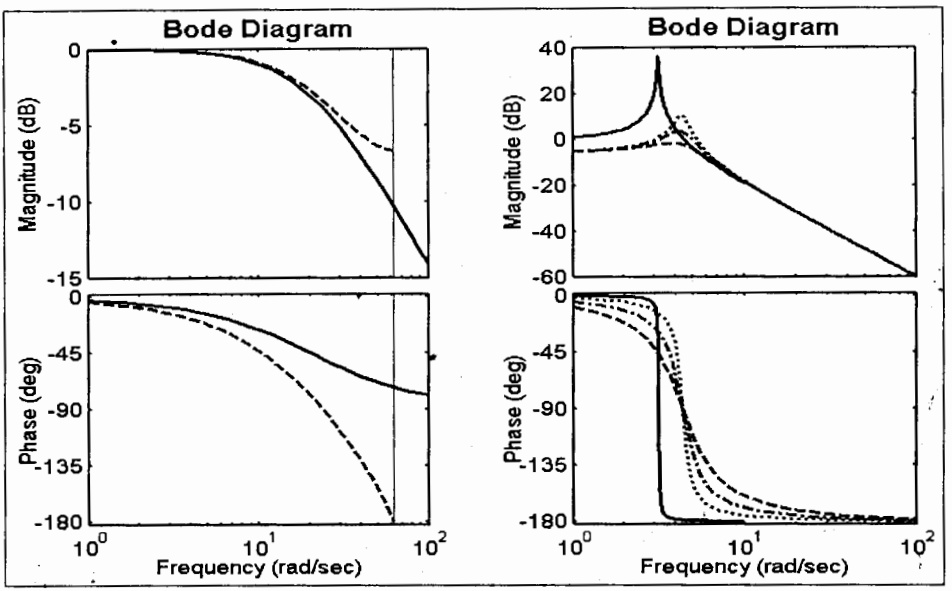
bode(*sys*1,*sys*2, ... ,*sysN*[,*w*])
bode(*sys*1,'*styl*1', ... ,*sysN*,'*stylN*')
nyquist(*sys*1,*sys*2, ... ,*sysN*[,*w*])
nyquist(*sys*1,'*styl*1', ... ,*sysN*,'*stylN*')

Trong cách viết lệnh ở trên, *sys*1, *sys*2 ... là các hệ LTI giống nhau về số lượng đầu vào, đầu ra. Nếu khai báo dạng của nét vẽ đồ thị bởi ('*styl*1', '*styl*2' ...), ta phải khai báo cho tất cả các hệ (không loại trừ hệ nào), và khi ấy ta sẽ không thể khai vector tần số được nữa. Để khắc phục nhược điểm này ta sẽ phải tự điền vào đồ thị nhờ các lệnh loglog và semilogx (xem mục 2.6.1).

Hình 3.14 minh họa tác dụng của lệnh bode. Nửa trái của hình được tạo bởi đoạn mã sau đây: Trước hết ta định nghĩa hàm truyền đạt sysPT1, tiếp theo ta gián đoạn hóa hàm đó với chu kỳ trích mẫu 0,05 và thu được sysPT1d. Đồ thị BODE của cả hai hệ được in ở nửa trái của hình 3.14.

```
>> sysPT1 = tf (1,[0.05 1])        % Khai báo khâu PT1
Transfer function:
    1
----------
0.05 s + 1
>> sysPT1d = c2d (sysPT1,0.05)     % Gián đoạn hóa khâu PT1
Transfer function:
  0.6321
----------
z - 0.3679
```

```
Sampling time: 0.05
>> subplot(121)                    % Vẽ nửa trái của hình 3.14
>> bode (sysPT1,'r-',sysPT1d,'b--')
```



Hình 3.14 *Đồ thị BODE của* `sysPT1`, `sysPT1d` *(trái) và* `sysPT2` *(phải)*

Việc tạo nửa bên phải hình 3.14 có phức tạp hơn: Ta vẽ đồ thị BODE của hàm truyền đạt `sysPT2` (khâu tỷ lệ có quán tính bậc 2) với các hệ số tắt dần có giá trị khác nhau. Trong vòng lặp `for` đầu tiên ta định nghĩa một mô hình TF với 4 đầu vào (chú ý: `size(sysPT2) = length(d)+1`) và 1 đầu ra. Hệ số tắt dần *D* của từng hàm truyền đạt con được lấy từ vector `d`.

Vì số lượng thành phần của vector `d` được chọn tùy ý và các đồ thị BODE cần được biểu diễn xếp chồng trên cùng một hệ trục, trước khi gọi lệnh `bode` ta phải gọi lệnh `hold on` để kích hoạt chức năng dừng - đợi sau từng đồ thị. Dạng nét của từng đồ thị BODE được khai báo và cất trong *Cell Array* `stil` và sẽ lần lượt được gọi bằng lệnh `stil{n}`.

```
>> sysPT2 = tf (10,[1 0.05 10])      % Định nghĩa sysPT2

Transfer function:
       10
-----------------
s^2 + 0.05 s + 10

>> d = [sqrt(2)/2 1.6 3];            % Vector d có 4 giá trị
```

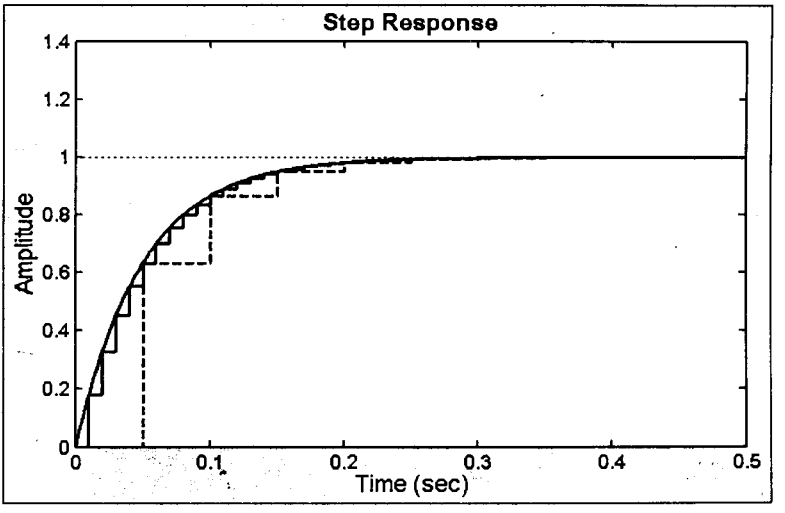
```
>> for n = 1:1:length(d) ,                % n = 1, 2, 3, 4
       sysPT2 = [sysPT2 ; tf(10,[1 d(n) 10])];
   end;                                   % Vòng for 1: Khai báo
                                          % 4 hệ con của sysPT2
>> subplot(122) ,                         % Vẽ nửa trái hình 3.14
>> hold on;                               % Lệnh dừng-đợi
>> stil = {'r-' 'b:' 'k-.' 'g--'};        % Khai báo nét vẽ
>> for n = 1:1:(length(d)+1) ,
       bode (sysPT2(n),stil{n});          % Vẽ sysPT2 thứ n
                                          % theo styl thứ n
   end;
```

Nửa trái của hình 3.14 chỉ cho ta thấy: Do chu kỳ trích mẫu của `sysPT1d` được chọn quá lớn, đồ thị BODE của `sysPT1` và `sysPT1d` không thể trùng với nhau. Để minh họa kết luận này, ta gián đoạn hóa `sysPT1` lại một lần nữa với chu kỳ trích mẫu 0,01 và thu được `sysPT1dnew`, sau đó ta so sánh đáp ứng bước nhẩy của cả 3 hàm trong hình 3.15.

```
>> sysPT1dnew = d2d (sysPT1d,0.01)

Transfer function:
  0.1813
----------
z - 0.8187

Sampling time: 0.01
>> step(sysPT1,'r-',sysPT1d,'b--',sysPT1dnew,'g-')
```



Hình 3.15 *Đáp ứng bước nhẩy của* `sysPT1`, `sysPT1d` *và* `sysPT1dnew`

Dự trữ biên và pha

Một công cụ quan trọng nữa, phục vụ khảo sát ổn định của hệ, là hai lệnh `margin` và `allmargin`. Hai lệnh đó tính dự trữ ổn định (ví dụ: dự trữ biên và pha) của hệ LTI.

- **Dự trữ biên**: Được định nghĩa là hệ số khuếch đại F_R, mà nếu ta bổ sung F_R thêm vào hàm truyền đạt của vòng hở, hệ khép kín sẽ vừa vặn đạt tới giới hạn giữa ổn định và không ổn định. Giá trị F_R được tính từ nghịch đảo của biên độ $|F(j\omega)|$ tại tần số đảo pha ω_φ (pha bắt đầu vượt $-180°$).
- **Dự trữ pha**: Được định nghĩa là khoảng cách góc φ_R (tính từ góc pha tại vị trí điểm cắt giữa $|F(j\omega)|$ của vòng hở với đường tròn đơn vị) tới $-180°$. Tần số ω_A tại điểm cắt đó được gọi là tần số cắt biên.

Để bảo đảm cho hệ ổn định cần có:

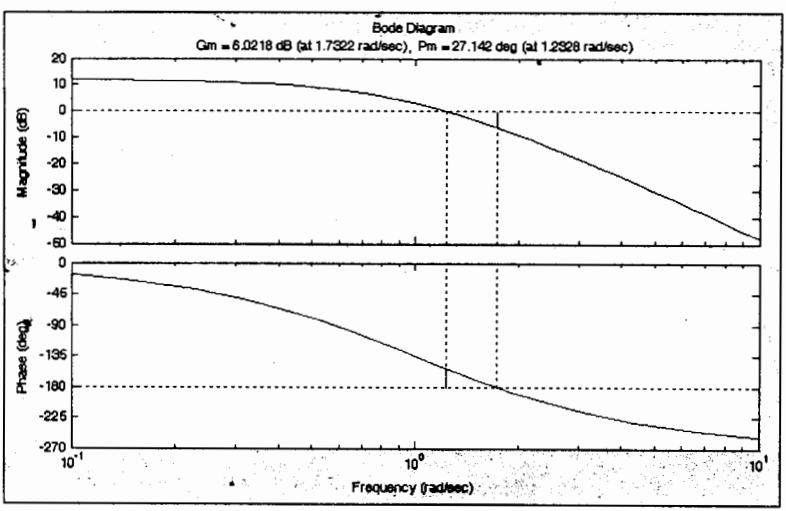
$$\omega_A < \omega_\varphi \quad \text{đồng nghĩa với} \quad F_R > 1 \quad \text{đồng nghĩa với} \quad \varphi_R > 0$$

Tại ranh giới ổn định ta có: $\omega_A = \omega_\varphi$ và $F_R = 1$ và $\varphi_R = 0$

Lệnh margin được viết như sau:

```
margin(sys)
[Gm,Pm,Wcg,Wcp] = margin(sys)
[Gm,Pm,Wcg,Wcp] = margin(mag, phase, w)
```

Nếu `margin` được gọi không có biến trả về, lệnh sẽ in đồ thị BODE như ở hình 3.16. Các biến được yêu cầu trả lại là dự trữ biên *Gm* với tần số đảo pha *Wcg*, dự trữ pha *Pm* với tần số cắt biên *Wcp*. Các giá trị *mag*, *phase*, *w* ứng với định nghĩa đã biết khi ta sử dụng lệnh `bode`.



***Hình 3.16** Đồ thị BODE với dự trữ biên và pha*

Đồ thị BODE ở hình 3.16 được vẽ bằng đoạn lệnh sau đây:

```
>> sys = zpk([],[-1 -1 -1],4)

Zero/pole/gain:
   4
-------
(s+1)^3
>> margin(sys)
```

Lệnh `allmargin` có tác dụng rộng hơn lệnh `margin`: Lệnh `allmargin` tính nhiều tham số liên quan đến ổn định của hệ trên cơ sở mạch vòng hở và sau đó cất các tham số tính được trong biến tên là *stabil* có dạng cấu trúc *Struct*.

stabil = `allmargin(sys)`

Các trường thuộc cấu trúc của *stabil* có ý nghĩa như sau (tần số tính bằng rad/s):

`GainMargin`	**Dự trữ biên**: Giá trị đảo của biên độ tại tần số `GMFrequency`
`GMFrequency`	Giá trị tần số mà tại đó đồ thị pha cắt đường thẳng nằm ngang –180°
`PhaseMargin`	**Dự trữ pha**: Khoảng cách góc (>0) từ vị trí `PMFrequency` tới –180°
`PMFrequency`	Giá trị tần số mà tại đó đồ thị biên cắt đường thẳng nằm ngang 0 dB (ứng với hệ số khuếch đại 1)
`DelayMargin`	**Dự trữ thời gian trễ**: Giá trị thời gian trễ mà nếu vượt quá, hệ sẽ mất ổn định
`DMFrequency`	Giá trị tần số ứng với `DelayMargin`
`Stable`	= 1 khi mạch vòng khép kín ổn định = 0 cho các trường hợp còn lại

Tiếp tục sử dụng ví dụ của hình 3.16 sẽ giúp ta làm sáng tỏ ý nghĩa của các tham số trên. Kết quả tính của `allmargin(sys)` được gán cho biến *stabil*: Hệ *sys* ổn định, dự trữ biên là 2,0003, dự trữ pha là 27,1424° và dự trữ thời gian trễ là 0,3843s. Ta cũng có thể đọc được các giá trị này từ hình 3.16.

```
>> stabil = allmargin (sys)
stabil =
    GMFrequency: 1.7322
     GainMargin: 2.0003
    PMFrequency: 1.2328
    PhaseMargin: 27.1424
    DMFrequency: 1.2328
    DelayMargin: 0.3843
         Stable: 1                    % Kết luận: sys ổn định
```

Nếu bây giờ ta thử gán thêm cho *sys* khoảng thời gian trễ với giá trị `stabil.DelayMargin+0.01`, tức là vượt quá dự trữ thời gian trễ 0,01 giây. Kết quả tính toán mới của `allmargin` sẽ thông báo tính không ổn định của *sys* sau khi bị gán thời gian trễ.

```
>> sys.ioDelay = stabil.DelayMargin + 0.01
                              % Gán thời gian trễ cho sys
Zero/pole/gain:
                    4
exp(-0.39*s) * -------
                (s+1)^3

>> allmargin (sys)
ans =
    GMFrequency: [1x8 double]
     GainMargin: [1x8 double]
    PMFrequency: 1.2328
    PhaseMargin: -0.7063
    DMFrequency: 1.2328
    DelayMargin: -0.0100
         Stable: 0          % Kết luận: sys không ổn định
```

Đồ thị NYQUIST

Một khả năng mô tả đặc tính tần số khác cũng rất hay được sử dụng là đồ thị NYQUIST: Biểu diễn các giá trị thực và ảo thuộc hàm truyền đạt phức của mạch vòng hở $F_0(j\omega)$ trong dải tần số từ $\omega = 0$ tới $\omega = \infty$ trên hệ tọa độ phức. Đường cong do các điểm tạo thành được gọi là qũy đạo biên pha $F_0(j\omega)$.

Trên cơ sở tiêu chuẩn ổn định NYQUIST ta có thể rút ra kết luận về tính ổn định của vòng điều chỉnh khép kín (có phản hồi đơn vị - âm) từ đồ thị NYQUIST.

"Vòng điều chỉnh khép kín sẽ là ổn định nếu, tia nối từ tọa độ nguy hiểm –1+j0 tới qũy đạo biên pha 1-$F_0(j\omega)$ quét một góc là:

$$\overset{\omega=+\infty}{\underset{\omega=0}{\Delta}} \phi = n_r\pi + n_a\frac{\pi}{2} \tag{3.45}$$

khi ω biến thiên từ 0 tới ∞. Trong đó, hàm $F_0(j\omega)$ cũng được phép chứa thành phần trễ.

$$F_0(s) = \frac{Z_0(s)}{N_0(s)}e^{-sT_t}; \qquad n_0 > m_0, T_t \geq 0 \tag{3.46}$$

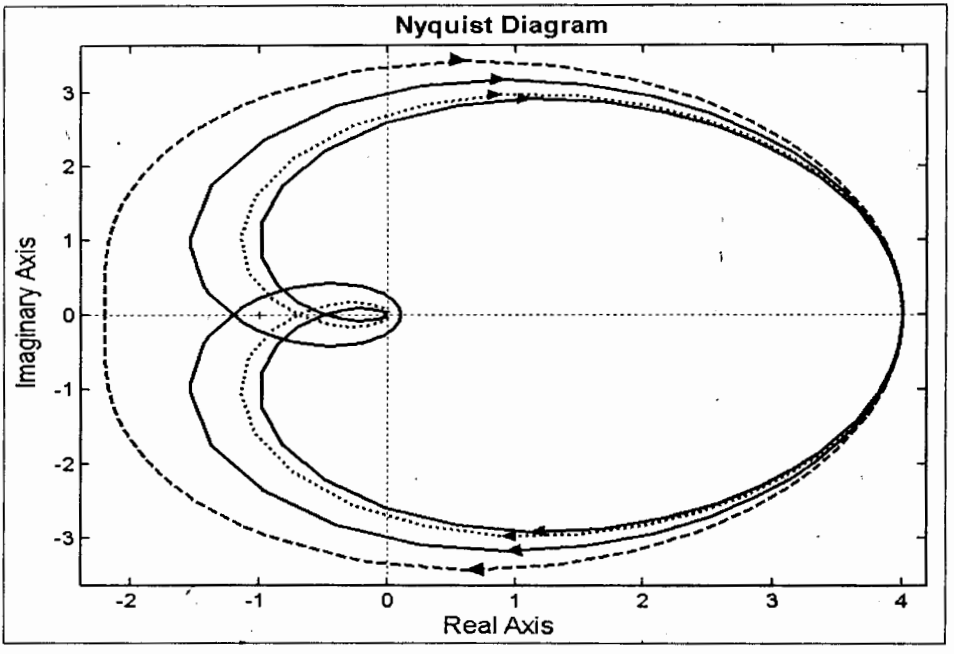
Trong (3.45) và (3.46), m_0 và n_0 là bậc của đa thức tử và mẫu số của F_0. n_r, n_a và n_t là nghiệm của $N_0(s)$ nằm phía phải (không ổn định), trên (giới hạn ổn định) và bên trái (ổn định) trục ảo của miền ảnh Laplace.

Cú pháp của lệnh vẽ đồ thị NYQUIST như sau:

```
nyquist(sys)
nyquist(sys,w)
[re,im,w] = nyquist(sys)
[re,im] = nyquist(sys,w)
```

Khi khai báo vector tần số *w*, hay khi vẽ nhiều đặc tính xếp chồng trên cùng một hệ trục tọa độ, bạn đọc có thể thực hiện giống hệt như đối với lệnh bode. Để minh họa hiệu lực của lệnh nyquist, ta thực hiện ví dụ sau: Trước hết, khai báo mô hình ZPK *sys*, sau đó tạo từ *sys* ba mô hình ZPK gián đoạn với chu kỳ trích mẫu khác nhau. Cuối cùng, ta vẽ đồ thị NYQUIST của cả ba chung trong hình 3.17.

```
>> sys = zpk([],[-1 -1 -1],4);
>> sysd1 = c2d(sys,0.3);
>> sysd2 = c2d(sys,1.5);
>> sysd3 = c2d(sys,4.0);
>> nyquist (sys,'r-',sysd1,'b:',sysd2,'k-',sysd3,'g--')
```



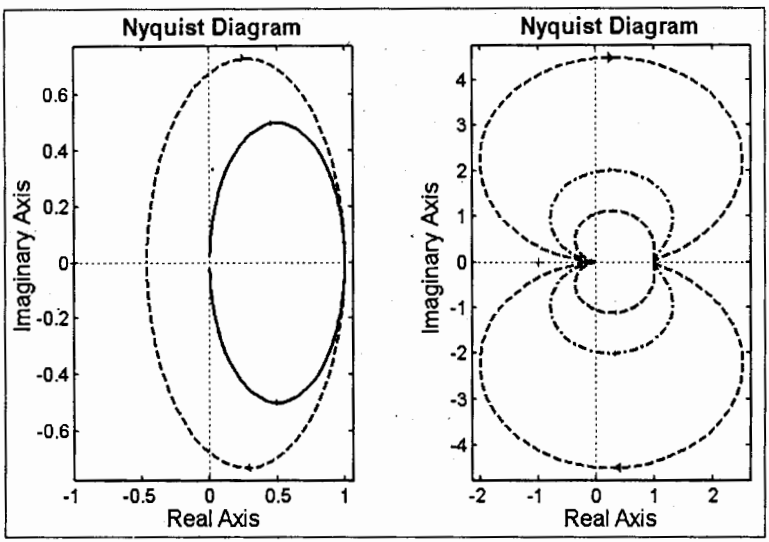
Hình 3.17 *Đồ thị NYQUIST của* sys, sysd1, sysd2 *và* sysd3 *với chu kỳ trích mẫu khác nhau*

Hình 3.18 minh họa đồ thị NYQUIST của các khâu PT_1, PT_2 đã sử dụng trong phần ví dụ thuộc lệnh bode.

```
>> sysPT1 = tf (1,[0.05 1]);          % Khai báo khâu PT1
>> sysPT1d = c2d (sysPT1,0.05);       % Gián đoạn hóa khâu PT1
>> sysPT2 = tf (10,[1 0.05 10]);      % Khai báo khâu PT2

>> subplot(121)                       % Vẽ nửa hình bên trái
>> nyquist (sysPT1,'r-',sysPT1d,'b--')

>> d = [sqrt(2)/2 1.6 3];             % Khai báo 4 hệ số D
>> for n = 1:1:length(d) ,
     sysPT2 = [sysPT2 ; tf(10,[1 d(n) 10])];
   end;                               % Khai báo 4 × sysPT2
>> subplot (122) ,                    % Vẽ nửa hình bên phải
>> hold on;
>> stil = {'r-' 'b--' 'k-.' 'g--'};
>> for n = 2:1:size(sysPT2,1) ,
     nyquist (sysPT2(n),stil{1+mod(n-1,size(sysPT2,1))});
   end;
```



Hình 3.18 *Đồ thị NYQUIST của sysPT1, sysPT1d (trái) và sysPT2 với các hệ số tắt dần D khác nhau (phải)*

Đáp ứng của hệ trên miền tần số	
bode	Vẽ đồ thị BODE
evalfr	Tìm đáp ứng đối với một tần số phức
freqresp	Tìm đáp ứng đối với nhiều tần số theo lựa chọn
margin	Tìm dự trữ biên và pha
allmargin	Tìm các tham số ổn định
nyquist	Vẽ đồ thị NYQUIST

3.3.5 Mô hình giảm bậc

Đôi khi,việc sử dụng mô hình giảm bậc rất có lợi trong công tác phân tích đặc điểm hệ thống. Ví dụ: Khi cần khảo sát đặc điểm tần số thấp hay tần số cao của hệ, mà không được phép bỏ qua các tính chất quan trọng.

Tự động giảm bậc

Bằng lệnh `minreal` ta có thể *loại trừ các trạng thái không quan sát được* và *các trạng thái không điều khiển được* ra khỏi mô hình SS, hay *triệt tiêu các điểm không - điểm cực trùng nhau* ra khỏi hàm truyền đạt củamô hình TF và ZPK. Tham số tùy chọn *tol* cho biết dung sai khi loại trừ trạng thái hay triệt tiêu điểm không - điểm cực. Khi không khai báo, *Control System Toolbox* mặc định đặt *tol* = `sqrt(eps)`.

sysr = `minreal(`*sys*`, [`*tol*`])`

msys = `sminreal(`*sys*`)`

Khi hệ được khai báo dưới dạng mô hình trạng thái, lệnh `sminreal` loại trừ tất cả các trạng thái tồn tại do đặc điểm riêng của cấu trúc, nhưng lại không hề ảnh hưởng tới quan hệ vào/ra. Ta có thể dễ dàng nhận thấy từ ví dụ dưới đây: Việc khai báo dung sai là 0.01 sẽ dẫn đến việc triệt tiêu điểm không –0,999 và điểm cực –1,001.

```
>> sys = zpk ([-0.999 1],[-1.001 2],0.5)
                                    % Khai báo mô hình ZPK
Zero/pole/gain:
0.5 (s+0.999) (s-1)
-------------------
  (s+1.001) (s-2)

>> minreal (sys)                    % tol mặc định quá nhỏ
                                    % không có tác dụng khử
Zero/pole/gain:
0.5 (s+0.999) (s-1)
-------------------
  (s+1.001) (s-2)

>> minreal (sys,0.01)               % Khử với dung sai 0.01

Zero/pole/gain:
0.5 (s-1)
---------
  (s-2)
```

Trong ví dụ tiếp theo, biến trạng thái thứ hai của mô hình *sys* (ở dạng SS) không hề có liên quan gì tới đầu vào và đầu ra. Sau khi dùng `sminreal`, hệ được hạ bậc xuống `sys(1,1)`.

```
>> sys = ss ([1 2 ; 0 3],[4 ; 0],[5 0],0) ;
>> msys = sminreal(sys);
>> [ msys.a msys.b msys.c msys.d ]
ans =
     1     4     5     0
```

Giảm bậc có mục đích

Nếu ta định đơn giản hóa một mô hình có bậc cao, hay đơn giản chỉ là dự định khảo sát riêng rẽ các trạng thái với động học khác nhau, khi ấy ta nên dùng công cụ `modred`.

rsys = `modred(`*sys*`,`*elim*`)`

rsys = `modred(`*sys*`,`*elim*`[,`*method*`])`

Bên cạnh *sys*, cần phải chuyển cho `modred` (thông qua tham số *elim*) vector chứa chỉ số của các trạng thái cần được triệt tiêu. Về cơ bản ta có hai phương pháp tính khác nhau, khai báo trong tham số *method* (giá trị mặc định: `'mdc'`).

`'mdc'` (*matching dc-gain*) có tác dụng tạo mô hình giảm bậc có **hệ số khuếch đại một chiều** (dc, khuếch đại tĩnh) **giống như của hệ gốc**. Đạo hàm (vế trái) của các trạng thái khai báo trong *elim* được đặt bằng không, như vậy các trạng thái đó sẽ bị loại trừ ra khỏi các phương trình còn lại.

`'del'` (*delete*) tách các trạng thái khai trong *elim* ra khỏi mô hình gốc. Việc tách này sẽ làm thay đổi hệ số khuếch đại một chiều (dc, khuếch đại tĩnh), nhưng (trên nguyên tắc) lại **bảo đảm đồng nhất tốt hơn giữa mô hình giảm bậc và mô hình gốc trên miền tần số**.

Ví dụ sau đây sẽ tạo mô hình *sys* từ hai hàm truyền đạt của hai khâu PT_1 (`sysPT1`) và PT_2 (`sysPT2`). Tiếp theo ta sử dụng hai phương pháp `'del'` và `'mdc'` để tìm các mô hình giảm bậc `sys1mdel`, `sys2mdel` và `sys1mmdc`, `sys2mmdc`. Cuối cùng ta vẽ đáp ứng bước nhảy của các mô hình giảm bậc trong hình 3.19 và đồ thị BODE trong hình 3.20 để so sánh.

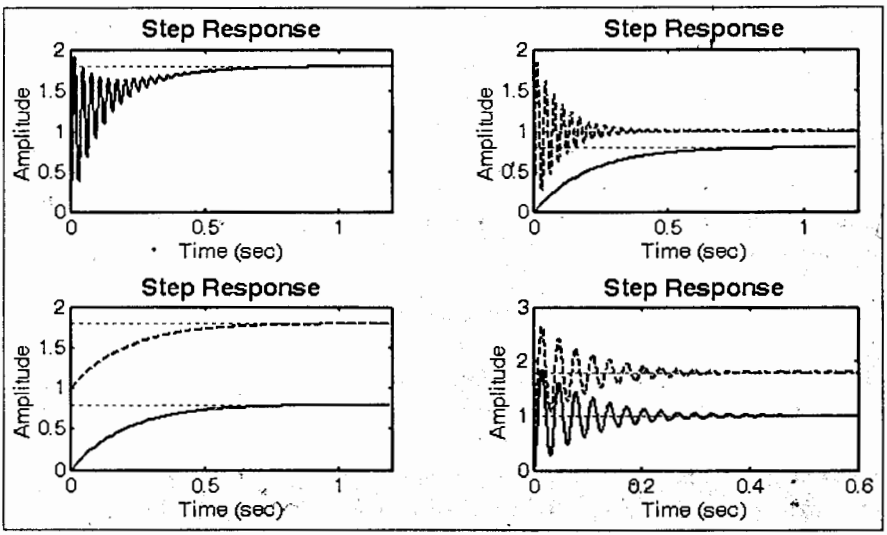
```
>> T = 0.2 ; V = 0.8;
>> wn = T/0.001 ; d = 0.05;
>> sysPT1 = ss (tf (V,[T 1]));          % Khai báo khâu PT1
>> sysPT2 = ss (tf (wn^2,[1 2*d*wn wn^2]));
                                        % Khai báo khâu PT2
>> sys = sysPT1 + sysPT2;               % sys có 5 trạng thái

>> sys1mdel = modred (sys,[2,3],'del'); % Giảm bậc theo
>> sys2mdel = modred (sys,[1],'del');   % ph² 'del'
>> sys1mmdc = modred (sys,[2,3],'mdc'); % Giảm bậc theo
>> sys2mmdc = modred (sys,[1],'mdc');   % ph² 'mdc'
```

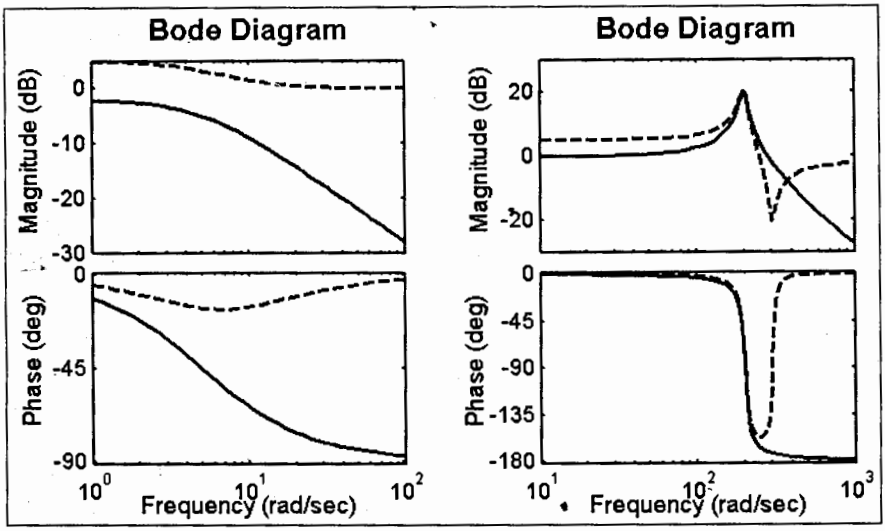
```
>> sys.a                                % Ma trận A của sys
ans =
   -5.0000         0         0
         0  -20.0000 -156.2500
         0  256.0000         0
>> [sysPT1.a , sys1mdel.a , sys1mmdc.a]
ans =                                   % Ma trận A của sysPT1
    -5    -5    -5                      % trước và sau giảm bậc
>> [sysPT2.a , sys2mdel.a , sys2mmdc.a]
ans =                                   % Ma trận A của sysPT2
                                        % trước và sau giảm bậc
  -20.0000 -156.2500 -20.0000 -156.2500  -20.0000 -156.2500
  256.0000         0 256.0000         0  256.0000         0

>> figure(1)                            % Vẽ đáp ứng bước nhảy
                                        % (hình 3.19)
>> subplot(221) , step (sys,'r-')
>> subplot(222) , step (sysPT1,'b-',sysPT2,'g--')
>> subplot(223) , step (sys1mdel,'b-',sys1mmdc,'g--')
>> subplot(224) , step (sys2mdel,'b-',sys2mmdc,'g--')

>> figure(2)                            % Vẽ đồ thị BODE
                                        % (hình 3.20)
>> subplot(121),bode(sysPT1,'r-',sys1mdel,'b-',sys1mmdc,'g--')
>> subplot(122),bode(sysPT2,'r-',sys2mdel,'b-',sys2mmdc,'g--')
```



Hình 3.19 *Đáp ứng bước nhảy trước và sau giảm bậc:* `sys` *(trên-trái),* `sysPT1` *và* `sysPT2` *(trên-phải),* `sys1mdel` *và* `sys1mmdc` *(dưới-trái),* `sys2mdel` *và* `sys2mmdc` *(dưới-phải)*



***Hình 3.20** Đồ thị BODE trước và sau giảm bậc:* `sysPT1`, `sys1mdel` *và* `sys1mmdc` *(trái);* `sysPT2`, `sys2mdel` *và* `sys2mmdc` *(phải)*

Mô hình giảm bậc	
`minreal`	Loại trừ trạng thái, triệt tiêu điểm không - điểm cực
`modred`	Giảm bậc có mục đích
`sminreal`	Tự động giảm bậc theo cấu trúc

3.3.6 Các phương pháp mô tả trên không gian trạng thái

Tùy theo mục đích nghiên cứu mà ta có thể sử dụng các phương pháp mô tả khác nhau trên không gian trạng thái. Nhờ sử dụng các ma trận chuyển thích hợp mà ta có thể tạo ra các mô hình khác nhau, ví dụ: Các ma trận mô tả đặc điểm điều khiển hay quan sát, cho phép khảo sát tính điều khiển được hay quan sát được của hệ.

Các mô hình dạng chuẩn tắc (canonical normal form)

Control System Toolbox cung cấp cho ta lệnh `canon`, cho phép thực hiện hai dạng mô tả chuẩn tắc trên cơ sở khai báo tham số *type*.

```
csys = canon(sys,'type')
[csys,T] = canon(sys,'type')
```

Khi *sys* là mô hình SS, MATLAB sẽ cất ma trận chuyển trạng thái trong biến *T*, là ma trận sẽ tạo nên vector trạng thái mới $\mathbf{x}_c = \mathbf{Tx}$. Nếu tham số *type* nhận giá trị `'modal'`, ma trận hệ thống của hệ mới *csys* sẽ chứa các giá trị riêng thực λ_i trên đường chéo, còn các giá trị riêng phức $\sigma \pm j\omega$ phân bố trên 2 × 2 ma trận con.

Giá trị riêng thực λ_1, λ_2
Giá trị riêng phức liên hợp $\sigma \pm j\omega$

$$\mathbf{A_m} = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \sigma & \omega & 0 \\ 0 & -\omega & \sigma & 0 \\ 0 & 0 & 0 & \lambda_2 \end{bmatrix} \tag{3.47}$$

Mô hình dạng chuẩn điều khiển

Là dạng chuẩn do `canon` tạo ra khi ta gán cho tham số *type* giá trị `'companion'`. Ma trận hệ thống (của mô hình mới) được hợp thành từ các hệ số của đa thức đặc tính $P(s)$ như sau (n: Số trạng thái):

$$P(s) = s^n + c_1 s^{n-1} + \cdots + c_{n-1}s + c_n$$

$$\Rightarrow \quad \mathbf{A_C} = \begin{bmatrix} 0 & 0 & \cdots & \cdots & 0 & -c_n \\ 1 & 0 & 0 & \cdots & 0 & -c_{n-1} \\ 0 & 1 & 0 & \cdots & \vdots & \vdots \\ \vdots & 0 & \ddots & \cdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 1 & 0 & -c_2 \\ 0 & \cdots & \cdots & 0 & 1 & -c_1 \end{bmatrix} \tag{3.48}$$

Tính quan sát được

Một thông tin rất quan trọng đối với kỹ sư điều khiển tự động khi khảo sát hệ thống, đó là: Liệu trạng thái của đối tượng có quan sát được hay không? Hay, nếu không phải tất cả các trạng thái đều quan sát được thì: Những trạng thái nào có thể quan sát được?

Để trả lời các câu hỏi trên ta sử dụng ma trận kiểm tra đặc điểm quan sát được **Ob** (*Observability Matrix*), tính từ ma trận hệ thống **A** và ma trận đầu ra **C** theo công thức sau:

$$\mathbf{Ob} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{n-1} \end{bmatrix} \tag{3.49}$$

Nếu ma trận **Ob** có hạng (*rang*) đầy đủ như ma trận hệ thống, khi ấy hệ là quan sát được toàn phần. Nếu không phải vậy: Hạng của **Ob** ứng với số trạng thái có thể quan sát được của hệ. Để kiểm tra ta sử dụng:

Ob = `obsv(A,C)`
Ob = `obsv(sys)`

Nếu cần biết cụ thể: Những trạng thái nào là không thể quan sát được, khi ấy ta sẽ phải sử dụng **dạng chuẩn quan sát** để khảo sát. Nếu một hệ là không quan sát được toàn phần, ta có thể dùng phương pháp chuyển hệ sang dạng bậc thang:

$$\overline{\mathbf{A}} = \mathbf{T}\mathbf{A}\mathbf{T}^T; \overline{\mathbf{B}} = \mathbf{T}\mathbf{B}; \overline{\mathbf{C}} = \mathbf{C}\mathbf{T}^T \tag{3.50}$$

với kết quả có dạng tổng quát:

$$\overline{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_{no} & \mathbf{A}_{12} \\ \mathbf{0} & \mathbf{A}_o \end{bmatrix}; \overline{\mathbf{B}} = \begin{bmatrix} \mathbf{B}_{no} \\ \mathbf{B}_o \end{bmatrix}; \overline{\mathbf{C}} = \begin{bmatrix} \mathbf{0} & \mathbf{C}_o \end{bmatrix} \tag{3.51}$$

Trong công thức (3.51), $\mathbf{A}_{no}$ là ma trận của các trạng thái không quan sát được và $\mathbf{B}_{no}$ là ma trận của phần đầu vào tương ứng. $\mathbf{A}_o$, $\mathbf{B}_o$ và $\mathbf{C}_o$ là các ma trận của phần trạng thái quan sát được.

Trong MATLAB ta tìm ma trận kiểm tra đặc điểm quan sát được như sau:

```
[Abar,Bbar,Cbar,T,k] = obsvf(A,B,C,[tol])
```

Ba biến *Abar*, *Bbar*, *Cbar* chứa các ma trận $\overline{\mathbf{A}}, \overline{\mathbf{B}}, \overline{\mathbf{C}}$. Vector *k* có chiều dài bằng kích cỡ của **A**, với tổng các phần tử của *k* chính là số các trạng thái quan sát được. *T* là ma trận chuyển hệ và tham số tùy chọn *tol* cho biết dung sai cho phép khi tính dạng chuẩn quan sát.

Ví dụ sau đây minh họa hiệu quả của hai lệnh `obsv` và `obsvf`.

Lệnh `obsv`

```
>> A = [ 6 -1 ; 1 4 ]
A =
     6    -1
     1     4
>> B = [ -2 2 ; -2 2 ]
B =
    -2     2
    -2     2
>> C = [ 1 0 ; 0 1 ]
C =
     1     0
     0     1
>> Ob = obsv (A,C)
Ob =
     1     0
     0     1
     6    -1
     1     4
>> rank (Ob)
ans =
     2
```

Lệnh `obsvf`

```
>>[Abar,Bbar,Cbar,T,k]=obsvf (A,B,C)
Abar =
     6    -1
     1     4
Bbar =
    -2     2
    -2     2
Cbar =
     1     0
     0     1
T =
     1     0
     0     1
k =
     2     0
```

Cả hai lệnh đều đưa ra kết luận: Hệ (khai báo ở bên trái) là quan sát được toàn phần.

Tính điều khiển được

Một đặc điểm quan trọng nữa là tính điều khiển được của hệ thống. Nếu ta muốn điều khiển một số trạng thái nhất định, ta phải biết: Các trạng thái đó có điều khiển được hay không. Để kiểm tra tính chất này ta có thể sử dụng một trong hai công cụ: Ma trận kiểm tra đặc điểm điều khiển được **Co** (*Controllability Matrix*), hay dạng chuẩn điều khiển của mô hình trạng thái.

Ma trận **Co** của hệ với n trạng thái được tính theo công thức sau đây:

$$\mathbf{Co} = \begin{bmatrix} \mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \cdots & \mathbf{A}^{n-1}\mathbf{B} \end{bmatrix} \tag{3.52}$$

Nếu **Co** có hạng (*rang*) đầy đủ như ma trận hệ thống, khi ấy hệ là điều khiển được toàn phần. Nếu không, hạng của **Co** ứng với số trạng thái có thể điều khiển được. Cú pháp của lệnh kiểm tra như sau:

Co = `ctrb`(*A*, *C*)

Co = `ctrb`(*sys*)

Nếu cần biết cụ thể về các trạng thái không điều khiển được, ta sử dụng **dạng chuẩn điều khiển** để khảo sát. Nếu hệ là không điều khiển được toàn phần, ta cũng có thể sử dụng phương pháp chuyển hệ sang dạng bậc thang với công thức như (3.50) và thu được kết quả:

$$\overline{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_{nc} & \mathbf{0} \\ \mathbf{A}_{21} & \mathbf{A}_c \end{bmatrix}; \overline{\mathbf{B}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{B}_c \end{bmatrix}; \overline{\mathbf{C}} = \begin{bmatrix} \mathbf{C}_{nc} & \mathbf{C}_c \end{bmatrix} \tag{3.53}$$

Tất cả các trạng thái không điều khiển được nằm trong $\mathbf{A}_{nc}$, phần điều khiển được cất trong các ma trận $\mathbf{A}_c$, $\mathbf{B}_c$ và $\mathbf{C}_c$.

Trong MATLAB ta tìm ma trận kiểm tra đặc điểm điều khiển được như sau:

[*Abar*, *Bbar*, *Cbar*, *T*, *k*] = `ctrbf`(*A*, *B*, *C*, [*tol*])

Tương tự khi khảo sát tính quan sát được: Ba biến *Abar*, *Bbar*, *Cbar* chứa các ma trận $\overline{\mathbf{A}}, \overline{\mathbf{B}}, \overline{\mathbf{C}}$. Vector k có chiều dài bằng kích cỡ của **A**, với tổng các phần tử của k chính là số các trạng thái điều khiển được. T là ma trận chuyển hệ và tham số tùy chọn *tol* cho biết dung sai cho phép khi tính dạng chuẩn quan sát.

Ví dụ sau đây minh họa hiệu quả của hai lệnh `ctrb` và `ctrbf`.

Lệnh `ctrb`

```
>> A = [ 6 -1 ; 1 4 ]
A =
     6    -1
     1     4
>> B = [ -2 2 ; -2 2 ]
B =
    -2     2
    -2     2
>> C = [ 1 0 ; 0 1 ]
C =
     1     0
```

Lệnh `ctrbf`

```
>> [Abar,Bbar,Cbar,T,k]=ctrbf (A,B,C)
Abar =
    5.0000   -0.0000
    2.0000    5.0000
Bbar =
         0         0
    2.8284   -2.8284
Cbar =
   -0.7071   -0.7071
    0.7071   -0.7071
```

```
     0     1
>> Co = ctrb (A,B)
Co =
    -2          2      -10
10
    -2          2      -10
10
>> rank (Co)
ans =
     1
```

```
T =
   -0.7071    0.7071
   -0.7071   -0.7071

k =
     1     0
```

Cả hai lệnh đều đưa ra kết luận: Hệ (khai báo ở bên trái) là không điều khiển được toàn phần

Mô tả trên không gian trạng thái	
canon	Mô hình dạng chuẩn modal
ctrb	Mô hình dạng chuẩn điều khiển
ctrbf	Mô hình dạng chuẩn điều khiển có dạng bậc thang
obsv	Mô hình dạng chuẩn quan sát
obsvf	Mô hình dạng chuẩn quan sát có dạng bậc thang

3.4 Thiết kế vòng điều chỉnh

Mục đích cuối cùng của quá trình nghiên cứu – khảo sát chính là việc thiết kế (còn gọi là: tổng hợp) vòng điều chỉnh. Nghĩa là: Thiết kế khâu điều khiển làm việc trong mạch vòng khép kín, có phản hồi đầu ra hay phản hồi trạng thái, nhằm gán cho hệ một đặc tính hệ thống mong muốn nào đó. Bước đầu tiên cần làm của giai đoạn này là lựa chọn khâu điều khiển, tiếp theo là bước tham số hóa khâu đã chọn.

Chỉ sau khi đã thu được các kết quả mô phỏng như ý ta mới bắt đầu quá trình thực hiện kỹ thuật trên hệ thống thiết bị thực và cuối cùng là thử nghiệm kiểm chứng. Nếu các kết quả thử nghiệm thỏa mãn chất lượng đặt ra, nhiệm vụ nghiên cứu – thiết kế mới có thể được coi là đã kết thúc.

Tuy nhiên, đôi khi nẩy sinh nhu cầu cải thiện, nâng cao chất lượng kết quả đã thu được, dẫn đến: Tất cả các bước (bắt đầu từ mô hình hóa, khảo sát hệ thống cho tới thiết kế khâu điều khiển) phải được thực hiện lặp lại nhiều lần.

Control System Toolbox cung cấp cho ta một số công cụ thiết kế theo các phương pháp:

- Phương pháp quỹ đạo điểm cực
- Phương pháp gán cực (tìm phản hồi và khâu quan sát trạng thái)
- Phương pháp tính theo tiêu chuẩn tích phân (tuyến tính, bình phương)
- Quan sát trạng thái khi có nhiễu bằng lọc Kalman

3.4.1 Thiết kế theo phương pháp quỹ đạo điểm cực

Phương pháp kinh điển để tham số hóa khâu điều khiển của vòng điều chỉnh hệ SISO là phương pháp quỹ đạo điểm cực (QĐĐC).

Quỹ đạo điểm cực

Quỹ đạo điểm cực là quỹ đạo hợp thành bởi các điểm cực của vòng điều chỉnh (ĐC) phụ thuộc vào hệ số khuếch đại phản hồi k và được biểu diễn trên mặt phẳng phức với thành phần thực $\mathrm{Re}\{\lambda\} = \sigma$ trên trục hoành x và thành phần ảo $\mathrm{Im}\{\lambda\} = \omega$ trên trục tung y.

Trong khuôn khổ của Control System Toolbox, hàm truyền đạt vòng hở $-G_0$ bao gồm các hàm truyền đạt G_o của đối tượng (*object*), G_c của khâu điều khiển (*controller*), G_M của cảm biến đo (*measurment*) và hệ số khuếch đại phản hồi k (n: đa thức tử số, d: đa thức mẫu số):

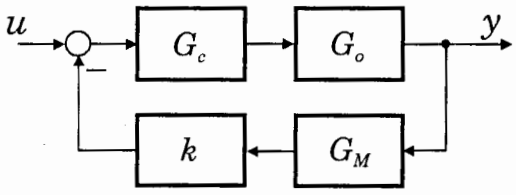
$$-G_0 = k G_c G_o G_M = k \frac{n_c n_o n_M}{d_c d_o d_M} \tag{3.54}$$

Hàm truyền đạt của vòng ĐC khép kín (có phản hồi âm đầu ra) có dạng:

$$G = \frac{G_c G_o}{1 + k G_c G_o G_M} = \frac{n_c n_o d_M}{d_c d_o d_M + k n_c n_o n_M} \tag{3.55}$$

Điểm cực của vòng ĐC chính là nghiệm của đa thức mẫu số của (3.55).

$$d_0 + k n_0 = d_c d_o d_M + k n_c n_o n_M = 0 \tag{3.56}$$



Hình 3.21 Vòng điều chỉnh có phản hồi âm đầu ra

Đối với hệ SISO có dạng mô hình LTI ta có thể sử dụng lệnh `rlocus`:

```
rlocus(sys[,k])
[r,k] = rlocus(sys)
r = rlocus(sys,k)
```

Mô hình LTI *sys* trong lệnh trên là hàm truyền đạt của mạch vòng hở $G_c G_o G_M$ mà chưa có hệ số khuếch đại phản hồi k, là tham số tuỳ chọn sẽ được khai báo riêng. Điều đó có nghĩa là: *sys* được tạo thành bởi sự ghép nối các mô hình riêng lẻ (cần đặc biệt chú ý tới trình tự khai báo).

Hàm truyền đạt của vòng hở: $G_0 = G_c G_o G_M$

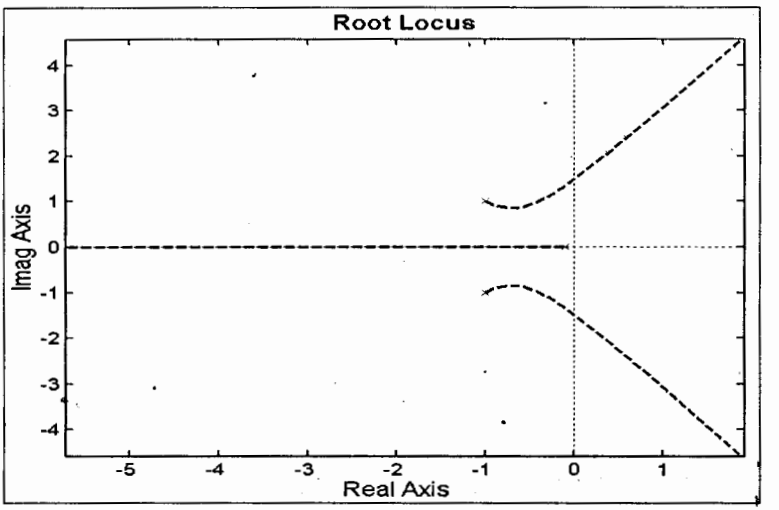
Cú pháp MATLAB:

```
sys = sysM * sysO * sysC
```

Khi gọi `rlocus(sys[,k])` mà không yêu cầu trả biến về, ta sẽ thu được đồ thị của quỹ đạo điểm cực. Nếu ta không chủ động khai báo các hệ số khuếch đại

trong vector tham số tùy chọn *k*, MATLAB sẽ tự động quyết định giá trị thích hợp. Nhân đây cũng xin nhắc lại khả năng in phân bố điểm không - điểm cực bằng lệnh `pzmap` (mục 3.3.2). Tương tự, tại đây ta cũng có thể sử dụng hai lệnh `sgrid` và `zgrid` để vẽ lưới tạo bởi các đường đẳng trị về hệ số tắt dần D trên miền ảnh Laplace *s* và miền ảnh *z*.

Nếu ta sử dụng `[r,k] = rlocus(sys)`, vị trí của các điểm cực (ứng với các hệ số khuếch đại cất trong vector *k*) được cất trong ma trận *r*. Ma trận *r* có `length(k)` cột, và các điểm cực thuộc giá trị *k*(*n*) được cất ở cột thứ *n*. Hình 3.22 biểu diễn quỹ đạo điểm cực của *sys* với *k* trong khoảng tối thiểu – tối đa.



Hình 3.22 *Quỹ đạo điểm cực của hàm sys*

```
>> sys = zpk([],[-0.1 -1-i -1+i],1)    % Khai báo sys

Zero/pole/gain:
          1
----------------------
(s+0.1) (s^2 + 2s + 2)

>> rlocus (sys)                         % Vẽ quỹ đạo điểm cực
>> [r,k] = rlocus (sys);
>> [k(1) , k(max(find(k<inf))) ; ...
   r(:,1) , r(:,max(find(k<inf))) ]
ans =
  1.0e+005 *
        0              4.5575          % Khoảng min<k<max
  -0.0000 + 0.0000i    0.0004 + 0.0007i
  -0.0000 - 0.0000i    0.0004 - 0.0007i
  -0.0000             -0.0008
```

Xuất vector hệ số khuếch đại

Giả sử đã sử dụng `rlocus` để vẽ quỹ đạo điểm cực (hình 3.22), MATLAB còn tạo điều kiện để ta tìm các giá trị liên quan (hệ số khuếch đại, các điểm cực khác cùng nhóm) tới một điểm cực bất kỳ nằm trên quỹ đạo đó: Đó là lệnh `rlocfind`.

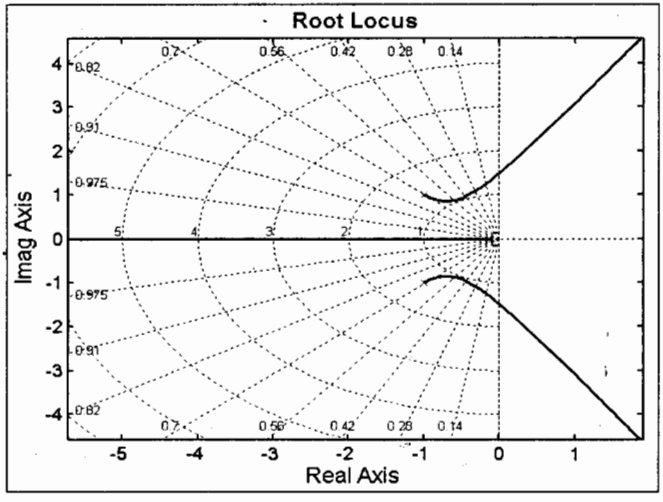
`[k,r] = rlocfind(sys)`

`[k,r] = rlocfind(sys,p)`

Sau khi gọi `rlocfind`, bằng chuột ta có thể chọn (nháy chuột trái) một điểm trên đồ thị đã vẽ, và nhận được các giá trị đã nói ở trên. Bạn đọc có thể tự mình kiểm chứng bằng cách lần lượt thực hiện chuỗi lệnh trong ví dụ dưới đây. Hình 3.23 hiển thị ba điểm cực (dấu +) sau khi ta chọn điểm có tọa độ:

`-0.42621892019453 + 0.98393434969275i`

Cuối cùng ta gọi `sgrid` để tạo lưới cho đồ thị.



Hình 3.23 *Tìm hệ số khuếch đại và các điểm cực liên quan nhờ* `rlocfind`

```
>> sys = zpk([],[-0.1 -1-i -1+i],1)    % Khai báo hàm sys

Zero/pole/gain:

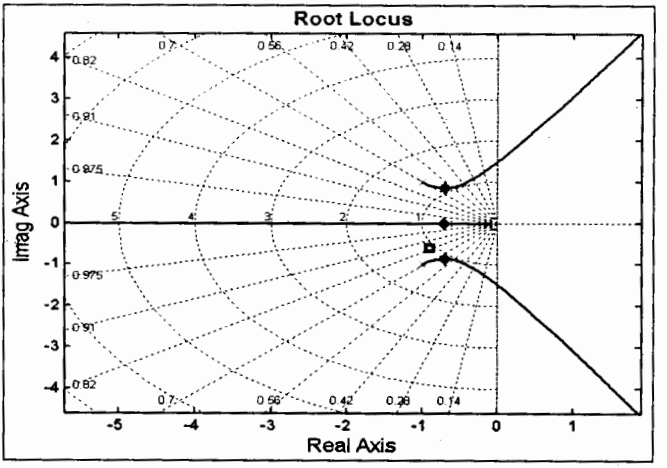
          1
----------------------
(s+0.1) (s^2 + 2s + 2)

>> rlocus (sys)                         % Vẽ quỹ đạo điểm cực
>> [kf,rf] = rlocfind (sys)             % Chọn điểm cực bằng
Select a point in the graphics window % mouse
selected_point =                        % Tọa độ điểm chọn:
 -0.42621892019453 + 0.98393434969275i
```

```
kf =                                          % Hệ số khuếch đại kf
   1.22885190921764                           % của điểm đã chọn
rf =                                          % Giá trị của ba điểm
 -1.25424252487196                            % cực ứng với kf
 -0.42287873756402 + 0.97999417461653i
 -0.42287873756402 - 0.97999417461653i
>> sgrid
```

Bằng việc khai báo thêm vector tham số p, `rlocfind` sẽ tìm các điểm thuộc quỹ đạo gần với các giá trị trong p. Các giá trị của k được tính theo $k = |den(p)/num(p)|$, còn các điểm cực là nghiệm của (3.56). Tiếp tục ví dụ trên: Trước hết ta khai báo điểm `p = [-0.9-0.6i]` và sau đó tính hệ số khuếch đại *kff*, các nghiệm *rff*. Cuối cùng ta biểu diễn p(□) và các giá trị tương ứng *rff*(◊) trong hình 3.24.

```
>> p = [-0.9-0.6*i]                           % Khai báo điểm p
p =
 -0.9000 - 0.6000i
>> [kff,rff] = rlocfind (sys,p)               % Tính:
kff =                                         % hệ số khuếch đại
   0.66098411478643
rff =                                         % và bộ điểm cực
 -0.69521695931838 + 0.85444053768211i        % lân cận
 -0.69521695931838 - 0.85444053768211i
 -0.70956608136323
>> hold on
>> plot(p,'ks')                               % Vẽ điểm p(□)
>> plot(rff,'kd')                             % Vẽ 3 điểm rff(◊)
```



Hình 3.24 *Tìm bộ điểm cực rff(◊) lân cận điểm p(□)*

Phương pháp quỹ đạo điểm cực	
`rlocus(sys[,k])`	Vẽ quỹ đạo điểm cực
`rlocfind(sys,p)`	Tìm bộ tham số gần điểm p

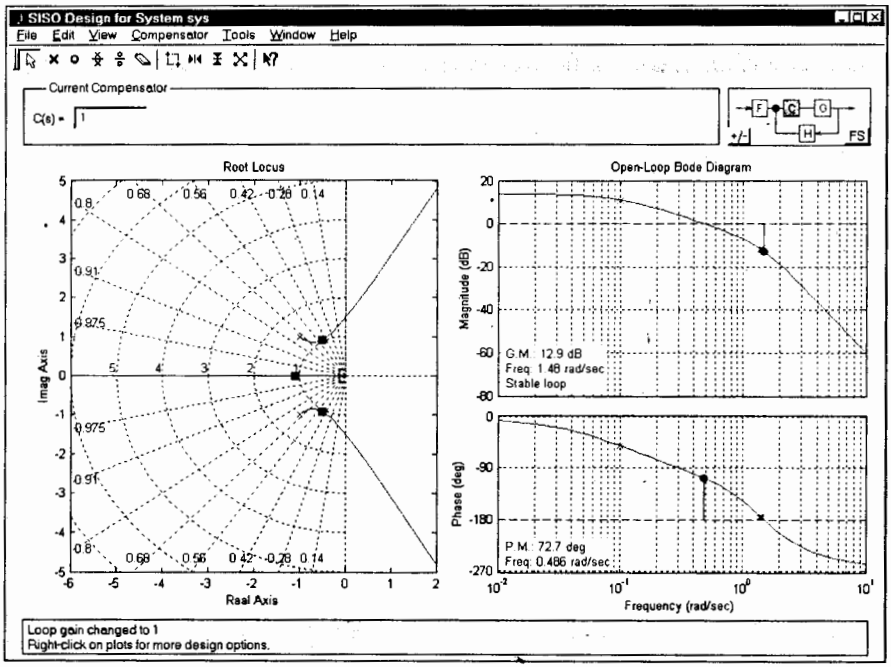
3.4.2 Thiết kế theo phương thức đối thoại giữa người và PC

Khi thiết kế các hệ thống SISO, *Control System Toolbox* cung cấp cho người sử dụng một công cụ hoạt động theo kiểu đối thoại (*interactive*) khá tiện lợi, đó là: *SISO Design Tool*. Quá trình khảo sát đối tượng và thiết kế vòng ĐC xảy ra trong khuôn khổ một văn bản có sẵn.

Để gọi ta sử dụng lệnh `sisotool` với các cách viết như sau:

`sisotool(sys[,comp])`

`sisotool(view,sys,comp,options)`



Hình 3.25 *Công cụ SISO Design Tool*

Nếu gọi `sisotool` không có tham số ta thu được cửa sổ *SISO Design Tool* rỗng. Thêm tham số *sys* ta trao cho lệnh một mô hình đối tượng dạng LTI đã khai báo, tham số tùy chọn *comp* là mô hình LTI của khâu điều khiển (*compensator*). Tham số *view* có thể nhận giá trị `'rlocus'` khi cần vẽ quỹ đạo điểm cực, hay `'bode'` khi cần đồ thị BODE. *options* là biến có dạng cấu trúc `struct` với hai trường:

- *options*.location: Khai báo khâu điều khiển nằm ở nhánh xuôi ('forward', mặc định nếu người sử dụng không chủ động khai) hay ở nhánh phản hồi ('feedback').
- *options*.sign: Quyết định phản hồi có dấu âm (-1, mặc định khi người sử dụng không chủ động khai) hay dương (1)

Ví dụ ở hình 3.25 được tạo nên bởi đoạn lệnh sau đây

```
>> options.location = 'forward' ;
>> options.sign = -1 ;
>> sisotool(zpk([],[-0.1 -1-i -1+i],1),1,options) ;
```

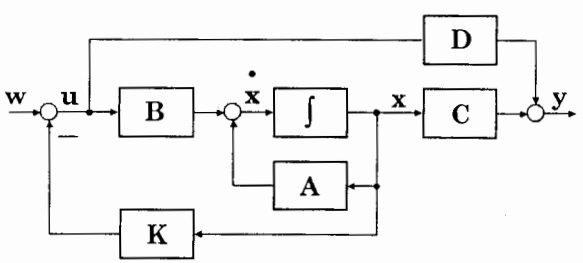
Mục này sẽ không tiếp tục đi sâu vào công cụ *SISO Design Tool*, bạn đọc quan tâm có thể tra cứu trực tiếp *User's Guide* của *Control System Toolbox* hay *On-Line Help* của MATLAB.

3.4.3 Điều khiển và quan sát trạng thái

Phương pháp thiết kế trình bầy trong mục này dựa trên cơ sở mô hình trạng thái của đối tượng, sử dụng lý luận của điều khiển và quan sát trạng thái. Các nội dung lý thuyết chỉ được trình bầy một cách ngắn gọn và không có dẫn dắt. Ngoài ra, chúng chỉ được trình bầy vừa đủ để hiểu tính năng và tác dụng của các công cụ thiết kế mà *Control System Toolbox* cung cấp. Bạn đọc nào cần tra cứu sâu về lý thuyết sẽ phải xem các tài liệu chuyên khác.

Điều khiển trạng thái

Về nguyên tắc, hệ thống có điều khiển trạng thái là hệ có phản hồi âm toàn bộ các biến trạng thái thông qua ma trận phản hồi **K** (hình 3.26).



Hình 3.26 Hệ có phản hồi trạng thái

- Phương trình trạng thái của đối tượng:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \qquad (3.57)$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

- Luật điều khiển (w = 0):

$$\mathbf{u} = -\mathbf{K}\mathbf{x} \qquad (3.58)$$

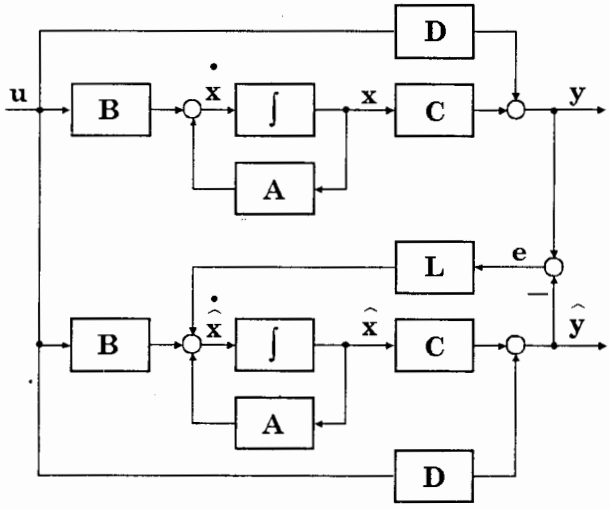
- Vòng ĐC khép kín:

$$\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} \qquad (3.59)$$

Điều kiện để thực hiện hệ thống điều khiển trạng thái như hình 3.26 là khả năng phản hồi tất cả các biến trạng thái, nghĩa là: Mọi biến trạng thái đều phải đo được, điều khó được thỏa mãn trong thực tiễn. Chính vì vậy ta sẽ phải cần đến khâu quan sát, cho phép tính vector biến trạng thái từ các giá trị đo được ở đầu ra.

Quan sát trạng thái (khâu quan sát Luenberger)

Khâu quan sát kinh điển sử dụng một mô hình tương đương với đối tượng, và một ma trận **L** phản hồi sai lệch giữa đầu ra thật và đầu ra của mô hình, có nhiệm vụ hiệu chỉnh đặc tính mô hình cho phù hợp với đặc tính của đối tượng.



Hình 3.27 *Quan sát vector biến trạng thái*

- Phương trình trạng thái của khâu quan sát:

$$\begin{aligned}\dot{\hat{\mathbf{x}}} &= \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}\mathbf{e} \\ \hat{\mathbf{y}} &= \mathbf{C}\hat{\mathbf{x}} + \mathbf{D}\mathbf{u}\end{aligned} \tag{3.60}$$

- Sai lệch quan sát:

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{C}\left(\mathbf{x} - \hat{\mathbf{x}}\right) \tag{3.61}$$

- Phương trình trạng thái mới:

$$\begin{aligned}\dot{\hat{\mathbf{x}}} &= \left(\mathbf{A} - \mathbf{L}\mathbf{C}\right)\hat{\mathbf{x}} + \left(\mathbf{B} - \mathbf{L}\mathbf{D}\right)\mathbf{u} + \mathbf{L}\mathbf{y} \\ &= \left(\mathbf{A} - \mathbf{L}\mathbf{C}\right)\hat{\mathbf{x}} + \begin{bmatrix}\mathbf{B} - \mathbf{L}\mathbf{D} & \mathbf{L}\end{bmatrix}\begin{bmatrix}\mathbf{u} \\ \mathbf{y}\end{bmatrix}\end{aligned} \tag{3.62}$$

Điều khiển trạng thái sử dụng khâu quan sát Luenberger

Khi thực hiện hệ thống điều khiển trạng thái, nếu ta sử dụng vector trạng thái quan sát được $\hat{\mathbf{x}}$ thay cho vector trạng thái thực $\mathbf{x}$, ta thu được phương trình của hệ thống có phản hồi như sau:

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{e}} \end{bmatrix} = \begin{bmatrix} \mathbf{A}-\mathbf{BK} & \mathbf{BK} \\ \mathbf{0} & \mathbf{A}-\mathbf{LC} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{e} \end{bmatrix}; \mathbf{e} = \mathbf{x} - \hat{\mathbf{x}} \tag{3.63}$$

Việc thiết kế khâu điều khiển (ma trận **K**) và khâu quan sát trạng thái (ma trận **L**) thường được thực hiện theo phương pháp gán cực. Tại đây ta cần chú ý: Động học của khâu quan sát phải bảo đảm nhanh hơn động học của khâu điều khiển.

3.4.4 Thiết kế theo phương pháp gán cực

Đây là phương pháp phổ cập khi thiết kế hệ thống điều khiển trạng thái: Ma trận phản hồi **K** được chọn sao cho các điểm cực mới xuất hiện (khi hệ thống đã khép kín) ứng với các điểm cực của một đa thức mong muốn cho trước.

Tính vector k / ma trận K phản hồi trạng thái

Control System Toolbox cung cấp cho ta hai lệnh `acker` và `place`. Cả hai lệnh đó đều tìm vector phản hồi trạng thái **k**, hay ma trận phản hồi trạng thái **K**, sao cho các điểm cực của hệ đồng nhất với các điểm cực mong muốn (khai báo trong vector *p*). Các điểm cực mong muốn nên tránh lặp lại nhiều lần.

```
k = acker(A,b,p)
K = place(A,B,p)
[K,prec,message] = place(A,B,p)
```

Hai lệnh trên khác nhau duy nhất ở đặc điểm: `acker` chỉ thích hợp với đối tượng SISO, còn `place` thích hợp cả với SISO và MIMO. Thêm vào đó, `place` mạnh hơn và cho phép ta khai báo cụ thể hơn về mục tiêu thiết kế: Đó là độ chính xác (tham số *prec*) của vị trí điểm cực thuộc hệ mới (hệ có phản hồi trạng thái) so với vị trí dự định gán, và sẽ có cảnh báo (cất trong *message*) khi sai số gán lớn hơn 10%. Một nhược điểm chính của `acker` là lệnh đó sẽ nhanh chóng trở nên mất ổn định khi phải thực hiện một khối lượng lớn tính toán số.

Tính ma trận L phản hồi sai lệch quan sát trạng thái

Ta cũng có thể dùng hai lệnh `acker` và `place` để tính ma trận **L** của khâu quan sát trạng thái. Tuy nhiên, trước khi sử dụng ta phải *thực hiện chuyển vị* (*transpose*) cho ma trận **A**, và thay vào vị trí của **B** ta khai báo ma trận chuyển vị của **C**. Cuối cùng, ma trận **L** là kết quả của phép chuyển vị ma trận do `acker` và `place` tìm được.

```
L = acker(A',c',p).'
L = place(A',C',p).'
```

Thiết kế khâu quan sát trạng thái

Nếu việc tính ma trận **L** đã kết thúc, ta có thể tổng hợp rất nhanh chóng và đơn giản khâu quan sát trạng thái. Lệnh `estim` có tác dụng tạo ra các phương trình hệ thống của khâu quan sát *est* như sau:

```
est = estim(sys,L)
est = estim(sys,L,sensors,known)
```

Có thể sử dụng lệnh theo dạng thứ nhất, nếu tất cả các tín hiệu vào đều là biến ngẫu nhiên (tạp âm hệ thống **w**, tạp âm đo lường **v**) và mọi biến ra đều có thể đo được. Phương trình cơ sở của tính toán là như sau:

Đối tượng

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{w}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{w}$$

Khâu quan sát

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{L}\left(\mathbf{y} - \mathbf{C}\hat{\mathbf{x}}\right)$$
$$\begin{bmatrix}\hat{\mathbf{y}}\\ \hat{\mathbf{x}}\end{bmatrix} = \begin{bmatrix}\mathbf{C}\\ \mathbf{I}\end{bmatrix}\hat{\mathbf{x}} \qquad (3.64)$$

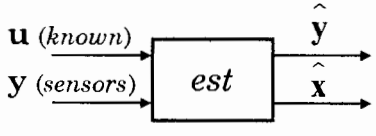
Nếu sử dụng thêm hai vector chỉ số *sensors* và *known*, ta có thể khai báo các biến ra đo được và các biến vào mà ta biết. Các biến ra không đo được sẽ được cất trong **z**. **w** và **v** là tạp âm hệ thống và tạp âm đo lường. Lúc này, phương trình cơ sở của tính toán là:

Đối tượng

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}_1\mathbf{w} + \mathbf{B}_2\mathbf{u}$$
$$\begin{bmatrix}\mathbf{z}\\ \mathbf{y}\end{bmatrix} = \begin{bmatrix}\mathbf{C}_1\\ \mathbf{C}_2\end{bmatrix}\mathbf{x} + \begin{bmatrix}\mathbf{D}_{11}\\ \mathbf{D}_{21}\end{bmatrix}\mathbf{w} + \begin{bmatrix}\mathbf{D}_{12}\\ \mathbf{D}_{22}\end{bmatrix}\mathbf{u}$$

Khâu quan sát

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}_2\mathbf{u} + \mathbf{L}\left(\mathbf{y} - \mathbf{C}_2\hat{\mathbf{x}} - \mathbf{D}_{22}\mathbf{u}\right)$$
$$\begin{bmatrix}\hat{\mathbf{y}}\\ \hat{\mathbf{x}}\end{bmatrix} = \begin{bmatrix}\mathbf{C}_2\\ \mathbf{I}\end{bmatrix}\hat{\mathbf{x}} + \begin{bmatrix}\mathbf{D}_{22}\\ \mathbf{0}\end{bmatrix}\mathbf{u} \qquad (3.65)$$

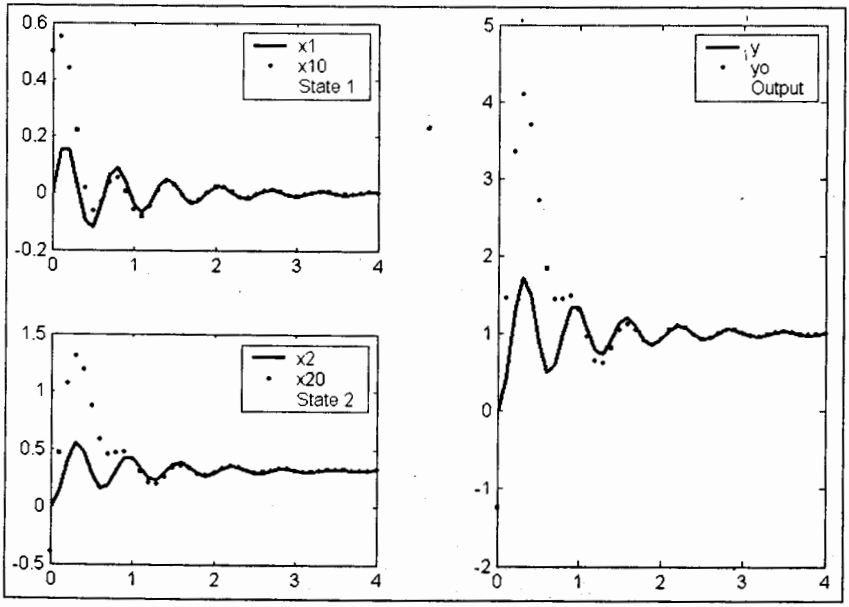


Hình 3.28 Mô hình est của khâu quan sát trạng thái

Mô hình khâu quan sát *est*, tìm thấy nhờ lệnh `estim` có các đầu vào **u**, **y** và các đầu ra $\hat{\mathbf{y}}, \hat{\mathbf{x}}$ được minh họa bởi hình 3.28. Ví dụ sau đây sẽ làm sáng tỏ thêm hiệu lực của các lệnh vừa được giới thiệu: Từ các điểm cực của đối tượng *sys* ta tìm các điểm cực `polo` của khâu quan sát và sau đó dùng lệnh `place` để

tìm vector phản hồi trạng thái k. Với *k* ta tạo mô hình *est* của khâu quan sát, và cuối cùng cất đáp ứng bước nhẩy của *sys* trong *y*, là tín hiệu sẽ được sử dụng làm đầu vào thứ hai (hình 3.28) của *est*. Đáp ứng bước nhẩy của *est* thu được nhờ lệnh lsim với xo0 là giá trị ban đầu của khâu quan sát. Kết quả được minh họa ở hình 3.29.

```
>> sys = ss(tf(100,[1 2 100])) ;       % Khai báo sys
>> polo = 3*real(pole(sys)) + imag(pole(sys))/3*i ;
                                       % Đ.cực của khâu QS
>> [ pole(sys) polo ]
ans =
  -1.0000 + 9.9499i  -3.0000 + 3.3166i
  -1.0000 - 9.9499i  -3.0000 - 3.3166i
>> L = place(sys.a',sys.c',polo).'     % Tính ma trận L
place: ndigits= 15
L =
   -1.7600
    1.2800
>> est = estim(sys,L,1,1) ;           % Dựng khâu QS
>> t = 0:0.1:4 ;                      % Khai báo trục t
>> [y,t,x] = step(sys,t) ;            % Đáp ứng của sys
>> xo0 = [0.5 -0.4] ;
>> [yo,to,xo] = lsim(est,[ones(size(t)) y],t,xo0) ;
                                       % Tính đáp ứng của est
```



Hình 3.29 *Kết quả tính của khâu quan sát trạng thái* est

Sau khi thực hiện đoạn lệnh trên, mọi giá trị đáp ứng bước nhẩy của *sys* (đối tượng) được cất trong `[y,t,x]`, của *est* (khâu quan sát) trong `[yo,t0,x0]`. Để vẽ các đồ thị ở hình 3.29, minh họa kết quả của đoạn lệnh trên, ta lần lượt thực hiện tiếp chùm lệnh sau đây:

```
>> subplot (221)                    % Đồ thị trạng thái 1
>> plot(t,x(:,1),'b-',t,xo(:,1),'r.')
>> legend ('x1','x10','State 1')
>> subplot (223)                    % Đồ thị trạng thái 2
>> plot(t,x(:,2),'b-',t,xo(:,2),'r.')
>> legend ('x2','x20','State 2')
>> subplot (122)                    % Đồ thị đầu ra y
>> plot(t,y(:,1),'b-',t,yo(:,1),'r.')
>> legend ('y','yo','Output')
```

Thiết kế hệ thống điều khiển sử dụng khâu quan sát

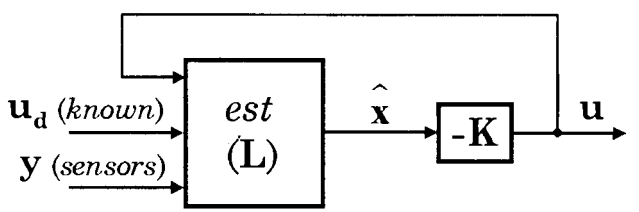
Sau khi đã tìm được các ma trận phản hồi **L** và **K**, ta có thể dựng hệ thống điều khiển sử dụng khâu quan sát trạng thái bằng lệnh `reg`.

rsys = `reg(`*sys*`,`*K*`,`*L*`)`

rsys = `reg(`*sys*`,`*K*`,`*L*`,`*sensors*`,`*known*`,`*controls*`)`

Trong lệnh trên, **K** và **L** là hai ma trận phản hồi được tính nhờ lệnh `acker` hay `place`, *sensors* là chỉ số của các biến ra đo được, *known* là chỉ số của các biến vào $\mathbf{u}_d$ mà ta biết, còn *controls* là biến vào của đối tượng. Phương trình mô tả đối tượng và hệ thống điều khiển trọn vẹn có dạng sau:

Đối tượng	**Hệ thống điều khiển**
$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$	$\dot{\hat{\mathbf{x}}} = \left(\mathbf{A} - \mathbf{LC} - \left(\mathbf{B} - \mathbf{LD}\right)\mathbf{K}\right)\hat{\mathbf{x}} + \mathbf{L}\mathbf{y}$ (3.66)
$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$	$\mathbf{u} = -\mathbf{K}\hat{\mathbf{x}}$



Hình 3.30 *Hệ thống điều khiển* `rsys` *sử dụng khâu quan sát trạng thái*

Hệ thống điều khiển trạng thái *rsys* được minh họa trong hình 3.30. Ví dụ dưới đây sẽ giúp ta hiểu rõ thêm cách sử dụng `reg`: Đối tượng *sys* cần được điều khiển trên không gian trạng thái. Để thực hiện ta lần lượt xác định các điểm

cực mong muốn của khâu quan sát và của khâu điều khiển trên cơ sở các điểm cực của đối tượng *sys*. Cần chú ý: Phải làm sao cho các điểm cực của khâu quan sát nhanh hơn các điểm cực của khâu điều khiển.

```
>> sys = ss(tf(100,[1 2 100])) ;
>> polo = 10*real(pole(sys)) + imag(pole(sys))/10*i ;
>> polc = 5*real(pole(sys)) + imag(pole(sys))/5*i ;
```

Trong đoạn lệnh trên, ta đã gán cho biến `polo` giá trị điểm cực của khâu quan sát và biến `polc` giá trị điểm cực của khâu điều khiển. Dễ dàng nhận thấy rằng: Phần thực của khâu quan sát (gấp 10 lần của đối tượng) lớn gấp 2 lần phần thực của khâu điều khiển (gấp 5 lần của đối tượng). Việc lựa chọn như vậy chính nhằm tạo động học lớn hơn cho khâu quan sát.

Sau đây ta dùng lệnh `place` để tính hai vector phản hồi **L** và **K**. Để tiện so sánh về sau, bên cạnh hệ thống điều khiển khép kín *rsys*, ta tạo thêm khâu quan sát *est*.

```
>> L = place(sys.a',sys.c',polo).'  % Tính vector L
place: ndigits= 15
L =
   -0.7002
    5.7600.
>> K = place(sys.a,sys.b,polc)      % Tính vector K
place: ndigits= 15
K =
    4.0000   -2.2200
>> est = estim(sys,L,1,1) ;         % Tìm khâu quan sát
>> rsys = reg(sys,K,L) ;            % Tìm khâu điều khiển
```

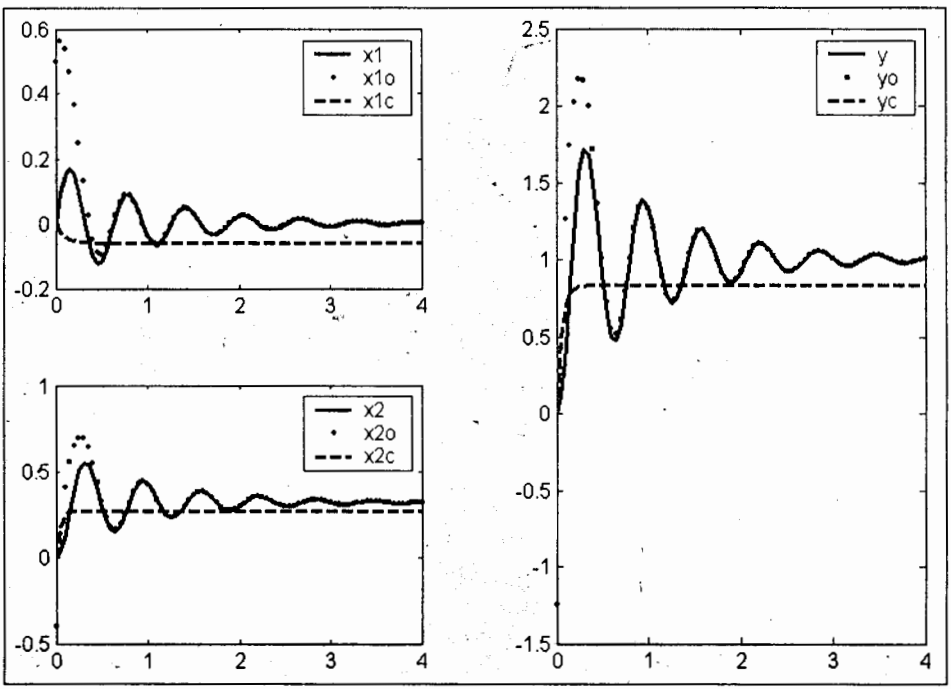
Tiếp theo ta lần lượt tính đáp ứng bước nhẩy của đối tượng *sys*, của khâu quan sát *est* (khi hệ chưa có ĐC) và của hệ thống *rsys* (khi hệ đã có ĐC).

```
>> t = 0:0.05:4 ;                   % Xác định trục t
>> [y,t,x] = step(sys,t) ;          % Đáp ứng của sys
>> xo0 = [0.5 -0.4] ;               % Giá trị ban đầu xo
>> [yo,to,xo] = lsim(est,[ones(size(t)) y],t,xo0) ;
                                    % Đáp ứng của est
>> xc0 = [0 0] ;                    % Giá trị ban đầu xc
>> [yc,tc,xc] = lsim(rsys,ones(size(t)),t,xc0) ;
                                    % Đáp ứng của rsys
```

Để so sánh các kết quả thu được, ta thực hiện tiếp chuỗi lệnh sau để vẽ đồ thị các đáp ứng bước nhẩy vừa tính được và in ra ở hình 3.31.

```
>> subplot (221)                    % Đáp ứng trạng thái 1
>> plot(t,x(:,1),'b-',t,xo(:,1),'r.',t,xc(:,1),'g--')
```

```
>> legend ('x1','x1o','x1c')
>> subplot (223)                          % Đáp ứng trạng thái 2
>> plot(t,x(:,2),'b-',t,xo(:,2),'r.',t,xc(:,2),'g--')
>> legend ('x2','x2o','x2c')
>> subplot (122)                          % Đáp ứng đầu ra
>> plot(t,y(:,1),'b-',t,yo(:,1),'r.',t,yc(:,1),'g--')
>> legend ('y','yo','yc')
```



Hình 3.31 *Đáp ứng bước nhẩy của hệ thống điều khiển sử dụng khâu quan sát trạng thái*

Qua hình 3.31 ta có thể rút ra hai nhận xét sau:

- Khâu quan sát có khả năng bám rất nhanh theo đặc điểm của đối tượng. Các giá trị quan sát được hội tụ nhanh và chính xác về giá trị thực tế.
- Khâu điều khiển có động học rất tốt, đưa các biến nhanh chóng tới một giá trị xác lập mà không xẩy ra quá điều khiển. Tuy nhiên, giá trị xác lập đầu ra không phải là 1 như ta trông đợi.

Sai lệch tĩnh kể trên có nguyên nhân ở sự thay đổi hàm truyền đạt của mạch vòng khép kín. Để khắc phục, thông thường ta phải bổ sung thêm một khâu phối hợp giá trị đặt đầu vào, nhằm đưa sai lệch tĩnh về không. Tuy nhiên, điều này lại dẫn đến sai lệch mới khi hệ bị nhiễu. Chính vì vậy, để khắc phục một cách triệt để, ta sẽ phải sử dụng một khâu điều khiển trạng thái có thêm khâu tích phân ở đầu vào.

Gán cực	
`acker(A,b,p)`	Gán cực cho hệ SISO theo công thức Ackermann
`place(A,B,p)`	Gán cực cho hệ MIMO
Tính khâu quan sát và điều khiển trạng thái	
`estim(sys,L)`	Tính khâu quan sát trạng thái
`reg(sys,K,L)`	Tính khâu điều khiển trạng thái

3.4.5 Thiết kế theo tiêu chuẩn tích phân tối ưu

Một phương pháp quan trọng hay được sử dụng để thiết kế hệ thống điều khiển trạng thái là phương pháp dựa trên cơ sở tiêu chuẩn tích phân tối ưu tuyến tính. Theo phương pháp này, các tham số của khâu điều khiển được chọn *xuất phát từ nỗ lực tìm cực tiểu cho một hàm chất lượng* (hàm mục tiêu), *chứ không xuất phát từ một dạng đáp ứng cho trước* (là bản chất của phương pháp gán cực).

Khi cần ĐC một đối tượng LTI trên không gian trạng thái, được mô tả bởi:

Mô hình liên tục	**Mô hình gián đoạn**	
$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$	$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$	(3.67)
$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$	$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k$	

ta sử dụng khâu phản hồi trạng thái:

Mô hình liên tục	**Mô hình gián đoạn**	
$\mathbf{u} = -\mathbf{K}\mathbf{x}$	$\mathbf{u}_k = -\mathbf{K}\mathbf{x}_k$	(3.68)

với ma trận phản hồi **K**, được thiết kế sao cho hàm chất lượng dưới đây là tối thiểu:

$$J(\mathbf{x},\mathbf{u}) = \int_{t=0}^{\infty} \left(\mathbf{x}^T\mathbf{Q}\,\mathbf{x} + \mathbf{u}^T\mathbf{R}\,\mathbf{u} + 2\mathbf{x}^T\mathbf{N}\,\mathbf{u}\right) dt \tag{3.69}$$

Bài toán đặt ra cho (3.69) sẽ quy tụ về việc giải phương trình đại số Riccati đối với **S** (sử dụng lệnh `care` hay `dare` của *Control System Toolbox*).

$$\mathbf{0} = \mathbf{A}^T\mathbf{S} + \mathbf{S}\mathbf{A} - (\mathbf{S}\mathbf{B} + \mathbf{N})\mathbf{R}^{-1}\left(\mathbf{B}^T\mathbf{S} + \mathbf{N}^T\right) + \mathbf{Q} \tag{3.70}$$

Sau khi tính **S**, ma trận phản hồi trạng thái **K** có dạng:

$$\mathbf{K} = \mathbf{R}^{-1}\left(\mathbf{B}^T\mathbf{S} + \mathbf{N}^T\right) \tag{3.71}$$

Khi đối tượng được mô tả bởi mô hình gián đoạn về thời gian, hàm chất lượng được viết theo công thức sau:

$$J(\mathbf{x},\mathbf{u}) = \sum_{n=1}^{\infty} \left(\mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k + 2\mathbf{x}_k^T \mathbf{N} \mathbf{u}_k \right) \tag{3.72}$$

với phương trình và công thức tính tương tự:

$$\mathbf{0} = \mathbf{A}^T\mathbf{S}\mathbf{A} - \mathbf{S} - \left(\mathbf{A}^T\mathbf{S}\mathbf{B} + \mathbf{N}\right)\left(\mathbf{B}^T\mathbf{S}\mathbf{B} + \mathbf{R}\right)^{-1}\left(\mathbf{B}^T\mathbf{S}\mathbf{A} + \mathbf{N}^T\right) + \mathbf{Q} \tag{3.73}$$

$$\mathbf{K} = \left(\mathbf{B}^T\mathbf{S}\mathbf{B} + \mathbf{R}\right)^{-1}\left(\mathbf{B}^T\mathbf{S}\mathbf{A} + \mathbf{N}^T\right) \tag{3.74}$$

Trong các công thức trên, **Q** là ma trận trọng lượng của các biến trạng thái, **R** là ma trận trọng lượng của các biến đầu vào. Ma trận **N** cho phép ta xét đến các tương tác giữa các biến vào và các biến trạng thái, đặc biệt quan trọng khi ma trận liên thông $\mathbf{D} \neq \mathbf{0}$. Tất cả các thành phần không điều khiển được của hệ phải bảo đảm ổn định tiệm cận.

Tính ma trận phản hồi K

Control System Toolbox cung cấp cho ta 4 lệnh để tìm ma trận phản hồi **K** theo tiêu chuẩn tích phân tối ưu tuyến tính.

```
[K,S,e]  = lqr(A,B,Q,R[,N])
[K,S,e]  = dlqr(A,B,Q,R[,N])
[Kd,S,e] = lqrd(A,B,Q,R[,N],Ts)
[K,S,e]  = lqry(sys,Q,R[,N])
```

Khi sử dụng các lệnh trên, ta luôn bắt buộc phải khai báo đầy đủ các ma trận hệ thống **A**, **B** hay *sys* (lệnh `lqry`) và hai ma trận trọng lượng **Q**, **R**. Ma trận **N** là tham số tuỳ chọn và được đặt mặc định bằng không. Đối với lệnh `lqrd` ta còn phải khai báo thêm chu kỳ trích mẫu *Ts*.

Kết quả sử dụng lệnh là ma trận phản hồi trạng thái gán cho biến *K*, lời giải của phương trình Riccati gán cho biến *S* và `e = eig(A-B*K)` nhận các giá trị riêng của hệ thống đã có phản hồi (vòng ĐC đã khép kín). Lệnh `lqr` có tác dụng tính ma trận phản hồi *K* cho hệ liên tục trên cơ sở các công thức (3.69) - (3.71). Lệnh `dlqr` có tác dụng tương tự nhưng đối với hệ gián đoạn theo (3.72) - (3.74). Ngược lại, lệnh `lqrd` tìm ma trận *Kd* của khâu điều khiển gián đoạn (*Digital Controller*) cho đối tượng liên tục tương ứng, được trích mẫu với chu kỳ *Ts* theo phương pháp ZOH (xem mục 3.2.8).

Trong trường hợp, thay vì nhằm vào các biến trạng thái, ta muốn đặt mục tiêu tối ưu vào các biến ra, lệnh `lqry` có tác dụng giúp ta tìm cực tiểu cho hàm chất lượng sau đây:

$$J(\mathbf{y},\mathbf{u}) = \int_{t=0}^{\infty} \left(\mathbf{y}^T\mathbf{Q}\mathbf{y} + \mathbf{u}^T\mathbf{R}\mathbf{u} + 2\mathbf{y}^T\mathbf{N}\mathbf{u} \right) dt \tag{3.75}$$

Khâu lọc Kalman

Lọc Kalman cho phép ta xét đến thực tế là: Cả hệ thống lẫn biến ra **y** đều có chứa một hàm lượng tạp âm **w**, **v** với kỳ vọng như sau:

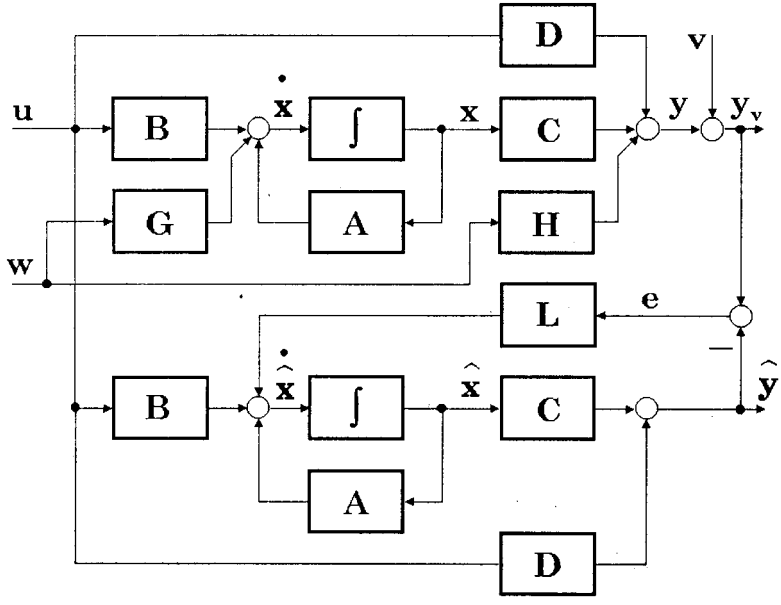
$$E\{\mathbf{w}\} = \mathbf{0}; \quad E\{\mathbf{v}\} = \mathbf{0} \tag{3.76}$$

$$E\{\mathbf{w}\,\mathbf{w}^T\} = \mathbf{Q}; \quad E\{\mathbf{v}\,\mathbf{v}^T\} = \mathbf{R}; \quad E\{\mathbf{w}\,\mathbf{v}^T\} = \mathbf{N} \tag{3.77}$$

Mục tiêu tối ưu khi tính toán lọc Kalman lúc này là: Quan sát (tính $\hat{\mathbf{x}}$) vector trạng thái $\mathbf{x}$ sao cho giá trị trung bình toàn phương của sai lệch trạng thái $\mathbf{x} - \hat{\mathbf{x}}$ là bé nhất.

$$\mathbf{P} = \lim_{t\to\infty} E\left\{\left(\mathbf{x}-\hat{\mathbf{x}}\right)\left(\mathbf{x}-\hat{\mathbf{x}}\right)^T\right\} \overset{!}{=} \min \tag{3.78}$$

Hình 3.32 mô tả khâu lọc Kalman cho trường hợp hệ liên tục.



Hình 3.32 Khâu lọc Kalman

- Phương trình trạng thái của đối tượng:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\,\mathbf{x} + \mathbf{B}\,\mathbf{u} + \mathbf{G}\,\mathbf{w}\\ \mathbf{y}_v &= \mathbf{C}\,\mathbf{x} + \mathbf{D}\,\mathbf{u} + \mathbf{H}\,\mathbf{w} + \mathbf{v}\end{aligned} \tag{3.79}$$

- Phương trình trạng thái của khâu quan sát:

$$\begin{aligned}\dot{\hat{\mathbf{x}}} &= \mathbf{A}\,\hat{\mathbf{x}} + \mathbf{B}\,\mathbf{u} + \mathbf{L}\,\mathbf{e}\\ \hat{\mathbf{y}} &= \mathbf{C}\,\hat{\mathbf{x}} + \mathbf{D}\,\mathbf{u}\end{aligned} \tag{3.80}$$

- Sai lệch quan sát:

$$\mathbf{e} = \mathbf{y}_v - \hat{\mathbf{y}} \tag{3.81}$$

Ma trận **L** phản hồi sai lệch quan sát được tính bằng cách giải phương trình Riccati. Trong trường hợp hệ gián đoạn ta còn cần đến ma trận **M**, dùng để tính hiệu đính lại kết quả ước lượng của chu kỳ trích mẫu trước đó. Phần trình bầy liên quan tới lý luận, chỉ nhằm giúp ta hiểu hiệu lực của các lệnh, được dừng tại đây. Bạn đọc nào quan tâm sẽ buộc phải tham khảo thêm các sách chuyên sâu và *User's Guide* của *Control System Toolbox*.

Thiết kế khâu lọc Kalman

Để thiết kế khâu lọc Kalman ta có thể sử dụng hai lệnh `kalman` và `kalmd`. Lệnh `kalman` tính khâu lọc Kalman liên tục từ mô hình *sys* của đối tượng. Khi đã thiết kế xong một khâu lọc liên tục và việc thử nghiệm đã đưa lại kết quả tốt, ta có thể sử dụng `kalmd` để tạo nên một khậu lọc Kalman gián đoạn tương ứng với chu kỳ trích mẫu *Ts*.

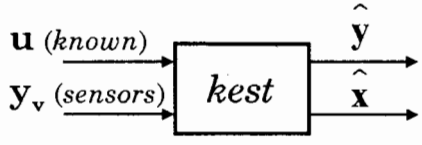
```
[kest,L,P] = kalman(sys,Qn,Rn[,Nn,sensors,known])
[kest,L,P,M,Z] = kalman(sys,Qn,Rn,Nn)
[kest,L,P,M,Z] = kalmd(sys,Qn,Rn,Ts)
```

Đối tượng *sys* có cấu trúc được mô tả bởi phương trình (3.79):

```
sys.a = A     sys.b = [ B G ]
sys.c = C     sys.d = [ D H ]
```

Bắt buộc phải được khai báo là các ma trận phương sai *Qn* và *Rn*, mô tả đặc điểm của tạp âm hệ thống và tạp âm đo lường. Ma trận *Nn* được đặt mặc định bằng không. Hai vector *sensors* và *known* chứa chỉ số của các biến ra đo được và các đầu vào ta biết. Khi dùng `kalmd` ta phải khai báo thêm *Ts*.

Kết quả tính *kest* (thu được khi sử dụng các lệnh trên) chính là mô hình trạng thái của khâu lọc Kalman (dạng SS) với các đầu vào/ra như hình 3.33. Ma trận *L* chính là ma trận **L** (còn gọi là ma trận khuếch đại Kalman) phản hồi sai lệch quan sát. *P* là ma trận phương sai của sai lệch tĩnh. Trong trường hợp gián đoạn ta còn thu được thêm ma trận *M* và ma trận *Z* phương sai của sai lệch điều khiển.



Hình 3.33 *Mô hình kest của khâu lọc Kalman*

Ví dụ sau đây sẽ minh họa quá trình thiết kế khá phong phú khâu lọc Kalman cho đối tượng bị nhiễu bởi tạp âm hệ thống và tạp âm đo lường. Trước hết, ta định nghĩa một khâu tỷ lệ có quán tính bậc 2 có tên là `object` (khâu PT_2) bị nhiễu hệ thống `w`.

```
>> sys = ss(tf(1000,[1 10 1000])) ;
```

```
>> object = ss (sys.a,[sys.b sys.b],sys.c,sys.d) ;
>> object.inputname = {'u' 'w'} ;
>> object.outputname = {'y'} ;
```

Tiếp theo ta gán cho hai phương sai (trong ví dụ này chỉ là vô hướng, scalar) Qn và Rn giá trị 1 và sử dụng lệnh kalman để tạo mô hình kest của khâu lọc Kalman:

```
>> Qn = 1 ; Rn = 1 ;
>> [kest,L,P] = kalman (object,Qn,Rn) ;
```

Để kiểm tra đáp ứng của hệ khi bị nhiễu ta tạo thêm hệ noise với ba biến vào là u, w và v, trong đó v chỉ ảnh hưởng trực tiếp ở đầu ra chứ không nhiễu vào hệ noise:

```
>> noise = ss(sys.a,[sys.b sys.b 0*sys.b],...
            [sys.c ; sys.c],[0 0 0 ; 0 0 1]) ;
>> noise.inputname = {'u' 'w' 'v'} ;
>> noise.outputname = {'y' 'yv'} ;
```

Cuối cùng, để mô phỏng bằng lệnh lsim ta còn phải tạo tín hiệu đầu vào u và hai tín hiệu tạp âm w và v. Tín hiệu u có dạng hàm xung vuông với biên độ là 2 và chu kỳ là 3:

```
>> [u,tt] = gensig ('square',3,6,0.01) ;
>> u = 2*u ;
>> randn('seed',0) ;
>> w = sqrt(Qn) * randn(length(tt),1) ;
>> v = sqrt(Rn) * randn(length(tt),1) ;
```

Quá trình tính đáp ứng đầu ra (mô phỏng) bắt đầu bằng việc dùng lệnh lsim để tính đáp ứng ra yideal khi không có tạp âm, tiếp theo là đáp ứng ynoise khi đối tượng bị nhiễu bởi w và v. Cuối cùng, khâu quan sát được nuôi bởi tín hiệu đầu vào u và tín hiệu đầu ra bị nhiễu ynoise(:,2) (chính là yv), tạo nên tín hiệu ra yestim:

```
>> [yideal,t] = lsim (noise,[u,0*w,0*v],tt) ;
>> [ynoise,t] = lsim (noise,[u,w,v],tt) ;
>> [yestim,t] = lsim (kest,[u,ynoise(:,2)],t) ;
```

Các lệnh gán dưới đây chỉ nhằm làm đơn giản phần viết lệnh vẽ đồ thị:

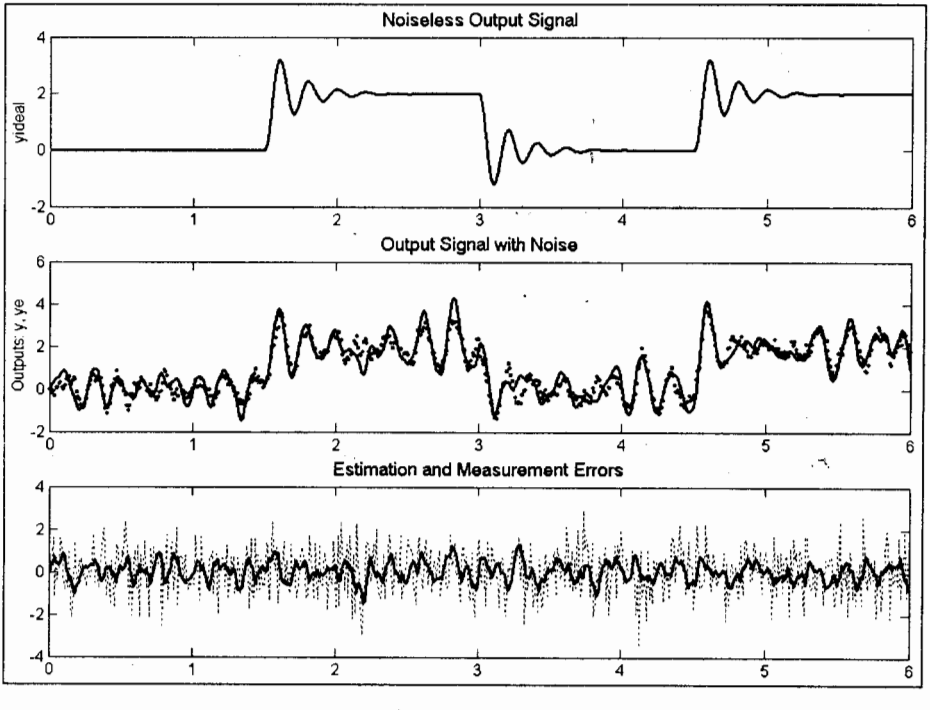
```
>> y = ynoise(:,1) ;                    % Đầu ra lý tưởng
>> yv = ynoise(:,2) ;                   % Đầu ra bị nhiễu
>> ye = yestim(:,1) ;                   % Đầu ra lọc Kalman
```

Để có thể đánh giá được chất lượng của các tín hiệu, ta định nghĩa thêm biến sai số đo lường `emeas` và biến sai số quan sát `eobs`, sau đó tính phương sai của chúng:

```
>> emeas = y - yv ;
>> eobs = y - ye ;
>> [ cov(emeas) cov(eobs) ]
ans =
   1.09988981779003   0.19287914850951
```

Bước cuối cùng của ví dụ là vẽ đồ thị của các đáp ứng trong hình 3.34:

```
>> figure (1)
>> subplot(311) , plot (t,yideal(:,1))
>> title ('Noiseless Output Signal')
>> ylabel ('yideal')
>> subplot(312) , plot (t,y,'b-',t,ye,'r.')
>> title ('Output Signal with Noise')
>> ylabel ('Outputs: y, ye')
>> subplot(313) , plot (t,emeas,':',t,eobs,'k-')
>> title ('Estimation and Measurement Errors')
```



Hình 3.34 *Khâu lọc Kalman sử dụng để quan sát trạng thái của hệ bị nhiễu*

Thiết kế theo tiêu chuẩn tích phân tối ưu sử dụng lọc Kalman

Tương tự lệnh `reg`, ta có thể sử dụng lệnh `lqgreg` để thiết kế khâu điều khiển trạng thái theo tiêu chuẩn tích phân tối ưu, sử dụng lọc Kalman để quan sát trạng thái. Tham số chuyển cho lệnh chính là phương trình hệ thống của lọc Kalman *kest* và ma trận phản hồi trạng thái *K*, tìm được từ trước bằng `kalman` và một trong các lệnh `lqr`, `dlqr`, `lqrd` hay `lqry`.

rlqg = `lqgreg` (*kest*, *K*)

rlqg = `lqgreg` (*kest*, *K*, *controls*)

Nếu ta khai báo thêm vector chỉ số *controls*, khâu lọc Kalman sẽ nhận thêm các đầu vào $\mathbf{u}_d$, không có tác dụng điều khiển trực tiếp đối tượng.

Phương trình mô tả đối tượng và hệ thống điều khiển có dạng sau đây:

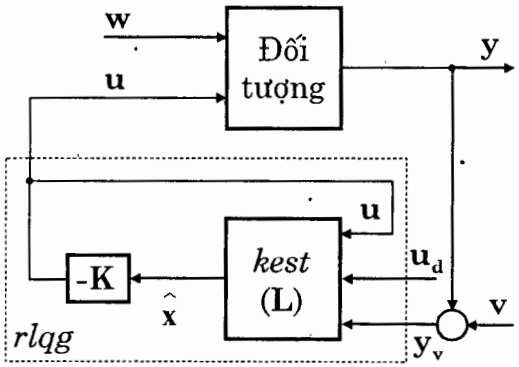
Đối tượng

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{G}\mathbf{w}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{H}\mathbf{w} + \mathbf{v}$$

Hệ thống điều khiển

$$\dot{\hat{\mathbf{x}}} = \left(\mathbf{A} - \mathbf{L}\mathbf{C} - (\mathbf{B} - \mathbf{L}\mathbf{D})\mathbf{K}\right)\hat{\mathbf{x}} + \mathbf{L}\mathbf{y}_v \qquad (3.82)$$
$$\mathbf{u} = -\mathbf{K}\hat{\mathbf{x}}$$

Hệ thống *rlqg* được ghép vào vòng điều khiển trạng thái như hình 3.35.



Hình 3.35 *Vòng điều chỉnh khép kín, thiết kế theo tiêu chuẩn tích phân tối ưu và sử dụng lọc Kalman để quan sát trạng thái*

Thiết kế theo tiêu chuẩn tích phân tối ưu	
`lqr` (*A*, *B*, *Q*, *R* [, *N*])	Khâu điều khiển cho hệ liên tục
`dlqr` (*A*, *B*, *Q*, *R* [, *N*])	Khâu điều khiển cho hệ gián đoạn
`lqrd` (*A*, *B*, *Q*, *R* [, *N*], *Ts*)	Khâu điều khiển gián đoạn cho hệ liên tục
`lqry` (*sys*, *Q*, *R* [, *N*])	Khâu điều khiển có trọng lượng đầu ra
`lqgreg` (*kest*, *K*)	Khâu điều khiển dùng lọc Kalman
	Lọc Kalman
`kalman` (*sys*, *Qn*, *Rn*, *Nn*)	Tính lọc Kalman
`kalmd` (*sys*, *Qn*, *Rn*, *Ts*)	Tính lọc Kalman gián đoạn cho hệ liên tục

3.5 Các vấn đề khi tính toán số

Thông thường, ngay từ khi mô hình hóa các vấn đề thực tiễn ta đã phải tiến hành đơn giản hóa, ví dụ: Tuyến tính hóa xung quanh một điểm công tác hay hạ bậc của đối tượng. Một mô hình toán thu được như vậy sẽ chỉ phản ánh đặc điểm của hệ đã đơn giản hóa, tuy nhiên tự thân nó là chính xác về toán học. Ngay việc biểu diễn, thực hiện tính toán bên trong máy tính cũng lại là động tác làm đơn giản hóa mô hình. Đơn giản hóa do máy tính gây nên có thể là: Phương pháp biểu diễn số, phạm vi giới hạn của số biểu diễn được, hay khả năng tính cũng như bộ nhớ rất hạn chế.

Ngành khoa học nghiên cứu và giải quyết các vấn đề trên là ngành toán số. Kết quả nghiên cứu là những thuật toán cho phép giải gần đúng các vấn đề toán của nhiều lĩnh vực rất khác nhau (lĩnh vực khoa học tự nhiên, hay kỹ thuật công nghệ ...) và đánh giá chất lượng phương pháp tính theo những quan điểm cũng khác nhau (ví dụ: theo đòi hỏi năng suất tính, theo nhu cầu bộ nhớ hay theo khả năng phân tích lỗi).

Vậy là, người sử dụng MATLAB phải đối đầu với câu hỏi: Anh ta sẽ phải chú ý điều gì khi thực hiện tính toán trên PC để có thể nhận được các kết quả tính chính xác và tin cậy cho vấn đề của mình.

Mục này sẽ chỉ trình bầy ngắn, có tính tổng kết các vấn đề xung quanh khả năng đánh giá, so sánh các dạng mô hình LTI như: mô hình trạng thái, mô hình truyền đạt hay mô hình điểm không - điểm cực.

3.5.1 Khái niệm lỗi

Để phân biệt khi tìm nguyên nhân gây lỗi, ta chia lỗi thành các loại sau:

- *Lỗi dữ liệu* hay *lỗi đầu vào*: Có cội nguồn từ các dữ liệu đầu vào không chuẩn xác, ví dụ: do bị tạp âm. Tác động loại này tới hệ thống là không tránh khỏi. Tuy nhiên, ta có thể và cần phải giảm thiểu ảnh hưởng của lỗi tới kết quả.
- *Lỗi phương pháp*: Xuất hiện do mô hình hoa vấn đề một cách chưa đầy đủ, đặc biệt khi gián đoạn hóa mô hình, hay do phép tính lặp (đệ quy, *iterative*) chỉ có số hữu hạn bước.
- *Lỗi làm tròn số*: Xuất phát từ phương pháp biểu diễn và phạm vi số của máy tính bị hạn chế. Đây là nguyên nhân không thể coi thường.

Gắn liền với khái niệm lỗi là khái niệm *sai số tuyệt đối* ε_k và *sai số tương đối* δ_k (khi $x_k \neq 0$) của giá trị xấp xỉ gần đúng $\tilde{\mathbf{x}} = \left(\tilde{x}_1, \ldots, \tilde{x}_n\right)$ so với giá trị chính xác $\mathbf{x} = \left(x_1, \ldots, x_n\right)$ với $1 \leq k \leq n$:

$$\varepsilon_k = x_k - \tilde{x}_k \qquad \delta_k = \frac{x_k - \tilde{x}_k}{x_k} \tag{3.83}$$

Vấn đề lỗi dữ liệu hay lỗi đầu vào được mô tả qua khái niệm làm *điều hòa* mô hình, lý giải ở mục 3.5.2. Lỗi phương pháp và lỗi làm tròn được trình bầy trong 3.5.3.

3.5.2 Khái niệm điều hòa mô hình

Khái niệm điều hòa (*conditioning*) chỉ vào ảnh hưởng của lỗi dữ liệu tới các kết quả tính toán. *Mô hình là điều hòa tốt* nếu các biến động nhỏ của dữ liệu đầu vào chỉ gây ra biến động rất nhỏ của kết quả tính. *Mô hình là điều hòa kém* nếu chỉ các biến động nhỏ của dữ liệu vào đã gây ra sai lệch lớn của kết quả tính (so với kết quả đúng). Dưới cách nhìn nhận như vậy ta có thể nói: Một mô hình điều hòa kém sẽ không thể đưa lại kết quả tính tốt, ngay cả khi ta sử dụng thuật tính hay và ổn định về mặt số.

Để đánh giá mức điều hòa của mô hình, ngành toán số đã định nghĩa *hệ số điều hòa của mô hình*, phản ánh mức độ ảnh hưởng của biến động của một giá trị dữ liệu vào tới một giá trị kết quả tính.

Việc giải hệ phương trình tuyến tính $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ và tính các giá trị riêng, vector riêng là một nhiệm vụ quan trọng của lĩnh vực mô phỏng số (*Numerical Simulation*). Vì hệ phương trình tuyến tính thường là điều hòa rất kém, để đánh giá được lỗi ta định nghĩa mức điều hòa của ma trận **A** như sau: Từ hệ phương trình tuyến tính bị nhiễu

$$\mathbf{A}\left(\mathbf{x}+\Delta\mathbf{x}\right)=\mathbf{b}+\Delta\mathbf{b} \tag{3.84}$$

ta thu được lỗi $\Delta\mathbf{x}=\mathbf{A}^{-1}\Delta\mathbf{b}$ với nhiễu $\Delta\mathbf{b}$. Lỗi đó có thể tính được bằng một chuẩn vector/ma trận bất kỳ:

$$\left\|\Delta\mathbf{x}\right\| \le \left\|\mathbf{A}^{-1}\right\| \left\|\Delta\mathbf{b}\right\| \tag{3.85}$$

Sai số tương đối được tính theo công thức:

$$\frac{\left\|\Delta\mathbf{x}\right\|}{\left\|\mathbf{x}\right\|} \le \left\|\mathbf{A}^{-1}\right\| \left\|\mathbf{A}\right\| \frac{\left\|\Delta\mathbf{b}\right\|}{\left\|\mathbf{b}\right\|} \tag{3.86}$$

Hệ số điều hòa của ma trận **A** được định nghĩa như sau:

$$\mathrm{cond}\left(\mathbf{A}\right) := \left\|\mathbf{A}^{-1}\right\| \left\|\mathbf{A}\right\| \tag{3.87}$$

Hệ số cond(**A**) có giá trị ≥1 đối với chuẩn tự nhiên. MATLAB cung cấp cho ta một số công cụ để kiểm tra mức điều hòa của ma trận như sau:

```
cond(A[,p])
condest(A)
condeig(A)
```

Lệnh `cond(A[,p])` tính hệ số điều hòa của ma trận **A** (nhằm mục đích đánh giá khả năng đảo **A**) với tham số p = 1 đối với chuẩn tự nhiên, p = 2 đối với chuẩn dạng phổ (giá trị mặc định), `'fro'` đối với chuẩn Frobenius và `inf` đối với chuẩn vô cùng. Lệnh `condest(A)` sử dụng dạng chuẩn tự nhiên và một

thuật toán khác với `cond(A,1)` để tính hệ số điều hòa, đánh giá khả năng nghịch đảo ma trận. Lệnh `condeig(A)` tính một vector hệ số điều hòa của các giá trị riêng thuộc ma trận **A**. Khi tính các giá trị riêng và để giải một hệ phương trình tuyến tính (cần phải đảo ma trận), ma trận hoàn toàn có thể có các đặc điểm điều hòa khác nhau.

Khi sử dụng các lệnh trên còn cần phải chú ý: Lệnh `log10(cond(A))` cho ta biết số vị trí sau dấu phẩy sẽ bị mất do lỗi làm tròn trong quá trình giải hệ phương trình tuyến tính. Một biểu hiện của đặc điểm điều hòa kém là: giá trị `cond(A)` lớn hơn nhiều so với `1/sqrt(eps)`.

Hệ số điều hòa	
`cond(A[,p])`	Hệ số điều hòa của ma trận **A** về khả năng đảo theo chuẩn *p*
`condest(A)`	Hệ số điều hòa của ma trận **A** về khả năng đảo theo chuẩn 1
`condeig(A)`	Hệ số điều hòa của ma trận **A** về giá trị riêng

3.5.3 Tính ổn định số học

Ngược lại với vấn đề điều hòa, tính ổn định số học *đánh giá ảnh hưởng của phương pháp giải* (thuật toán giải) tới kết quả.

Về nguyên tắc, các máy tính số (*Digital Computer*) đều sử dụng các phép tính số học với dấu phẩy động và tập các số có thể biểu diễn được hoàn toàn phụ thuộc vào độ dài hạn chế (*Word Length*) của *Bus* máy tính. Điều này dẫn đến việc thường xuyên phải làm tròn số, và có thể làm cho nhiều vị trí số bị xóa. Nếu điều này xẩy ra trong một quá trình tính nhiều bước, lỗi trên có thể được tích luỹ và dẫn đến tình trạng mất ổn định số học. Hiện tượng này có thể được gọi hiện tượng lan truyền lỗi.

Vì các phép tính số học cơ sở "cộng, trừ, nhân và chia" có sự nhạy cảm khác nhau đối với việc xóa kể trên, *trình tự thực hiện các phép tính số học trong cùng một thuật toán có thể có ảnh hưởng quyết định tới kết quả*. Đối với các phép tính số học giữa hai số ta có thể có nhận xét sau:

- Phép nhân và phép chia là hai phép tính "thiện".
- Phép cộng và phép trừ sẽ là "thiện", nếu hai toán hạng có cùng (phép cộng) hoặc khác dấu (phép trừ). Trường hợp này ứng với việc cộng trị tuyệt đối của hai toán hạng, sau đó nhân với dấu chung của kết quả.
- Phép cộng và phép trừ sẽ là "ác" và có thểdẫn đến mất ổn định số học, nếu hai toán hạng khác (phép cộng) hoặc có cùng dấu (phép trừ), đồng thời lại có giá trị xấp xỉ như nhau. Trường hợp này ứng với việc trừ trị tuyệt đối của hai toán hạng, sau đó nhân với dấu chung của kết quả.

3.5.4 Đánh giá mô hình LTI theo quan điểm tính số

Có thể nói ngay: Phương thức biểu diễn trên không gian trạng thái (mô hình LTI loại SS) là *thích hợp nhất đối với tính toán số*. Chính vì vậy, nhiều thuật toán của MATLAB và của *Control System Toolbox* được cài đặt trên cơ sở mô hình LTI-SS. Mô hình LTI dưới các dạng khác sẽ được hoặc tự động hoặc chủ động đảo dạng sang LTI-SS. Tuy nhiên, dù có lợi thế so với các loại mô hình khác, mô hình LTI-SS vẫn phải đảm bảo điều hòa tốt mới có thể cung cấp được các lời giải chính xác. Để thỏa mãn yêu cầu này, ta có thể dùng lệnh `ssbal` để tạo từ mô hình ban đầu một mô hình mới *giảm thiểu sự khác biệt về kích cỡ giữa các phần tử của vector/ma trận*.

Mô hình hàm truyền đạt (mô hình LTI-TF) chỉ thích hợp với các đối tượng có bậc thấp (<10) và thường có *hệ số điều hòa thấp khi mức chênh lệch giữa các hệ số là lớn*. Một vấn đề nữa của mô hình LTI-TF là: Khi đảo sang mô hình LTI-SS (dùng lệnh `ssbal`), ta thường thu được một ma trận điều hòa kém về các vector giá trị riêng, đặc biệt là các hệ bậc cao.

Nếu có thể, nên ưu tiên dùng mô hình điểm không - điểm cực (mô hình LTI-ZPK), bởi vì việc đảo sang mô hình trạng thái sẽ không làm sai lệch giá trị các điểm cực.

3.6 Tóm tắt nội dung chương 3

Sau khi đã nghiên cứu chương 3, người đọc cần nắm vững các nội dung sau đây:

1. Có thể mô tả hệ dưới những dạng mô hình nào của *Control System Toolbox* ?
2. Sự khác nhau giữa các dạng mô hình đó ?
3. Đối tượng LTI là gì và làm thế nào để kiểm tra hay thay đổi đặc điểm của chúng ?
4. Làm thế nào để tạo các hệ thống phức hợp từ mô hình LTI ?
5. Có bao nhiêu phương pháp chuyển mô hình LTI từ liên tục sang gián đoạn về thời gian ? Cần chú ý điều gì khi chuyển ?
6. Có thể kiểm tra những đặc điểm nào của mô hình ?
7. Làm thế nào để tính và biểu diễn vị trí điểm không - điểm cực ?
8. *Control System Toolbox* cung cấp lệnh có sẵn nào để tìm đáp ứng của hệ trên miền thời gian ? Cách sử dụng lệnh ra sao ?
9. Làm thế nào để khảo sát đáp ứng của hệ khi sử dụng một tín hiệu thử bất kỳ ?
10. Những lệnh nào giúp ta khảo sát mô hình LTI trên miền tần số ?
11. Làm thế nào để khảo sát tính quan sát được và tính điều khiển được của mô hình LTI ?
12. *Control System Toolbox* hỗ trợ những phương pháp nào để thiết kế khâu điều khiển ?

13. Làm thế nào để vẽ quỹ đạo điểm cực ? Nguyên tắc cơ bản sử dụng *SISO Design Tool* ?
14. Thiết kế khâu điều khiển theo phương pháp gán cực trong MATLAB ? Cần sử dụng những ma trận nào khi thiết kế khâu điều khiển và khâu quan sát trạng thái ? Cách tính chúng ?
15. Cách tính ma trận phản hồi của khâu điều khiển trạng thái thiết kế theo tiêu chuẩn tích phân tối ưu tuyến tính ?
16. Cần sử dụng khâu quan sát gì khi hệ bị nhiễu bởi tạp âm ?
17. Những điều cần chú ý khi biểu diễn và tính toán số trên máy tính ?

4 Optimization Toolbox: Công cụ tính toán tìm tối ưu

Trong khi đi tìm giải pháp cho các vấn đề kỹ thuật, đôi khi ta phải tiến hành tìm cực tiểu hay cực đại cho các hàm mục tiêu nhiều chiều. Việc tìm cực trị đó thường còn bị ràng buộc bởi các điều kiện phụ, mô tả dưới dạng phương trình hay bất phương trình, có nghĩa là: Ngay từ đầu, tập các lời giải đã bị hạn chế. Quá trình tính toán trên được gọi là quá trình tìm tối ưu dưới các điều kiện phụ (*constrained optimization*). Một phương pháp rất phổ biến khi đi tìm nghiệm gần đúng cho các hệ phương trình thiếu là phương pháp bình phương sai phân bé nhất (*least squares*). Phương pháp này còn hay được sử dụng để tìm đường cong xấp xỉ (*curve fitting*). Tuy nhiên, ứng dụng của phương pháp này không chỉ dừng ở đó. Ngoài ra, *least squares* còn có các biến dạng khác với điều kiện phụ, hay phương pháp *least squares* tuyến tính và phi tuyến. Một bài toán con hay gặp khi tìm tối ưu là tìm nghiệm của các hệ phương trình phi tuyến. Để giải bài toán tính toán tìm tối ưu, *Optimization Toolbox* cung cấp cho ta nhiều thuật toán khác nhau.

Mục đích của *Optimization Toolbox* không phải là phân tích hay tổng hợp (thiết kế) các thuật toán tìm tối ưu, mục đích của bộ công cụ này là: Làm sao sử dụng một cách tiện lợi các thuật toán đã biết. Đối với từng thuật toán, chương 4 sẽ chỉ mô tả những đặc điểm chính cần thiết, nếu muốn tìm hiểu kỹ hơn, bạn đọc bắt buộc phải tra cứu các tài liệu chuyên sâu tương ứng. Sau đây, khi đi tìm cực trị ta sẽ không phân biệt giữa cực tiểu và cực đại nữa, mà chỉ tập trung vào bài toán tìm cực tiểu. Tất cả các bài toán tìm cực đại với hàm mục tiêu $F(\mathbf{x})$ đều có thể được quy về bài toán tương ứng: Tìm cực tiểu cho hàm mục tiêu $-F(\mathbf{x})$. Tại đây, ta quy ước một số cách viết trong chương này: Các hàm vector sẽ được viết bằng chữ hoa to F, hàm vô hướng (*scalar*) được viết bằng chữ thường nhỏ f, vector viết bằng chữ thường đậm $\mathbf{v}$ và ma trận bằng chữ hoa đậm $\mathbf{M}$.

Để sử dụng *Optimization Toolbox*, hàm mục tiêu và các điều kiện phụ phải được khai báo dưới dạng hàm MATLAB (MATLAB `function`) viết thành *m-File* hay *inline object*. Việc xây dựng các hàm MATLAB đã được đề cập đến ở mục 1.7.2. Ngoài ra, việc sử dụng *inline object* thay cho hàm MATLAB (trong khuôn khổ MATLAB *scripts*) có thể rất có lợi.

4.1 Inline Objects

Thay vì phải cất hàm MATLAB lên đĩa cứng dưới dạng *m-File*, ta có thể tạo hàm trực tiếp dưới dạng *inline object* cất trong MATLAB *Workspace*. Về nguyên tắc, mọi hàm MATLAB đều có thể được dựng dưới dạng *inline object*, đặc biệt

thích hợp là các hàm nhỏ được gọi lặp lại nhiều lần trong các MATLAB *scripts*. Bằng cách ấy ta có thể làm cho các MATLAB *scripts* phức hợp trở nên ngắn gọn rất nhiều.

Ví dụ: Để định nghĩa hàm $f_1(x) = x^2 + x - 1$ ta phải gọi lệnh:

```
>> f1 = inline('x^2+x-1','x')
f1 =
     Inline function:
     f1(x) = x^2+x-1
```

Kết quả thu được là *inline object* mang tên `f1` cất trong MATLAB *Workspace*. Cùng trên dòng lệnh `inline` ta cũng có thể khai báo nhiều biến độc lập khác nhau. Hàm hai biến $f_2(x_1, x_2) = x_1^2 - x_2^2 + 3x_1x_2^2$ được tạo bởi:

```
>> f2 = inline('x1^2-x2^2+3*x1*x2^2','x1','x2')
f2 =
     Inline function:
     f2(x1,x2) = x1^2-x2^2+3*x1*x2^2
```

Việc tính hàm được thực hiện bằng cách gọi tại vị trí cần tính. Giá trị của các biến được viết tách bởi dấu phẩy.

```
>> y1 = f1(3), y2 = f2(3,2)
y1 =
    11
y2 =
    41
```

Tất cả các *inline objects* cất trong MATLAB *Workspace* có thể được hiển thị qua lệnh `whos`. Nội dung của *inline objects* được hỏi bằng cách gọi chính nó:

```
>> f2
f2 =
     Inline function:
     f2(x1,x2) = x1^2-x2^2+3*x1*x2^2
```

Bằng lệnh `inline` ta có thể khai báo các biểu thức MATLAB bất kỳ. Vì vậy, ta có thể tạo nên các hàm với vector giá trị.

Ví dụ: Hàm vector $f_{3vec}(x_1, x_2) = \left[x_1^2 - x_2^2 + 3x_1x_2^2 \quad x_1^2 + x_2 - 1\right]^T$ được định nghĩa bởi lệnh sau:

```
>>f3_vec=inline('[x1^2-x2^2+3*x1*x2^2;x1^2+x2-1]','x1','x2')
```

```
f3_vec =
     Inline function:
     f3_vec(x1,x2) = [x1^2-x2^2+3*x1*x2^2; x1^2+x2-1]
```

Để tính giá trị cụ thể của hàm vector ta gọi:

```
>> y3 = f3_vec(3,2)
y3 =
    41
    10
```

Nếu hàm nhận các vector làm biến và phải tính từng phần tử một, khi ấy thay vào vị trí các toán tử *, /, ^, ta có thể sử dụng các toán tử điểm .*, ./, .^ để giảm bớt số vòng lặp `for`. Nhờ vậy có thể sử dụng lệnh cho cả các biến vector cũng như các biến *scalar*.

Inline Objects	
`f = inline('function', 'var1', 'var2', ...)`	Tạo *inline object*

4.2 Điều khiển thuật toán

Tất cả các thuật toán của *Optimization Toolbox* sử dụng một biến có tên là `options` (thuộc loại *structure*) để điều khiển các bước tính toán. Nếu khi hỏi ta nhận được hai ngoặc vuông rỗng `[]`, có nghĩa là biến rỗng. Nếu khi gọi ta không khai báo tham số cụ thể, lúc bấy giờ các giá trị mặc định sẽ được chọn. Hai lệnh `optimget` và `optimset` được sử dụng để thay đổi biến `options`. Khi nhập lệnh:

```
>> options = optimset
options =
       ActivConstrTol: []
       DerivativeCheck: []
           Diagnostics: []
         DiffMaxChange: []
         DiffMinChange: []
               Display: []
     GoalsExactAchieve: []
           GradConstr: []
              GradObj: []
              Hessian: []
             HessMult: []
          HessPattern: []
           HessUpdate: []
             Jacobian: []
```

```
            JacobMult: []
          JacobPattern: []
            LargeScale: []
    LevenbergMarquardt: []
        LineSearchType: []
          MaxFunEvals: []
              MaxIter: []
            MaxPCGIter: []
            MaxSQPIter: []
         MeritFunction: []
             MinAbsMax: []
       Preconditioner: []
    PrecondBandWidth: []
    ShowStatusWindow: []
               TolCon: []
               TolFun: []
               TolPCG: []
                 TolX: []
            TypicalX: []
```

ta sẽ thu được một cấu trúc với các trường rỗng `[]`, trong đó ký hiệu `[]` có nghĩa là: trường đã tự động nhận tham số mặc định do *Optimization Toolbox* gán. Để thay đổi giá trị của từng trường cụ thể ta sử dụng lệnh:

```
options=optimset(oldopts,'param1',value1,'param2',value2,...)
```

Ví dụ: Để thay đổi các giá trị hiện tại của hai trường `TolX` và `MaxIter` mà không cần phải định nghĩa thêm biến `oldopts`, ta nhập:

```
>> options = optimset(options, 'TolX', 1e-6, 'MaxIter', 30);
```

Tất cả các trường của `options` (giống như các *structure* trong *Workplace*) có thể được hiển thị bằng cách gọi `options`. Để hiển thị từng trường riêng rẽ ta gọi `optimget` kèm theo tên của trường đó. Ví dụ:

```
>> par = optimget(options, 'TolX')
par =
  1.0000e-006
```

Đối với từng lệnh cụ thể của *Optimization Toolbox*, không phải bao giờ cũng cần sử dụng tất cả các trường. Để kiểm tra xem một lệnh cụ thể (giả sử có tên là: `command`) sử dụng những trường nào ta nhập `optimset('command')`. Khi gọi như vậy, giá trị của các trường được `command` sử dụng cũng sẽ được hiển thị. Một trường quan trọng, được hầu hết các lệnh sử dụng là trường `Display`. Các giá trị được phép đặt của `Display` bao gồm `'off'`, `'final'`, `'notify'`,

và 'iter'. Nếu là 'off', sẽ không thực hiện xuất; là 'notify', sẽ chỉ xuất nếu thuật toán không hội tụ; là 'final', sẽ chỉ xuất kết quả cuối cùng, còn nếu là 'iter', sẽ xuất thông tin cụ thể của từng bước lặp (*iteration steps*).

Có thể sử dụng biến options để điều khiển thuật toán của tất cả các lệnh trong *Optimization Toolbox*. Nếu cần sử dụng trình tự điều khiển thuật toán khác nhau cho các lệnh khác nhau, ta chỉ việc định nghĩa các biến điều khiển khác nhau, ví dụ: options1, options2, options3 ...

Đặt chế độ điều khiển thuật toán

`options = optimset`	Khai báo biến rỗng options
`options = optimset(oldopts,'par1',value1)`	Thay đổi giá trị của trường cụ thể
`optimset('command')`	Hiển thị các trường mà command sử dụng
`optimget(options, 'par1')`	Hiển thị giá trị của par1

4.3 Tìm điểm không

Có nhiều bài toán con (xuất hiện khi giải quyết vấn đề tìm tối ưu) đòi hỏi phải tìm nghiệm của hệ phương trình phi tuyến, hay xác định điểm không của hàm phi tuyến. Tạm gọi chung đó là vấn đề "*tìm điểm không*". Thông thường, chỉ có thể tìm điểm không của hàm phi tuyến theo phương pháp số (*numerical methods*), vì ta không thể có các công thức tìm nghiệm tổng quát. Việc giải phương trình phi tuyến luôn có thể quy về bài toán tìm điểm không, vì vậy, trong mục này ta sẽ chỉ đề cập đến vấn đề "*tìm điểm không*". Công cụ tìm điểm không trong MATLAB là hai lệnh: fzero dành cho hàm scalar và fsolve dành cho hàm vector phi tuyến cũng như hệ phương trình phi tuyến.

4.3.1 Hàm Scalar

Để nắm chắc vấn đề, ta sẽ bắt đầu bằng việc sử dụng lệnh fzero để tìm điểm không của một hàm scalar. Lệnh fzero cài đặt một thuật toán số tìm điểm không, xuất phát từ ý tưởng: Tại những điểm x thực nào hàm $f(x)$ sẽ đổi dấu. Điều này dẫn tới hai hệ quả: 1. các điểm không với giá trị phức sẽ không được quan tâm, và 2. các điểm tiếp xúc giữa $f(x)$ và trục x sẽ không phải là điểm không theo ý tưởng trên. fzero có thể được sử dụng theo hai cách sau đây:

```
nullst = fzero('function', x0, options)
[nullst,f_value] = fzero('function', x0, options)
```

Tham số *function* khai báo tên của hàm (tên của *m-File* hay của *inline objects*) cần tìm điểm không. Nếu $x0$ được khai báo dưới dạng scalar, `fzero` sử dụng $x0$ làm giá trị xuất phát, giá trị đó được tăng dần cho đến khi phát hiện ra hàm đổi dấu, khoảng giá trị tìm thực tế sẽ phụ thuộc vào câu trả lời: Tìm theo hướng nào sẽ phát hiện được điểm không trước. Nếu cần xác định rõ ràng khoảng giá trị tìm nghiệm, ta khai báo $x0$ dưới dạng vector có độ dài là 2, trong đó dấu của hàm sẽ thay đổi tại 2 giá trị biên của khoảng giá trị tìm (khi ấy sẽ tồn tại chắc chắn ít nhất là 1 điểm không).

Hãy lấy hàm

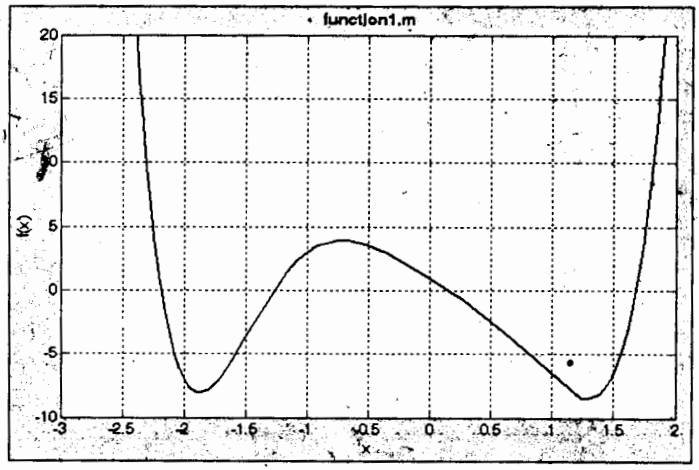
$$f(x) = x^6 + x^5 - 3x^3 - x^2 - 6x + 1 \tag{4.1}$$

để thử hiệu lực của lệnh `fzero`. Trước hết ta hãy soạn thảo một *script* (*m-File*) có tên `function1.m` sau đây và sau đó cất *script* vào một thư mục nào đó đã được khai báo trong *menue File/Set Path*.

```
%function1.m
function y = function1(x)
y = x.^6+x.^5-3*x.^4-x.^2-6*x+1;
```

Để tiện theo dõi, ta sử dụng đoạn lệnh sau để vẽ đồ thị của hàm (4.1) như ở hình 4.1.

```
>> fplot('x^6+x^5-3*x^4-x^2-6*x+1',[-3 2],'b-')
>> grid on
>> axis([-3,2,-10,20])
>> title ('function1.m')
>> xlabel ('x')
>> ylabel ('f(x)')
```



Hình 4.1 *Đồ thị của hàm* $f(x) = x^6 + x^5 - 3x^3 - x^2 - 6x + 1$

Vị trí điểm không ở lân cận $x \approx 0{,}2$ (xem hình 4.1) được phát hiện nhờ lệnh `fzero` xuất phát từ `x0=0` sau đây:

```
>> x0 = 0; [nullst fvalue] = fzero('function1', x0)
nullst =
    0.1620
fvalue =
     0
```

Nếu giá trị xuất phát được chọn là `x0=-0.8`, ta sẽ tìm được điểm không thứ hai nằm ở lân cận $x \approx -1{,}25$ (xem hình 4.1):

```
>> x0 = -0.8; [nullst fvalue] = fzero('function1', x0)
nullst =
   -1.2766
fvalue =
 -8.8818e-016
```

Như vậy là nhờ có đồ thị ở hình 4.1 ta đã xác định khoảng cần tìm và do đó đã dẫn đến đích một cách khá thuận lợi. Để xác định được cả 4 điểm không ta sẽ phải thực hiện một chuỗi lệnh như sau:

```
>> [nullst1 fvalue1] = fzero('function1', x01)
nullst1 =
   -2.1854
fvalue1 =
  2.1316e-014
>> [nullst2 fvalue2] = fzero('function1', x02)
nullst2 =
   -1.2766
fvalue2 =
 -8.8818e-016
>> [nullst3 fvalue3] = fzero('function1', x03)
nullst3 =
    0.1620
fvalue3 =
     0
>> [nullst4 fvalue4] = fzero('function1', x04)
nullst4 =
    1.6788
fvalue4 =
     0
```

Lệnh `fzero` bao giờ cũng tự động ngừng sau khi tìm thấy một điểm không và không tiếp tục tìm các điểm khác. Vì vậy, khi cần tìm nhiều điểm không khác nhau ta sẽ phải khai báo nhiều khoảng giá trị tìm khác nhau.

Nếu không cần tính tuyệt đối chính xác vị trí điểm không, ta có thể báo cho `fzero` biết độ chính xác mà lệnh phải bảo đảm. Độ chính xác được đặt thông qua tham số `TolX` của biến `options`, là tham số quyết định sai số tối đa theo hướng trục x. Nếu không trực tiếp khai báo, MATLAB sẽ đặt mặc định `TolX=eps`. Ta thử khảo sát ảnh hưởng của `TolX` thông qua giá trị điểm không thứ 4 `nullst4`. Nếu giảm độ chính xác xuống còn `1e-6`, `fzero` sẽ đưa ra kết quả kém chính xác hơn trước nhiều, tuy nhiên, số bước lặp khi tính lại giảm đi đáng kể:

```
>> tol=1e-3; options=optimset;
>> options=optimset(options,'TolX',tol);
>> [nullst4 fvalue4] = fzero('function1', x04, options)
nullst4 =
    1.6781
fvalue4 =
   -0.0372
```

Dễ dàng thấy rằng: Khi giảm độ chính xác, giá trị `fvalue4` của hàm sẽ không hoàn toàn chính xác bằng không nữa.

Vì `function1` là một đa thức của x, ta cũng có thể sử dụng một lệnh khác để tìm đồng thời tất cả các điểm không (kể cả các điểm có giá trị phức). Lệnh `roots` được cung cấp các hệ số của đa thức, khai báo dưới dạng vector, và sau khi tính sẽ cho kết quả là toàn bộ các điểm không:

```
>> poly = [1 1 -3 0 -1 -6 1]; nullst = roots(poly)
nullst =
  -2.1854
   1.6788
  -1.2766
   0.3106 + 1.1053i
   0.3106 - 1.1053i
   0.1620
```

Tuy vậy, đối với hàm phi tuyến, duy nhất lệnh `fzero` có thể giúp ta tới đích. Ví dụ, để giải phương trình: $\text{arctg}(x) = 0.5x$. Trước hết ta viết `function2.m`, và sau đó có thể tìm ba nghiệm chỉ bằng một chuỗi lệnh đơn giản:

```
%function2.m
function y = function2(x)
y = atan(x)-0.5.*x;

>> x01 = [-3 -1]; nullst1 = fzero('function2', x01)
nullst1 =
   -2.3311
```

```
>> x02 = [-1 1]; nullst2 = fzero('function2', x02)
nullst2 =
     0
>> x03 = [2 3]; nullst3 = fzero('function2', x03)
nullst3 =
    2.3311
```

4.3.2 Hàm vector và hệ phương trình

Để tìm nghiệm của hệ phương trình phi tuyến ta dùng lệnh `fsolve`, đối với hệ phương trình tuyến tính ta dùng toán tử \. Trước hết ta xét các hệ không thừa và cũng không thiếu phương trình, là các hệ hoặc không có nghiệm hoặc có một lượng hữu hạn nghiệm. Đối với hệ tuyến tính, điều đó có nghĩa là hệ hoặc vô nghiệm, hoặc có duy nhất một nghiệm. Hệ phương trình tuyến tính:

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 1 \\ 2x_1 + 3x_2 + 6x_3 = 3 \\ 3x_1 + 2x_2 + 7x_3 = 0 \end{cases} \tag{4.2}$$

có thể được viết dưới dạng ma trận:

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}; \quad \mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T \tag{4.3}$$

và sau đó giải bằng chuỗi lệnh dưới đây:

```
>> A = [1 2 3; 2 3 6; 3 2 7]
A =
     1     2     3
     2     3     6
     3     2     7
>> b = [1; 3; 0]
b =
     1
     3
     0
>> x = A\b
x =
   -7.5000
   -1.0000
    3.5000
```

Nói chung, lệnh `A\b` sẽ có đáp án tương tự như `inv(A)*b` (xem lệnh `inv` ở mục 1.2). Điểm khác ở đây là: Thay vì tính ma trận đảo, `A\b` đã giải hệ bằng phương pháp thế của Gauβ, là phương pháp có lợi hơn về mặt tính số trong nhiều trường hợp. `A\b` còn được gọi là phép chia ma trận từ bên trái. Khả năng giải hệ có thể được kiểm tra trước bằng cách tính định thức của ma trận **A**.

```
>> determinante = det(A)
determinante =
     2
```

Đối với hệ phương trình phi tuyến có thể xảy ra hai khả năng: Hoặc vô nghiệm, hoặc vô số nghiệm. Lệnh `fsolve` cung cấp cho ta nhiều thuật toán khác nhau để giải hệ với dạng $F(\mathbf{x}) = 0$, trong đó $F(\mathbf{x})$ là hàm vector của nhiều biến. Cú pháp của lệnh như sau:

x = fsolve('*function*', $x0$, *options*)
[x *fval*] = fsolve('*function*', $x0$, *options*)
[x *fval exitflag*] = fsolve('*function*', $x0$, *options*)

Bằng biến *options* ta có thể chọn các thuật toán khác nhau để giải hệ, có thể chủ động khai báo (ví dụ) ma trận Jacobi và sai số cần thiết. Để biết chi tiết về các thuật toán, bạn đọc nên xem trực tiếp *User's Guide* (*Optim_tb.pdf* trong thư mục ...\HELP\PDF_DOC\OPTIM). Nếu bỏ qua *options*, `fsolve` sẽ tự động sử dụng các khai báo mặc định. Hệ phương trình $F(\mathbf{x})$ cần được định nghĩa trước dưới dạng *m-File* hay *inline object*. $x0$ là giá trị xuất phát của thuật toán. x là kết quả tìm được và *fval* giữ giá trị của hàm thuộc giá trị x. *exitflag* cho ta biết điều kiện đã dẫn đến kết thúc thuật toán. *exitflag* > 0 nói rằng đã tìm được một nghiệm. Ngược lại, *exitflag* < 0 cho biết thuật toán đã không hội tụ. Nếu *exitflag* = 0, khi ấy ta biết là số bước lặp (*iteration steps*) đã đạt tới giá trị tối đa. Để minh họa ta hãy theo dõi hệ phương trình sau:

$$F(\mathbf{x}) = \begin{bmatrix} x_1^2 - 3x_2 + e^{-x_1} \\ \sin(x_1) + x_2 - e^{-x_2} \end{bmatrix} = 0 \tag{4.4}$$

Trước hết ta định nghĩa F bằng một hàm có tên `function3.m`. `fsolve` sẽ gọi hàm đó tại mỗi bước lặp.

```
%function3.m
function F = function3(x)
F = [x(1)^2-3*x(2)+exp(-x(1)), sin(x(1))+x(2)-exp(-x(2))];
```

Bây giờ ta có thể gọi chạy thuật toán `fsolve` với hàm `function3.m`. Dưới đây, trong phần khai báo *options* ta đã yêu cầu `fsolve` hiển thị trên màn hình kết quả của từng bước lặp. Giá trị xuất phát được chọn là `x0 = [1 1]`.

```
>> x0 = [1 1];
>> options=optimset('fsolve');options=optimset(options,'Display','iter');
>> [x, fval, exitflag] = fsolve('function3', x0, options)
                                        Norm of      First-order
 Iteration  Func-count     f(x)          step          optimality   CG-iterations
     1          4         4.83529            1            6.91           0
     2          7        0.244343       1.02472            1.74           1
     3         10    7.64713e-005      0.144197          0.0241           1
```

```
     4          13    1.81679e-009      0.00591701         0.00014           1
     5          16    8.51255e-020      1.888e-005          1e-009           1
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
x =
   0.48791614046545   0.28398883749062
fval =
  1.0e-009 *
   0.27238244992844  -0.10456246979373
exitflag =
     1
```

Bên cạnh nghiệm x và giá trị của hàm fval ta có exitflag=1, cho biết thuật toán đã hội tụ rất tốt. Việc hiển thị kết quả của từng bước tính cho phép ta theo dõi diễn biến của quá trình tìm nghiệm: Func-count cho biết số lần tính hàm, f(x) cho biết giá trị của hàm, Norm of step cho biết giá trị của bước được sử dụng. Vì thuật toán có sử dụng đến các đạo hàm bậc nhất (ma trận Jacobi), số lần tính hàm sẽ lớn hơn số bước lặp. Ma trận Jacobi được xác định (bằng số) ở lân cận của vị trí hiện tại **x**. Số lượng bước lặp phụ thuộc nhiều vào việc chọn giá trị xuất phát. Nếu ta chọn giá trị xuất phát xa vị trí nghiệm (chưa biết), số lượng bước lặp tăng và khối lượng tính sẽ trở nên lớn. Vì vậy, không bao giờ có thể xác định được trước chính xác số lượng phép tính sẽ dẫn tới đích. Thay đổi giá trị x0 của ví dụ trên ta sẽ chứng nghiệm được điều đó.

```
>> x0 = [10 10];
>> options=optimset('fsolve');options=optimset(options,'Display','iter');
>> [x, fval, exitflag] = fsolve('function3', x0, options)
```

Iteration	Func-count	f(x)	Norm of step	First-order optimality	CG-iterations
1	4	4989.42	1	1.39e+003	0
2	7	545.43	10	243	1
3	10	20.8371	2.16159	24.6	1
4	13	20.8371	4.39244	24.6	1
5	16	4.94664	1.09811	4.65	0
6	19	4.94664	2.19622	4.65	1
7	22	2.76698	0.549055	1.85	0
8	25	0.79441	1.09811	2.18	1
9	28	0.0864245	0.481581	1	1
10	31	5.26005e-005	0.106422	0.0252	1
11	34	2.68477e-012	0.00208816	3.03e-006	1
12	37	8.08509e-024	1.48047e-006	9.29e-012	1

```
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
x =
   0.48791614041662   0.28398883757459
```

```
fval =
  1.0e-011 *
   0.27967628213332  -0.05130340596793
exitflag =
     1
```

Đối với các hệ phương trình phi tuyến có thể tồn tại không phải chỉ một mà vô số nghiệm. Nếu ta biết ước chừng vị trí của nghiệm, ta nên chọn điểm xuất phát ở lân cận vị trí đó. Nghiệm thứ 2 của hệ (4.4) có thể được tìm thấy nhanh chóng khi ta chọn điểm xuất phát là `x0 = [-2 2]`.

```
>> x0 = [-2 2];
>> options=optimset('fsolve');options=optimset(options,'Display','iter');
>> [x, fval, exitflag] = fsolve('function3', x0, options)
                                        Norm of      First-order
 Iteration  Func-count     f(x)          step          optimality   CG-iterations
     1          4         29.9547             1          61.8            0
     2          7         2.63653      0.879018          10.7            1
     3         10       0.0877609      0.373429          1.42            1
     4         13     0.000731681      0.131078         0.116            1
     5         16    9.46893e-008     0.0141595        0.0013            1
     6         19    1.69041e-015    0.000163721      1.74e-007          1
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
x =
  -0.92286750724255   1.12272683994279
fval =
  1.0e-007 *
   0.40709509097070   0.05757234466941
exitflag =
     1
```

Qua các ví dụ trên, một điều trở nên rất rõ ràng là: *Thuật toán chỉ có thể tìm thấy một nghiệm* duy nhất, các nghiệm còn lại chỉ có thể tìm được sau khi ta đã thay đổi giá trị xuất phát một cách phù hợp. Vì vậy, nếu cần tìm một nghiệm đặc biệt nào đó, ta cần phải tìm cách thu được thông tin sơ bộ về vị trí của nó. Ví dụ: Bằng cách vẽ đồ thị, để rồi qua đó chọn các giá trị xuất phát.

Đối với các hệ bậc cao hay các thuật toán với khối lượng tính lớn, ta có thể giảm bớt số lần tính hàm (`Func-count`) bằng cách không để cho `fsolve` phải tự tính xấp xỉ bằng số các ma trận Jacobi nữa, mà cho phép `fsolve` sử dụng các ma trận Jacobi đã được *Optimization Toolbox* chuẩn bị sẵn. Để làm điều đó ta sẽ phải đặt tham số `options.Jacobian` về `'on'`. Ma trận Jacobi được định nghĩa như sau:

$$J_{ij} = \frac{\partial F_i}{\partial x_j} \tag{4.5}$$

Hàm mới `function4.m` sử dụng cho ví dụ (4.4) được mở rộng thêm ma trận Jacobi như sau:

```
%function4.m
function [F, J] = function4(x)
F = [x(1)^2-3*x(2)+exp(-x(1)), sin(x(1))+x(2)-exp(-x(2))];
J = [2*x(1)-exp(-x(1)), -3; cos(x(1)), 1+exp(-x(2)) ];
```

Chùm lệnh đã được sử dụng với `function3.m` được sửa lại với hiệu quả như sau:

```
>> x0 = [-2 2];
>> options = optimset('fsolve');
>> options = optimset(options,'Display','iter','Jacobian','on');
>> [x, fval, exitflag] = fsolve('function4', x0, options)
                                        Norm of      First-order
 Iteration  Func-count     f(x)          step          optimality   CG-iterations
     1          2         29.9547             1          61.8            0
     2          3         2.63653      0.879018          10.7            1
     3          4       0.0877609      0.373429          1.42            1
     4          5     0.000731682      0.131078          0.116           1
     5          6    9.46897e-008     0.0141595         0.0013          1
     6          7    1.69041e-015    0.000163722       1.74e-007        1
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
x =
  -0.92286750724359   1.12272683994438
fval =
  1.0e-007 *
   0.40709284387930   0.05758714283211
exitflag =
     1
```

Kết quả thu được rõ ràng là giống hệt như khi không khai báo ma trận Jacobi. Điểm khác dễ nhận thấy chỉ là: Số `Func-count` đã giảm đi một cách đáng kể.

Tìm điểm không	
[*nullst*,*f_value*] = fzero('*function*', *x*0, *options*)	Tìm điểm không cho hàm scalar
\ (slash)	Giải hệ pt. tuyến tính **Ax = b**
[*x fval exitflag*] = fsolve('*function*', *x*0, *options*)	Giải hệ pt. phi tuyến

4.4 Tìm cực tiểu cho hàm phi tuyến

Công cụ *Optimization Toolbox* cung cấp rất nhiều thuật toán giúp ta tìm cực tiểu của các hàm phi tuyến. Đó chính là các công cụ tối thiểu hóa (*minimization*) các hàm mục tiêu khi giải quyết bài toán tìm tối ưu. Mục này đề cập đến vấn đề tối thiểu hóa các hàm scalar phi tuyến với một hay nhiều biến và không kèm theo điều kiện phụ (*unconstrained optimization*).

Trường hợp đơn giản nhất là tìm cực tiểu của hàm scalar một biến trong một khoảng cho trước. Bài toán có thể giải quyết được bằng cách đặt đạo hàm bậc nhất của hàm bằng không và tìm nghiệm của phương trình mới thu được. Lúc này nhiệm vụ đặt ra được quy về bài toán tìm điểm không (bằng phương pháp số) của một hàm phi tuyến mà ta đã giải quyết. Ngoài ra, điểm tìm được theo cách đó không nhất thiết phải là một điểm cực trị, mà có thể là một điểm với diễn biến hàm dạng hình yên ngựa. Các thuật toán của *Optimization Toolbox* đều đã chuẩn bị sẵn sàng cho tất cả các tình huống đó.

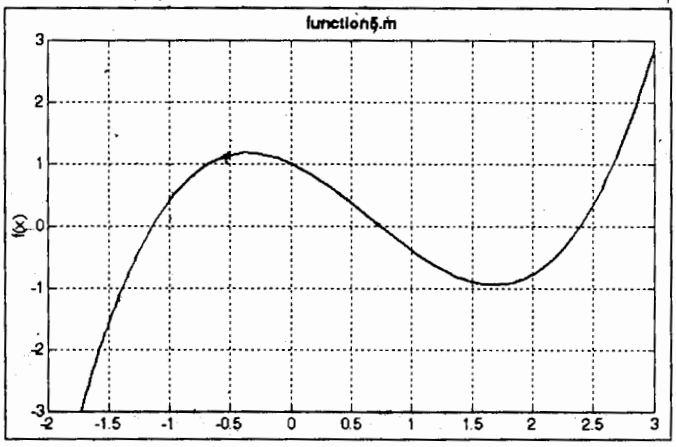
Lệnh có chức năng tìm cực tiểu của hàm scalar một biến trong khoảng cho trước là lệnh `fminbnd`. Thuật toán `fminbnd` cung cấp giá trị cực tiểu cục bộ trong dải giá trị $x_1 < x < x_2$. Lệnh được gọi như sau:

```
[x, fval, exitflag] = fminbnd('function', x1, x2, options)
```

Trong đó ba biến *fval*, *exitflag* và *options* chỉ là các biến tùy chọn, không bắt buộc. *function* phải là một hàm liên tục không có điểm gián đoạn đột biến. Ngoài ra ta cần lưu ý: Tốc độ hội tụ kết quả của thuật toán sẽ càng chậm thêm, nếu điểm cần tìm nằm càng gần một trong hai biên của dải giá trị.

Ta hãy thử tìm cực tiểu và cực đại của hàm sau:

$$f(x) = 0{,}5x^3 - x^2 - x + e^{0{,}1x} \tag{4.6}$$



Hình 4.2 *Đồ thị của hàm* $f(x) = 0{,}5x^3 - x^2 - x + e^{0{,}1x}$

Để tiện theo dõi ta đã vẽ đồ thị của (4.6) ở hình 4.2 bằng chùm lệnh dưới đây. Ta định nghĩa hàm (4.6) trong *script* có tên `function5.m`.

```
>> fplot('0.5*x^3-x^2-x+exp(0.1*x)',[-2 3],'b-')
>> grid on
>> axis([-2,3,-3,3])
>> title ('function5.m')
>> xlabel ('x')
>> ylabel ('f(x)')
```

Giá trị cực tiểu được tính bằng chùm lệnh sau đây:

```
>> options = optimset('fminbnd');
>> options = optimset(options, 'Display', 'iter');
>> x1 = 1; x2 = 2;
>> [x, fval, exitflag] = fminbnd('function5',x1,x2,options)
  Func-count     x          f(x)          Procedure
     1        1.38197    -0.823935        initial
     2        1.61803    -0.942405        golden
     3        1.76393    -0.938283        golden
     4         1.6808    -0.948653        parabolic
     5        1.68179    -0.948657        parabolic
     6        1.68268    -0.948658        parabolic
     7        1.68265    -0.948658        parabolic
     8        1.68262    -0.948658        parabolic
Optimization terminated successfully:
 the current x satisfies the termination criteria using
OPTIONS.TolX of 1.000000e-004
x =
   1.68264892760592
fval =
  -0.94865814609520
exitflag =
     1
```

Bên cạnh kết quả x=1.6826 ta thu được giá trị hàm fval tại vị trí cực tiểu, đồng thời *Optimization Toolbox* báo cho ta biết cụ thể thuật toán (cột Procedure) đã được sử dụng tại từng bước lặp. Vị trí cực đại của hàm có thể tìm được bằng cách tính cực tiểu của hàm function6.m = -1*function5.m. Đoạn lệnh sau đây tìm giá trị cực đại đó và không xuất dữ liệu của từng bước.

```
>> options = optimset(options, 'Display', 'final');
>> x1 = -2; x2 = 0;
>> [x, fval, exitflag] = fminbnd('function6',x1,x2,options)
Optimization terminated successfully:
 the current x satisfies the termination criteria using
OPTIONS.TolX of 1.000000e-004
x =
 -0.35645793682000
```

```
fval =
  -1.17173153613543
exitflag =
     1
```

Thuật toán cài đặt trong `fminbnd` được xây dựng trên cơ sở phương pháp tìm có tên "*Nhát cắt vàng*" và nội suy parabol. Lệnh `fminsearch` cũng có tác dụng như `fminbnd`, nhưng lại có thuật toán được cài đặt trên cơ sở phương pháp "*Simplex*". Phương pháp "*Simplex*" không đòi hỏi tính gradient (số hay giải tích), vì vậy có khả năng tìm được kết quả trông đợi cả khi hàm là không liên tục. Bạn đọc muốn tìm hiểu kỹ về hai phương pháp "*Nhát cắt vàng*" và "*Simplex*" sẽ buộc phải tìm đọc các tài liệu toán chuyên sâu, sách này sẽ không đi sâu vào giới thiệu hai phương pháp đó.

Để tìm cực tiểu của các hàm scalar nhiều biến và không có các điều kiện ràng buộc (*unconstrained optimization*) ta sử dụng lệnh `fminunc` với cách gọi như sau:

`[`*x, fval, exitflag*`] = fminunc('`*function*`',` *x*0, *options*`)`

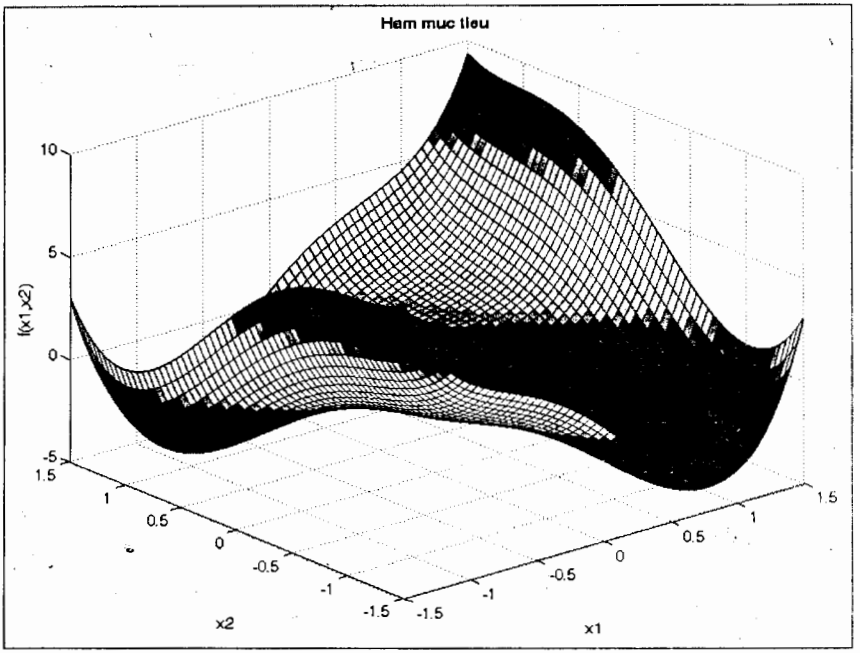
fval và *exitflag* là hai tham số tùy chọn. Nếu không có yêu cầu cụ thể, `fminunc` sẽ mặc định hiểu rằng nhiệm vụ tối ưu đặt ra thuộc phạm trù *Large Scale*. Nghĩa là, số lượng biến $x_1, x_2, \ldots, x_n$ của hàm là rất lớn. Để cho khối lượng tính số không trở nên quá lớn, ta có thể chuyển sang phạm trù *Medium Scale* nhờ lệnh `options = optimset(options, 'LargeScale', 'off')` (ngừng *Large Scale* trong *options*). Thuật toán tìm sử dụng phương pháp *Newton* có biến đổi, nếu cần hiểu sâu, bạn đọc có thể xem thêm trong *User's Guide*. Trình tự tìm cực tiểu được minh họa thông qua ví dụ với hàm:

$$f(x_1, x_2) = 2x_1^4 + x_2^4 - 2x_1^2 - 2x_2^2 + 4\sin(x_1 x_2) \tag{4.7}$$

Để tiện theo dõi, ta vẽ đồ thị 3-D của hàm (4.7) bằng chùm lệnh dưới đây. Xung quanh vấn đề vẽ đồ họa 3-D có phối cảnh bạn đọc xem mục 2.7.2, tại đó có giải thích rõ ý nghĩa của các lệnh:

```
>> x = -1.5:0.05:1.5; y = -1.5:0.05:1.5;
>> [X,Y] = meshgrid(x, y);
>> Z = 2*X.^4+Y.^4-2*X.^2-2*Y.^2+4*sin(X.*Y.*1);
>> figure
>> surf (X, Y, Z);
>> axis([-1.5,1.5,-1.5,1.5,-5,10])
>> title ('Ham muc tieu')
>> xlabel ('x1')
>> ylabel ('x2')
>> zlabel ('f(x1,x2)')
>> view (-40, 30)
```

Sau khi thực hiện chùm lệnh ta thu được kết quả ở hình 4.3.



Hình 4.3 *Đồ thị 3-D của hàm* $f(x_1, x_2) = 2x_1^4 + x_2^4 - 2x_1^2 - 2x_2^2 + 4\sin(x_1 x_2)$

Ta lại soạn thảo một *script* có tên `function7.m` với nội dung như dưới đây và cất vào một thư mục đã khai báo với MATLAB:

```
%function7.m
function F = function7(x)
F = [2*x(1)^4+x(2)^4-2*x(1)^2-2*x(2)^2 + 4*sin(x(1)*x(2))];
```

Vị trí cực tiểu đầu tiên nằm ở lân cận **x** = [1 –1], (điểm [x_1 x_2] = [1 –1]), được tìm thấy nhờ chùm lệnh sau đây:

```
>> options = optimset('fminunc');
>> options=optimset(options,'Display','iter','LargeScale','off');
>> x0 = [1 -1];
>> [x, fval] = fminunc('function7', x0, options)
                                                    Directional
 Iteration  Func-count     f(x)          Step-size      derivative
     1          2         -4.36588              1          -8.05
     2          8         -4.63276      0.0642678         -0.272
     3         14          -4.6476      0.0525359       3.21e-005
     4         21         -4.64761       0.965714      -1.16e-008
Optimization terminated successfully:
 Search direction less than 2*options.TolX
```

```
x =
    0.9039   -1.1732
fval =
   -4.6476
```

Điểm cực tiểu thứ hai ở gần **x** = [-1 1] sẽ dễ dàng được tìm thấy sau khi ta thay đổi giá trị xuất phát một cách thích hợp. Chùm lệnh cần thiết (không xuất từng bước lặp) là như sau:

```
>> options = optimset('fminunc');
>>options=optimset(options,'Display','final','LargeScale','off');
>> x0 = [-1 1];
>> [x, fval] = fminunc('function7', x0, options)
Optimization terminated successfully:
 Search direction less than 2*options.TolX
x =
   -0.9039    1.1732
fval =
   -4.6476
```

Độ chính xác của kết quả (đi theo là khối lượng tính) có thể được tăng lên bằng việc giảm `TolX` hay `TolFun` trong biến `options`. Ngoài ra, việc lựa chọn các giá trị xuất phát có ảnh hưởng quyết định tới khả năng hội tụ của thuật toán tìm tối ưu. Ví dụ: Ở lân cận của gốc tọa độ, hàm (4.7) có đồ thị tương đối phẳng, do đó đạo hàm bậc nhất và bậc hai của (4.7) có thể có giá trị rất nhỏ. Điều ấy dẫn đến, thuật toán tìm tối ưu trong phạm vi sai số cho phép (do `TolX` và `TolFun` quyết định) sẽ có thể đưa ra một kết quả, mà thực tế tại đó không hề tồn tại cực tiểu. Bạn đọc có thể tự kiểm chứng điều này bằng cách chọn $\mathbf{x}_0$ = [0,01 0,01]. Lưu ý này nhắc nhở ta không được phép tin một cách mù quáng vào kết quả do thuật toán đưa ra mà phải luôn có ý thức hiểu và sử dụng thuật toán một cách thích hợp.

4.5 Tìm cực tiểu khi có điều kiện phụ

Khi giải quyết các bài toán tìm tối ưu thường ta không chỉ quan tâm tới điểm cực tiểu của hàm mục tiêu. Hơn thế nữa, khi đi tìm cực tiểu, biến độc lập x còn phải thỏa mãn một số điều kiện phụ. Các điều kiện đó hoặc có thể do người sử dụng đặt ra, hoặc do các hạn chế về vật lý gây nên. Các điều kiện phụ thường được mô tả dưới dạng phương trình, hoặc bất phương trình, hoặc hỗn hợp cả hai. Thuật ngữ mô tả việc tìm tối ưu dưới điều kiện phụ là *constrained nonlinear optimization*.

Optimization Toolbox cung cấp cho ta lệnh `fmincon`, nhằm tìm cực tiểu của $f(\mathbf{x})$ dưới nhiều điều kiện phụ khác nhau. Các điều kiện được phân thành điều kiện phụ phi tuyến và điều kiện phụ tuyến tính. Thuật toán `fmincon` tìm:

$$\min_{\mathbf{x}} f(\mathbf{x}) \tag{4.8}$$

dưới các điều kiện:

$$\mathbf{c}(\mathbf{x}) \le \mathbf{0} \tag{4.9}$$

$$\mathbf{c}_{eq}(\mathbf{x}) = \mathbf{0} \tag{4.10}$$

$$\mathbf{A}\,\mathbf{x} \le \mathbf{b} \tag{4.11}$$

$$\mathbf{A}_{eq}\,\mathbf{x} = \mathbf{b}_{eq} \tag{4.12}$$

$$\mathbf{lb} \le \mathbf{x} \le \mathbf{ub} \tag{4.13}$$

Trong các công thức trên, $\mathbf{c}$ và $\mathbf{c}_{eq}$ là các hàm vector phi tuyến, $\mathbf{A}$ và $\mathbf{A}_{eq}$ là các ma trận hằng, $\mathbf{b}$ và $\mathbf{b}_{eq}$ là các vector hằng, còn $\mathbf{lb}$ và $\mathbf{ub}$ là các giới hạn dưới và giới hạn trên của vector kết quả $\mathbf{x}$. Lệnh gọi thuật toán có dạng sau đây:

```
[x,fval,exitflag]
= fmincon('function',x0,A,b,Aeq,beq,lb,ub,'nonlcon',options)
```

Khi gọi lệnh không nhất thiết phải khai báo tất cả các tham số. Tại vị trí của những tham số không cần sử dụng đến, ta chỉ cần thay vào đó dấu rỗng `[]`. *function* là hàm mục tiêu cần tìm cực tiểu, *x*0 là giá trị xuất phát. Các tham số từ *A* tới *ub* chính là các điều kiện phụ mô tả bởi các công thức (4.9) ... (4.13). Biến *options* có tác dụng điều khiển trình tự tính của thuật toán như ta đã biết. Các điều kiện được mô tả dưới dạng phương trình hay bất phương trình như (4.11), (4.12) được khai báo nhờ hàm *nonlcon*. Các điều kiện phi tuyến phải được khai báo đúng như các công thức ở trên, tức là phải kết thúc bởi "=0" hay "≤0". Các *script* viết dưới dạng *m-File* để mô tả điều kiện phụ phi tuyến phải có dạng tổng quát sau:

```
% Nonlinear Constrained Conditions
function [c, ceq] = nonlcon(x)
c   = ...                          % Nonlinear Non-Equation
ceq = ...                          % Nonlinear Equation
```

Trong *nonlcon*, điều kiện dưới dạng bất phương trình phải được viết trước, sau đó mới đến điều kiện dưới dạng phương trình. Ý nghĩa của *x*, *fval* và *exitflag* đã được giải thích ở trên. Tại đây cũng tồn tại vấn đề tìm tối ưu *Large Scale* (rất nhiều chiều) hay *Medium Scale* (nhiều chiều vừa phải) mà ta cần chú ý. Nếu tìm tối ưu theo *Large Scale*, hàm mục tiêu phải trả lại các giá trị Gradient sau mỗi bước tính hàm. Hàm mục tiêu *function* có dạng tổng quát sau:

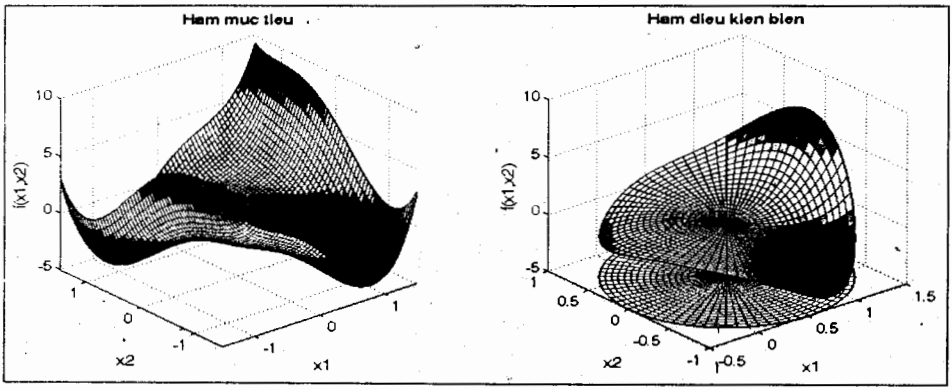
```
% Function for Optimizing
function [f, g] = functionx(x)
```

```
f = ...                          % Function for optimizing
g = [ ... ]                      % Gradient as vector of x
```

Nếu sử dụng thuật toán theo *Medium Scale*, ta phải khai báo chế độ đó bằng cách dùng lệnh `options = optimset(options, 'LargeScale', 'off')` để loại *Large Scale*. Trong trường hợp này sẽ không cần đến việc trả lại các giá trị Gradient. Đối với cả hai, hàm mục tiêu *function* và hàm điều kiện *nonlcon*, đều phải thỏa mãn đòi hỏi: Chúng phải là các hàm liên tục. Ngoài ra, tùy theo giá trị xuất phát $x0$, kết quả thu được chỉ là điểm cực tiểu cục bộ.

Để hiểu rõ chức năng của lệnh `fmincon` ta tiếp tục sử dụng hàm (4.7) với đồ thị ở hình 4.3. Ta bổ sung thêm điều kiện phụ, vector nghiệm chỉ được phép nằm trên đường tròn sau đây:

$$(x_1 - 0,5)^2 + x_2^2 = 1 \tag{4.14}$$



Hình 4.4 *Đồ thị 3-D của hàm mục tiêu* $f(x_1,x_2) = 2x_1^4 + x_2^4 - 2x_1^2 - 2x_2^2 + 4\sin(x_1x_2)$ *(hình trái) và hàm điều kiện phụ* $(x_1 - 0,5)^2 + x_2^2 = 1$ *chiếu vào hàm mục tiêu (hình phải)*

Hình 4.4 được tạo bởi chuỗi lệnh dưới đây. Dễ dàng nhận thấy: Chỉ bằng những lệnh cơ bản đã làm quen ở mục 2.7.2 ta đã có thể thực hiện đồ họa 3-D phức tạp một cách tương đối dễ dàng.

```
>> subplot(121);                  % Tạo ô hình trái
>> x1 = -1.5:0.05:1.5; y1 = -1.5:0.05:1.5;
                                  % Dải giá trị x1, x2
>> [X1,Y1] = meshgrid(x1, y1);    % Tạo ma trận giá trị
                                  % của các điểm trong dải
>> Z1 = 2*X1.^4+Y1.^4-2*X1.^2-2*Y1.^2+4*sin(X1.*Y1.*1);
>> surf (X1, Y1, Z1);            % Vẽ hàm mục tiêu
>> axis([-1.5,1.5,-1.5,1.5,-5,10])
>> title ('Ham muc tieu','FontSize',12)
```

```
>> xlabel ('x1','FontSize',12)
>> ylabel ('x2','FontSize',12)
>> zlabel ('f(x1,x2)','FontSize',12)
>> view (-40, 30)
>> subplot(122);                    % Tạo ô hình phải
>> [theta,radius] = meshgrid((0:5:360)*pi/180,0:0.05:1);
                                    % Ma trận giá trị của các
                                    % điểm trong đ.tròn r=1
>> [X2,Y2] = pol2cart(theta,radius);
                                    % Chuyển từ tọa độ cực
                                    % sang tọa độ Cartesian
>> X2 = X2+0.5;                     % Dịch tâm đường tròn
>> Z2 = 2*X2.^4+Y2.^4-2*X2.^2-2*Y2.^2+4*sin(X2.*Y2.*1);
>> surf (X2, Y2, Z2);               % Vẽ hàm mục tiêu bị chặn
                                    % trong đường tròn
>> axis([-0.5,1.5,-1,1,-5,10])
>> title ('Ham dieu kien bien','FontSize',12)
>> xlabel ('x1','FontSize',12)
>> ylabel ('x2','FontSize',12)
>> zlabel ('f(x1,x2)','FontSize',12)
>> view (-40, 30)
>> hold on
>> surf(X2, Y2, zeros(size(X2))-5,'FaceColor','w')
                                    % Vẽ hình chiếu xuống nền
                                    % và tô mầu trắng
>> hold off
```

Hình chiếu xuống nền (hình 4.4, bên phải) của phần mặt cong (4.7) bị giới hạn bởi điều kiện phụ (4.14) giúp ta có ngay được nhận xét: Phần mặt cong đó có hai điểm cực tiểu cục bộ ở gần $\mathbf{x} = [0,5\ -1]$ và $\mathbf{x} = [-0,2\ 1]$. Trước hết ta khai báo:

- *Hàm mục tiêu*

```
%function8.m
function [f, g] = function8(x)
f = [2*x(1)^4+x(2)^4-2*x(1)^2-2*x(2)^2 + 4*sin(x(1)*x(2))];
g = [];
```

- *Hàm điều kiện*

```
%nonlcon.m
function [c, ceq] = nonlcon(x)
c = [];
ceq = (x(1)-0.5)^2 + x(2)^2 - 1;
```

Điểm cực tiểu thứ nhất ở gần $\mathbf{x} = [0,5\ -1]$ sẽ được tìm thấy nhờ chùm lệnh:

```
>> options = optimset('fmincon');
>> options = optimset(options,'Display','iter','LargeScale',
'off');
>> x0 = [0.5 -1];
>> [x,fval] = fmincon('function8',x0,[],[],[],[],[],[],'nonlcon',
options)
                                  max                Directional
 Iter  F-count    f(x)          constraint  Step-size  derivative  Procedure
    1      3     -3.2927            0         1         -20.3
    2     10    -4.19884          0.3179     0.125       -1.11
    3     14    -4.25756         0.09838       1        -0.101
    4     18    -4.26616         0.01449       1         0.0204
    5     22    -4.24513       0.0001185       1        0.00018
    6     26    -4.24495       6.181e-009      1      8.54e-009      Hessian
                                                                    modified
Optimization terminated successfully:
 Magnitude of directional derivative in search direction
  less than 2*options.TolFun and maximum constraint violation
  is less than options.TolCon
Active Constraints:
     1
x =
    0.8523   -0.9359
fval =
   -4.2450
```

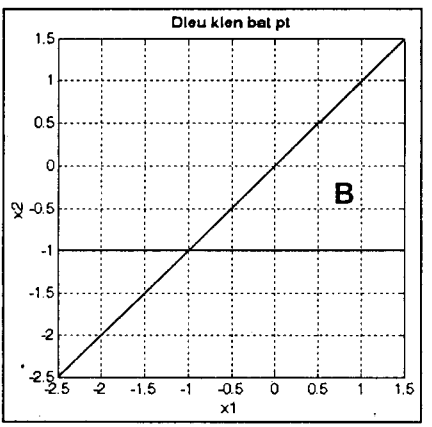
Trong khi tìm điểm cực tiểu ta đã ngắt chế độ *Large Scale*, vì vậy không cần khai báo Gradient (`g=[]`, xem `function8.m`). Điểm cực tiểu thứ 2 nằm lân cận**x** = [-0,2 1] (hình 4.4, bên phải) được tìm thấy nhờ chùm lệnh (đã bỏ qua việc xuất từng bước lặp):

```
>> options = optimset('fmincon');
>> options = optimset(options,'Display','final','LargeScale',
'off');
>> x0 = [-0.2 1];
>> [x,fval] = fmincon('function8',x0,[],[],[],[],[],[],'nonlcon',
options)
Optimization terminated successfully:
 Magnitude of directional derivative in search direction
  less than 2*options.TolFun and maximum constraint violation
  is less than options.TolCon
Active Constraints:
     1
x =
   -0.2868    0.6172
fval =
   -1.4721
```

Kết quả nhận được là vector **x** (vị trí điểm cực tiểu), điều kiện kết thúc tính toán và giá trị hàm tại điểm **x**. Ngoài ra ta còn biết thêm: Liệu có điều kiện phụ, hay điều kiện phụ nào đã có hiệu lực khi tìm cực tiểu? Câu trả lời `Active Constraints: 1` có nghĩa là: Điều kiện phụ thứ 1 có hiệu lực (trong ví dụ này cũng chỉ tồn tại 1 điều kiện (4.14)). Nếu các điểm cực tiểu cục bộ nằm trong một phạm vi (chứ không phải trên đường cong như ví dụ (4.14)), khi ấy điều kiện phụ được mô tả dưới dạng bất phương trình.

Lại sử dụng hàm (4.7) làm ví dụ chịu điều kiện phụ dưới dạng bất phương trình tuyến tính.

$$x_1 \geq x_2; \quad x_2 \geq -1 \tag{4.15}$$



Hình 4.5 Miền giới hạn (B) theo (4.15)

Ta dùng chùm lệnh sau để biểu diễn vùng giới hạn bởi các điều kiện phụ.

```
>>fplot('x1',[-2.5 1.5],'b-')
>>hold on
>>fplot('-1',[-2.5 1.5],'b-')
>>hold off
>>title ('Dieu kien bat pt',
        'FontSize',12)
>>xlabel ('x1','FontSize',12)
>>ylabel ('x2','FontSize',12)
>>axis([-2.5,1.5,-2.5,1.5])
>>axis square
>>grid on
```

Để biểu diễn điều kiện phụ hợp thức theo (4.11) ta phải định nghĩa ma trận **A** và vector **b** như sau:

$$\mathbf{A} = \begin{bmatrix} -1 & 1 \\ 0 & -1 \end{bmatrix}; \mathbf{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4.16}$$

A và **b** sẽ được trao cho thuật toán tìm cực tiểu. Việc tính toán được thực hiện bằng chùm lệnh:

```
>> A = [-1 1; 0 -1]; b = [0; 1];
>> options = optimset('fmincon');
>>options=optimset(options,'Display','final','LargeScale','off');
>> x0 = [1 0];
>> [x, fval] = fmincon('function7',x0,A,b,[],[],[],[],[],options)
Optimization terminated successfully:
 Magnitude of directional derivative in search direction
  less than 2*options.TolFun and maximum constraint violation
  is less than options.TolCon
```

```
Active Constraints:
     2
x =
    0.9135   -1.0000
fval =
   -4.4428
```

Từ kết quả thu được ta có thể thấy rõ là điều kiện phụ thứ 2 (`Active Constraints:2`) của (4.15) đã có hiệu lực khi tìm tối thiểu. Trong trường hợp đơn giản này thực ra ta đã có thể xuất phát từ đáp án $x_2 = -1$ để rút ra kết luận rằng điều kiện $x_2 \geq -1$ sẽ có hiệu lực. Kết quả tìm được nằm chính trên đường thẳng $x_2 = -1$.

Lệnh `fmincon` là một lệnh rất mạnh và có khả năng xét đồng thời nhiều bất phương trình hay phương trình điều kiện phụ. Tuy nhiên, cần phải lưu tâm để giành lại được một không gian nghiệm, tránh trường hợp các điều kiện phụ loại trừ lẫn nhau. Cũng tại đây cần phải nhắc lại: Như mọi thuật toán đã đề cập đến ở các mục trước, `fmincon` chỉ có thể tìm được một điểm tối thiểu ở lân cận điểm xuất phát.

Nếu hàm mục tiêu không phải là scalar (vô hướng) mà là hàm có giá trị vector, khi ấy ta đứng trước một *bài toán tối ưu với nhiều mục tiêu* (*multiobjective optimization problem*). Sẽ có nhiều hàm phải đồng thời thỏa mãn nhiều điều kiện tối ưu khác nhau (*goal attainment problem*). *Optimization Toolbox* có lệnh `fgoalattain` để giành cho trường hợp này. Một bài toán tiêu biểu đối với các hàm mục tiêu dạng vector là việc tối thiểu hóa khả năng xảy ra kết quả xấu nhất (*worst case*), tức là giảm khả năng xảy ra tối đa của từng thành phần riêng lẻ thuộc hàm mục tiêu. Bài toán tối ưu dạng đó thường được gọi là vấn đề *Minimax*, và để giải bài toán đó ta có thể dùng lệnh `fminimax` của *Optimization Toolbox*. Bạn đọc muốn tìm hiểu kỹ hai lệnh `fgoalattain` và `fminimax` sẽ phải xem *User's Guide* hạy *Online Help*.

Tìm cực tiểu khi có điều kiện phụ

`[x,fval]=fmincon('function',x0,A,b,Aeq,beq,lb,ub,'nonlcon',options)`
Tìm tối thiểu của một hàm nhiều biến chịu điều kiện phụ mô tả dưới dạng phương trình hay bất phương trình

`[x,fval]=fminimax('function',x0,A,b,Aeq,beq,lb,ub,'nonlcon',options)`
Giải bài toán *minimax*

`[x,fval]=fgoalattain('function',x0,'goal',weight,A,b,Aeq,beq,lb,...,ub,'nonlcon',options)`
Giải bài toán *goal attainment*

4.6 Phương pháp bình phương sai phân bé nhất

Khái niệm bình phương sai phân bé nhất (*least squares*, LQ) bao gồm nhiều phương pháp tìm tối ưu khác nhau nhưng đều có chung một mục tiêu: *Tìm tối thiểu của tổng các giá trị sai số bình phương*. Ta phân biệt giữa phương pháp LQ tuyến tính hay phi tuyến, và có hay không có điều kiện phụ. Vì LQ rất hay được sử dụng để tìm đa thức xấp xỉ (tìm đường cong xấp xỉ), trong tài liệu chuyên môn ta còn hay gặp khái niệm *curve fitting* nhằm vào ứng dụng này.

Phương pháp LQ tuyến tính không kèm theo điều kiện phụ có thể được sử dụng để giải một hệ phương trình tuyến tính thừa (số phương trình lớn hơn số ẩn). Hệ phương trình:

$$\mathbf{C}\mathbf{x} = \mathbf{d} \quad \Leftrightarrow \quad \mathbf{C}\mathbf{x} - \mathbf{d} = \mathbf{0}; \quad \mathbf{C} \in \mathbb{R}^{n \times m}, \mathbf{d} \in \mathbb{R}^{n \times 1} \tag{4.17}$$

là một hệ thừa khi $n > m$ và vì vậy không có nghiệm. Vì (4.17) không có nghiệm chính xác, ta có thể coi (4.17) là vấn đề tìm tối ưu, với bài toán tìm tối thiểu được diễn đạt bởi:

$$\min_{\mathbf{x}} \left\| \mathbf{C}\mathbf{x} - \mathbf{d} \right\|_2^2 \tag{4.18}$$

Trong (4.18), $\| \ \|_2^2$ là chuẩn toàn phương Euclid của vector. Nếu ta thay $F(\mathbf{x})$ = $\mathbf{Cx}$-$\mathbf{d}$ vào và tính chuẩn Euclid, ta thu được:

$$\min_{\mathbf{x}} \sum_{i=1}^{n} F_i(\mathbf{x})^2 \tag{4.19}$$

$F_i(\mathbf{x})$ là sai lệch giữa thành phần thứ i của (4.17) với nghiệm chính xác. Ta sẽ phải tìm vector $\mathbf{x}$, sao cho tổng bình phương sai phân (4.19) là bé nhất (là tối thiểu). Một ví dụ ứng dụng kinh điển là việc sử dụng phương pháp LQ tuyến tính để tìm một đa thức xấp xỉ cho các giá trị đo. Các giá trị đầu vào u_i được cất trong vector $\mathbf{u} = [u_1 \ldots u_n]$ và các giá trị đo được y_i cất trong vector $\mathbf{y} = [y_1 \ldots y_n]$. Quan hệ $y = f(u)$ cần được xấp xỉ bởi đa thức bậc m sau đây với $m < n$.

$$y_i = a_0 + a_1 u_i + a_2 u_i^2 + \cdots + a_m u_i^m \tag{4.20}$$

Mỗi cặp giá trị của điểm đo cần phải nằm rất gần đa thức cần tìm. Hệ phương trình tuyến tính mô tả vấn đề đặt ra có thể được quy về dạng tương tự với (4.17). Các ma trận sẽ có dạng tổng quát như sau:

$$\mathbf{C} = \begin{bmatrix} 1 & u_1 & u_1^2 & \cdots & u_1^m \\ 1 & u_2 & u_2^2 & \cdots & u_2^m \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & u_n & u_n^2 & \cdots & u_n^m \end{bmatrix}; \mathbf{x} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}; \mathbf{d} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} \tag{4.21}$$

Việc tìm nghiệm tối ưu của hệ phương trình thừa dạng (4.19) được thực hiện bởi lệnh \ (`slash`) của MATLAB. Vấn đề tối ưu (4.18) cũng có thể giải được theo phương pháp giải tích và sẽ luôn tường minh khi $n \geq m$, tức là khi số phương

trình lớn hơn số lượng các hệ số của đa thức. Ví dụ sau đây lấy từ kết quả thực tế của một phép nhận dạng *off-line* điện cảm stator của một động cơ dị bộ phụ thuộc dòng kích từ $L_s = f(I_{sd})$. Các giá trị đầu vào là dòng kích từ đã chuẩn hóa bởi dòng kích danh định I_{sdN}, tăng dần từ 0 ... I_{sdN} với bước tăng là $0,1I_{sdN}$, được cất trong vector **u**. Các giá trị L_s tương ứng do phép nhận dạng thu được có thứ nguyên là mH được cất trong vector **y**. Ta khai báo hai vector cột chứa cặp dữ liệu do phép nhận dạng *off-line* cung cấp:

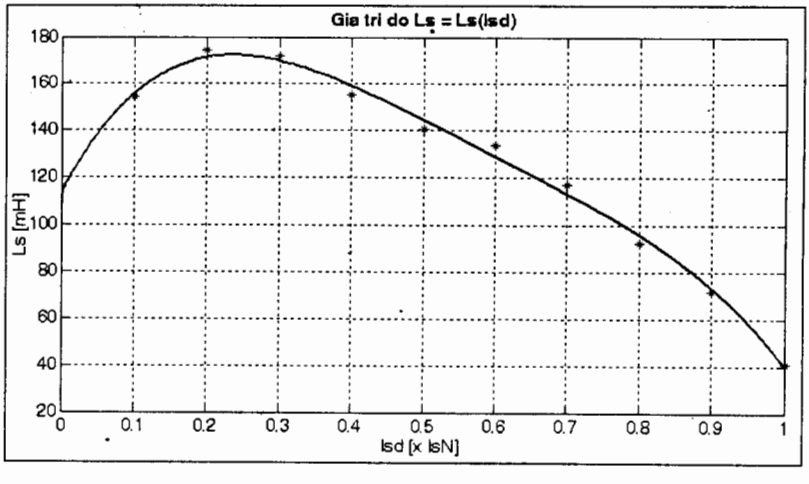
```
>> u = [0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0]';
>> y = [154.1 174.5 171.7 155 140.4 133.7 ...
       116.9 92.4 71.7 41]';
```

Đa thức xấp xỉ theo dạng (4.20) mô tả đặc tính từ hóa $L_s = f(I_{sd})$ được chọn là đa thức bậc 4. Ta dễ dàng thu được các hệ số của đa thức bằng chùm lệnh sau:

```
>> d = y; C = [ones(length(u),1) u u.^2 u.^3 u.^4];
>> x = C \ d
x =
  1.0e+003 *
    0.1132
    0.5978
   -1.9395
    2.2077
   -0.9394
```

Đa thức thu được có dạng:

$$L_s = 113,2 + 597,8I_{sd} - 1939,5I_{sd}^2 + 2207,7I_{sd}^3 - 939,4I_{sd}^4 \qquad (4.22)$$



Hình 4.6 *Đa thức* $L_s = 113,2 + 597,8I_{sd} - 1939,5I_{sd}^2 + 2207,7I_{sd}^3 - 939,4I_{sd}^4$ *xấp xỉ kết quả nhận dạng đường đặc tính từ hóa của động cơ dị bộ ba pha*

Để dễ so sánh và đánh giá hiệu quả, các cặp giá trị nhận dạng và đồ thị của (4.22) được biểu diễn ở hình 4.6 sau khi thực hiện chùm lệnh dưới đây:

```
>> plot(0.1,154.1,'b*',0.2,174.5,'b*',0.3,171.7,'b*', ...
     0.4,155,'b*',0.5,140.4,'b*',0.6,133.7,'b*', ...
     0.7,116.9,'b*',0.8,92.4,'b*',0.9,71.7,'b*',1,41,'b*');
>> hold on
>> fplot('113.2 + 597.8*x - 1939.5*x^2 + 2207.7*x^3 ...
       - 939.4*x^4', 0:1, 'k-');
>> hold off
>> title('Gia tri do Ls = Ls(Isd)','FontSize',12);
>> xlabel('Isd [x IsN]','FontSize',12);
>> ylabel('Ls [mH]','FontSize',12);
>> grid on
```

Vì vấn đề vừa giải quyết chỉ là bài toán tối ưu tuyến tính, sẽ luôn tồn tại đáp án rõ ràng và không phụ thuộc vào giá trị xuất phát. Nếu giả sử vì các điều kiện kỹ thuật – vật lý, hay vì lý do nào đó ta đòi hỏi tất cả các hệ số của đa thức phải không âm:

$$\min_{\mathbf{x}} \|\mathbf{C}\mathbf{x} - \mathbf{d}\|_2^2 ; \mathbf{x} \geq \mathbf{0} \tag{4.23}$$

Lúc này bài toán tìm tối ưu (4.23) có thêm điều kiện phụ và được gọi bằng thuật ngữ *nonnegative least squares*. Để tìm nghiệm tối ưu ta sử dụng lệnh `lsqnonneg`. Do có điều kiện phụ, có thể tồn tại nhiều điểm cực tiểu cục bộ, và ta sẽ phải khai báo giá trị xuất phát. Nếu không khai báo giá trị xuất phát, `lsqnonneg` sẽ tự động lấy gốc tọa độ làm điểm xuất phát. Lệnh được viết như sau:

```
x = lsqnonneg(C,d,x0,options)
[x,resnorm] = lsqnonneg(C,d,x0,options)
```

Vector **x** chứa các giá trị kết quả tính và độ dư sai số của chuẩn toàn phương Euclid được cất trong biến *resnorm*. Bằng độ dư đó ta có thể đánh giá chất lượng của các nghiệm. Ta lại lấy các giá trị nhận dạng ở trên để đặt thêm điều kiện phụ $\mathbf{x} \geq \mathbf{0}$.

```
>> d = y; C = [ones(length(u),1) u u.^2 u.^3 u.^4];
>> x0 = [10 10 10 10 10]; options = optimset('lsqnonneg');
>> options = optimset(options, 'Display', 'final');
>> [x,resnorm] = lsqnonneg(C,d,x0,options)
Optimization terminated successfully.
x =
  1.0e+002 *
   1.25140000000000
                   0
                   0
                   0
                   0
```

```
resnorm =
    1.771586400000000e+004
```

Nếu không cần đến giá trị xuất phát, ta viết `x0=[]`. Để so sánh với trường hợp tối thiểu không có điều kiện phụ và để đánh giá được chất lượng, ta hãy tính các giá trị chuẩn. Nên chú ý rằng, lệnh `norm` đầu tiên dưới đây tính chuẩn của kết quả tối ưu *nonnegative least squares*, lệnh `norm` thứ hai tính chuẩn khi không có điều kiện phụ. Lệnh `norm` tính chuẩn Euclid (không bình phương), vì vậy ta sẽ phải lấy bình phương kết quả của `norm` sẽ thu được giá trị của *resnorm*.

```
>> norm(C*lsqnonneg(C,d,x0,options)-d)
                                        % Có điều kiện phụ
Optimization terminated successfully.
ans =
    1.331009541663770e+002
>> norm(C*(C\d)-d)                      % Không điều kiện phụ
ans =
  10.22908007858367
```

Để có thể xét đến tất cả các điều kiện phụ dưới dạng phương trình hay bất phương trình, ta sử dụng lệnh `lsqlin`, dùng để giải bài toán:

$$\min_{\mathbf{x}} \|\mathbf{C}\,\mathbf{x}-\mathbf{d}\|_2^2 \tag{4.24}$$

dưới các điều kiện phụ:

$$\mathbf{A}\,\mathbf{x} \le \mathbf{b} \tag{4.25}$$

$$\mathbf{A}_{eq}\,\mathbf{x} = \mathbf{b}_{eq} \tag{4.26}$$

$$\mathbf{lb} \le \mathbf{x} \le \mathbf{ub} \tag{4.27}$$

Ý nghĩa của các ký hiệu đã được giải thích ở mục 4.5. Lệnh `lsqlin` được gọi như sau:

$$[x, resnorm] = \text{lsqlin}(C, d, A, b, Aeq, beq, lb, ub, x0, options)$$

Những tham số không sử dụng đến khi gọi lệnh cần được thay thế bởi dấu `[]`. Để minh họa ta lại lấy ví dụ tìm xấp xỉ đường đặc tính từ hóa của động cơ dị bộ với 10 cặp giá trị nhận dạng. Đa thức cần tìm sẽ phải chịu thêm điều kiện phụ: $a_2 + a_3 = 0$. Điều kiện đó sẽ được thỏa mãn nếu ta khai báo:

$$\mathbf{A}_{eq} = [0 \quad 0 \quad 1 \quad 1 \quad 0]; \quad \mathbf{b}_{eq} = 0 \tag{4.28}$$

Chùm lệnh tìm tối ưu là như sau:

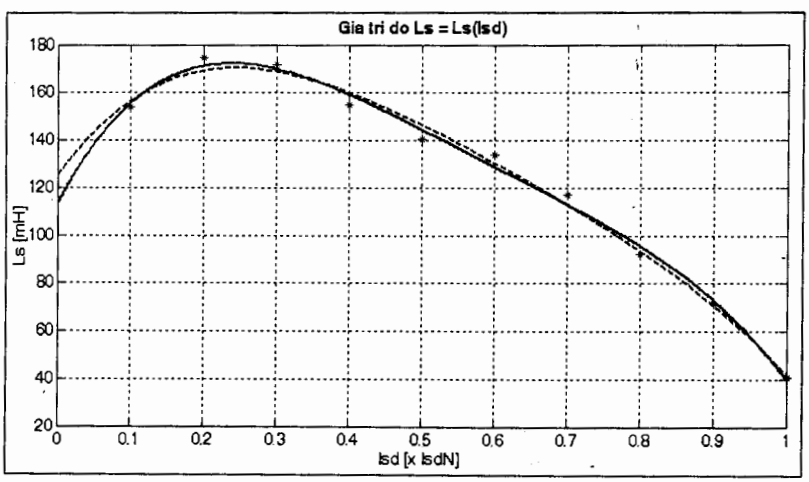
```
>> u = [0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0]';
                          % Khai báo các giá trị đầu vào
>> y = [154.1 174.5 171.7 155 140.4 133.7 ...
116.9 92.4 71.7 41]';     % Khai báo các giá trị Ls
                          % do nhận dạng tìm được
```

```
>> d = y; C = [ones(length(u),1) u u.^2 u.^3 u.^4];
>> options = optimset('lsqlin');
>> options = optimset(options, 'Display', 'final', ...
            'LargeScale', 'off');
>> Aeq = [0 0 1 1 0]; beq = [0];
>> [x,resnorm] = lsqlin(C,d,[],[],Aeq,beq,[],[],[],options)
Optimization terminated successfully.
x =
  1.0e+003 *
    0.1251
    0.4329
   -1.3012
    1.3012
   -0.5160
resnorm =
  136.3343
```

Với kết quả mới tìm được theo các điều kiện phụ (4.26) và (4.28), đặc tính từ hóa (4.22) lúc này có dạng:

$$L_s = 125{,}1 + 432{,}9 I_{sd} \underbrace{-1301{,}2}_{a_2} I_{sd}^2 \underbrace{+1301{,}2}_{a_3} I_{sd}^3 - 516 I_{sd}^4 \qquad (4.29)$$

Để dễ so sánh, ta vẽ các giá trị nhận dạng, đặc tính xấp xỉ (4.22) và (4.29) trên hình 4.7.



Hình 4.7 *Đặc tính từ hóa theo đa thức (4.22) (nét liền), đa thức (4.29) (nét đứt) và các giá trị nhận dạng (chấm *)*

Hình 4.7 cho thấy cả hai cách đặt bài toán LQ đều cho kết quả tốt và khá giống nhau. Tuy nhiên, chỉ nhằm mục đích minh họa cách sử dụng các lệnh, người viết đã trên cơ sở (4.22) để cố tình chọn điều kiện $a_2 + a_3 = 0$ cho dễ hội tụ.

Hình 4.7 đã được vẽ bởi chùm lệnh dưới đây:

```
>> plot(0.1,154.1,'b*',0.2,174.5,'b*',0.3,171.7,'b*', ...
   0.4,155,'b*',0.5,140.4,'b*',0.6,133.7,'b*', ...
   0.7,116.9,'b*',0.8,92.4,'b*',0.9,71.7,'b*', ...
   1,41,'b*');                          % Vẽ các cặp giá trị
>> hold on
>> fplot('113.2 + 597.8*x - 1939.5*x^2 + 2207.7*x^3 ...
   - 939.4*x^4', 0:1, 'r-');            % Vẽ đặc tính (4.22)
>> fplot('125.1 + 432.9*x - 1301.2*x^2 + 1301.2*x^3 ...
   - 516*x^4', 0:1, 'b--');             % Vẽ đặc tính (4.29)
>> grid on
>> title('Gia tri do Ls = Ls(Isd)','FontSize',12);
>> xlabel('Isd [x IsdN]','FontSize',12);
>> ylabel('Ls [mH]','FontSize',12);
>> hold off
```

Nếu các giá trị đo không thể được xấp xỉ bởi một đa thức, mà bởi một hàm phi tuyến bất kỳ, bài toán đó được gọi là *curve fitting* phi tuyến được mô tả bởi:

$$\min_{\mathbf{x}} \left\| F\left(\mathbf{x}, \mathbf{x}_{data}\right) - \mathbf{y}_{data} \right\|_2^2 = \min_{\mathbf{x}} \sum_{i=1}^{n} \left[F\left(\mathbf{x}, \mathbf{x}_{data,i}\right) - \mathbf{y}_{data,i} \right]^2 \quad (4.30)$$

Trong (4.30), $\mathbf{x}_{data}$ và $\mathbf{y}_{data}$ là vector các giá trị đo cần được xấp xỉ bởi hàm vector phi tuyến $F(\mathbf{x},\mathbf{x}_{data})$. Vector $\mathbf{x}$ chứa các tham số của hàm cần tìm. Để tìm tối ưu theo (4.30) ta sử dụng lệnh `lsqcurvefit` với cách gọi sau:

```
[x,resnorm]=lsqcurvefit('function',x0,xdata,ydata,lb,ub,options)
```

Hàm *function* được định nghĩa dưới dạng *inline object* hay *m-File* như sau:

```
function F = function9(x,xdata)
F = ...;            % Phụ thuộc tham số x và vector xdata
```

Các tham số còn lại đều đã được giải thích ý nghĩa và hướng dẫn cách khai báo. Ta hãy theo dõi cách sử dụng lệnh thông qua một ví dụ thực tế. Một đối tượng điều khiển được kích thích bởi một xung ở đầu vào và đo đáp ứng $\mathbf{y} = [y_1 \ldots y_n]$ (ứng với *ydata*) ở đầu ra trong khoảng thời gian $\mathbf{t} = [t_1 \ldots t_n]$ (ứng với *xdata*). Vì có tạp âm đo lường, vector **y** chứa các giá trị bị nhiễu. Giả thiết ta biết khái quát về đối tượng đó: Trong ví dụ là khâu PT_2 (khâu tỷ lệ có quán tính bậc 2). Trên miền thời gian, khâu đó được mô tả bởi một đáp ứng có dao động tắt dần, đặc trưng bởi ba tham số K, T và ω theo công thức:

$$y(t) = Ke^{-t/T} \sin(\omega t) \quad (4.31)$$

Trước hết ta viết một *script* có tên `function9.m`, với bộ tham số cần tìm K, T và ω, chứa trong vector `x = [x(1) x(2) x(3)]`.

```
%function9.m
function F = function9(x,t)
F = x(1).*(exp(-t./x(2)).*sin(x(3).*t));
```

Sau khi đã có function9.m, ta thực hiện ví dụ qua chuỗi lệnh dưới đây. Trước hết ta khai báo một đối tượng *sys* là khâu PT_2 dưới dạng mô hình TF. Sau đó tìm đáp ứng xung của *sys*, vector các giá trị đầu ra (chưa hề bị nhiễu) và vector thời gian tương ứng được cất trong hai biến y và t. Tiếp theo ta định nghĩa biến y_noise (đặc trưng cho đáp ứng ra đo được và bị nhiễu do tạp âm đo) bằng cách cộng thêm vào vector biến y một lượng tạp âm randn nào đó. Hai vector y_noise và t (ứng với *ydata* và *xdata* của công thức (4.30)) sẽ là cơ sở để thuật toán lsqcurvefit tìm bộ tham số x = [x(1) x(2) x(3)] định nghĩa trong function9.m.

```
>> sys = tf (5,[1 1 10])                % Khai báo đối tượng
Transfer function:
     5
-------------
s^2 + s + 10
>> [y,t] = impulse(sys,0:0.02:10);      % Đáp ứng chưa nhiễu
>> y_noise = y + 0.1*randn(length(t),1);
                                        % Đáp ứng có nhiễu đo
>> options = optimset('lsqcurvefit');
>> options = optimset(options, 'Display', 'final', ...
             'LargeScale', 'off');
>> x0 = [1 1 4];                        % Giá trị xuất phát
>> [x,resnorm] = lsqcurvefit('function9',x0,t,y_noise, ...
             [],[],options)             % Tính Curve Fitting
Optimization terminated successfully:
 Search direction less than tolX
x =
   1.61742519421002   1.98339388319347   3.12667250622918
resnorm =
   5.25168500083788
```

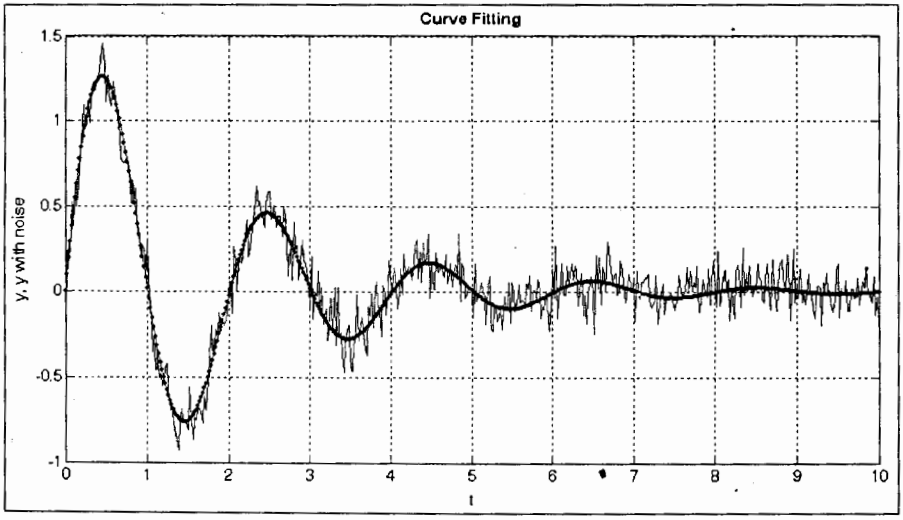
Thuật toán LQ *curve fitting* đã cho ra kết quả:

$$y_{curvefit}(t) = 1.61742\, e^{-t/1.98339} \sin(3.12667t) \tag{4.32}$$

Để dễ so sánh và đánh giá được hiệu lực của lệnh lsqcurvefit, ta vẽ đồ thị y, y_noise và (4.32) ở hình 4.8 bằng chùm lệnh sau đây.

```
>> fplot('1.61742*(exp(-x/1.98339)*sin(3.12667*x))', ...
        [0 10],'g-');
>> hold on
>> plot(t,y,'k.',t,y_noise,'r-');
>> grid on;
```

```
>> title('Curve Fitting','FontSize',12);
>> xlabel('t','FontSize',12);
>> ylabel('y, y with noise','FontSize',12);
```



Hình 4.8 *Đáp ứng đầu ra của đối tượng sau khi nhận xung kích thích: y (không chứa tạp âm, nét liền), y_noise (chứa tạp âm) và $y_{curvefit}$ (nét chấm)*

Hình 4.8 cho thấy: Đường cong xấp xỉ $y_{curvefit}$ (nét chấm đậm) và đường cong của đáp ứng lý tưởng y thực sự trùng khít với nhau, cho thấy khả năng mạnh của `lsqcurvefit` trong việc tìm xấp xỉ tối ưu cho một chuỗi giá trị đo bị nhiễu `y_noise`. Tuy nhiên, cần phải lưu ý thêm về vai trò quan trọng của giá trị xuất phát $x0$ đối với khả năng và tốc độ hội tụ khi tìm xấp xỉ.

Một phương pháp mở rộng của LQ tuyến tính theo (4.18) là phương pháp LQ phi tuyến, được coi là trường hợp tổng quát của LQ. Mục 4.7 trình bầy một ví dụ chi tiết minh họa ứng dụng của phương pháp.

Mục tiêu của bình phương sai phân bé nhất phi tuyến là tìm bộ tham số tối ưu của hàm vector phi tuyến $F(\mathbf{x})$ với các tham số cần tìm $\mathbf{x}$ theo phương trình:

$$\min_{\mathbf{x}} \left\| F(\mathbf{x}) \right\|_2^2 = \min_{\mathbf{x}} \sum_{i=1}^{n} F_i(\mathbf{x})^2 \tag{4.33}$$

Để giải bài toán (4.33) ta sử dụng lệnh `lsqnonlin` của *Optimization Toolbox*. Trong lệnh `lsqnonlin`, tổng các giá trị bình phương của (4.33) không được tính riêng một cách tường minh. Thay vào đó, ta trao cho `lsqnonlin` một hàm vector *function* với cú pháp gọi:

`[`*x*`,`*resnorm*`] = lsqnonlin(`*'function'*`,`*x0*`,`*lb*`,`*ub*`,`*options*`)`

Cách khai báo các tham số còn lại ta đều đã biết, bỏ không khai tham số nào ta phải thay `[]` vào vị trí của tham số đó. Tốc độ hội tụ phụ thuộc rất nhiều vào việc lựa chọn giá trị xuất phát $x0$.

Để giảm bớt khối lượng tính, ta có thể khai báo thêm tham số tùy chọn `options = optimset('Jacobian', 'on')`, đồng thời thực hiện xuất ma trận Jacobi dưới dạng biến ra thứ hai của hàm *function*. Ngoài ra, khi lựa chọn giá trị xuất phát cần phải lưu ý rằng: `lsqnonlin` chỉ luôn tìm được bộ giá trị cực tiểu cục bộ.

Phương pháp bình phương sai phân bé nhất (Least Squares, LQ)

`/` (slash)
LQ tuyến tính không có điều kiện phụ

`[x,resnorm]=lsnonneg(C,d,x0,options)`
LQ tuyến tính với $\mathbf{x} \geq \mathbf{0}$

`[x,resnorm]=lsqlin(C,d,A,b,Aeq,beq,lb,ub,x0,options)`
LQ tuyến tính có điều kiện phụ

`[x,resnorm]=lsqcurvefit('function',x0,xdata,ydata,lb,ub,options)`
Curve Fitting phi tuyến

`[x,resnorm]=lsqnonlin('function',x0,lb,ub,options)`
LQ phi tuyến

4.7 Tìm bộ tham số tối ưu cho mô hình SIMULINK

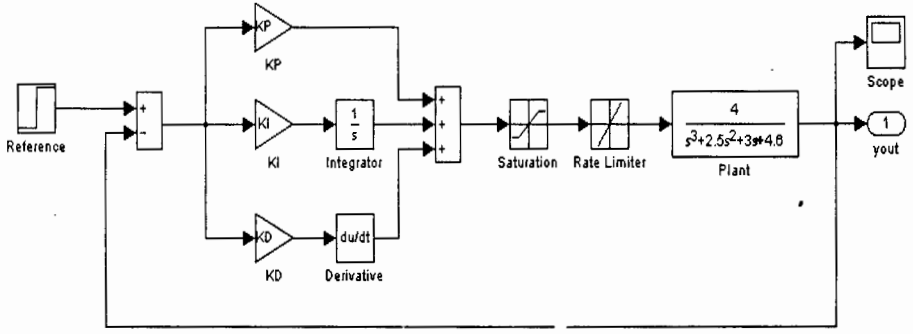
Về cơ bản, mọi tìm tối thiểu cho các hàm phi tuyến vector đều có thể dùng để tìm bộ tham số tối ưu cho mô hình SIMULINK. Trong các chương 6 – 8 ta sẽ đề cập kỹ hơn về việc xây dựng mô hình. Để minh họa ứng dụng của `lsqnonlin`, ta hãy lấy việc sử dụng phương pháp LQ tìm bộ tham số tối ưu K_P, K_I và K_D của khâu ĐC theo luật PID làm ví dụ. *Tổng các bình phương của sai lệch đáp ứng ra* (khi có kích thích bước nhẩy ở đầu vào) *phải là tối thiểu trong một khoảng thời gian cho trước*. Đáp ứng bước nhẩy $g(t,\mathbf{x})$ của hệ phụ thuộc phi tuyến vào thời gian t và bộ tham số $\mathbf{x}$ đang cần tìm. Khi mô phỏng hệ bằng SIMULINK, ta có thể ghi lại các giá trị đầu ra sau mỗi bước tính. Tùy bộ tham số $\mathbf{x}$ mà ta thu được tín hiệu ra $y(t,\mathbf{x})$ khác nhau. Nếu tập hợp tất cả các giá trị $y_i(\mathbf{x}) = y(t_i,\mathbf{x})$ trong vector tín hiệu ra $\mathbf{y}(\mathbf{x})$, ta thu được một hàm vector và có thể dùng `lsqnonlin` để tìm cực tiểu của hàm đó. Khi giá trị đặt y_{ref} là hằng và n là số bước tính (bước tích phân), hàm vector cần tìm cực tiểu có dạng:

$$F(\mathbf{x}) = \begin{bmatrix} y_1(\mathbf{x}) - y_{ref} \\ y_2(\mathbf{x}) - y_{ref} \\ \vdots \\ y_n(\mathbf{x}) - y_{ref} \end{bmatrix} \tag{4.34}$$

Lúc này có thể áp dụng lệnh `lsqnonlin` để tìm bộ tham số ĐC tối ưu theo phương pháp LQ :

[*x*, *resnorm*]=lsqnonlin('*function*', *x*0, *lb*, *ub*, *options*)

Hàm MATLAB *function* chứa các khai báo của $F(\mathbf{x})$ theo (4.34). Giả thiết, ta có nhiệm vụ đi tìm bộ tham số tối ưu của khâu ĐC với đặc tính PID, nhằm điều khiển một đối tượng quán tính bậc 3 (có khả năng dao động) như ở hình 4.9.



Hình 4.9 *Cấu trúc điều khiển đối tượng quán tính bậc 3, sử dụng khâu PID*

Các tham số của khâu PID được khai báo tổng quát ở phần *Block Parameters* là KP, KI và KD. Đầu ra của khâu PID bị chặn về biên độ (khâu *Saturation*: ±8) và độ dốc (khâu *Rate Limiter*: ±3). Cấu trúc được cất dưới tên `fig04_09.mdl`. Vì lệnh `lsqnonlin` của *Optimization Toolbox* chỉ có khả năng tìm tối ưu cho hàm, ta sẽ phải viết một hàm dưới dạng *m-File*, có nhiệm vụ trả về các giá trị sai lệch điều chỉnh (hiệu số giữa giá trị đặt và giá trị thực) tại tất cả các thời điểm mô phỏng. Ta đặt tên cho hàm (ví dụ) là `simopterror.m`, chứa ba biến chính là các tham số cần tìm của khâu PID. Hàm có nhiệm vụ gọi mô hình `fig04_09.mdl` (hình 4.9), tính các vector sai lệch điều chỉnh cung cấp cho thuật toán tìm tối ưu `lsqnonlin`. *Script* `simopterror.m` gồm chuỗi lệnh như sau:

```
% simopterror.m
function F = simopterror(x)
KP = x(1);                              % Đổi tên biến thành
KI = x(2);                              % tên các tham số của
KD = x(3);                              % cấu trúc fig04_09

% Khai báo chế độ làm việc của fig04_09.mdl và báo cho
% Workspace biết để trao đổi biến
opt = simset('solver','ode5','SrcWorkspace','Current');

% Mô phỏng fig04_09 trong khoảng 0...10s
[tout,xout,yout] = sim('fig04_09',[0 10],opt);
% Tính sai lệch ĐC; Giá trị đặt là 1
F = yout - 1;
```

Việc đổi tên các biến của hàm `simopterror.m` thành KP, KI và KD là cần thiết, vì đó chính là tên của các tham số sử dụng trong mô hình SIMULINK. Bằng `simset` ta đặt chế độ mô phỏng cho mô hình `fig04_09.mdl`. Đặc biệt, `simset('SrcWorkspace','Current')` có tác dụng buộc quá trình mô phỏng phải được thực hiện trong khuôn khổ *Workspace* (nơi lưu giữ tất cả các biến hiện tại của MATLAB), và do đó có trao đổi biến giữa mô hình và *Workspace*. Điều này vô cùng quan trọng vì trong quá trình tìm bộ tham số tối ưu, mô hình `fig04_09.mdl` luôn phải sử dụng đến KP, KI và KD nằm trong *Workspace*. Quá trình mô phỏng sẽ được `lsqnonlin` lặp lại nhiều lần cho tới khi thỏa mãn điều kiện tối ưu (4.33). Cuối cùng, để đơn giản hóa công việc, ta cất chùm lệnh tìm tối ưu trong một *script* có tên `simopt.m`.

```
%simopt.m
%Optimization of simulink model fig04_09.mdl
x0 = [1, 1, 1];                    % Giá trị xuất phát của
                                   % bộ tham số của PID
options = optimset('LargeScale','off','Display','iter', ...
                   'TolX',0.05,'TolFun',0.05);
x = lsqnonlin('simopterror',x0,[],[],options);
                                   % Tìm tham số tối ưu
KP = x(1); KI = x(2); KD = x(3); % Đổi tên trở lại
```

Sau khi đã hoàn tất công việc chuẩn bị, ta gọi:

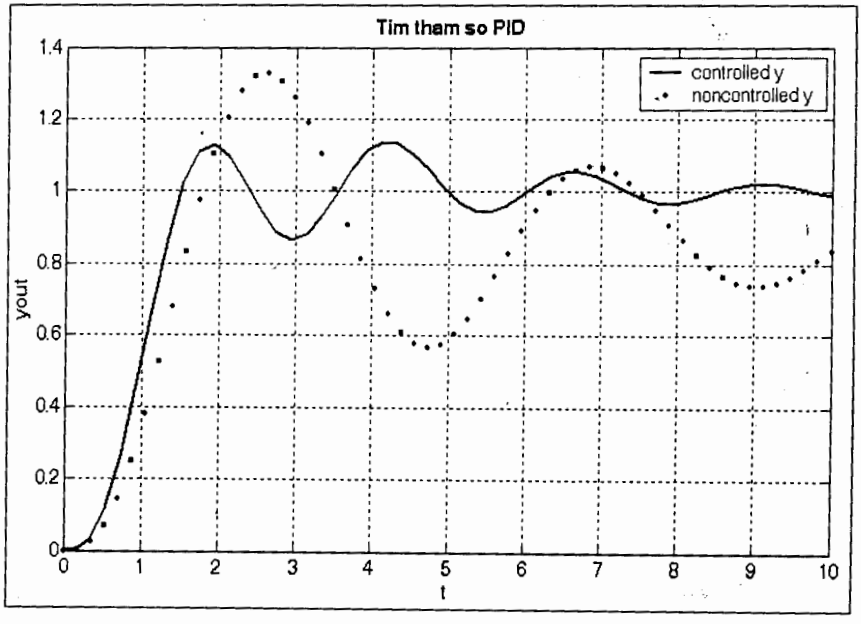
```
>> simopt
                                          Directional
Iteration  Func-count  Residual  Step-size   derivative    Lambda
   1            3       5.1112       1         -0.77
   2           10      4.50158       1        -0.0453   2.33614
Optimization terminated successfully:
 Gradient in the search direction less than tolFun
 Gradient less than 10*(tolFun+tolX)
KP =
    1.5582
KI =
    1.3535
KD =
    1.3947
```

Sau khi đã tìm được bộ tham số KP, KI và KD tối ưu, ta thay chúng vào mô hình `fig04_09.mdl` và gọi trực tiếp, cho `fig04_09.mdl` chạy mô phỏng với bộ tham số mới để thu được đáp ứng đầu ra `yout` (xem hình 4.9), cất trong *Workspace*. Để so sánh hiệu quả ta vẽ đồ thị của `yout` (đáp ứng bước nhẩy của đối tượng có điều chỉnh) và `yout2` (đáp ứng bước nhẩy của đối tượng không có điều chỉnh). Trước khi vẽ, ta còn phải tìm cách thu thập `yout2` như dưới đây.

```
>> sys = tf(4, [1 2.5 3 4.6])  % Khai báo đối tượng sys

Transfer function:
            4
-------------------------
s^3 + 2.5 s^2 + 3 s + 4.6

>> t = 0:10/57:10;                % Chú ý: Phải đúng với kích
                                  % cỡ tout trong Workspace
>> [yout2,tout2] = step(sys,t);
                                  % Đáp ứng bước nhẩy của sys
>> plot(tout,yout,'k-',tout2,yout2,'r.')
>> grid on
>> title('Tim tham so PID','FontSize',12);
>> xlabel('t','FontSize',12);
>> ylabel('yout','FontSize',12);
>> legend('controlled y','noncontrolled y')
```



Hình 4.10 *Đáp ứng bước nhẩy khi chưa có ĐC (nét chấm) và khi có ĐC bởi khâu PID (nét liền) với bộ tham số tối ưu tìm được theo phương pháp LQ phi tuyến*

Theo hình 4.10, rõ ràng nhờ có bộ tham số ĐC tối ưu mà đối tượng bậc 3 dao động đã được tăng hệ số tắt dần một cách đáng kể. Tuy nhiên, cần phải chú ý tới các đặc điểm phi tuyến của hệ (khâu ĐC bị chặn về biên độ và độ dốc ở đầu ra). Thuật toán LQ cài đặt trong *Optimization Toolbox* tìm tối ưu phụ thuộc vào biên độ của bước nhẩy kích thích và vì vậy rất dễ bị ảnh hưởng tới kết quả.

4.8 Tóm tắt nội dung chương 4

Sau khi đã nghiên cứu kỹ chương 4, người đọc cần nắm vững các nội dung sau đây:

1. Khai báo và sử dụng *Inline Objects* ?
2. Hiển thị và khai báo *Options* (đặt chế độ điều khiển) cho các thuật toán tìm tối ưu khác nhau ?
3. Tìm điểm không của các hàm scalar ?
4. Giải hệ phương trình tuyến tính và phi tuyến ?
5. Tìm tối thiểu của hàm scalar có hay không có điều kiện phụ ?
6. Sử dụng phương pháp bình phương sai phân bé nhất tuyến tính khi có hay không có điều kiện phụ ?
7. Tìm bộ tham số tối ưu của một hàm theo phương pháp *Curve Fitting* trên cơ sở chuỗi giá trị đo được ?
8. Áp dụng phương pháp bình phương sai phân bé nhất phi tuyến đối với các hàm vector. Ví dụ: Nhằm tìm tối ưu cho mô hình SIMULINK ?

5 Signal Processing Toolbox: Công cụ xử lý tín hiệu

Các phương pháp xử lý tín hiệu trước hết phục vụ chế biến, lọc và phân tích các số liệu thu thập được. Thông thường, các số liệu đó tồn tại dưới dạng giá trị đo và - để có thể chế biến tiếp - được MATLAB đọc vào dưới dạng vector. Trong điều khiển tự động, có rất nhiều bài toán (thuộc các phạm trù như nhận dạng, quan sát và điều khiển thích nghi vv...) cần các giá trị đo đã qua xử lý - chế biến thích hợp. Chính vì vậy, *Signal Processing Toolbox* có một ý nghĩa đặc biệt đối với lĩnh vực xử lý tín hiệu nói chung, và điều khiển - tự động hóa nói riêng.

5.1 Phương pháp nội suy (Interpolation)

Các phép đo thường được thực hiện tại các thời điểm gián đoạn (cách đều hoặc không cách đều). Tuy nhiên, lúc sử dụng các giá trị đo được, đôi khi ta cần cả các giá trị nằm giữa các thời điểm đo. Công cụ nội suy (*Interpolation*) sẽ cung cấp cho ta các giá trị nằm giữa đó. Trong MATLAB ta có thể sử dụng lệnh `interp1(x_value,y_value,x_processing,method)` để thực hiện nội suy một chiều, hay `interp2` và `interp3` để thực hiện nội suy nhiều chiều hơn. Khi thực hiện nội suy, các số liệu đo phải được sắp xếp theo trình tự tọa độ tăng dần (giống lệnh vẽ `surf`).

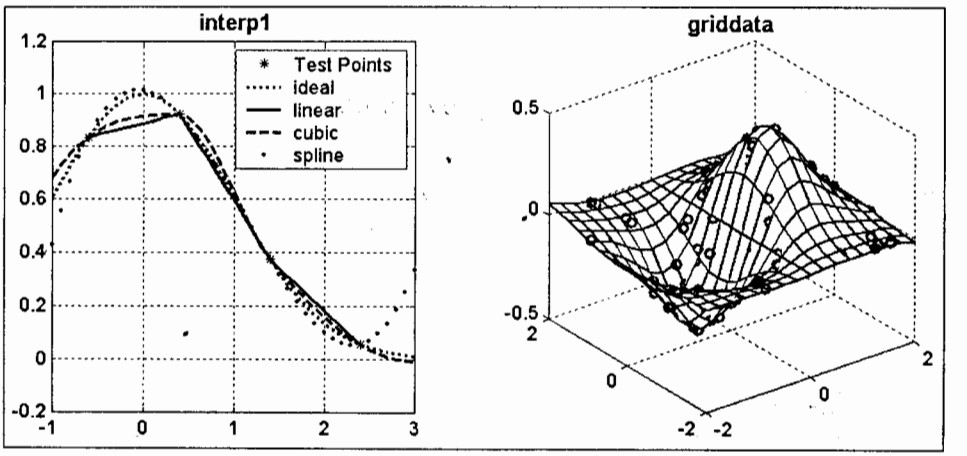
Ví dụ ở hình 5.1 (bên trái) được thực hiện bởi chùm lệnh sau đây. Hình đó cho ta thấy rất rõ, duy nhất hai phương pháp *Splines* và *Cubic* cho phép ngoại suy (*Extrapolation*) vượt ra khỏi khoảng bị chặn bởi các bước thô.

```
>> x_rough = -0.6:2.4;                    % Chọn bước thô
>> x_fine = -1:0.1:3;                     % Chọn bước nội suy
>> data = exp(-x_rough.^2/2);             % Số liệu đo được

% Nội suy số liệu theo các phương pháp khác nhau
>> inter_linear = interp1(x_rough, data, x_fine, 'linear');
>> inter_cubic = interp1(x_rough, data, x_fine, 'cubic');
>> inter_spline = interp1(x_rough, data, x_fine, 'spline');

% Vẽ đồ thị để so sánh
>> figure
>> plot(x_rough, data, 'k*');
>> hold on;
```

```
>> plot(x_fine, exp(-x_fine.^2/2), 'k:');
>> plot(x_fine, inter_linear, 'b-');
>> plot(x_fine, inter_cubic, 'g--');
>> plot(x_fine, inter_spline, 'r.');
>> grid on;
>> legend('Test Points','ideal','linear','cubic','spline');
>> title('interp1','FontSize',12);
```



Hình 5.1 *Các phương pháp nội suy khác nhau*

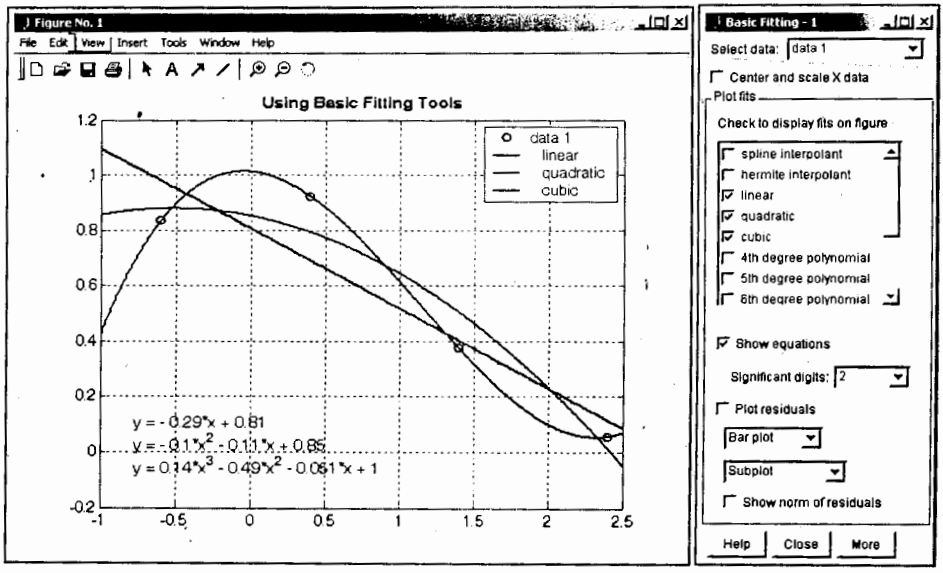
Nếu ta có bộ dữ liệu nhiều chiều chưa được xử lý, ta có thể dùng `griddata(x_value,y_value,z_value,x_proc,y_proc,method)` để tạo ra khung số liệu. Lệnh `griddata` phù hợp để tạo ra (và hiển thị đồ họa 3-D) một lưới từ các số liệu đo có phân bố không đều như hình 5.1 (bên phải), được tạo bởi chùm lệnh dưới đây. Lệnh `rand(rows,columns)` tạo ra một ma trận với kích cỡ tùy theo ta khai báo với giá trị ngẫu nhiên trong khoảng 0...1.

```
>> x = rand (100,1)*4-2;          % Tạo vector số ngẫu nhiên
>> y = rand (100,1)*4-2;
>> z = x.*exp(-x.^2-y.^2);        % Các giá trị thử
>> [XI, YI] = meshgrid (-2:0.25:2, -2:0.25:2);
                                  % Tạo khung tọa độ
>> ZI = griddata (x, y, z, XI, YI, 'v4');
                                  % Tạo lưới ZI từ giá trị thử
>> mesh (XI, YI, ZI);             % Vẽ lưới ZI (nối các điểm)
>> hold on;
>> plot3(x, y, z, 'o');           % Vẽ các điểm trên lưới
>> title('griddata','FontSize',12);
>> hold off;
```

Nếu cần phải tìm mối quan hệ hàm giữa các số liệu đo bằng các hàm nội suy, ta có thể sử dụng công cụ *Basic Fitting*. Để sử dụng, trước hết ta biểu diễn số liệu đo bằng một công cụ đồ họa. Sau đó gọi *Basic Fitting-Tool* trong menue *Tools* của đồ họa mới vẽ. Để minh họa ta hãy vẽ các giá trị của ví dụ ở hình 5.1 (bên trái) bằng các lệnh dưới đây:

```
>> x_rough = -0.6:2.4;
>> data = exp(-x_rough.^2/2);
>> figure
>> plot(x_rough,data,'ko')
```

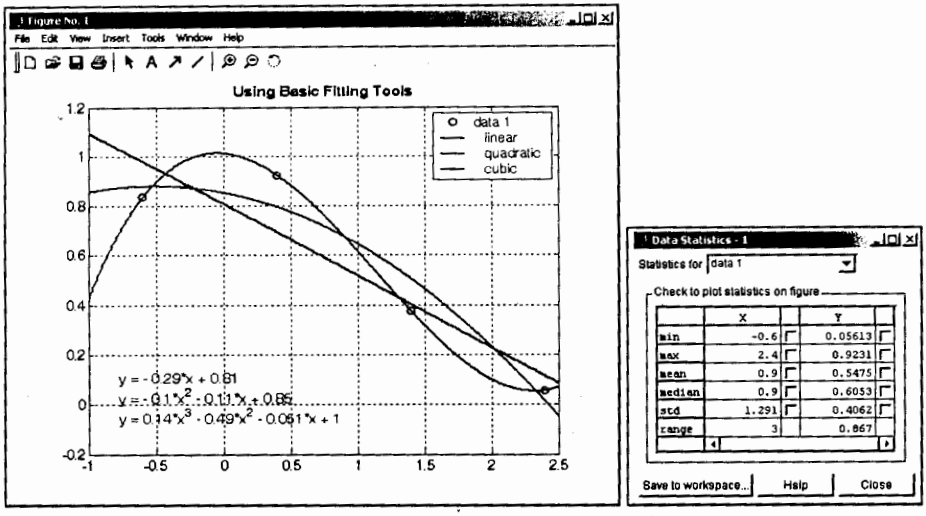
Sau đó ta vào menue *Tools* của GUI thu được và chọn *Basic Fitting*, một cửa sổ mới (hình 5.2, bên phải) mở ra và ta có nhiều khả năng để lựa chọn phương pháp nội suy, kèm theo thuyết minh về đường cong (*legend*, bên phải phía trên của đồ họa). Nếu ta chọn thêm ô *Show Equations*, thậm chí thu được cả phương trình của các đường cong nội suy (hình 5.2, bên trái).



Hình 5.2 *Nội suy bằng công cụ Basic Fitting trong menue Tools của GUI*

Lợi thế lớn nhất của công cụ *Basic Fitting* là tạo cho người sử dụng khả năng đánh giá nhanh hiệu quả nội suy bằng đồ họa. Tuy nhiên, nếu cần các kết quả có độ chính xác cao, hay cần xử lý – chế biến trong các *scripts* thì dùng lệnh `interp1` vẫn thuận lợi hơn.

Một công cụ khá tốt nữa của GUI là *Data Statistics* (hình 5.3, bên phải), chọn từ menue *Tools*. Trong cửa sổ *Data Statistics* ta có thể đọc được các giá trị trung bình, sai lệch chuẩn, giá trị *median* (trung bình thống kê) và một số giá trị thống kê khác nữa.



Hình 5.3 Đánh giá kết quả nội suy của Basic Fitting bằng công cụ Data Statistics

Nội suy	
`interp1(x_value,y_value,x_processing,method)`	Nội suy
`griddata(x_value,y_value,z_value,x_proc, ... y_proc,method)`	Tạo khung số liệu
Basic Fitting	Nội suy bằng GUI
Data Statistics	Thống kê bằng GUI

5.2 Biến đổi Fourier gián đoạn

Biến đổi Fourier (*Fourier Transformation*) là công cụ quan trọng sử dụng trong phân tích số liệu trên miền tần số. Công thức biến đổi Fourier liên tục như sau:

$$H(j\omega) = \int_{-\infty}^{\infty} h(t)e^{-j\omega t}dt \tag{5.1}$$

Để có thể xử lý được các số liệu đo thu được tại các thời điểm gián đoạn, ta cần phép biến đổi Fourier gián đoạn (*Discrete Fourier Transformation*: DFT). Để tính xấp xỉ công thức (5.1), ta thay phép tích phân bởi phép tính tổng N diện tích hình chữ nhật có chiều cao là $h(nT)$ với T là chiều rộng của các hình chữ nhật. T chính là chu kỳ trích mẫu của phép đo, $F = 1/T$ là tần số trích mẫu và N là số lượng giá trị đo.

$$H_d(\omega_k) = \sum_{n=0}^{N-1}\left[h(nT)e^{-j\omega_k nT}\right] \tag{5.2}$$

DFT chỉ được định nghĩa cho tần số gián đoạn $\omega_k = k\Delta\omega$. Độ phân giải tần số $\Delta\omega = 2\pi/(NT)$ hoặc $\Delta F = 1/(NT)$ là tỷ lệ nghịch với khoảng thời gian đo. Tần số tối đa có thể đo được là $F_{max} = F/2 = 1/(2T)$. Tùy theo góc pha của từng tần số thành phần, có thể xuất hiện các hệ số Fourier với giá trị phức.

Lệnh `fft(x)` của MATLAB có tác dụng chế biến vector số liệu x, trả lại kết quả dưới dạng một vector có cùng độ dài với x, và các hệ số Fourier[1]. Các hệ số Fourier có hai khoảng đối xứng $1 \le k < N/2$ và $N/2 < k \le N$.

Nếu cần xác định các hệ số thực a_i, b_i của chuỗi Fourier:

$$h(t) = \frac{a_0}{2} + \sum_{k=1}^{K}\left[a_k \cos(k\omega_k t) + b_k \sin(k\omega_k t)\right] \tag{5.3}$$

bằng công cụ DFT, ta cần chú ý tới chu kỳ trích mẫu T và chuẩn hóa kết quả bằng thời gian đo NT. Để làm điều đó và để xét tới chiều rộng T của lượng tử diện tích chữ nhật, ta chia các phần tử của H_d cho N. Đồng thời ta tách dải tần trên F_{max} ra, và gấp đôi số phần tử còn lại (không kể thành phần một chiều có chỉ số 1).

$$a_k = \frac{2}{N}\mathrm{Re}\{H_d(k)\}; b_k = -\frac{2}{N}\mathrm{Im}\{H_d(k)\} \tag{5.4}$$

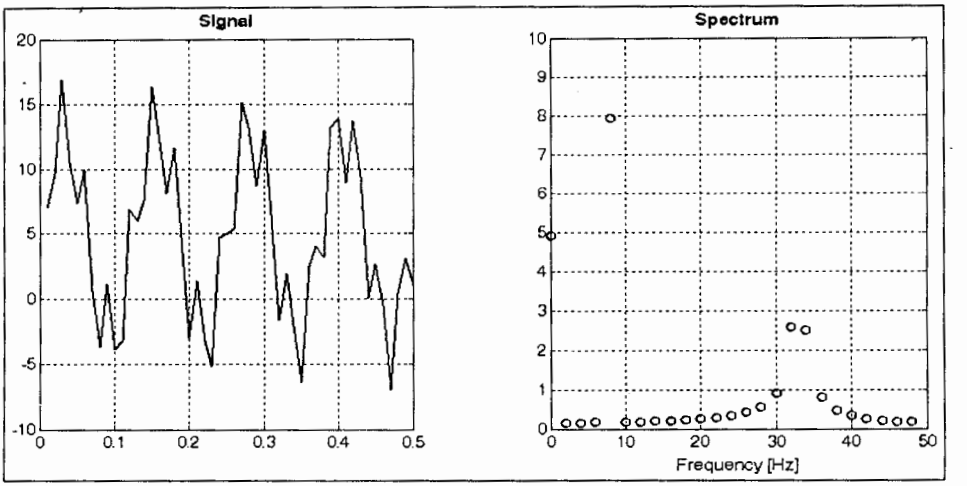
Trong ví dụ dưới đây, trước hết ta tạo ra một vector giá trị đo x kèm theo vector thời gian t. Sau đó ta xác định chu kỳ trích mẫu T, số lượng N của giá trị đo và vector tần số f. Kết quả biến đổi H được chuẩn hóa và hạn chế vào phạm vi $\le F_{max}$. Tín hiệu đo và phổ tần được minh họa ở hình 5.4.

```
>> t = 0.01:0.01:0.5;          % Vector thời gian (F=100Hz)
>> x = 5 + 8*sin(2*pi*8*t) + 4*cos(2*pi*33*t);
                               % Vector số liệu đo
>> T = diff (t(1:2));          % Chu kỳ trích mẫu
>> N = length(x);              % Chiều dài vector số liệu
>> f = [0:(N-1)/2] / (N*T);    % Vector tần số (phục vụ in)
>> H = fft(x);                 % Biến đổi Fourier
>> H = H/N;                    % Chuẩn hóa
>> H = [H(1) 2*H(2:N/2)];      % Giới hạn vào <F_max
```

Để hiển thị kết quả biến đổi Fourier vừa thực hiện, ta vẽ đồ thị như sau:

```
>> figure;
>> subplot(121);               % Vẽ đồ thị tín hiệu đo
>> plot(t,x);
>> title('Signal','FontSize',12);
>> subplot(122);               % Vẽ phổ tần thu được
>> plot(f,abs(H),'o');
>> xlabel('Frequency [Hz]','FontSize',12);
>> title('Spectrum','FontSize',12);
```

[1] Trong MATLAB chỉ số (*index*) được đếm từ 1 đến N, chứ không phải từ 0 tới N-1



Hình 5.4 *Tín hiệu ban đầu và phổ tần số*

Trong ví dụ trên, chu kỳ trích mẫu T = 10ms và tần số cao nhất có thể đo được là F_{max} = 50Hz. Có tất cả 50 giá trị đo đã được xử lý. Độ phân giải tần số là ΔF = 2Hz. Phổ được tính cho các tần số 0, ΔF, $2\Delta F$, ... , F_{max}, nếu xuất hiện một thành phần tần số của tín hiệu đo được nằm giữa các tần số tính (hình 5.4: nằm giữa các chấm tròn), đó chính là do hiện tượng dò tần số[1] (*Leakage Effect*) gây nên: Biên độ có phân bố rải sang cả các tần số lân cận. Hiện tượng này sẽ xuất hiện khi khoảng thời gian đo NT không phải là bội số (gấp một số nguyên lần) của chu kỳ của thành phần tín hiệu cần được phân tích.

Việc chọn độ phân giải tần số ΔF mịn hơn nhằm mục đích loại trừ dò tần số cũng chỉ có tác dụng hạn chế, vì qua đó tạp âm sẽ có ảnh hưởng mạnh hơn. Biện pháp tốt hơn là sử dụng các hàm cửa sổ (*windows function*) hoặc tạo giá trị trung bình (phương pháp *Averaging*).

5.2.1 Phương pháp Averaging

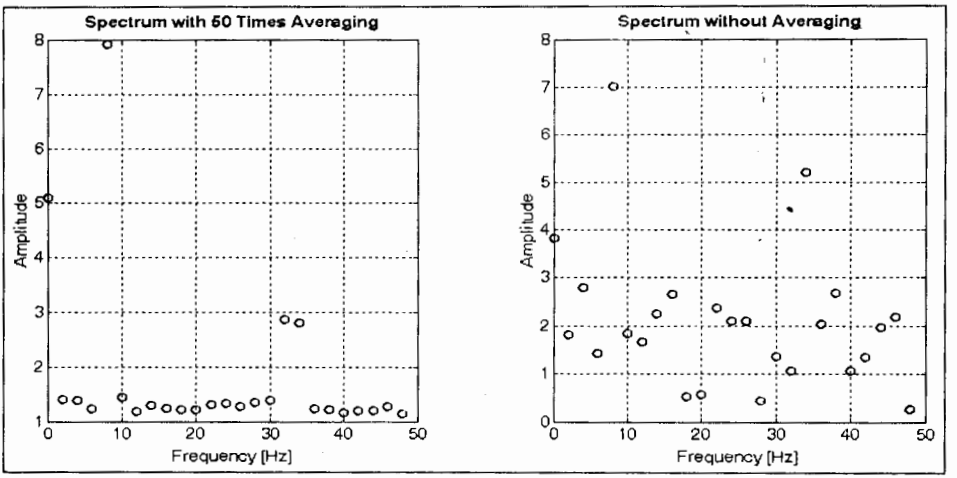
Nếu có một bộ số liệu đo đủ dài, ta có thể chia thành nhiều chùm (*sequences*) để có thể sử dụng biến đổi Fourier (phân tích phổ) cho từng chùm, tiếp theo ta tính giá trị trung bình cho từng phổ vừa thu được.

Ví dụ sau đây sử dụng bộ số liệu đo được của một tín hiệu trong khoảng hơn 25s. Ta chồng thêm vào tín hiệu đó một lượng tạp âm có phân bố chuẩn bằng lệnh `randn(`*rows*`,`*columns*`)`. Độ dài của chùm được chọn trước là `N_seq=50` điểm đo:

```
>> t = 0.01:0.01:25;                    % Vector thời gian
```

[1] Bạn đọc quan tâm tới *Leakage Effect* sẽ phải tự tra cứu tài liệu chuyên về xử lý tín hiệu

```
>> x = 5 + 8*sin(2*pi*8*t) + 4*cos(2*pi*33*t);
                                % Vector số liệu (giá trị đo)
>> x = x + 5*(randn(1,length(t)));
                                % Tạo thêm thành phần tạp âm
>> N_seq = 50;                  % Chiều dài của sequence
>> T = diff(t(1:2));            % Chu kỳ trích mẫu
>> N = length(x);               % Chiều dài vector số liệu
>> f = [0:(N_seq-1)/2] / (N_seq*T);
                                % Vector tần số
>> H = zeros(1,N_seq);          % Khai vector phổ tần số
>> for k = 1:N/N_seq ,
>>     x_seq = x(1+(k-1)*N_seq:k*N_seq);% Cắt sequence
>>     H = H + abs(fft(x_seq));          % DFT và tính tổng
>> end;
>> H = [H(1) 2*H(2:N_seq/2)] / N;
                                % Giới hạn, chuẩn hóa
% Vẽ phổ tần số: Hình 5.5, bên trái
>> figure;
>> subplot(121);
>> plot(f,abs(H),'o');
>> xlabel('Frequency [Hz]','FontSize',12);
>> title('Spectrum with Averaging','FontSize',12);
>> grid on;
```



Hình 5.5 *Phổ tần thu được bằng DFT khi có (hình trái) và không (hình phải) sử dụng phương pháp Averaging để loại trừ dò tần số*

Để dễ dàng so sánh, ta vẽ phổ tần của trường hợp không sử dụng phương pháp *Averaging* bằng đoạn lệnh sau đây (hình 5.5, bên phải).

```
>> t = 0.01:0.01:0.5;          % Vector thời gian
>> x = 5 + 8*sin(2*pi*8*t) + 4*cos(2*pi*33*t);
>> x = x + 5*(randn(1,length(t))); % Vector số liệu (giá
                                % trị đo) có tạp âm
>> T = diff(t(1:2));           % Chu kỳ trích mẫu
>> N = length(x);              % Chiều dài vector số liệu
>> f = [0:(N-1)/2] / (N*T);    % Vector tần số

>> H = zeros(1,N);             % Khai báo vector phổ tần
>> H = fft(x);                 % Biến đổi DFT
>> H = [H(1) 2*H(2:N/2)] / N;  % Giới hạn, chuẩn hóa

>> subplot(122);               % Vẽ hình 5.5 bên phải
>> plot(f,abs(H),'o');
>> xlabel('Frequency [Hz]','FontSize',12);
>> title('Spectrum without Averaging','FontSize',12);
>> grid on;
```

5.2.2 Phương pháp tạo cửa sổ số liệu

Ngoài *Averaging* ta có thể sử dụng phương pháp tạo cửa sổ số liệu (trước khi thực hiện DFT) để hạn chế hiện tượng dò tần số. Ý tưởng cơ bản của phương pháp có thể được diễn đạt như sau: Việc thực hiện phép đo một tín hiệu tuần hoàn dài vô hạn trong một khoảng thời gian hữu hạn (quan sát dưới giác độ toán học) tương đương với việc nhân tín hiệu cần đo với một hàm có dạng hình chữ nhật, gọi là *hàm cửa sổ hình chữ nhật*. Hàm có giá trị 1 trong khoảng thời gian đo, ngoài ra là bằng 0. Trên miền tần số, điều đó ứng với phép tích chập giữa phổ tần số hữu ích và phổ tần của cửa sổ hình chữ nhật. Tốc độ tắt dần chậm của phổ tần cửa sổ (ứng với sườn có độ dốc rất lớn của cửa sổ) chính là nguyên nhân của hiện tượng dò tần số. Nếu thay vào đó ta sử dụng một hàm cửa sổ có sườn không quá dốc (có nghĩa là: không phải số liệu đo nào cũng có hệ số trọng lượng 1), khi ấy có thể hạn chế dò tần số rất nhiều. Trong MATLAB đã có sẵn một số lệnh tạo hàm cửa sổ với chiều dài n như: `boxcar(n)`, `hamming(n)`, `barlett(n)`, `blackman(n)` và `hann(n)`.

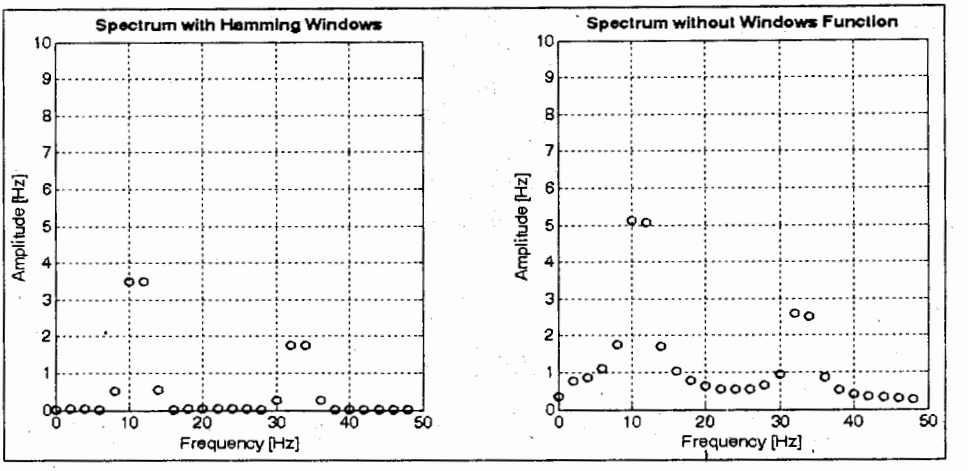
Khi sử dụng hàm cửa sổ, ta sẽ không thực hiện DFT cho toàn bộ tập số liệu nữa mà chỉ áp dụng cho phần số liệu do cửa sổ cắt ra. Cú pháp gọi lệnh `fft` đối với bộ số liệu x sử dụng cửa sổ *Hamming* là như sau:

```
H = fft (hamming(length(x))'.*x);
```

Ví dụ sau đây sẽ minh họa rõ hơn cách sử dụng các lệnh trên.

```
>> t = 0.01:0.01:0.5;                  % Vector t (F=100Hz)
>> x = 8*sin(2*pi*11*t) + 4*cos(2*pi*33*t);
                                       % Vector số liệu đo
>> T = diff(t(1:2));                   % Chu kỳ trích mẫu
```

```
>> N = length(x);
>> f = [0:(N-1)/2] / (N*T);
>> H = fft (hamming(N)'.*x);
>> H = H/N;
>> H = [H(1) 2*H(2:N/2)];
>> subplot(121);
>> plot(f,abs(H),'o');
>> axis([0 50 0 10]);
>> xlabel('Frequency [Hz]','FontSize',12);
>> ylabel('Amplitude [Hz]','FontSize',12);
>> title('Spectrum with Hamming Windows','FontSize',12);
>> grid on;
```



Hình 5.6 *Phổ tần thu được bằng DFT khi có (hình trái) và không (hình phải) sử dụng phương pháp cửa sổ Hamming để loại trừ dò tần số*

Đồ thị của phổ tần ở bên phải của hình 5.6 được vẽ bởi chùm lệnh tương tự như hình 5.5. So sánh hai phổ ta dễ dàng rút ra nhận xét: Khi sử dụng cửa sổ *Hamming*, hiện tượng dò chỉ còn xẩy ra ở lân cận hai tần số 11Hz và 33Hz và đồng thời biên độ (*amplitude*) cũng giảm đi. Quyết định cuối cùng: Có hay không sử dụng hàm cửa sổ, sẽ phải được quyết định tùy theo ứng dụng cụ thể, xem liệu có được phép hay không.

Một công cụ phân tích phổ không kém phần lợi hại khác của MATLAB là lệnh `spectrum(x)`, có khả năng tính và hiển thị phổ mật độ công suất[1]. Cho đến nay ta mới đề cập đến lệnh `fft(x)`, có tác dụng biến đổi Fourier cho một tập số liệu x, tức là biến đổi Fourier một chiều. Trong trường hợp ta có nhiều tập số liệu đo, được sắp xếp trong ma trận X, khi ấy ta có thể sử dụng lệnh

[1] Phổ mật độ công suất mô tả năng lượng của các thành phần tần số thuộc phổ và được xác định nhờ bình phương của các hệ số Fourier.

`Y=fft2(X)` để tìm ảnh Fourier hai chiều của X. Có tác dụng ngược lại với `fft` và `fft2` là các lệnh `ifft` và `ifft2`. `ifft(y)` có tác dụng biến đổi ngược ảnh y, nếu y = `fft(x)`, ta có: x = `ifft(y) = ifft(fft(x))`. Tương tự, `ifft2(Y)` có tác dụng biến đổi ngược ảnh Y, nếu Y = `fft2(X)`, ta có: X = `ifft2(Y) = ifft(fft(X))`.

Biến đổi Fourier gián đoạn	
`fft(x)`	Biến đổi Fourier gián đoạn (DFT) một chiều
`fft2(X)`	Biến đổi Fourier gián đoạn (DFT) hai chiều
`ifft(x)`	Biến đổi ngược ảnh `fft`
`ifft2(X)`	Biến đổi ngược ảnh `fft2`
`spectrum(x)`	Phân tích kèm theo phổ công suất
`boxcar(order)`	Hàm cửa sổ chữ nhật
`hamming(order)`	Hàm cửa sổ Hamming
`bartlett(order)`	Hàm cửa sổ Bartlett
`blackmann(order)`	Hàm cửa sổ Blackmann
`hann(order)`	Hàm cửa sổ Hann
`rand(rows,columns)`	Số ngẫu nhiên trong khoảng 0 ... 1
`randn(rows,columns)`	Số ngẫu nhiên (tạp âm) phân bố chuẩn

5.3 Hàm tương quan (Correlation Functions)

Hàm tương quan được sử dụng để phân tích sự tương đồng, hay thậm chí khả năng đồng nhất giữa các tín hiệu. Hàm tương quan được định nghĩa cho cả tín hiệu liên tục lẫn tín hiệu gián đoạn. Để phân tích được bằng MATLAB, các tín hiệu đều có sẵn dưới dạng gián đoạn, vì vậy tại đây ta sẽ chỉ quan tâm tới hàm tương quan của các tín hiệu gián đoạn. *Hàm tương quan chéo* (*cross corelation functions*) Φ_{xy} giữa hai tín hiệu $x(n)$ và $y(n)$ được định nghĩa bởi:

$$\Phi_{xy}(k) = \frac{1}{N}\sum_{n=1}^{N}\left[x(n+k)\,y(n)\right] \tag{5.5}$$

Có thể tính hàm tương quan của một tín hiệu duy nhất, gọi là *hàm tự tương quan* Φ_{xx} (*autocorelation functions*) định nghĩa như sau:

$$\Phi_{xx}(k) = \frac{1}{N}\sum_{n=1}^{N}\left[x(n+k)\,x(n)\right] \tag{5.6}$$

Các hàm tương quan đều là hàm chẵn, vì vậy:

$$\Phi_{xx}(-k) = \Phi_{xx}(k);\ \Phi_{xy}(-k) = \Phi_{xy}(k) \tag{5.7}$$

Để tính được hàm tương quan một cách chính xác, thực ra ta cần sử dụng một tập số liệu dài vô hạn, nghĩa là $N \to \infty$. Các công thức (5.6), (5.7) chỉ là công

thức gần đúng, với kết quả tính tiệm cận trường hợp lý tưởng khi N tăng. Tuy nhiên, tài liệu này sẽ không đi sâu vào vấn đề tương quan.

Trong MATLAB, hàm tương quan chéo Φ_{xy} giữa hai tín hiệu được tính bởi lệnh `xcorr`:

```
cxy = xcorr(x, y, 'options')
```

Trong lệnh trên, x và y là giá trị trích mẫu của các hàm thời gian. Nếu không khai báo *options*, hàm tương quan Φ_{xy} sẽ không được tính cho toàn bộ chiều dài N của tín hiệu, nghĩa là: Hệ số $1/N$ trong công thức (5.6) sẽ mất đi. Nếu gán cho `options` giá trị `biased`, khi ấy ta thu được kết quả theo (5.6). Hàm tự tương quan Φ_{xx} chỉ cần một vector biến vào x là đủ.

```
cxx = xcorr(x, 'options')
```

Kết quả của `xcorr` là một vector có độ dài $2N$-1. Khi chiều dài của tập số liệu là N, hàm tương quan tìm được chỉ xê dịch trong khoảng $-(N-1) \le k \le N-1$. Ngoài khoảng đó, hàm tương quan có giá trị là 0. Trong phạm vi hai vector *cxx* và *cxy*, các phần tử của hàm tương quan được sắp xếp theo thứ tự như sau:

$$\begin{aligned}
\Phi_{xx}(-N+1) &= cxx(1) \\
\Phi_{xx}(-N+2) &= cxx(2) \\
&\vdots \quad \vdots \\
\Phi_{xx}(0) &= cxx(N) \\
\Phi_{xx}(1) &= cxx(N+1) \\
&\vdots \quad \vdots \\
\Phi_{xx}(N-1) &= cxx(2N-1)
\end{aligned} \tag{5.8}$$

Ví dụ: Hàm tự tương quan

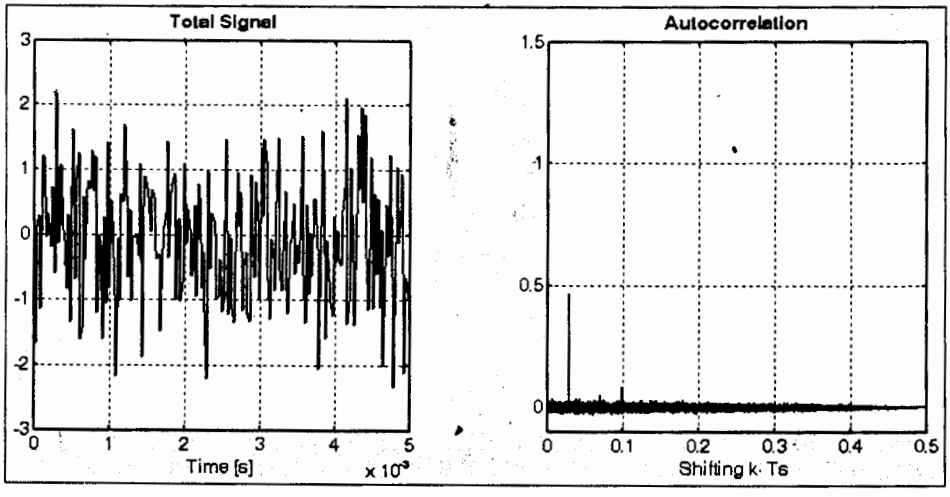
Để minh họa phương thức tìm hàm tự tương quan ta hãy quan sát một khâu truyền đạt đơn giản. Trong ví dụ đó, tín hiệu có ích (tiếng nói, giá trị đo) bị nhiễu bởi tiếng vọng. Bằng công cụ hàm tự tương quan, ta xác định thời gian di chuyển của tiếng vọng (hình 5.1).

Trước hết ta tạo ra tín hiệu có ích dưới dạng chuỗi ngẫu nhiên trong khoảng thời gian 0,5s với tần số trích mẫu F_s. Tiếp theo, từ tín hiệu ban đầu ta tạo hai tín hiệu vọng (bằng cách làm trễ 30ms và 100ms) có suy giảm. Cuối cùng, ta kiểm tra tự tương quan của tín hiệu hỗn hợp mới tạo.

```
% Đã có tín hiệu (âm thanh, tín hiệu đo, vv...)
>> Fs = 44.1e3;                          % Tần số 44,1kHz
>> Ts = 1/Fs;                            % Chu kỳ trích mẫu
>> t = 0:Ts:0.5;
>> endIndex = length(t);
% Tạo tín hiệu có ích dưới dạng số ngẫu nhiên
>> signal = randn(1,endIndex);
```

```
% Tại vị trí bất kỳ xuất hiện tín hiệu vọng đã bị suy giảm
>> echo1 = zeros(1,endIndex);
>> delay1 = 30e-3;
>> delayIndex1 = ceil(delay1/Ts);
>> echo1(delayIndex1:endIndex) = 0.5*signal(1:endIndex ...
                                 -(delayIndex1-1));
>> echo2 = zeros(1,endIndex);
>> delay2 = 100e-3;
>> delayIndex2 = ceil(delay2/Ts);
>> echo2(delayIndex2:endIndex) = 0.1*signal(1:endIndex ...
                                 -(delayIndex2-1));
% Hỗn hợp tín hiệu có ích và tiếng vọng
>> compound = signal + echo1 + echo2;
% Tìm tự tương quan (autocorrelation) của tín hiệu hỗn hợp
>> cxx = xcorr(compound,'biased');

% Vẽ đồ thị phía trục dương
>> subplot(121)
>> plot(t(1:221),compound(1:221));
>> title('Total Signal','FontSize',12);
>> xlabel('Time [s]','FontSize',12); axis([0 0.005 -3 3]);
>> grid on;
>> subplot(122)
>> plot(t,cxx(endIndex:2*(endIndex)-1));
>> title('Autocorrelation','FontSize',12);
>> xlabel('Shifting k\cdot Ts','FontSize',12);
>> axis([0 0.5 -0.1 1.5]);
>> grid on;
```



Hình 5.7 *Hàm tự tương quan phục vụ tính thời gian chạy của tín hiệu vọng*

Ví dụ: Hàm tương quan chéo

Với sự trợ giúp của bốn máy phát, ta cần xác định được vị trí của một máy thu, bố trí cùng trong một phòng với các máy phát. Vị trí của các máy phát được coi là đã biết. Tất cả bốn máy phát cùng phát theo chu kỳ các chuỗi tín hiệu khác nhau, bao gồm hai giá trị -1 và +1. Máy thu thu được cả bốn tín hiệu đã bị trễ và tạp âm gây nhiễu. Bằng hàm tương quan chéo ta xác định được khoảng cách của máy thu tới các máy phát.

Trước hết ta tạo ra chuỗi Bits +1 và -1. Sau đó, ta khai báo các khoảng thời gian chạy của tín hiệu từ bốn máy phát và tạo các tín hiệu bị trễ với khoảng thời gian chạy đó. Cần chú ý là cả bốn máy đều phát tín hiệu lặp lại. Tín hiệu hỗn hợp từ bốn nguồn được xếp chồng thêm một thành phần tạp âm. Để tìm được khoảng cách (tức là: tìm được thời gian tín hiệu chạy từ máy phát tới máy thu), cần phải tìm hàm tương quan chéo giữa từng chuỗi Bits phát riêng rẽ với tín hiệu hỗn hợp mà máy phát thu được.

Chuỗi lệnh MATLAB thực hiện quá trình vừa mô tả và vẽ đồ thị ở hình 5.8 là như sau:

```
>> Fs = 10e3;                    % Tần số trích mẫu đo 10kHz
>> Ts = 1/Fs;                    % Chu kỳ trích mẫu đo
>> t = 0:Ts:0.5;
>> endIndex = length(t);

>> seq1 = sign(randn(1,5001)); % Bốn tín hiệu
>> seq2 = sign(randn(1,5001)); % phát giống nhau
>> seq3 = sign(randn(1,5001));
>> seq4 = sign(randn(1,5001));

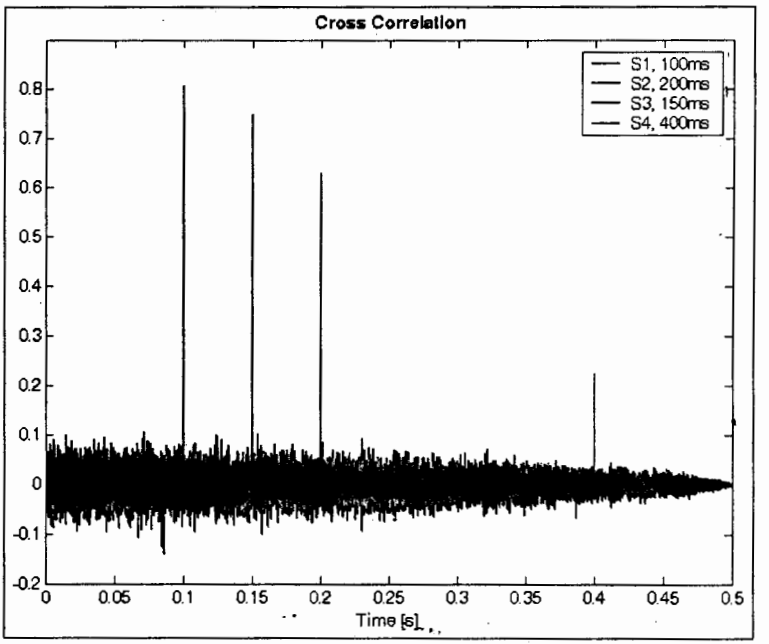
>> delay1 = 100e-3;              % Khai báo thời gian trễ của
>> delay2 = 200e-3;              % từng tín hiệu tại máy thu
>> delay3 = 150e-3;
>> delay4 = 400e-3;
>> delayIndex1 = ceil(delay1/Ts);
>> delayIndex2 = ceil(delay2/Ts);
>> delayIndex3 = ceil(delay3/Ts);
>> delayIndex4 = ceil(delay4/Ts);

% Tạo 4 tín hiệu bị trễ bắt được tại máy thu
>> delayedSeq1 = [seq1(length(seq1)-delayIndex1: ...
          length(seq1)) seq1(1:length(seq1)-delayIndex1)];
>> delayedSeq2 = [seq2(length(seq2)-delayIndex2: ...
          length(seq2)) seq2(1:length(seq2)-delayIndex2)];
>> delayedSeq3 = [seq3(length(seq3)-delayIndex3: ...
          length(seq3)) seq3(1:length(seq3)-delayIndex3)];
>> delayedSeq4 = [seq4(length(seq4)-delayIndex4: ...
          length(seq4)) seq4(1:length(seq4)-delayIndex4)];
>> noise = randn(1,endIndex);  % Vector tạp âm
```

```
% Tín hiệu hỗn hợp do máy thu thu được
>> compound = delayedSeq1(1:endIndex) ...
            + delayedSeq2(1:endIndex) ...
            + delayedSeq3(1:endIndex) ...
            + delayedSeq4(1:endIndex) + noise;

% Lần lượt tìm tương quan chéo giữa tín hiệu hỗn hợp
% compound và từng tín hiệu seq1, seq2, seq3 và seq4
>> cxy1 = xcorr(compound,seq1,'biased');
>> cxy2 = xcorr(compound,seq2,'biased');
>> cxy3 = xcorr(compound,seq3,'biased');
>> cxy4 = xcorr(compound,seq4,'biased');

% Biểu diễn các hàm tương quan chéo trên cùng một đồ thị
>> plot(t,cxy1((length(cxy1)-1)/2+1:length(cxy1)), ...
       t,cxy2((length(cxy2)-1)/2+1:length(cxy2)), ...
       t,cxy3((length(cxy3)-1)/2+1:length(cxy3)), ...
       t,cxy4((length(cxy4)-1)/2+1:length(cxy4)));
>> legend('S1, 100ms','S2, 200ms','S3, 150ms', ...
         'S4, 400ms',-1);
>> xlabel('Time [s]','FontSize',12);
>> axis([0 0.5 -0.2 0.9]);
>> title('Cross Correlation','FontSize',12);
```



Hình 5.8 *Hàm tương quan chéo sử dụng để tìm thời gian tín hiệu chạy*

Để xác định phổ của hàm tự tương quan và hàm tương quan chéo, ta sử dụng lệnh `csd` (*cross spectral density*). Lệnh `csd` tính phổ mật độ công suất tự tương quan và tương quan chéo.

`[Pxy f] = csd(x, y, nfft, fs, window)`

Giá trị mà `csd` trả lại là phổ mật độ công suất *Pxy* và các tần số kèm theo *f*. Bên cạnh hai tập giá trị đo *x* và *y*, ta còn có thể khai báo chiều dài của phép biến đổi Fourier cần thực hiện, tần số trích mẫu và loại hàm cửa sổ (mục 5.2.2) ta muốn sử dụng. Lệnh `csd` sẽ thực hiện nhiều phép biến đổi Fourier như số lượng giá trị đo. Phổ tổng hợp thu được là giá trị trung bình của các phổ riêng lẻ. Phổ mật độ công suất tự tương quan được tính bằng `csd(x,x, ...)`, hoặc `psd(x, ...)`. Cách sử dụng `psd` giống như `csd`.

Hàm tương quan	
`xcorr(x, y, 'options')`	Hàm tự tương quan và tương quan chéo
`csd(x, y, nfft, fs, window)`	Phổ mật độ công suất tương quan chéo
`psd(x, nfft, fs, window)`	Phổ mật độ công suất tự tương quan

5.4 Lọc số (Digital Filter)

Để có thể làm nhụt bớt hay tôn thêm một dải tần số nhất định nào đó của tín hiệu, ta thường sử dụng bộ lọc. Lọc số được phân thành hai loại FIR[1] và IIR[2]. Hàm truyền đạt của bộ lọc số có dạng tổng quát trên miền ảnh *z* sau đây:

$$H\left(z^{-1}\right)=\frac{y\left(z^{-1}\right)}{x\left(z^{-1}\right)}=\frac{B\left(z^{-1}\right)}{A\left(z^{-1}\right)}$$
$$=\frac{b_1+b_2z^{-1}+b_3z^{-2}+\cdots+b_{m+1}z^{-m}}{a_1+a_2z^{-1}+a_3z^{-2}+\cdots+a_{n+1}z^{-n}} \quad (5.9)$$

Trong (5.9), *x* và *y* là tín hiệu vào và tín hiệu ra của bộ lọc. Chuyển sang dạng phương trình sai phân, giá trị thứ *k* của đầu ra *y*(*k*) chính là tổng các giá trị quá khứ của đầu vào, đầu ra, sau khi nhân với một hệ số trọng lượng nhất định.

$$a_1y(k)=b_1x(k)+b_2x(k-1)+\cdots+b_{m+1}x(k-m)$$
$$-a_2y(k-1)-\cdots-a_{n+1}y(k-n) \quad (5.10)$$

Đối với bộ lọc FIR, các hệ số $a_2 = ... = a_{n+1} = 0$, vì giá trị đầu ra chỉ được tính từ các giá trị đầu vào. Dòng thứ 2 của (5.10) sẽ không tồn tại.

[1] Finite Impulse Response

[2] Infinite Impulse Response

5.4.1 Bộ lọc FIR và hàm cửa sổ

Bộ lọc FIR có đáp ứng xung hữu hạn và vì vậy (dưới giác độ hệ thống điều khiển) rất ổn định. Bộ lọc có tác dụng làm lệch tuyến tính góc pha của tín hiệu vào và thời gian trễ trung bình là $T_t = mT/2$, T là chu kỳ trích mẫu. So với bộ lọc IIR, nhược điểm chính của FIR là bậc m (số lượng các phần tử) khá cao.

Việc tính toán, thiết kế một bộ lọc FIR thông thấp được thực hiện nhờ lệnh:

```
fir1(order, limitfrequency, window)
```

Kết quả do lệnh `fir1` cung cấp là vector hệ số có độ dài m+1 (hình 5.9, bên trái, nét *). Sau khi chuẩn hóa, tần số có giá trị nằm trong khoảng 0...1, trong đó 1 ứng với ½ của tần số trích mẫu. Tham số *order* là bậc của bộ lọc, *limitfrequency* là tần số giới hạn và *window* đặc trưng cho hàm cửa sổ ta định sử dụng. Nếu biến *limitfrequency* chứa hai giá trị, ta có thể khai báo giới hạn trên và dưới của một dải tần số mà ta muốn cho đi qua hay chặn lại. Một vài cách sử dụng đặc trưng được tập hợp lại dưới đây:

`fir1(20,0.2)`	Lọc thông thấp: < $0{,}2F_{max}$
`fir1(20,0.2,'high')`	Lọc thông cao: > $0{,}2F_{max}$
`fir1(20,[0.2 0.4])`	Thông băng tần: $0{,}2F_{max}$ ... $0{,}4F_{max}$
`fir1(20,[0.2 0.4],'stop')`	Chặn băng tần: $0{,}2F_{max}$ ... $0{,}4F_{max}$

Hình 5.9 (bên phải) minh họa hàm truyền đạt của một bộ lọc thông thấp. Ta có thể nhận thấy hiện tượng mấp mô dạng sóng ở gần tần số tối đa F_{max} (hiện tượng có tên *Gibbs*). Hiện tượng đó có thể được hạn chế nhờ các hàm cửa sổ khác nhau. Lệnh `fir1` mặc định sử dụng cửa sổ *Hamming*. Trong mục 5.2 bạn đọc đã làm quen với tác dụng của hàm cửa sổ, nhằm hạn chế hiện tượng dò tần số khi thực hiện biến đổi Fourier. Hàm truyền đạt của bộ lọc được tìm nhờ lệnh:

```
freqz(num,den,points,samplingfreq)
```

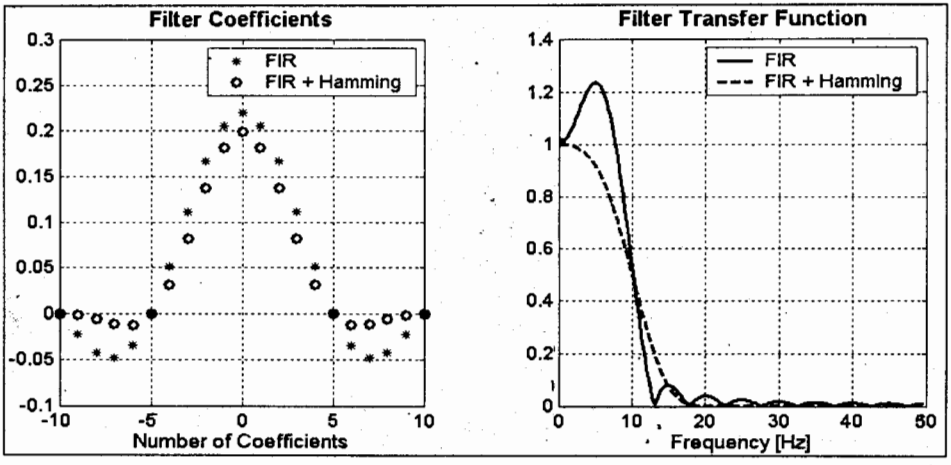
Lệnh `freqz` tính hàm (5.9) với $z^{-k} = \exp(-jk\omega T)$. Tham số *points* xác định độ phân giải, tần số trích mẫu *samplingfreq* được khai báo với thứ nguyên là Hz. Mẫu số *den* có giá trị 1 (chỉ đối với bộ lọc FIR), tử số *num* chứa chính các hệ số của bộ lọc vừa thu được sau khi thực hiện lệnh `fir1`. Khi thiết kế bộ lọc FIR cần lưu ý: Chiều dài của hàm cửa sổ bao giờ cũng là m+1. Ta hãy thực hiện ví dụ ở hình 5.9 bằng chùm lệnh dưới đây:

```
% Thiết kế bộ lọc
>> B = fir1(20,0.2,boxcar(20+1));
>> Bw = fir1(20,0.2,hamming(length(B)));
% Hàm truyền đạt của bộ lọc không có và có cửa sổ Hamming
>> [H,F] = freqz(B,1,200,100);
>> [Hw,Fw] = freqz(Bw,1,200,100);

% Vẽ hình 5.9
>> subplot(121);                          % Tạo nửa hình trái
>> number = -10:1:10;
```

```
>> plot(number,B,'r*',number,Bw,'go');  % Biểu đồ hệ số
>> grid on;
>> title('Filter Coefficients','FontSize',12);
>> xlabel('Number of Coefficients','FontSize',12);
>> legend('FIR','FIR + Hamming');
>> axis([-10 10 -0.1 0.3]);

>> subplot(122);                        % Tạo nửa hình phải
>> plot (F, abs(H),'r-',Fw, abs(Hw),'g--');
>> grid on;
>> title('Filter Transfer Function','FontSize',12);
>> xlabel('Frequency [Hz]','FontSize',12);
>> legend('FIR','FIR + Hamming');
```



Hình 5.9 *Hệ số (hình trái) và hàm truyền đạt (hình phải) của bộ lọc FIR khi không dùng và có dùng hàm cửa sổ Hamming*

Bộ lọc FIR ở hình 5.9 có bậc $m = 20$, tần số trích mẫu là $F = 100$Hz và tần số giới hạn là$(F/2).0,2 = 10$Hz. Với bộ lọc FIR dùng cửa sổ *Hamming* (có tên `Bw`) vừa được thiết kế, ta lọc tín hiệu x ở hình 5.4 (bên trái) bằng lệnh:

`filter(`*num*`,`*den*`,`*data*`)`

Trong lệnh trên, biến *num* là vector hệ số của đa thức tử số (chính là `Bw`, kết quả của lệnh thiết kế `fir1`), còn *den* là vector hệ số của đa thức mẫu số. Vì ta thiết kế bộ lọc FIR, *den* sẽ chỉ có giá trị 1. Trên hình 5.10 ta nhận thấy rất rõ quá trình quá độ của bộ lọc FIR khi dùng lệnh `filter`, và cả thời gian trễ gần ½ chu kỳ ($T_t = 20.0,005s/2 = 0,05s$).

Để khắc phục quán tính (gây nên quá trình quá độ) và thời gian trễ, ta sử dụng lệnh:

`filtfilt(`*num*`,`*den*`,`*data*`)`

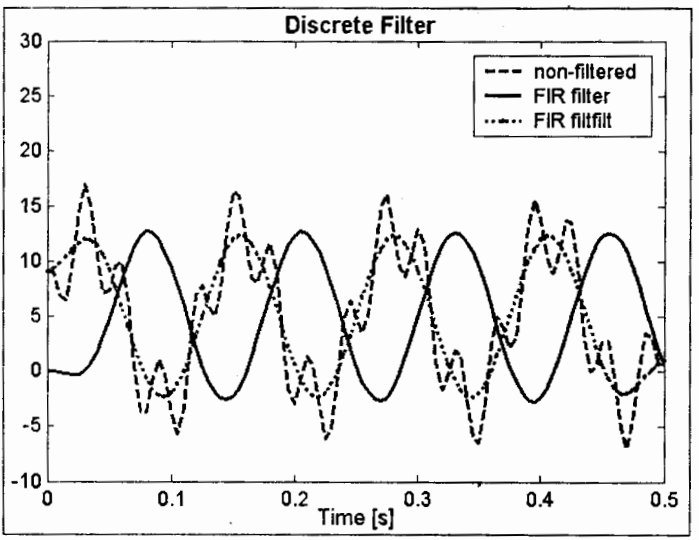
Thực chất, lệnh `filtfilt` là một *m-File* sử dụng lệnh `filter` đối với tập số liệu *x* như bình thường. Tiếp theo, chuỗi số liệu đã qua lọc lại được đưa ngược trở lại đầu vào để qua lọc một lần nữa. Vì qua lọc FIR hai lần, bạn đọc có thể nhận thấy sự suy giảm nhẹ biên độ (hình 5.10) của tín hiệu ra. Khi sử dụng `filtfilt` ta phải lưu ý: `filtfilt` đòi hỏi *chiều dài của tập số liệu x phải lớn hơn ít nhất 3 lần* so với bậc *m* của FIR. Trong ví dụ dưới đây, do ta khai báo `t=0:0.005:0.5`, *x* sẽ là một vector với 101 phần tử (độ dài: 101), bảo đảm lớn hơn 3 lần so với bậc *m*=20 của bộ lọc FIR mang tên `Bw`:

```
% Tạo tập số liệu x có chiều dài length(x)=101
>> t = 0:0.005:0.5;
>> x = 5 + 8*sin(2*pi*8*t) + 4*cos(2*pi*33*t);

% Thiết kế bộ lọc FIR
>> Bw = fir1(20,0.2,hamming(20+1));

% Dùng Bw để lọc x theo 2 cách: filter và filtfilt
>> x_f = filter(Bw,1,x);
>> x_ff = filtfilt(Bw,1,x);

% Vẽ đồ thị các tín hiệu chưa qua và đã qua lọc (hình 5.10)
>> plot(t,x,'g-',t,x_f,'r-',t,x_ff,'b.');
>> axis([0 0.5 -10 30]);
>> title('Discrete Filter','FontSize',12);
>> xlabel('Time [s]','FontSize',12);
>> legend('non-filtered','FIR filter','FIR filtfilt');
```



Hình 5.10 *Tín hiệu không qua lọc và tín hiệu qua lọc FIR dùng* `filter` *và* `filtfilt`

Bộ lọc FIR và hàm cửa sổ	
`fir1(`*order, limitfrequency, window*`)`	Thiết kế bộ lọc FIR (lọc thông thấp)
`filter(`*num,den,data*`)`	Lọc số liệu
`filtfilt(`*num,den,data*`)`	Lọc số liệu có hiệu chỉnh pha
`freqz(`*num,den,points,samplingfreq*`)`	Đáp ứng tần số gián đoạn

5.4.2 Bộ lọc IIR

Khác với FIR, bộ lọc IIR hoạt động theo phương thức đệ quy (*recursive*), có sử dụng các giá trị quá khứ của đầu ra. Bằng cách đó, ta có thể thực hiện bộ lọc chỉ cần bậc thấp nhưng dải thông có độ dốc sườn trước và sau giống nhau. Chính vì bậc thấp, IIR rất hay được sử dụng trong các hệ thống trích mẫu, đặc biệt là các hệ thống điều khiển tự động.

Bên cạnh đòi hỏi dải thông có độ dốc sườn giống nhau, mức độ mấp mô trong dải thông hay dải chặn cũng là một tiêu chuẩn hay được đặt ra. Chính vì vậy, ta thường phân biệt các loại bộ lọc khác nhau, đáp ứng những tiêu chuẩn nhất định đặt ra. Ta có thể thiết kế bộ lọc IIR bằng một trong các công cụ sau:

`butter(`*order,limitfreq*`)`
`cheby1(`*order,ripple,limitfreq*`)`
`cheby2(`*order,ripple,limitfreq*`)`
`ellip(`*order,ripple,attenuation,limitfreq*`)`

Khi sử dụng lệnh `cheby1`, ta khai báo mức mấp mô *ripple* của dải thông, đối với `cheby2`, ta khai báo mức mấp mô *ripple* của dải chặn. Mức mấp mô *ripple* và độ suy giảm *attenuation* được khai báo bằng dB[1] (decibel). Tùy theo lựa chọn, kết quả thiết kế có thể được trả lại dưới dạng đa thức tử số / mẫu số của hàm truyền đạt, hay điểm không / điểm cực của mô hình trạng thái (đều trên miền ảnh z). MATLAB phân biệt đòi hỏi trả kết quả thông qua số lượng biến nhận kết quả mà ta khai. Tương tự FIR, tần số được chuẩn hóa bởi giá trị $F_{max} = F/2$. Việc khai báo thiết kế lọc thông cao hay thông thấp, lọc thông băng hay chặn băng cũng tương tự như FIR, ví dụ thiết kế một bộ lọc IIR bậc 6:

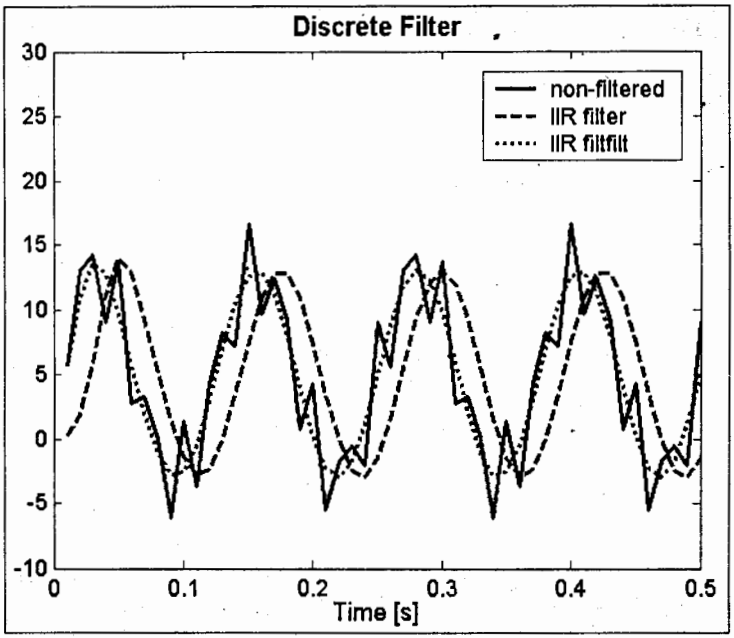
```
% Lọc thông thấp. Xuất kết quả: dạng hàm truyền đạt
>> [B,A] =butter(6,0.6);
% Lọc thông cao, độ mấp mô 3dB. Xuất kết quả: mô hình
% điểm không, điểm cực và hệ số khuếch đại
>> [Z,P,K] =cheby1(6,3,0.6,'high');
% Lọc chặn băng. Xuất kết quả: mô hình trạng thái
>> [A,B,C,D] =ellip(6,3,20,[0.3 0.6],'stop');
```

Ví dụ sau minh họa việc thiết kế và sử dụng một bộ lọc IIR *Butterworth* bậc 4 để loại thành phần tần số 40Hz ra khỏi tín hiệu có ích với tần số 8Hz.

[1] Mức mấp mô hay độ suy giảm 20dB ứng với 1/10 biên độ của tín hiệu chưa qua lọc

Tần số giới hạn của bộ lọc được chọn là 20Hz. Kết quả (hình 5.11) cho thấy: Chỉ với bộ lọc bậc 4 (khá thấp) ta đã thu được kết quả lọc rất tốt.

```
>> t = 0.01:0.01:1;
>> x = 5 + 8*sin(2*pi*8*t) + 4*cos(2*pi*40*t);
>> [B,A] = butter(4,20/50);     % Thiết kế bộ lọc IIR
>> x_f = filter(B,A,x);         % Lọc tín hiệu x
>> x_ff = filtfilt(B,A,x);      % Lọc tín hiệu x có bù pha
>> plot(t,x,'g-',t,x_f,'r-',t,x_ff,'b:');
>> axis([0 0.5 -10 30]);
>> title('Discrete Filter','FontSize',12);
>> xlabel('Time [s]','FontSize',12);
>> legend('non-filtered','IIR filter','IIR filtfilt');
```



Hình 5.11 Tín hiệu không qua lọc và tín hiệu qua lọc IIR kiểu Butterworth bậc 4

Bộ lọc IIR	
`butter(`*order*`,`*limitfreq*`)`	Thiết kế lọc *Butterworth*
`cheby1(`*order*`,`*ripple*`,`*limitfreq*`)`	Thiết kế lọc *Tschebyscheff* Typ 1
`cheby2(`*order*`,`*ripple*`,`*limitfreq*`)`	Thiết kế lọc *Tschebyscheff* Typ 2
`ellip(`*order*`,`*ripple*`,`*attenuation*`,`*limitfreq*`)`	Thiết kế lọc *Elliptic* (*Cauer*)
`filter(`*num*`,`*den*`,`*data*`)`	Lọc số liệu
`filtfilt(`*num*`,`*den*`,`*data*`)`	Lọc số liệu có hiệu chỉnh pha
`freqz(`*num*`,`*den*`,`*points*`,`*samplingfreq*`)`	Đáp ứng tần số gián đoạn

5.5 Lọc tương tự (Analog Filter)

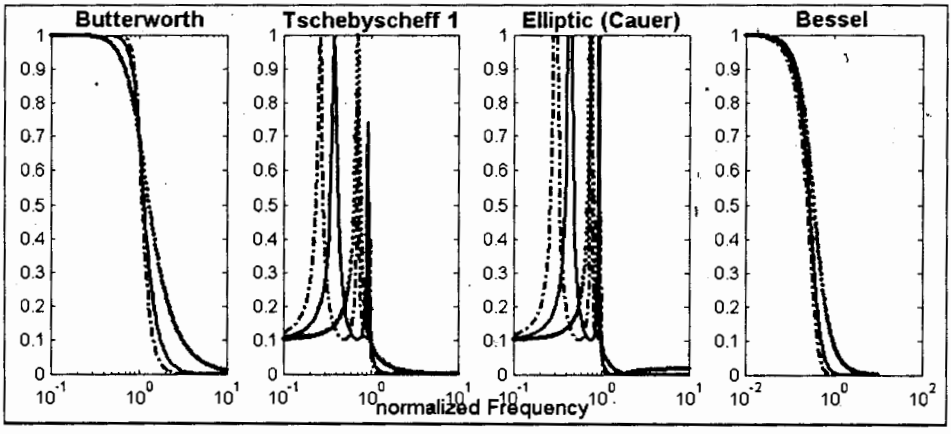
Bộ lọc tương tự đôi khi được sử dụng để mô phỏng các hệ thống liên tục, ví dụ: mô phỏng với SIMULINK. Việc thiết kế có thể được thực hiện với các lệnh thiết kế lọc IIR đã biết, chỉ cần khai báo thêm tham số `'s'`. Ngoài ra ta có thêm lệnh:

`besself(`*order*`,`*omega*`)`

Tần số giới hạn *omega* được khai báo với thứ nguyên `rad/s` và có thể mang giá trị dương bất kỳ.

```
>> [B,A] = butter(4,1,'s');        % Lọc Butterworth analog
>> [H,W] = freqs(B,A);             % Tìm hàm truyền đạt và
                                   % vector tần số
>> semilogx (W,abs(H));            % Xuất đặc tính tần số
```

Hình 5.12 minh họa đặc điểm truyền đạt của một vài khâu lọc tương tự với các bậc 2, 4 và 6. Đặc tính truyền đạt được tính bằng lệnh `freqs(`*num*`,`*den*`)` như ta đã biết. Kết quả thu được có thứ nguyên là `rad/s`.



Hình 5.12 *Đặc tính truyền đạt của vài bộ lọc tương tự có bậc 2, 4 và 6*

Lọc tương tự (Analog Filter)	
`besself(`*order*`,`*omega*`)`	Thiết kế lọc *Bessel* (analog)
`butter(`*order*`,`*limitfreq*`,'s')`	Thiết kế lọc *Butterworth*
`cheby1(`*order*`,`*ripple*`,`*limitfreq*`,'s')`	Thiết kế lọc *Tschebyscheff* Typ 1
`cheby2(`*order*`,`*ripple*`,`*limitfreq*`,'s')`	Thiết kế lọc *Tschebyscheff* Typ 2
`ellip(`*order*`,`*ripple*`,`*attenuation*`,`*limitfreq*`,'s')`	Thiết kế lọc *Elliptic* (*Cauer*)
`freqs(`*num*`,`*den*`,`*func*`)`	Đặc ứng tần số (analog)

5.6 Tóm tắt nội dung chương 5

Sau khi đã nghiên cứu kỹ chương 5, người đọc cần nắm vững các nội dung sau đây:

1. Nội suy các tập số liệu đo một hoặc nhiều chiều ?
2. Phân tích tín hiệu bằng công cụ biến đổi Fourier gián đoạn ?
3. Tìm chuỗi Fourier (ảnh Fourier) của tín hiệu tuần hoàn từ kết quả biến đổi Fourier ?
4. Thực hiện Averaging để hạn chế hiện tượng dò tần số ?
5. Tính hàm tự tương quan và hàm tương quan chéo của các tín hiệu thu từ phép trích mẫu ?
6. Thiết kế bộ lọc FIR có sử dụng các hàm cửa sổ khác nhau ?
7. Thiết kế các loại bộ lọc IIR ?
8. Thiết kế các bộ lọc tương tự (analog) phục vụ mô phỏng xấp xỉ liên tục ?

6 Cơ sở về SIMULINK

SIMULINK là phần chương trình mở rộng của MATLAB nhằm mục đích mô hình hóa, mô phỏng và khảo sát các hệ thống động học. Giao diện đồ họa trên màn hình của SIMULINK cho phép thể hiện hệ thống dưới dạng sơ đồ tín hiệu với các khối chức năng quen thuộc. SIMULINK cung cấp cho người sử dụng một thư viện rất phong phú, có sẵn với số lượng lớn các khối chức năng cho các hệ tuyến tính, phi tuyến và gián đoạn. Hơn thế, người sử dụng cũng có thể tạo nên các khối riêng của mình.

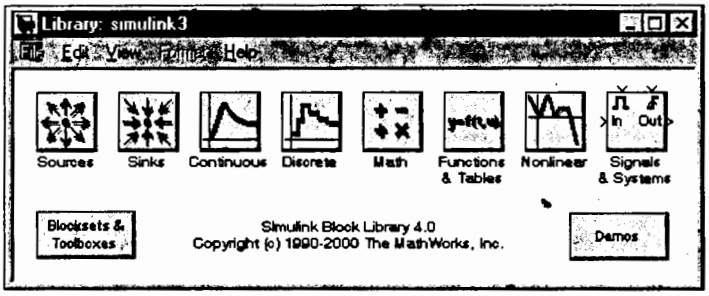
Trong chương này bạn đọc sẽ làm quen với các đặc điểm cơ bản của SIMULINK, với cách thức tạo nên và sử dụng một mô hình mô phỏng. Ngoài ra, chương 6 sẽ giới thiệu chi tiết một số thư viện quan trọng của SIMULINK.

Sau khi đã xây dựng mô hình của hệ thống cần nghiên cứu, bằng cách ghép các khối cần thiết thành sơ đồ cấu trúc của hệ, ta có thể khởi động quá trình mô phỏng. Trong quá trình mô phỏng ta có thể trích tín hiệu tại vị trí bất kỳ của sơ đồ cấu trúc và hiển thị đặc tính của tín hiệu đó trên màn hình. Hơn thế nữa, nếu có nhu cầu ta còn có thể cất giữ các đặc tính đó vào môi trường nhớ (ví dụ: cất lên đĩa cứng). Việc nhập hoặc thay đổi tham số của tất cả các khối cũng có thể được thực hiện rất đơn giản bằng cách nhập trực tiếp hay thông qua MATLAB. Để khảo sát hệ thống, ta có thể sử dụng thêm các *Toolbox* như *Signal Processing* (xử lý tín hiệu), *Optimization* (tối ưu) hay *Control System* (hệ thống điều khiển).

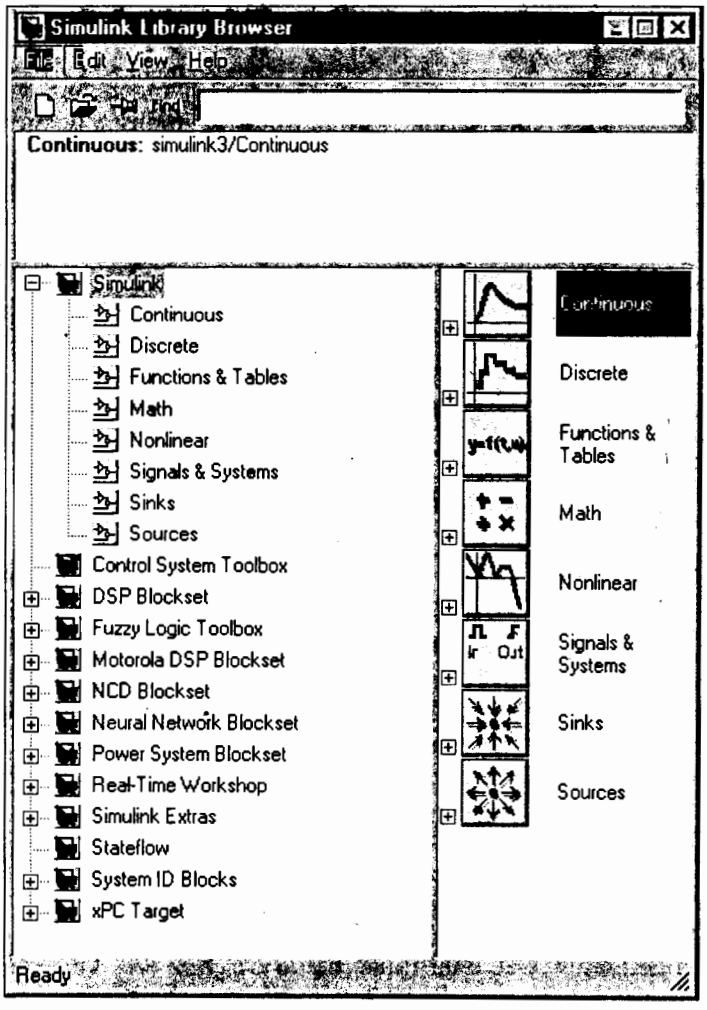
6.1 Khởi động SIMULINK

Để có thể làm việc với SIMULINK, trước hết ta phải khởi động MATLAB (mục 1.1). Nếu chạy dưới hệ điều hành LINUX, sau khi thực hiện lệnh *simulink3* ta sẽ thu được cửa sổ thư viện của SIMULINK (hình 6.1). Nếu làm việc dưới WINDOWS, sau khi gọi *simulink* ta có cửa sổ tra cứu (*Browser*) thư viện như hình 6.2. Nếu nháy phím chuột phải vào dòng "*Simulink*" của nửa cửa sổ phía trái hình 6.2 ta thu được cửa sổ thư viện với nội dung giống như hình 6.1.

Các thư viện con *Sources* (các khối nguồn tín hiệu), *Sinks* (các khối xuất tín hiệu), *Math* (các khối ghép nối toán học) và *Signals & Systems* (các khối tín hiệu và hệ con) sẽ được giới thiệu trong phạm vi chương này. Các nhóm *Continuous* (các khối liên tục), *Nonlinear* (các khối phi tuyến) và *Functions & Tables* là đối tượng của chương 7. Các khối thuộc nhóm *Discrete* (các khối gián đoạn) được đề cập đến trong chương 8.

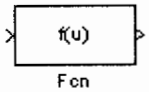


Hình 6.1 *Cửa sổ thư viện các nhóm chức năng của SIMULINK*



Hình 6.2 *Cửa sổ tra cứu (Browser) thư viện của SIMULINK*

Tính chất của các khối chức năng



***Hình 6.3** Khối chức năng của SIMULINK*

Tất cả các khối chức năng đều được xây dựng theo một mẫu (hình 6.3) giống nhau: Mỗi khối có một hay nhiều đầu vào/ra (trừ trường hợp ngoại lệ: các khối thuộc hai thư viện con *Sources* và *Sinks*), có tên và ở trung tâm của hình khối chữ nhật có biểu tượng (hàm truyền đạt, đồ thị đặc tính hay tên *file*) thể hiện đặc điểm riêng của khối. Người sử dụng có thể tùy ý thay đổi tên của khối (nháy kép phím chuột trái vào vị trí tên), tuy nhiên, mỗi tên chỉ có thể sử dụng một lần duy nhất trong phạm vi cửa sổ mô hình mô phỏng. Khi nháy kép phím chuột trái trực tiếp vào khối ta sẽ mở cửa sổ tham số *Block Parameters* (trừ các khối *Scope*, *Slider Gain*, *Subsystem*) và có thể nhập thủ công các tham số đặc trưng của khối. Khi đã nhập xong, nháy chuột trái vào nút *OK* (khép cửa sổ) hay nút *Apply* (cửa sổ vẫn mở) để SIMULINK chấp nhận các tham số vừa nhập. Nếu nháy kép phím chuột trái vào nút *Help* ta sẽ mở cửa sổ của tiện ích trợ giúp trực tuyến. Nháy một lần phím chuột phải trực tiếp vào khối có tác dụng mở menu chứa các lệnh (ví dụ) cho phép soạn thảo và lập định dạng (*format*) khối.

SIMULINK phân biệt (không phụ thuộc vào thư viện con) hai loại khối chức năng: *khối ảo* (*virtual*) và *khối thực* (*not virtual*). Các khối thực đóng vai trò quyết định khi chạy mô phỏng mô hình SIMULINK. Việc thêm hay bớt một khối thực sẽ thay đổi đặc tính động học của hệ thống đang được mô hình SIMULINK mô tả. Có thể nêu nhiều ví dụ về khối thực như: Khối tích phân *Integrator* hay khối hàm truyền đạt *Transfer Fcn* của thư viện con *Continuous* (mục 7.1), khối *Sum* hay khối *Product* của thư viện con *Math* (mục 6.5). Ngược lại, các khối ảo không có khả năng thay đổi đặc tính của hệ thống, chúng chỉ có nhiệm vụ thay đổi diện mạo đồ họa của mô hình SIMULINK. Đó chính là các khối như *Mux*, *Demux* hay *Enable* thuộc thư viện con *Signal & System* (mục 6.7.2). Một số khối chức năng mang đặc tính ảo hay thực tùy thuộc theo vị trí hay cách thức sử dụng chúng trong mô hình SIMULINK, các mô hình đó được xếp vào loại *ảo có điều kiện*. Khối chức năng nào, với đặc điểm thuộc loại nào, khi đề cập đến từng khối cụ thể ta sẽ quay trở lại trả lời câu hỏi này.

Mô hình SIMULINK

Từ cửa sổ thư viện khối (*Library*) hay từ cửa sổ truy cập thư viện (*Library Browser*) ta có thể tạo các cửa sổ mô phỏng mới bằng cách đi theo menu *File / New / Model*, hoặc mở các *File* đã có sẵn qua menu *File / Open*. Một *File* SIMULINK khi được cất giữ (menu *File / Save* hoặc *File / Save As* trong cửa sổ mô phỏng) sẽ có đuôi *.mdl*.

6.2 Tạo mới và soạn thảo lưu đồ tín hiệu

SIMULINK gần như chỉ có thể sử dụng được nhờ chuột. Bằng cách nháy kép phím chuột trái vào một trong số các thư viện con thuộc cửa sổ thư viện chính *Library* ta sẽ thu được một cửa sổ mới có chứa các khối thuộc thư viện con đó. Hoặc cũng có thể thu được kết quả tương tự bằng cách nháy kép chuột trái vào nhánh của thư viện con, nằm ở phần bên phải của cửa sổ truy cập *Library Browser*. Từ các khối chứa trong thư viện con ta có thể xây dựng được lưu đồ tín hiệu mong muốn. Để tạo định dạng (*Format*) và soạn thảo ta có các khả năng sau đây:

- **Sao chép**: Bằng cách gắp và thả "*Drag & Drop*" nhờ phím chuột phải ta có thể chép một khối từ thư viện con (cũng có thể từ một cửa sổ khác ngoài thư viện).
- **Di chuyển**: Ta có thể dễ dàng di chuyển một khối trong phạm vi cửa sổ của khối đó nhờ phím chuột trái.
- **Đánh dấu**: Bằng cách nháy phím chuột trái vào khối ta có thể đánh dấu, lựa chọn từng khối, hoặc kéo chuột đánh dấu nhiều khối một lúc.
- **Xóa**: Có thể xóa các khối và các đường nối đã bị đánh dấu bằng cách gọi lệnh menu *Edit / Clear*. Bằng menu *Edit / Undo* hoặc tổ hợp phím *Ctrl+Z* ta có thể cứu vãn lại động tác xóa vừa thực hiện.
- **Hệ thống con** (*Subsystem*): Bằng cách đánh dấu nhiều khối có quan hệ chức năng, sau đó gom chúng lại thông qua menu *Edit / Create Subsystem*, ta có thể tạo ra một hệ thống con mới (xem mục 6.7).
- **Nối hai khối**: Dùng phím chuột trái nháy vào đầu ra của một khối, sau đó di mũi tên của chuột (giữ nguyên trạng thái nhấn phím chuột trái) tới đầu vào cần nối. Sau khi thả ngón tay khỏi phím chuột, đường nối tự động được tạo ra. Có thể rẽ nhánh tín hiệu bằng cách nháy phím chuột phải vào một đường nối có sẵn và (giữ nguyên trạng thái nhấn phím chuột) kéo đường nối mới xuất hiện tới đầu vào cần nối.
- **Di chuyển đường nối**: Để lưu đồ tín hiệu thoáng và dễ theo dõi, nhiều khi ta phải di chuyển, bố trí lại vị trí các đường nối. Khi nháy chọn bằng chuột trái ta có thể di chuyển tùy ý các điểm góc hoặc di chuyển song song từng đoạn thẳng của đường nối.
- **Tạo vector đường nối**: Để dễ phân biệt giữa đường nối đơn (*scalar*) và đường nối các tín hiệu theo định dạng vector, hoặc ma trận, hoặc mảng (*Array*), ta có thể chọn menu *Format / Wide nonscalar lines* để tăng bề dầy (tăng độ đậm nhạt) của đường nối.
- **Chỉ thị kích cỡ và dạng dữ liệu của tín hiệu**: Lệnh chọn qua menu *Format / Signal dimensions* sẽ hiển thị kích cỡ (mục 6.3.1) của tín hiệu

đi qua đường nối. Lệnh menu *Format / Port data types* chỉ thị thêm loại dữ liệu (mục 6.3.2) của tín hiệu qua đường nối.

- **Định dạng** (*Format*) **cho một khối**: Sau khi nháy phím chuột phải vào một khối, cửa sổ định dạng khối sẽ mở ra. Tại mục *Format* ta có thể lựa chọn kiểu và kích cỡ chữ, cũng như vị trí của tên khối, có thể lật hoặc xoay khối. Hai mục *Foreground Color* và *Background Color* cho phép ta đặt chế độ mầu bao quanh cũng như mầu nền của khối.
- **Định dạng cho đường nối**: Sau khi nháy phím chuột phải vào một đường nối, cửa sổ định dạng đường (của cả đường dẫn tới đường nối đó) sẽ mở ra. Tại đây ta có các lệnh cho phép cắt bỏ, chép (*Copy*) hoặc xóa đường nối.
- **Hộp đối thoại** (*Dialog Box*) **về đặc tính của khối** (*Block Properties*): Hoặc đi theo menu của cửa sổ mô phỏng *Edit / Block Properties*, hoặc chọn mục *Block Properties* của cửa sổ định dạng khối, ta sẽ thu được hộp đối thoại cho phép đặt một vài tham số tổng quát về đặc tính của khối. Ví dụ: ở mục *Description* ta có thể viết vài lời mô tả ngắn về khối, hay ở mục *Open Function* ta có thể nhập tên của MATLAB *File* (*m-File*) sẽ được kích hoạt khi nháy kép chuột vào khối.
- **Hộp đối thoại về đặc tính của tín hiệu** (*Signal Properties*): Có thể tới được hộp thoại *Signal Properties* của một đường nối (của tín hiệu) hoặc bằng cách nháy chuột đánh dấu đường nối trên cửa sổ mô phỏng, sau đó đi theo menu *Edit / Signal Properties*, hoặc chọn mục *Signal Properties* từ cửa sổ định dạng đường. Trong hộp đối thoại ta có thể đặt tên cho đường nối hoặc nhập một đoạn văn bản mô tả. Tuy nhiên, để đặt tên cho đường nối cũng còn cách khác đơn giản hơn: Nháy kép phím chuột trái vào đường nối ta sẽ tự động tới được chế độ nhập văn bản.

6.3 Tín hiệu và các loại dữ liệu

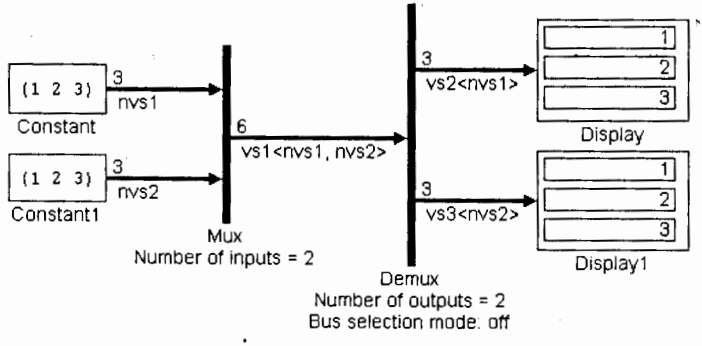
6.3.1 Làm việc với tín hiệu

Đối với SIMULINK, khái niệm tín hiệu nhằm chỉ vào dữ liệu xuất hiện ở đầu ra của các khối chức năng trong quá trình mô phỏng. Ta tạm hình dung rằng: Các tín hiệu (các dữ liệu) đó *chạy dọc theo đường nối từ đầu ra của khối chức năng này tới đầu vào của các khối chức năng khác mà không tốn thời gian*. Tín hiệu trong khuôn khổ MATLAB có những đặc điểm riêng do người sử dụng xác định, ví dụ: đặt tên (mục 6.2), quy định kích cỡ (tiếp sau đây) hay loại dữ liệu (mục 6.3.2).

Trong SIMULINK ta phân biệt 3 loại kích cỡ tín hiệu:

- *Tín hiệu đơn* (*scalar*).
- *Vector tín hiệu*: Còn được gọi là tín hiệu 1-D, vì kích cỡ của tín hiệu chỉ được xác định theo một chiều với độ dài n.
- *Ma trận tín hiệu*: Còn được gọi là tín hiệu 2-D, vì kích cỡ của tín hiệu được xác định theo hai chiều [m×n]. Cả vector hàng [1×n] và vector cột [m×1] cũng thuộc về phạm trù ma trận tín hiệu. Đôi khi, ví dụ: lúc khai báo định dạng, ma trận cũng được gọi là mảng (*Arrays*).

Tùy từng trường hợp cụ thể, các khối chức năng có thể chấp nhận ở đầu vào (hay xuất ở đầu ra) các tín hiệu với một hay nhiều kích cỡ khác nhau.



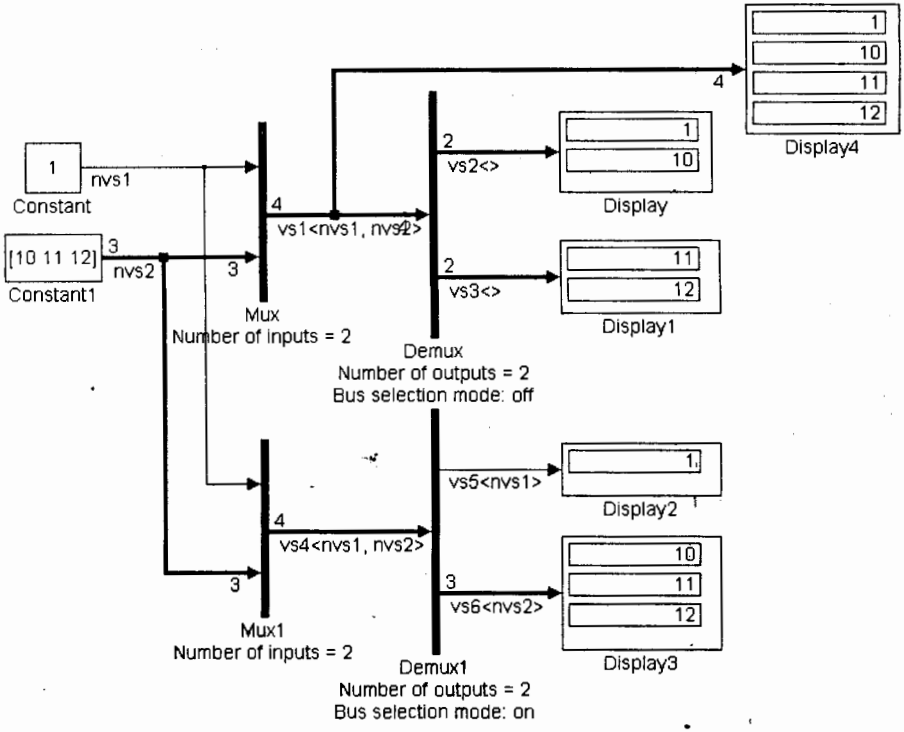
Hình 6.4 *Ví dụ về tín hiệu ảo trong sơ đồ mô phỏng SIMULINK*

Khi tạo một cấu trúc SIMULINK, các khối ảo sẽ tạo nên các đường *tín hiệu ảo* (hình 6.4), duy nhất nhằm mục đích làm cho sơ đồ cấu trúc trở nên đỡ rối mắt, người sử dụng dễ quản lý hơn. Tín hiệu ảo có thể được coi là sự tập hợp hình ảnh của nhiều tín hiệu ảo, không ảo, hay hỗn hợp cả hai loại. Trong quá trình mô phỏng, SIMULINK sử dụng một thủ tục tên *signal propagation* để nhận biết: Những tín hiệu thực (không ảo) nào được ghép vào (chùm) tín hiệu ảo. Diễn đạt một cách khác: Những khối chức năng nào được ghép thực sự ở đầu cuối của tín hiệu. Ví dụ ở hình 6.4 minh họa nội dung trên. Các tín hiệu thực *nvs1* và *nvs2* được các khối *Constant* và *Constant1* phát ra và được bộ dồn kênh (*multiplexer*) *Mux* ghép lại thành tín hiệu *vs1*[1] (vector với chiều dài 6). Sau khi chọn *Number of Outputs* = 2, bộ phân kênh *Demux* sẽ lại tách *vs1* trở lại hai thành phần tín hiệu ban đầu. *vs1* là một tín hiệu ảo vì *vs1* chỉ là đại diện hình học của *nvs1* và *nvs2* trên sơ đồ cấu trúc. Vì lý do ấy, hai đầu ra của *Demux* là *vs2* và *vs3* sẽ phải là tín hiệu ảo. Tuy nhiên, bằng thủ tục *signal propagation*, SIMULINK nhận biết rằng *vs1* cấu tạo bởi *nvs1*, *nvs2* (tự động bổ sung vào nhãn của tên *vs1* ký hiệu <*nvs1*, *nvs2*>) và *vs2*, *vs3* thực chất là *nvs1*, *nvs2* (tự động bổ sung vào nhãn của tên *vs2*, *vs3* ký hiệu <*nvs1*>, <*nvs2*>). Nghĩa là, SIMULINK biết rằng các khối *Constant*, *Constant1* nối với *Display*, *Display1*.

[1] vs, nvs: virtual signal, not virtual signal (tín hiệu ảo, tín hiệu thực)

Đối với các tín hiệu ảo ta có thể mở hộp thoại *Signal Properties* và khai chọn *Show propagated signals*. Sau khi khai chọn (như hình 6.4 cho thấy) tên của tín hiệu sẽ tự động được bổ sung thêm phần (trong ngoặc < >), cho biết các tín hiệu chứa trong đó.

Bus tín hiệu (*signal buses*) là tập hợp các tín hiệu ảo riêng rẽ. Khi tách Bus bởi bộ phân kênh *Demux* ta sẽ không thể truy cập vào từng phần tử của mỗi tín hiệu, mà chỉ có thể truy cập vào từng tín hiệu. Ví dụ sau minh họa rõ điều này: Trong sơ đồ ở hình 6.5, hai bộ dồn kênh *Mux* và *Mux1* nhận đầu vào là *nvs1* và *nvs2*.



Hình 6.5 *Ví dụ về tín hiệu ảo và Bus tín hiệu trong sơ đồ mô phỏng SIMULINK*

Trong hình 6.5, chế độ *Bus selection mode* của bộ phân kênh *Demux* bị đặt là *off*, vì vậy *Demux* chỉ có thể tách vector *vs1* thành hai tín hiệu *vs2* và *vs3* với kích cỡ là 2. Tuy nhiên, trên cửa sổ nhập lệnh (*Command Windows*) của MATLAB ta sẽ nhận được cảnh báo là MATLAB không thể thực hiện *signal propagation* đối với *vs2* và *vs3*. Chính vì vậy, *vs2* và *vs3* chỉ được bổ sung thêm vào tên dấu ngoặc < > rỗng. Ngược lại, *Bus selection mode* của *Demux1* được khai báo *on*; do đó *vs4* là Bus tín hiệu. Do đó *Demux1* không thể tách *vs4* thành hai tín hiệu với kích cỡ là 2, mà chỉ có thể tách thành hai tín hiệu *vs5* và *vs6* ứng với xuất sứ của chúng là *nvs1* và *nvs2*.

Chức năng của *Mux* và *Demux* được đề cập đến kỹ hơn ở mục 6.7.2.

6.3.2 Làm việc với các loại số liệu

Bên cạnh các đặc điểm đã được giới thiệu, mỗi tín hiệu thuộc sơ đồ cấu trúc SIMULINK đều được gán một loại số liệu nhất định, và do đó quyết định đến dung lượng bộ nhớ dành cho một tín hiệu. SIMULINK cũng hỗ trợ tất cả các loại số liệu của MATLAB.

- *double* (*double-precision floating point*: chính xác cao, dấu phẩy động).
- *single* (*single-precision floating point*: chính xác vừa, dấu phẩy động).
- *int8*, *uint8*, *int16*, *uint16*, *int32*, *uint32* (*signed / unsigned* 8-, 16- or 32-*bit integer*: số nguyên 8-, 16- hay 32- bit có / không có dấu).
- *boolean* (0 hoặc 1, logic, được SIMULINK xử lý như *uint8*).

Loại số liệu mặc định sẵn của SIMULINK là *double*. Trong quá trình mô phỏng, SIMULINK sẽ kiểm tra xem việc đảo giữa các loại số liệu (nếu sơ đồ mô phỏng sử dụng nhiều loại số liệu khác nhau) có đúng hay không, nhằm loại trừ các kết quả sai lầm có thể xảy ra.

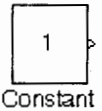
Khả năng khai báo, xác định loại số liệu của tín hiệu cũng như của tham số thuộc các khối chức năng trong SIMULINK là đặc biệt có ý nghĩa, nếu ta dự định tạo ra từ mô hình SIMULINK mã chạy cho các ứng dụng thời gian thực. Nhu cầu về bộ nhớ và tốc độ tính toán phụ thuộc vào loại số liệu được ta chọn.

6.4 Thư viện SOURCES và SINKS

6.4.1 Thư viện Sources

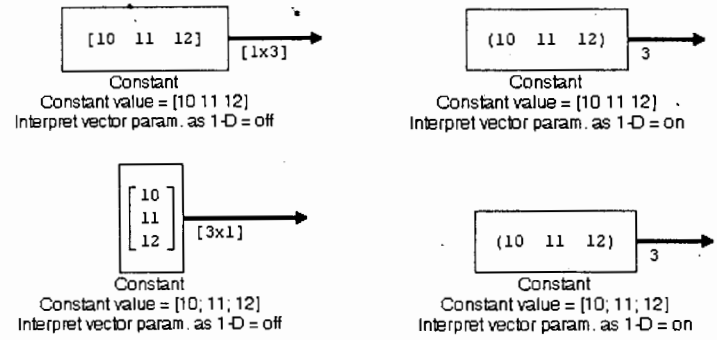
Khi nháy chuột kép vào ký hiệu *Sources*, cửa sổ của thư viện con với các khối chức năng sẽ mở ra. Các khối chuẩn trong đó bao gồm các nguồn phát tín hiệu, các khối cho phép nhập số liệu từ một *File*, hay từ MATLAB *Workspace*. Sau đây ta lần lượt điểm qua ý nghĩa của từng khối.

Constant

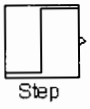


Khối *Constant* tạo nên một hằng số (không phụ thuộc thời gian) thực hoặc phức. Hằng số đó có thể là scalar, vector hay ma trận, tùy theo cách ta khai báo tham số *Constant Value* và ô *Interpret vector parameters as 1-D* có được chọn hay không. Nếu ô đó được chọn, ta có thể khai báo tham số *Constant Value* là vector hàng hay cột với kích cỡ $[1 \times n]$ hay $[n \times 1]$ dưới dạng ma trận. Nếu ô đó không được chọn, các vector hàng hay cột đó chỉ được sử dụng như vector với chiều dài n, tức là tín hiệu 1-D.

Ví dụ về cách khai báo khối *Constant*:



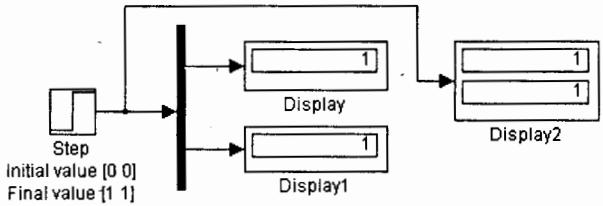
Step và Ramp



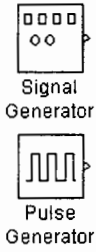
Nhờ hai khối *Step* và *Ramp* ta có thể tạo nên các tín hiệu dạng bậc thang hay dạng dốc tuyến tính, dùng để kích thích các mô hình SIMULINK. Trong hộp thoại *Block Parameters* của khối *Step* ta có thể khai báo giá trị đầu / giá trị cuối và cả thời điểm bắt đầu của tín hiệu bước nhẩy. Đối với *Ramp* ta có thể khai báo độ dốc, thời điểm và giá trị xuất phát của tín hiệu ở đầu ra.

Đối với cả hai khối (giống như khối *Constant*), ta có thể sử dụng tham số tùy chọn *Interpret vector parameters as 1-D* để quyết định các tín hiệu dạng bước nhẩy hay dạng dốc tuyến tính có giá trị scalar hay vector hay ma trận.

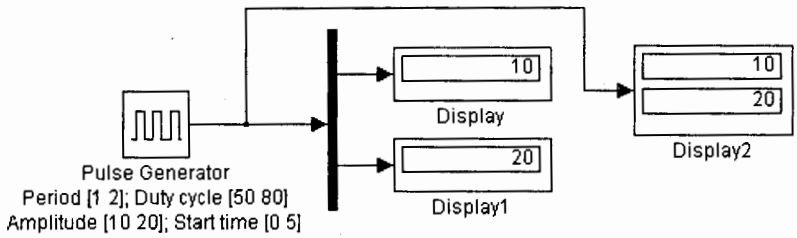
Chú ý: Hai khối *Step* và *Ramp* không phải chỉ tạo ra một tín hiệu như nhiều người vẫn hiểu nhầm, mà có thể tạo ra một tập các tín hiệu được xử lý dưới dạng vector (hàng hay cột) hoặc ma trận như ví dụ dưới đây minh họa.



Signal Generator và Pulse Generator



Bằng *Signal Generator* ta tạo ra các dạng tín hiệu kích thích khác nhau (ví dụ: hình sin, răng cưa), còn *Pulse Generator* tạo chuỗi xung hình chữ nhật. Biên độ và tần số có thể khai báo tùy ý. Đối với *Pulse Generator* ta còn có khả năng chọn tỷ lệ cho bề rộng xung (tính bằng % so với cả chu kỳ). Đối với cả hai khối (giống như khối *Constant*), ta có thể sử dụng tham số tùy chọn *Interpret vector parameters as 1-D* để quyết định các tín hiệu có giá trị scalar hay vector hay ma trận (xem ví dụ tiếp theo).



Đối với các hệ gián đoạn hay hệ lai (sơ đồ có cả hai loại khối liên tục và gián đoạn) ta sử dụng khối *Discrete Pulse Generator* để tạo chuỗi xung chữ nhật.

Repeating Sequence



Khối *Repeating Sequence* cho phép ta tạo nên một tín hiệu tuần hoàn tùy ý. Tham số *Time values* phải là một vector thời gian với các giá trị đơn điệu tăng. Vector biến ra *Output values* phải có kích cỡ (chiều dài) phù hợp với chiều dài của tham số *Time values*. Giá trị lớn nhất của vector thời gian quyết định chu kỳ lặp lại (chu kỳ tuần hoàn) của vector biến ra.

Sine Wave



Khối *Sine Wave* được sử dụng để tạo tín hiệu hình sin cho cả hai loại mô hình: liên tục (tham số *Sample time* = 0) và gián đoạn (tham số *Sample time* = 1). Tín hiệu đầu ra *y* phụ thuộc vào ba tham số chọn: *Amplitude*, *Frequency* và *Phase* trên cơ sở quan hệ *y* = *Amplitude*.`sin`(*Frequency*.`time` + *Phase*). Vì thứ nguyên (đơn vị) của *Phase* là `[rad]`, ta có thể khai báo trực tiếp giá trị của *Phase* là một hệ số nào đó nhân với `pi`. Giống như khối *Constant*, ta có thể sử dụng tham số tùy chọn *Interpret vector parameters as 1-D* để quyết định các tín hiệu có giá trị scalar hay vector hay ma trận.

From Workspace



Khối *From Workspace* có nhiệm vụ lấy số liệu từ cửa sổ MATLAB *Workspace* để cung cấp cho mô hình SIMULINK. Các số liệu lấy vào phải có dạng của biểu thức MATLAB, khai báo tại dòng *Data*.

Nếu số liệu lấy vào là tín hiệu scalar hay vector (tín hiệu 1-D), khi ấy biểu thức khai tại *Data* phải là một ma trận với vector thứ nhất là vector thời gian có giá trị tăng dần, gán cho vector (hay ma trận) các giá trị tín hiệu lấy vào. Nếu số liệu đọc vào là *n* tín hiệu scalar / vector *u*1 ... *un*, khi ấy biểu thức MATLAB khai tại dòng *Data* (tên mặc định là `simin`) sẽ phải có dạng:

$$\text{simin} = \begin{bmatrix} t_1 & u1_1 & u2_1 & \cdots & un_1 \\ t_2 & u1_2 & u2_2 & \cdots & un_2 \\ \vdots & & & & \vdots \\ t_{final} & u1_{final} & u2_{final} & \cdots & un_{final} \end{bmatrix} \tag{6.1}$$

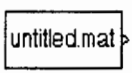
Nếu sử dụng định dạng *Structure* (phần dành cho thời gian bỏ trống) và *Structure with time*[1], ta có thể đọc từ *Workspace* không chỉ các số liệu scalar và vector (số liệu 1-D), mà lấy cả số liệu ma trận (2-D). Ví dụ sau sẽ làm rõ thêm cấu tạo của định dạng *Structure with time*.

Giả sử ta phải lấy số liệu của các tín hiệu $u1$ (scalar), $u2$ (vector với chiều dài là 4) và $u3$ (ma trận [3 × 2]) sau đây:

$$\begin{aligned} structurname.time &= \begin{bmatrix} t_1 & t_2 & \cdots & t_{final} \end{bmatrix}^T \\ structurname.signals(1).values &= \begin{bmatrix} u1_1 & u1_2 & \cdots & u1_{final} \end{bmatrix}^T \\ structurname.signals(1).dimensions &= 1 \\ structurname.signals(2).values &= \begin{bmatrix} u2_1 & u2_2 & \cdots & u2_{final} \end{bmatrix}^T \\ structurname.signals(2).dimensions &= 4 \\ structurname.signals(3).values &= \begin{bmatrix} u3_1 & u3_2 & \cdots & u3_{final} \end{bmatrix}^T \\ structurname.signals(3).dimensions &= \begin{bmatrix} 3 & 2 \end{bmatrix} \end{aligned} \tag{6.2}$$

Định dạng của ma trận (mảng, *Array*) và của cấu trúc cũng giống hệt như khi đọc số liệu qua *Workspace I/O* của hộp thoại *Simulation Parameters* (mục 6.6). Một định dạng *Structure with time* do khối *To Workspace* gửi vào *Workspace* bao giờ cũng có thể lấy vào được một cách dễ dàng. Bằng tham số *Form output after final data value* ta có thể xác định dạng tín hiệu ra của *From Workspace* sau giá trị cuối cùng (*Extrapolation*: ngoại suy; hay *SettingToZero*: xóa về không; hay *HoldingFinalValue*: giữ nguyên giá trị cuối; hay *CyclicRepetition*: lặp lại theo chu kỳ)

From File



Bằng khối *From File* ta có thể lấy số liệu từ một MAT-*File* (xem mục 2.3) có sẵn. MAT-*File* có thể là kết quả của một lần mô phỏng trước đó, đã được tạo nên và cất đi nhờ khối *To File* trong sơ đồ SIMULINK. Số liệu cất trong MAT-*File* phải có định dạng như sau:

[1] Ô *Format* thuộc trang *Workspace I/O*: tới bằng menue *Simulation / Simulation Parameters*

$$\begin{bmatrix} t_1 & t_2 & \cdots & t_{final} \\ u1_1 & u1_2 & \cdots & u1_{final} \\ u2_1 & u2_2 & \cdots & u2_{final} \\ \vdots & \vdots & & \vdots \\ un_1 & un_2 & \cdots & un_{final} \end{bmatrix} \quad (6.3)$$

Trong (6.3), t là vector bao gồm các giá trị thời gian đơn điệu tăng, $u1$ ... un là các vector giá trị của các tín hiệu ra. Như vậy, tín hiệu ra của khối *From File* là một vector 1-D với chiều dài n.

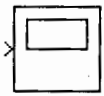
Tập các số liệu do khối *To File* tạo nên (mảng hay ma trận) có thể được khối *From File* đọc vào mà không phải qua xử lý, hay chế biến gì thêm. Nếu cần lấy tập số liệu do khối *To Workspace* tạo nên và cất thành *File*, cần phải chú ý hai điều sau đây:

- Khối *To Workspace* không cất các giá trị thời gian. Tuy nhiên, có thể buộc SIMULINK phải cất t bằng cách chọn ô *Time* ở menue *Simulation / Simulation Parameters / Workspace I/O*.
- Để khối *From File* có thể đọc được, mảng (*Array*, ma trận) các giá trị tín hiệu do khối *To Workspace* tạo nên cần phải được chuyển vị[1] trước khi cất.

6.4.2 Thư viện Sinks

Thư viện con *Sinks* bao gồm các khối xuất chuẩn của SIMULINK. Bên cạnh khả năng hiển thị đơn giản bằng số, còn có các khối dao động ký để biểu diễn các tín hiệu phụ thuộc thời gian hay biểu diễn hai tín hiệu trên hệ tọa độ x-y. Ngoài ra còn có khả năng xuất số liệu vào cửa sổ MATLAB *Workspace* hay cất dưới dạng *File*.

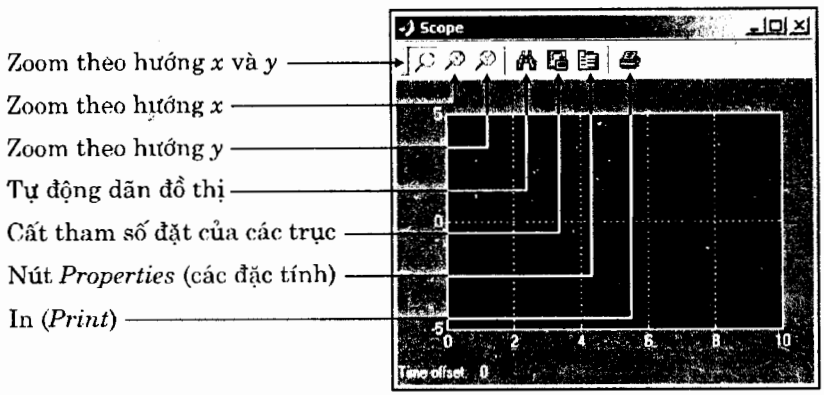
Scope



Nhờ khối *Scope* ta có thể hiển thị các tín hiệu của quá trình mô phỏng. Nếu mở cửa sổ *Scope* sẵn từ trước khi bắt đầu mô phỏng, ta có thể theo dõi trực tiếp diễn biến của tín hiệu. Ý nghĩa của các nút trên cửa sổ đã được minh họa ở hình 6.6.

Khi nhấn vào nút *Properties*, hộp thoại *'Scope' Properties* (đặc điểm của *Scope*) sẽ mở ra. Tại trang chọn *General* ta có thể đặt chế độ cho các trục. Khi đặt *Number of axes* > 1, cửa sổ *Scope* sẽ có nhiều đồ thị con, tương tự như lệnh `subplot` của MATLAB. Nếu điền một số cụ thể vào ô *Time range*, đồ thị sẽ chỉ được biểu diễn tới thời điểm do giá trị của số xác định.

[1] Ma trận chuyển vị: *transposed matrix*



Hình 6.6 *Cửa sổ của Scope sau khi mở*

Tại ô *Sampling*, nếu chọn chế độ *Decimation* là ta đã chọn một "hệ số loại bỏ". Ví dụ, giả sử ghi 100 có nghĩa là: Cứ đọc vào 100 giá trị *Scope* mới hiển thị 1 giá trị (loại bỏ 99 giá trị).

Ô *floating scope* giúp chọn chế độ hiển thị một hay đồng thời nhiều tín hiệu của mô hình SIMULINK, nếu nhiều thì các đường tín hiệu không cần nối với khối *Scope* nữa: Khi chọn ô *floating scope*, đường tín hiệu tới *Scope* sẽ tự động ẩn đi.

Trong khi mô phỏng, những tín hiệu nào đã được đánh dấu sẽ được hiển thị. Việc đánh dấu thực hiện bằng cách nhấn giữ phím *Shift* của PC, đồng thời nháy chuột vào đường tín hiệu muốn đánh dấu, hoặc thông qua Signal Selector (nháy chuột phải vào cửa sổ *Scope* khi mô hình mô phỏng đang chạy). Khi chọn *float scope* cần bảo đảm rằng: Tham số *Signal storage reuse* thuộc trang *Advanced* của hộp thoại *Simulation Parameters* được đặt về *off*. Nếu nháy chuột phải vào trục của cửa sổ *Scope* và chọn *Axes properties* ta sẽ có thêm một số khả năng đặt chế độ cho trục.

Khi nháy chuột vào nút *Properties* của cửa sổ *Scope*, sau đó tới trang chọn *Data history*, tại đó ta có thêm một số khả năng tác động tới khối lượng số liệu được hiển thị và được cất. Để có thể hiển thị toàn bộ số liệu của quá trình mô phỏng, hoặc ta loại không chọn ô *Limit data points to last*, hoặc ta khai báo một số ≥ số lượng điểm thực tế lấy vào. *Save data to workspace* sẽ cất các tín hiệu hiển thị trên *Scope* vào *Workspace* với tên khai báo tại *Variable name* và *Format* (*Array* khi *Number of axes* = 1, *Structure* và *Structure with time* khi *Number of axes* ≥ 1). Một mảng (*Array*) được cất dưới dạng như sau:

$$\begin{bmatrix} t_1 & u1_1 & u2_1 & \cdots & un_1 \\ t_2 & u1_2 & u2_2 & \cdots & un_2 \\ \vdots & & & & \vdots \\ t_{final} & u1_{final} & u2_{final} & \cdots & un_{final} \end{bmatrix} \quad (6.4)$$

Định dạng trên có thể được khối *From Workspace* đọc vào ngay mà không cần qua xử lý / chế biến gì.

Một *Structure* khi được cất sẽ có ba mảng *time*, *signals* và *blockName*. Mảng thời gian *time* là rỗng. Nếu muốn cất cả thời gian mô phỏng ta phải chọn *Structure with time*. Giả sử ta nối một Bus tín hiệu với bề rộng n tới đầu vào của *Scope*, *Structure with time* được cất sẽ có dạng dưới đây:

$$structurname.time = \begin{bmatrix} t_1 & t_2 & \cdots & t_{final} \end{bmatrix}^T$$

$$structurname.signals.values = \begin{bmatrix} u1_1 & u2_1 & \cdots & un_1 \\ u1_2 & u2_2 & \cdots & un_2 \\ \vdots & \vdots & & \vdots \\ u1_{final} & u2_{final} & \cdots & un_{final} \end{bmatrix}$$

$$structurname.signals.dimensions = n$$

$$structurname.signals.label = 'Label\ of\ Signal\ coming\ to\ Block\ Scope'$$

$$structurname.signals.title = 'Title\ of\ Scope\ Plot'$$

$$structurname.signals.plotStyle = \begin{bmatrix} 1 \times n \end{bmatrix}$$

$$structurname.blockName = 'Model\ Name / Name\ of\ Block\ Scope'$$

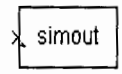
(6.5)

XY Graph



XY Graph biểu diễn hai tín hiệu đầu vào scalar trên hệ tọa độ *xy* dưới dạng đồ họa (*Figure*) MATLAB. Đầu vào thứ nhất (bên trên) ứng với trục *x*, đầu thứ hai với trục *y*. Trong hộp thoại Block Parameters ta có thể đặt giới hạn cho hai trục.

To Workspace



Khối *To Workspace* gửi số liệu ở đầu vào của khối tới môi trường MATLAB *Workspace* dưới dạng *Array*, *Structure* hay *Structure with time* và lấy chuỗi ký tự khai tại *Variable name* để đặt tên cho tập số liệu được ghi.

Nếu tại ô *Save format* ta đã chọn tham số *Array*, để thu thập không bỏ sót toàn bộ khoảng giá trị mô phỏng, ta sẽ phải nhập giá trị `inf` cho tham số *Limit data points to last*. *Array* có cấu trúc như sau:

$$\begin{bmatrix} u1_1 & u2_1 & \cdots & un_1 \\ u1_2 & u2_2 & \cdots & un_2 \\ & \vdots & & \vdots \\ u1_{final} & u2_{final} & \cdots & un_{final} \end{bmatrix} \tag{6.6}$$

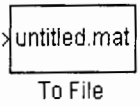
Một cấu trúc *Structure with time* do khối *To Workspace* ghi có cấu tạo bởi hai mảng *time* và *signals* với dạng như sau:

$$\begin{aligned} structurname.time &= \begin{bmatrix} t_1 & t_2 & \cdots & t_{final} \end{bmatrix}^T \\ structurname.signals.values &= \begin{bmatrix} u1_1 & u2_1 & \cdots & un_1 \\ u1_2 & u2_2 & \cdots & un_2 \\ \vdots & \vdots & & \vdots \\ u1_{final} & u2_{final} & \cdots & un_{final} \end{bmatrix} \\ structurname.signals.dimensions &= n \\ structurname.signals.label &= 'Label\ of\ Signal\ coming\ to \\ &\quad Block\ To\ Workspace' \\ structurname.blockName &= 'Model\ Name/\ Name\ of \\ &\quad Block\ To\ Workspace' \end{aligned} \tag{6.7}$$

Khi chọn *Structure*, mảng *time* sẽ là mảng rỗng. Tham số *Decimation* quyết định (giống như khối *Scope*): Bao nhiêu số liệu sẽ bị bỏ qua khi vẽ *XY Graph*. Khi mô phỏng bằng thuật giải (*Solver*) với bước linh hoạt *Variable step* (xem mục 6.6) ta cần tận dụng ưu thế của tham số *Sampling time*: Khi chọn bước linh hoạt *Variable step*, khoảng cách giữa các thời điểm của quá trình mô phỏng là linh hoạt (không cố định), ta có thể dùng *Sampling time* để khai báo một chu kỳ (bước tính bằng s) cố định cho quá trình gửi số liệu tới môi trường MATLAB *Workspace*. Nếu *Sampling time* giữ giá trị mặc định –1, bước gửi số liệu sẽ do *Solver* quyết định. Để có thể sử dụng khối *From Workspace* đọc *Array* (lấy số liệu) cất trong môi trường MATLAB *Workspace*, ta cần vector thời gian tương ứng với thời điểm thu thập số liệu. Để tạo vector thời gian, ta phải kích hoạt ô *time* tại trang *Workspace I/O*, tới qua menue *Simulation / Simulation Parameters* của cửa sổ mô hình

mô phỏng SIMULINK. Khối *From Workspace* có thể đọc *Structure with time* mà không cần xử lý gì.

To File

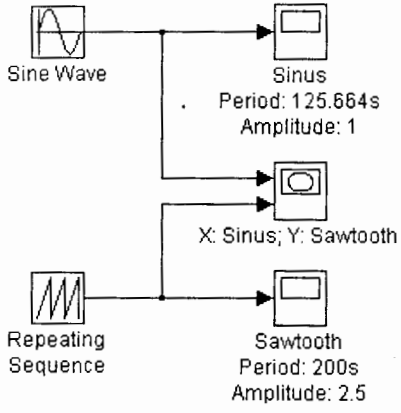


Khối *To File* cất tập số liệu (mảng, *Array* hay ma trận) ở đầu vào của khối cùng với vector thời gian dưới dạng MAT-*File* (mục 2.3). *Array* có định dạng giống như định dạng mà khối *From File* cần, và chính vì vậy: Số liệu do *To File* cất có thể được *From File* đọc trực tiếp mà không cần phải xử lý / chế biến gì.

Để khối *From Workspace* đọc được mảng ta phải tiến hành chuyển vị (*transpose*) cho mảng. Trong hộp thoại *Block Parameters* của khối *To File* ta có thể đặt tên cho MAT-*File* và *Array*. Các tham số *Decimation* và *Sample time* có tác dụng và cách khai báo giống như khối *To Workspace*.

Ví dụ:

Ví dụ sau đây sẽ minh họa rõ hơn chức năng của các khối *Sine Wave*, *Repeating Sequence*, *Scope* và *XY Graph*. Hình 6.7 mô tả cấu trúc mô phỏng có tên *Fig06_07.mdl*, trong đó hai tín hiệu hình sin (*sinus*) và răng cưa (*sawtooth*) được hiển thị độc lập, đồng thời trên hệ tọa độ *xy*.



Hình 6.7 *Ví dụ sử dụng các khối Sources và Sinks: mô hình SIMULINK có tên Fig06_07.mdl*

Tham số của khối *Sine Wave*:

Biên độ (Amplitude):	1
Tần số (Frequency, rad/sec):	0,05
Pha (Phase):	0
Sample time:	0

Bằng các tham số trên ta đã khai báo tín hiệu hình sin với chu kỳ $T = 2\pi/\omega = 2\pi/0{,}05s = 125{,}664Hz$.

Tham số của khối *Repeating Sequence*:

Time values:	[0:20:200]
Output values:	[0:0.25:2.5]

Với các tham số trên ta đã chọn chu kỳ của răng cưa là 200s, biên độ là 2,5.

Tham số của khối *X: Sinus; Y: Sawtooth*:

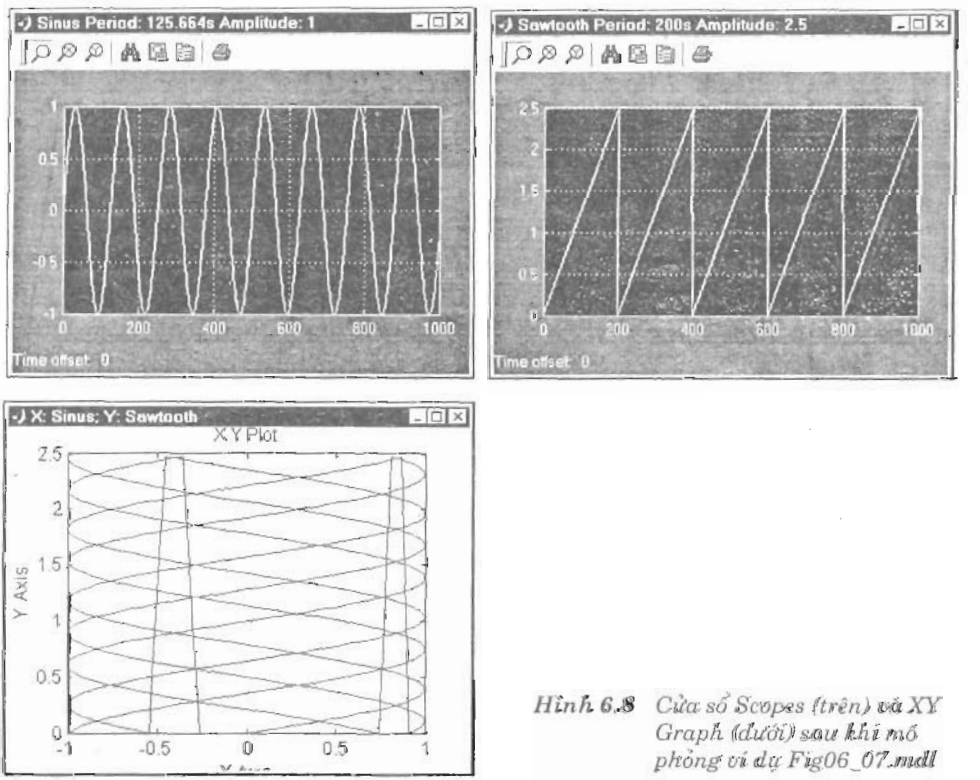
x-min:	-1
x-max:	1
y-min:	0
y-max:	2.5
Sample time:	-1

Bằng các tham số trên, đồ thị sẽ được biểu diễn theo trục x và y của *XY Graph* trong khoảng biên độ đã chọn cho các tín hiệu đầu vào.

Các tham số mô phỏng được khai báo tại trang *Solver* của hộp thoại *Simulation Parameters* (tới qua menue *Simulation* tại cửa sổ mô hình SIMULINK) như sau:

Simulation time:
Start time: 0.0
Stop time: 1000.0
Solver options:
Type: Variable-step, ode45(Dormand-Prince)
Max step size: 2.0
Min step size: auto
Initial step size: auto

Với các tham số như trên, quá trình mô phỏng xảy ra khá chậm và ta sẽ có cơ hội để theo dõi diễn biến của các tín hiệu trong khi vẽ đồ thị. Trong hai khối *Scope*, ô *Limit data points to last* đã không được chọn để bảo đảm hiển thị toàn bộ quá trình mô phỏng, không phụ thuộc vào bước. Vì đây là một hệ thống rất đơn giản, ta có thể sử dụng thuật toán *ode45* (mặc định chọn) để tích phân. Kết quả mô phỏng được giới thiệu tại hình 6.8.

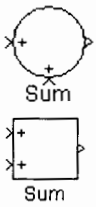


Hình 6.8 *Cửa sổ Scopes (trên) và XY Graph (dưới) sau khi mô phỏng ví dụ Fig06_07.mdl*

6.5 Thư viện Math

Thư viện con *Math* có một số khối với chức năng ghép toán học các tín hiệu khác nhau. Bên cạnh các khối đơn giản nhằm cộng hay nhân tín hiệu, trong *Math* còn có nhiều hàm (toán, lượng giác và logic) được chuẩn bị sẵn. Sau đây ta chỉ mô tả ngắn một số khối quan trọng nhất. Khối *Algebraic Constraint* sẽ được đề cập đến trong mục 7.6.

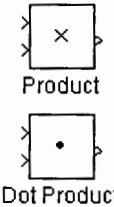
Sum



Đầu ra của khối *Sum* là tổng của các tín hiệu vào. Nếu các tín hiệu vào là scalar, tín hiệu tổng cũng là scalar. Nếu đầu vào có nhiều tín hiệu hỗn hợp, *Sum* tính tổng từng phần tử. Ví dụ, đầu vào bao gồm 3 tín hiệu: 1, sin(x) và [4 4 5 6], tín hiệu ra sẽ có dạng [5+ sin(x) 5+ sin(x) 6+ sin(x) 7+ sin(x)].

Nếu khối *Sum* chỉ có một đầu vào dạng vector, khi ấy các phần tử của vector sẽ được cộng thành scalar. Ví dụ, ở đầu vào chỉ tồn tại vector [4 4 5 6], ở đầu ra sẽ xuất hiện giá trị 19. Tại ô *List of signs* ta có thể khai báo cực tính và số lượng đầu vào bằng cách viết một chuỗi các ký hiệu + và -.

Product và Dot Product

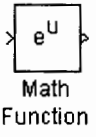


Khối *Product* thực hiện phép nhân từng phần tử hay nhân ma trận, cũng như phép chia giữa các tín hiệu vào (dạng 1-D hay 2-D) của khối, phụ thuộc vào giá trị đặt của tham số *Multiplication* và *Number of inputs*. Việc chọn *Multiplication* = *element-wise* có nghĩa: Kết quả là tích hay thương của từng phần tử của tín hiệu vào (tương đương với phép tính `y = u1.*u2` của MATLAB). Ví dụ: Nếu một khối Product có tham số *Number of inputs* = */*, với ba tín hiệu vào là 5, sin(x) và [4 4 5 6], khi ấy tín hiệu đầu ra có dạng [20/sin(x) 20/sin(x) 25/sin(x) 30/sin(x)]. Nếu khối *Product* chỉ có một đầu vào dạng vector, khi ấy các phần tử của vector sẽ được nhân với nhau thành scalar ở đầu ra.

Nếu *Multiplication* = *Matrix*, với ba tín hiệu dạng ma trận (có kích cỡ thích hợp) A, B, C, và (giả sử) *Number of inputs* = **/, ở đầu ra ta thu được kết quả ABC^{-1}.

Khối *Dot Product* tính tích scalar (vô hướng) của các vector đầu vào. Giá trị đầu ra của khối tương đương với lệnh MATLAB `y = sum(conj(u1).*u2)`.

Math Function và Trigonometric Function



Trong khối *Math Function* có một lượng khá lớn các hàm toán đã được chuẩn bị sẵn tại ô *Function*, cho phép ta lựa chọn theo nhu cầu sử dụng. Tương tự, khối *Trigonometric Function* có tất cả các hàm lượng giác quan trọng. Ngược với khối *Fcn* (thư viện con



Trigonometric Function

Function & Tables), cả hai khối *Math Function* và *Trigonometric Function* đều có thể xử lý tín hiệu 2-D.

Gain, Slider Gain, Matrix Gain



Gain



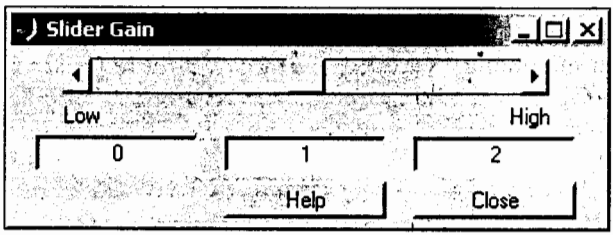
Slider Gain



Matrix Gain

Khối *Gain* có tác dụng khuếch đại tín hiệu đầu vào (định dạng 1-D hay 2-D) bằng biểu thức khai báo tại ô *Gain*. Biểu thức đó có thể chỉ là một số hay một biến. Nếu là biến, biến đó phải tồn tại trong môi trường MATLAB *Workspace*, chỉ khi ấy SIMULINK mới có thể tính toán được với biến. Nhờ thay đổi giá trị của tham số *Multiplication* ta có thể xác định: Phép nhân của biến vào với *Gain* được thực hiện theo phương thức nhân ma trận hay nhân từng phần tử.

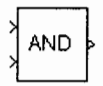
Khối *Slider Gain* cho phép người sử dụng thay đổi hệ số khuếch đại scalar trong quá trình mô phỏng. Khi nháy kép chuột trái vào khối, cửa sổ của *Slider Gain* mở ra như dưới đây:



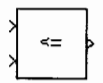
Dễ dàng thấy ở cửa sổ trên: Khi đã khai báo giá trị lớn nhất (ô *High*) và bé nhất (ô *Low*), ta có thể thay đổi hệ số khuếch đại trong khoảng *Low* ... *High* bằng thanh trượt hoặc bằng số cụ thể.

Khối *Matrix Gain* cũng giống như *Gain*, điểm khác chỉ là: Phải khai báo các tham số thích hợp để thực hiện phép nhân giữa ma trận *Gain* với đầu vào.

Logical Operator và Relational Operator



Logical Operator



Relational Operator

Khối *Logical Operator* thực hiện kết hợp các biến vào của khối theo hàm Logic đã chọn tại ô *Operator*. Biến ra sẽ nhận các giá trị 1 (TRUE, đúng) hay 0 (FALSE, sai). Nếu các biến vào có định dạng vector (tính hiệu 1-D) hay ma trận (tín hiệu 2-D), các phần tử của chúng sẽ được kết hợp theo hàm Logic đã chọn, và ở đầu ra sẽ xuất hiện một vector hay ma trận. Khi các tín hiệu không thuộc loại *boolean* (không có giá trị 0 hoặc 1, ví dụ: *double*) được đưa tới đầu vào của *Logical Operator*, cần phải chú ý: Tham số *Boolean logic signals* (trang *Advanced* của hộp thoại *Simulation Parameters*) phải được chọn là *off*. Khi ấy, các tín hiệu $\neq 0$ được coi là TRUE, còn = 0 là FALSE.

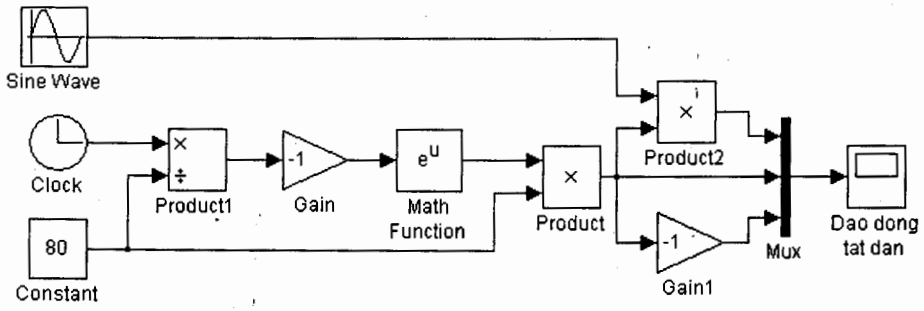
Khối *Relational Operator* thực hiện kết hợp hai tín hiệu đầu vào theo toán tử so sánh đã chọn tại ô *Operator*. Biến ra sẽ nhận các giá trị 1 (TRUE, đúng) hay 0 (FALSE, sai). Ví dụ: Tín hiệu $u1$ nối với đầu vào phía trên và tín hiệu $u2$ nối với đầu vào phía dưới, tín hiệu ra sẽ là $y = u1 < u2$ hay $y = u1 \geq u2$ ..., tùy theo toán tử so sánh đã chọn. Các tín hiệu vào dạng 2-D được xử lý giống như khối *Logical Operator*. Nếu tham số *Boolean logic signals* (trang *Advanced* của hộp thoại *Simulation Parameters*) được chọn là *off*, tín hiệu đầu ra sẽ không phải là *boolean* mà là *double*.

Ví dụ:

Ví dụ vật lý đơn giản sau đây nhằm minh họa chức năng của các khối *Product*, *Gain* và *Math Function* (khối *Mux* thuộc thư viện con *Signals & Systems* sẽ được đề cập đến ở mục 6.7). Hình 6.9 giới thiệu mô hình SIMULINK có tên *Fig06_09.mdl* với nhiệm vụ mô phỏng phương trình sau:

$$f(t) = 80\exp\left(-\frac{1}{80}t\right)\sin\left(0,25t + \frac{\pi}{3}\right) \tag{6.8}$$

Dao động hình sin tắt dần và đường bao sẽ được ghi và hiển thị. Khối *Clock* xác định thời gian mô phỏng t và được sử dụng để tính hàm exp của đường bao. Phép chia của thời gian mô phỏng t cho hằng số 80 được thực hiện khi đã đặt giá trị */ cho tham số *Number of inputs* của khối *Product1*.



Hình 6.9 *Ví dụ minh họa chức năng các khối thuộc thư viện con Sources, Sinks và Math*

Tham số của khối *Sine Wave*:

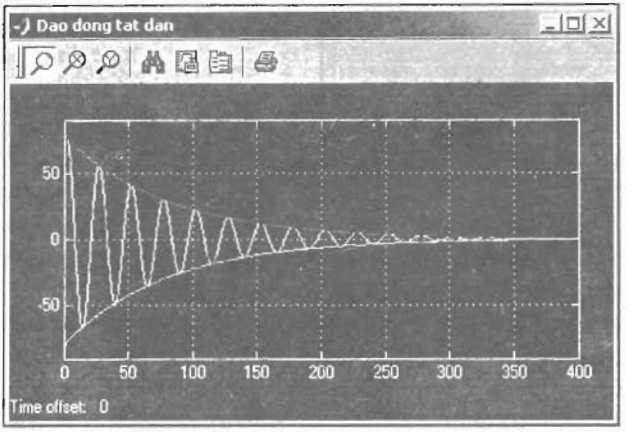
Amplitude:	1
Frequency (rad/sec):	0,25
Phase:	pi/3
Sample time:	0

Theo các tham số trên, tín hiệu hình sin có chu kỳ $T = 2\pi/\omega = 2\pi/0,25s = 25,133s$ và biên độ là 1.

Chế độ mô phỏng (trang *Solver* của hộp thoại *Simulation Parameters*) được chọn như sau:

Simulation time:
Start time: 0.0
Stop time: 400.0
Solver options:
Type: Variable-step, ode45(Dormand-Prince)
Max step size: 0.005
Min step size: auto
Initial step size: auto

Kết quả mô phỏng được giới thiệu ở hình 6.10.



Hình 6.10 Cửa sổ Scope sau khi mô phỏng ví dụ Fig06_09.mdl

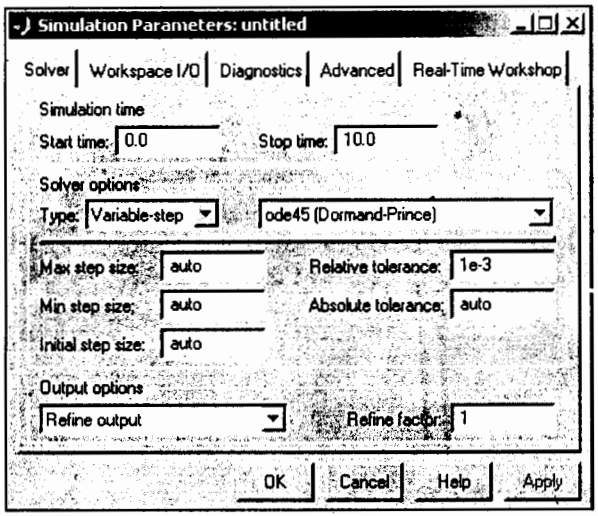
6.6 Chuẩn bị mô phỏng: Khai báo tham số và phương pháp tích phân

Trước khi tiến hành mô phỏng ta phải có những thao tác chuẩn bị nhất định: Đó là khai báo tham số và phương pháp mô phỏng. Các thao tác chuẩn bị được thực hiện tại hộp thoại *Simulation Parameters* (tới qua nhánh menue *Simulation / Simulation Parameters*). Tại đó, tất cả các tham số đều đã có một giá trị mặc định (*default*) sẵn, nghĩa là: Có thể khởi động mô phỏng trực tiếp không cần chuẩn bị. Tuy nhiên, để thu được kết quả mô phỏng tốt nhất, phải thực hiện chuẩn bị, đặt các tham số phù hợp với mô hình SIMULINK cụ thể.

Hộp thoại *Simulation Parameters* bao gồm bốn trang:

Solver

Tại trang *Solver* ta có thể khai báo *thời điểm bắt đầu và kết thúc, thuật toán tích phân* và *phương pháp xuất kết quả* của mô phỏng. Hình 6.11 mô tả cửa sổ của trang *Solver* với các tham số mặc định.



Hình 6.11 *Trang Solver (với các tham số mặc định) của hộp thoại Simulation Parameters*

SIMULINK cung cấp cho ta một số thuật toán (*Solver*) khác nhau để giải bằng số phương trình vi phân[1], đáp ứng một phổ khá rộng các bài toán đặt ra. Đối với hệ gián đoạn ta có thể chọn thuật toán *discrete* với bước tích phân linh hoạt (*Variable-step*) hay cố định (*Fixed-step*). Đối với hệ liên tục ta có các thuật toán *Variable-step* khác nhau như *ode45*, *ode23* (cả hai đều dựa trên phương pháp Runge-Kutta), hay *ode2* (phương pháp Heun), hay *ode1* (phương pháp Euler tiến). Ngoài ra, SIMULINK còn có các thuật toán *Variable-step* dành cho mô phỏng *hệ thống cứng*[2] như *ode15s*, *ode23s*, *ode23t* hay *ode23tb*.

Thuật toán *Variable-step* làm việc với bước tích phân linh hoạt. Việc giải các PTVP được bắt đầu với bước tích phân khai báo tại *Initial step size*. Nếu ngay khi vừa bắt đầu, đạo hàm của các biến trạng thái đã quá lớn, *Solver* sẽ chọn giá trị bé hơn giá trị ghi tại *Initial step size*. Trong quá trình mô phỏng, SIMULINK sẽ cố gắng giải PTVP bằng bước cho phép lớn nhất ghi tại *Max step size*. Kích cỡ *Max step size* có thể tính (phụ thuộc vào *Start time* và *Stop time*) như sau:

$$Max\ step\ size = \frac{Stop\ time - Start\ time}{50}$$

Do có khả năng thích nghi bước tích phân, thuật toán *Solver* với *Variable-step* có thể giám sát biến thiên (giám sát lỗi) của các biến trạng thái từ thời điểm vừa qua tới thời điểm hiện tại. Thêm vào đó, thuật toán có thể nhận biết các vị trí không liên tục của hàm (*zero crossing detection*) như các đột biến dạng bước nhảy.

Khi mới sử dụng chương trình lần đầu, về nguyên tắc ta có thể sử dụng ngay thuật toán mặc định *ode45*.

---

[1] Phương trình vi phân: viết tắt PTVP

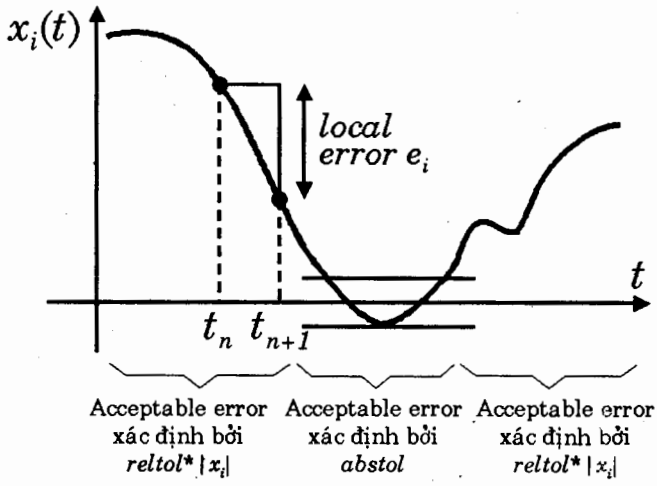
[2] *Hệ thống cứng* là những hệ có tồn tại tần số (cộng hưởng) cao hơn động học của hệ rất nhiều. Ví dụ: Các hệ thống cơ với trục kích cỡ lớn và cứng nhưng có cộng hưởng.

Giám sát sai số

Để có thể thích nghi bước tích phân với động học của các biến trạng thái, tại mỗi bước tích phân, SIMULINK lại tính độ biến thiên của biến trạng thái từ thời điểm vừa qua tới thời điểm hiện tại. Độ biến thiên đó được gọi là sai số cục bộ *local error* e_i (i = 1 ... : số biến trạng thái của hệ, hình 6.12). Cứ mỗi bước tích phân, thuật toán *Solver* (dạng *Variable-step*) lại kiểm tra xem *local error* của mỗi biến trạng thái có thỏa mãn điều kiện *acceptable error* (sai số có thể chấp nhận) được xác định bởi các tham số *Relative tolerance* và *Absolute tolerance* ở hộp thoại *Simulation Parameters* (viết tắt: *reltol* và *abstol*). Điều kiện *acceptable error* được mô tả bằng công thức sau:

$$e_i \le \underbrace{\max\left(reltol.|x_i|, abstol\right)}_{acceptable\ error}$$

Nếu một trong số các biến trạng thái không thỏa mãn điều kiện trên, bước tích phân tự động được giảm và quá trình tính của bước sẽ được lặp lại. Việc *acceptable error* được xác định trên cơ sở lựa chọn tối đa (max) có nguyên do như sau: Giá trị khai báo tại *Relative tolerance* là ứng với biến thiên cho phép tính bằng % của giá trị tức thời của biến trạng thái x_i. Nếu *acceptable error* chỉ được quyết định bởi *Relative tolerance*, vậy khi $|x_i|$ bé thì *Relative tolerance* có thể trở nên quá bé, đồng nghĩa với việc: Biến trạng thái không được phép biến thiên (tăng/giảm) gì nữa. Điều này sẽ không xảy ra nếu *acceptable error* được chọn theo công thức ở trên. Nếu ta khai báo cho *Absolute tolerance* giá trị *auto*, khi ấy SIMULINK sẽ bắt đầu bằng 10^{-6}. Sau đó *abstol* được đặt về *reltol*.max($|x_i|$). Nhờ cách chọn bước linh hoạt như vậy, SIMULINK cho phép các biến trạng thái vẫn được phép biến thiên ngay cả khi $|x_i|$ rất bé, hoặc thậm chí bằng 0. Hình 6.12 minh họa những nội dung vừa giải thích.

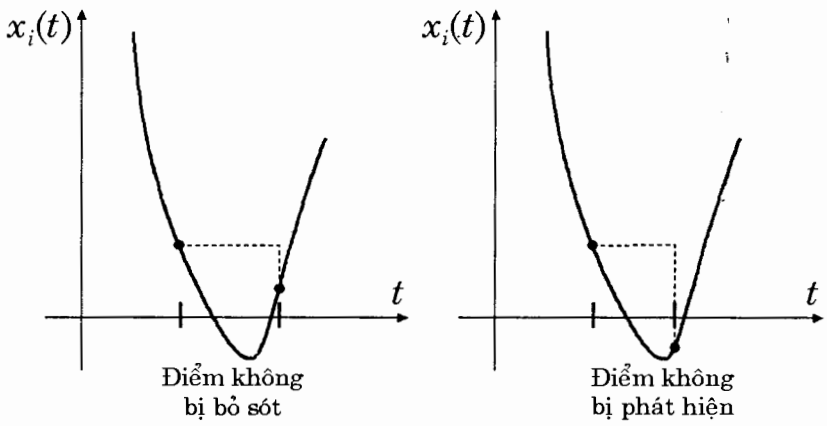


Hình 6.12 *Giám sát sai số của bước tích phân khi Solver sử dụng Variable-step*

Nói chung, do có khả năng giám sát sai số SIMULINK sẽ tự động giảm bước tích phân, nếu trong một dải nào đó đồ thị thời gian của các biến trạng thái có độ dốc lớn. Ngược lại, trong những dải gần như không có sự thăng giáng của biến trạng thái lại có thể tiết kiệm thời gian tính toán nhờ sử dụng bước tích phân lớn nhất được phép *Max step size*.

Zero crossing detection

Khái niệm *zero crossing* trong SIMULINK được hiểu là tính không liên tục trong diễn biến của trạng thái hay là các điểm không thông thường. Các tín hiệu không liên tục thường do một số khối nhất định gây ra như *Abs*, *Backslash*, *Dead Zone*, *Saturation* hay *Switch*. Mỗi khối hàm loại này có kèm theo một biến *zero crossing*, phụ thuộc vào các biến trạng thái không liên tục và đổi dấu mỗi khi gặp điểm không liên tục. Cứ sau mỗi bước tích phân, SIMULINK lại kiểm tra các biến *zero crossing* và qua đó nhận biết: Trong bước hiện tại có xảy ra *zero crossing* hay không. Nếu có, SIMULINK nỗ lực tính chính xác tối đa thời điểm xuất hiện bằng phương pháp nội suy giữa giá trị vừa qua và giá trị hiện tại của biến *zero crossing* đó. Khi đã biết chính xác, SIMULINK sẽ chỉ tính cho tới cận trái của điểm không liên tục, để rồi bước tích phân sau đó bắt đầu tính tiếp từ cận phải. Vì vậy, nếu chọn sai số quá thô sẽ có nguy cơ bỏ sót các điểm không (của biến trạng thái x_i hay của biến *zero crossing*). Nếu có nghi vấn bỏ sót điểm không, cần phải giảm sai số đã khai báo để bảo đảm là *Solver* với *Variable-step* sẽ chọn bước tính đủ nhỏ. Hình 6.13 minh họa vấn đề vừa trình bầy.

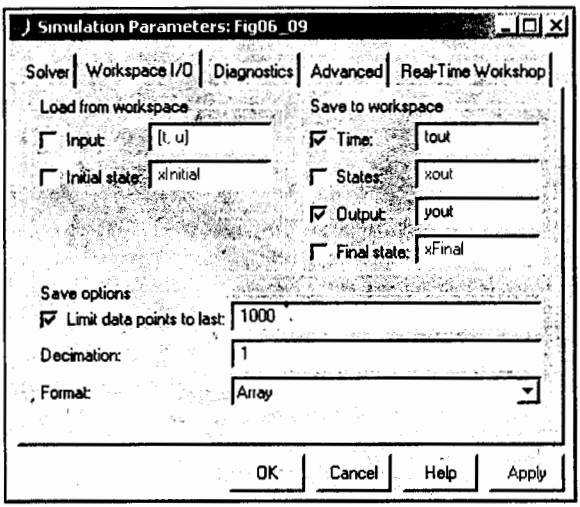


Hình 6.13 *Việc phát hiện điểm không hoàn toàn phụ thuộc vào sai số*

Solver với *Fixed-step* hoạt động với bước cố định và việc giám sát – phát hiện các điểm không liên tục là không thể. Song vì biết chính xác số lượng bước tích phân, ta có thể ước lượng khá chính xác thời gian tính của mô hình mô phỏng. Điều này đặc biệt có ý nghĩa nếu ta dự kiến sẽ cài đặt mô hình (sau khi mô phỏng thành công) trên một cấu hình Hardware nào đó.

Workspace I/O

Nhờ khai báo thích hợp tại trang *Workspace I/O* ta có thể gửi số liệu vào, hoặc đọc số liệu từ môi trường MATLAB *Workspace* mà không cần sử dụng các khối như *To Workspace*, *From Workspace* trong mô hình SIMULINK. Ngoài ra, ta có thể khai báo giá trị ban đầu cho các biến trạng thái tại đây. Hình 6.14 cho ta thấy cấu tạo của trang *Workspace I/O* với các tham số mặc định.



Hình 6.14 *Trang Workspace I/O với các tham số mặc định*

Nếu chọn ô *Input* (nằm dưới dòng *Load from Workspace*), ta có thể khai báo ở phần dành để điền chữ bên phải: Tên của các tập số liệu cần đọc từ *Workspace*. Các tập số liệu đó có thể có định dạng quen biết *Array*, *Structure* và *Structure with time*. Tất cả các yêu cầu về định dạng là hoàn toàn giống như đối với khối *From Workspace*.

Để đặt giá trị ban đầu cho biến trạng thái, ta phải chọn ô *Initial State* và ghi vào phần dành để điền chữ bên phải: Tên của biến đang giữ giá trị ban đầu, biến đó có thể có định dạng *Array* hay *Structure*. Việc tận dụng khả năng khai báo biến giữ giá trị ban đầu là rất có ý nghĩa khi ta cần sử dụng các giá trị trạng thái của một lần mô phỏng trước đó, đang còn nằm trong *Workspace* nhờ đã kích hoạt ô *Save to Workspace* và khai báo *Final state* (hình 6.14), hay nhờ đã sử dụng khối *To Workspace*. Diễn đạt cách khác: Nhờ *Workspace I/O* ta có thể bắt đầu lần mô phỏng mới bằng chính các giá trị trạng thái mà ta đã kết thúc lần mô phỏng trước đó.

Biến ra của mô hình SIMULINK được cất bằng cách điền tên biến ra vào phần điền chữ bên cạnh ô *Output*, sau khi đã kích hoạt *Output*. Cần chú ý: SIMULINK chỉ chấp nhận là biến ra những biến được nối với một khối *Outport*. Tương tự, bằng cách kích hoạt ô *States* (hình 6.14) và viết tên biến trạng thái vào phần điền chữ bên cạnh, SIMULINK sẽ cất biến trạng thái đó vào *Workspace*. Khi ta kích hoạt một trong các ô thuộc trang *Workspace I/O*, ta cần sử dụng phần *Save Options* để khai thêm các yêu cầu phụ về dung lượng số liệu

và về định dạng (*Format*) của các số liệu muốn cất. Định dạng *Array* khi cất đi có cấu trúc như sau (giống như khối *To Workspace*).

$$\begin{bmatrix} u1_1 & u2_1 & \cdots & un_1 \\ u1_2 & u2_2 & \cdots & un_2 \\ \vdots & \vdots & & \vdots \\ u1_{final} & u2_{final} & \cdots & un_{final} \end{bmatrix} \tag{6.9}$$

Cấu trúc *Structure with time* có hai mảng: *time* và *signals*. Đối với *Structure* mảng *time* là mảng rỗng.

$$\begin{aligned} structurname.time &= \begin{bmatrix} t_1 & t_2 & \cdots & t_{final} \end{bmatrix}^T \\ structurname.signals(1).values &= \begin{bmatrix} u1_1 & u1_2 & \cdots & u1_{final} \end{bmatrix}^T \\ structurname.signals(1).label &= 'Label\ of\ Signal_1' \\ structurname.signals(1).blockName &= 'Model\ Name/\ Name\ of \\ &\qquad Block\ generating\ Signal_1' \\ &\vdots \\ structurname.signals(n).values &= \begin{bmatrix} un_1 & un_2 & \cdots & un_{final} \end{bmatrix}^T \\ structurname.signals(n).label &= 'Label\ of\ Signal_n' \\ structurname.signals(n).blockName &= 'Model\ Name/\ Name\ of \\ &\qquad Block\ generating\ Signal_n' \end{aligned} \tag{6.10}$$

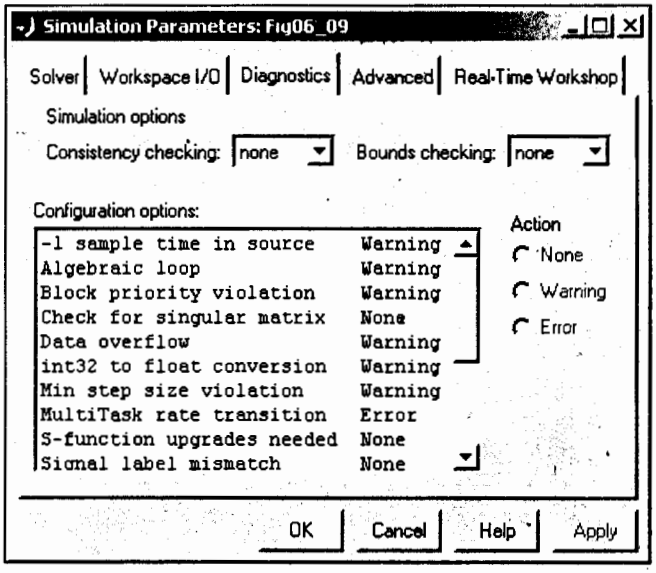
Để biết chính xác về trật tự sắp xếp các đại lượng trạng thái thành cột (đối với *Array*) và thành mảng (đối với *Structure* và *Structure with time*) ta hãy gọi lệnh `[size,x0,xstord] = modelname`.

Dự báo (Diagnostics)

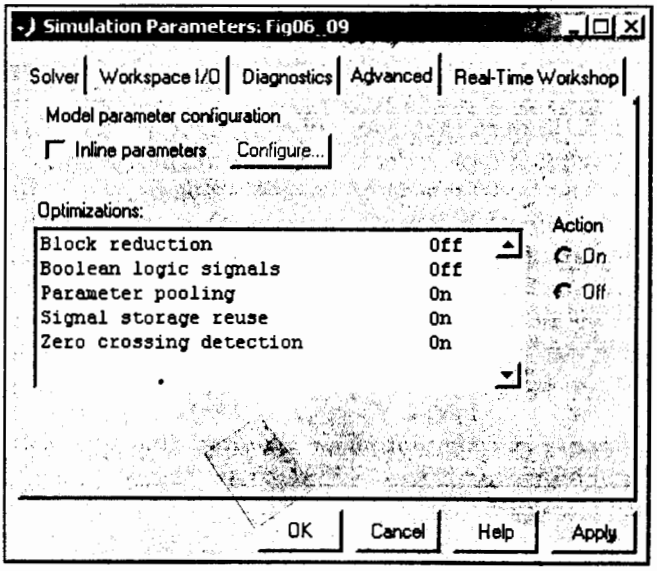
Trang tiếp theo của menue *Simulation / Simulation Parameters* là *Diagnostics*. Tại ô *Configuration options* ta có thể khai báo phương thức xử lý của SIMULINK đối với các sự kiện xảy ra trong quá trình mô phỏng. Hình 6.15 minh họa trang *Diagnostics* với tham số mặc định. Sau khi lựa chọn bằng cách nháy chuột vào một trong các sự kiện (*Algebraic Loop*: vòng lặp đại số, *Min step size violation*: vi phạm bước bé nhất vv...), ta có thể chọn và báo cho SIMULINK biết để sử dụng một trong ba cách xử lý nằm dưới *Action*: *Warning* sẽ chỉ có tác dụng cảnh báo trong cửa sổ MATLAB, còn *Error* là báo lỗi sẽ dẫn đến ngừng mô phỏng.

Ô *Simulation options* có hai vị trí khai báo *Consistency checking* và *Bounds checking*. Nếu được kích hoạt (khi giá trị chọn là *warning* hay *error*), thời gian

mô phỏng có khả năng bị tăng lên đáng kể. *Consistency checking* chỉ cần thiết khi sơ đồ SIMULINK có chứa những khối là hàm S[1] tự viết, có nhiệm vụ bảo đảm tính năng của các khối đó giống như các khối thông thường của SIMULINK. *Bounds checking* có nhiệm vụ khống chế, bảo đảm rằng không có số liệu nào của khối bị ghi ra ngoài vùng nhớ giành riêng cho khối.



Hình 6.15 *Trang Diagnostics với các tham số mặc định*



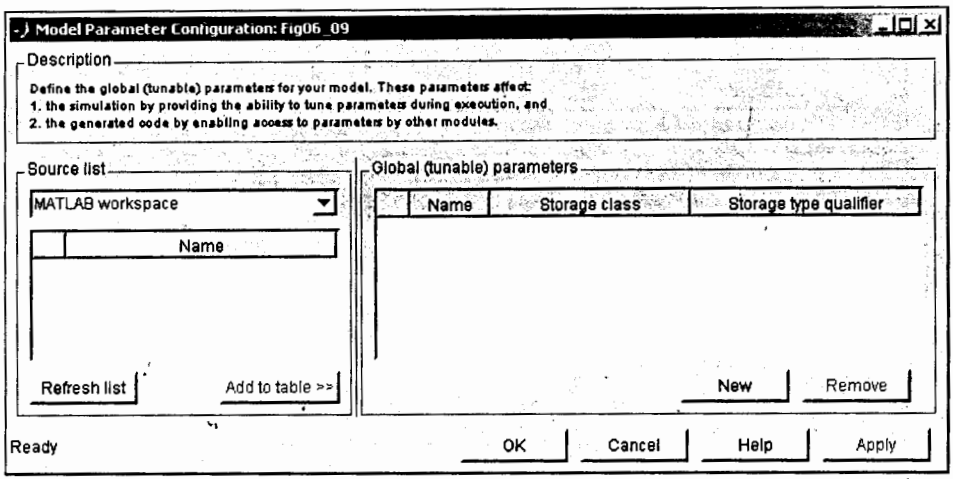
Hình 6.16 *Trang Advanced với các tham số mặc định*

[1] Hàm S: *S Function*

Advanced (khai báo nâng cao)

Hình 6.16 minh họa trang *Advanced* của *Simulation Parameters* với các tham số khai báo mặc định.

Việc kích hoạt ô *Inline parameters* sẽ phủ định khả năng thay đổi tham số của các khối trong quá trình mô phỏng. Duy nhất những tham số liệt kê trong danh sách *Global (tunable) parameters* là vẫn có thể thay đổi được (hình 6.17: thu được sau khi nhấn nút *Configure*). Vì những tham số không thay đổi được sẽ bị coi là hằng số, thời gian tính toán (so với cấu trúc tham số mặc định, cấu trúc mà hầu hết tham số có thể thay đổi được) sẽ giảm đi đáng kể.



Hình 6.17 *Sau khi nhấn nút Configure của ô Inline parameters, ta thu được cửa sổ mới để khai báo cấu trúc tham số của mô hình (Model Parameter Configuration)*

Tại ô *Optimization* (hình 6.16) ta có thêm vài khả năng khai báo tác động tới khối lượng tính toán mô phỏng. *Block reduction* và *Parameter pooling* ảnh hưởng tới việc tạo mã chạy (tạo *Code*) của *Real-Time Workshop*. Nếu tham số *Boolean logic signals* được chọn là *off*, khi ấy tất cả những tham số không thuộc loại *boolean* (ví dụ: thuộc loại *double*) vẫn được phép nối tới đầu vào của những khối (ví dụ: khối *Logical Operator*) bình thường chỉ chấp nhận tín hiệu vào dạng *boolean*. Các tín hiệu có giá trị > 0 được coi là TRUE, còn giá trị = 0 là FALSE. Khi *Boolean logic signals* = *off*, khối *Relational Operator* sẽ tạo tín hiệu đầu ra có dạng *double* thay vì *boolean*, làm đơn giản bớt việc xử lý tín hiệu trong các khối tiếp theo.

Tham số *Signal storage reuse*[1] chỉ nên đặt về *on* khi sơ đồ SIMULINK có sử dụng *floating scope* (xem lại khối *Scope*, thư viện *Sinks*) hay *floating display*, vì nhu cầu sử dụng bộ nhớ của một sơ đồ như vậy là rất lớn.

[1] *Signal storage reuse*: Cho phép sử dụng lặp nhiều lần vùng nhớ (giành riêng cho giá trị) của tín hiệu

Việc chọn *Zero crossing detection = off* cho phép chủ động loại trừ khả năng phát hiện các vị trí không liên tục của tín hiệu trong quá trình mô phỏng (sử dụng *Solver* với *Variable-step*) và nhờ đó giảm bớt khối lượng tính toán. Việc chọn *Zero crossing detection = off* cũng phủ định khả năng nhận biết vị trí không liên tục của các khối có đặc điểm gây nên sự không liên tục của tín hiệu. Đó là khối *MinMax*, *Abs*, *Sign* và *Relational Operator* (*Math*), *Saturation*, *Backslash* và *Dead Zone*, cũng như *Integrator* và *Step*. Tham số không có ảnh hưởng gì tới khối *Hit Crossing*.

6.6.1 Khởi động và ngừng mô phỏng

Quá trình mô phỏng của mô hình SIMULINK được khởi động qua menue *Simulation / Start*. Trong khi mô phỏng, có thể chọn *Simulation / Pause* để tạm ngừng, hay *Simulation / Stop* để ngừng hẳn quá trình mô phỏng.

Thêm vào đó ta còn có thể điều khiển quá trình mô phỏng bằng các dòng lệnh viết tại cửa sổ lệnh (cửa sổ *Command*) của MATLAB. Điều này đặc biệt có ý nghĩa khi ta muốn tự động hóa toàn bộ các chu trình mô phỏng, không muốn khởi động, ngừng hay xử lý vv... bằng tay. Đó là các lệnh `set_param` và `sim`.

- Lệnh `set_param` được gọi như sau:

  ```
  set_param('sys','SimulationCommand','cmd')
  ```

  Trong lệnh trên, mô hình mô phỏng có tên *sys* sẽ được khởi động khi *cmd = start*, hay ngừng lại khi *cmd = stop*. Sau khoảng thời gian nghỉ *pause*, ta ra lệnh tiếp tục mô phỏng bằng *continue*. Nếu chọn *cmd = update*, mô hình sẽ được cập nhật mới. Có thể kiểm tra tình trạng mô phỏng của *sys* bằng lệnh:

  ```
  get_param('sys','SimulationStatus')
  ```

  Bên cạnh tác dụng điều khiển quá trình mô phỏng, có thể sử dụng `set_param` lập các tham số của khối, các tham số mô phỏng.

- Lệnh `sim` được gọi như sau:

  ```
  [t,x,y] = sim('model')
  ```

  Nếu muốn chuyển giao cả tham số mô phỏng, hãy gọi:

  ```
  [t,x,y] = sim('model',timespan,options,ut)
  ```

  Bằng lệnh trên ta chủ động được quá trình đặt tham số mô phỏng từ môi trường MATLAB (mà thông thường phải thực hiện tại các trang *Solver* và *Workspace I/O* của hộp thoại *Simulation Parameters*). Vế trái của lệnh gồm các vector thời gian *t*, ma trận biến trạng thái *x* và ma trận biến ra *y* của mô hình. Các tham số của `sim` có ý nghĩa như sau: *model* là tên của mô hình SIMULINK, *timespan* viết dưới dạng `[tStart tFinal]` định nghĩa thời điểm bắt đầu và thời điểm ngừng chạy mô phỏng. Tham số *ut* cho phép đọc tập số liệu đã có vào khối *Inport*, có tác dụng tương tự như khi khai ô *Input* thuộc trang *Workspace I/O* của hộp thoại *Simulation Parameters*.

Bằng *options* ta chuyển giao cho mô hình các tham số mô phỏng quan trọng như thuật toán và bước tích phân, sai số, các điều kiện xuất số liệu vv... Việc tạo cấu trúc tham số *options* (định dạng ***structure***) được thực hiện bằng lệnh:

```
options = simset(property,value, ...)
```

Với lệnh trên, các tham số đã đặt trong hộp thoại *Simulation Parameters* sẽ không bị thay đổi, mà chỉ bị vô hiệu hóa khi `sim` khởi động quá trình mô phỏng. Bằng lệnh:

```
newopts = simset(oldopts,property,value, ...)
```

ta có thể thay đổi bộ tham số đã có *oldopts* (chỉ thay đổi những chỗ khai tại *property*) bởi bộ tham số mới *newopts*. Khi gọi `simset` không có khai báo đi kèm, khi ấy toàn bộ "*properties*" và các giá trị của chúng được xuất ra màn hình. Với lệnh:

```
struct = simget('model')
```

ta sẽ thu được trọn vẹn bộ tham số *options* đã được khai báo nhờ lệnh `simset` hay nhờ hộp thoại *Simulations Parameters*.

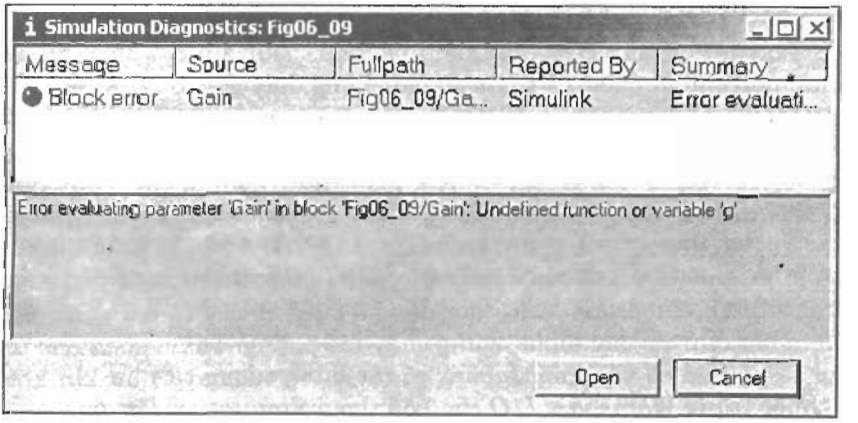
Ví dụ, lệnh:

```
[t,x,y] = sim('model',[],simset(simget('model'), ...
                'solver','ode2','MaxStep',0.01));
```

sẽ đưa thuật toán tích phân của sơ đồ SIMULINK có tên *model* về *ode2* với bước lớn nhất là 0,01 giây. Tại vị trí của *timespan* ta viết `[]`, nghĩa là: Các giá trị đặt của *Start time* và *Stop time* ở hộp thoại *Simulation Parameters* được giữ nguyên.

6.6.2 Xử lý lỗi

Nếu xuất hiện lỗi trong quá trình mô phỏng, SIMULINK sẽ ngừng mô phỏng và mở hộp thoại thông báo lỗi *Simulation Diagnostics* như hình 6.18.



Hình 6.18 Thông báo lỗi bằng hộp thoại Simulation Diagnostics

Trong phần phía trên của hộp thoại báo lỗi (hình 6.18) ta thấy có danh sách các khối gây nên lỗi. Khi chuyển vạch chọn tới khối nào (thuộc danh sách), ta sẽ thấy ở phần dưới hộp thoại các mô tả kỹ về lỗi của khối đó. Nếu nháy chuột trái vào nút *Open*, cửa sổ *Block Parameters* của khối sẽ mở ra để ta thay đổi, sửa lại các tham số khai báo tại đó. Đôi khi, nguồn gây lỗi trên sơ đồ còn được tôn nổi bật thêm bằng mầu, giúp ta nhanh chóng xác định được vị trí của khối gây lỗi.

6.6.3 Tập hợp các tham số trong Script của MATLAB

Đối với các sơ đồ SIMULINK phức hợp, ta không nên trực tiếp khai báo tham số cho từng khối cụ thể, mà nên tập hợp chúng lại trong một *script* (*m-File*). Bằng cách ấy, mọi công việc khai báo hay thay đổi tham số đều có thể được thực hiện một cách rất rõ ràng, tường minh và khó nhầm lẫn.

Để làm như vậy, thay vì viết các giá trị cụ thể, ta chỉ cần viết tên của các biến. Các biến đó sẽ được gán giá trị cụ thể sau này, trong khuôn khổ của *script*. Trước khi bắt đầu mô phỏng hay sau khi thay đổi tham số, ta sẽ phải gọi *script* để nạp các biến vào môi trường *Workspace* của MATLAB. Nhờ vậy, trong quá trình mô phỏng SIMULINK có thể truy cập và sử dụng các biến đã nạp.

Một khả năng khác để kích hoạt một *script* chứa các tham số mô hình, là việc sử dụng các thủ tục *Callback* (*Callback Routines*). Khả năng này cho phép ta tiết kiệm, không cần mất công gọi *script* đó bằng dòng lệnh trong cửa sổ lệnh (*Command Windows*). Một *script*, khi đã được liên kết với tham số *InitFcn* của sơ đồ SIMULINK nhờ lệnh `set_param`, lúc bắt đầu mô phỏng sẽ được kích hoạt, nhưng luôn luôn trước khi đọc *Block Parameters*.

Ví dụ lệnh:

```
set_param('model','InitFcn','model_ini')
```

sẽ liên kết *script* có tên *model_ini.m* với tham số *InitFcn* của mô hình SIMULINK có tên *model.mdl*. Mối liên kết đó sẽ bị hủy nếu ta gọi:

```
set_param('model','InitFcn','')
```

Thông tin: Thủ tục *Callback* nào được gọi và được gọi vào lúc nào, sẽ do lệnh sau đây quyết định:

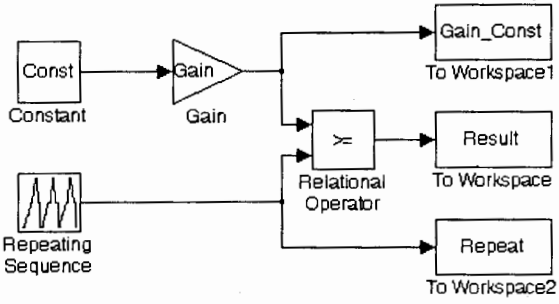
```
set_param(0,'CallbackTracing','on')
```

Lệnh đó sẽ buộc SIMULINK phải liệt kê toàn bộ các thủ tục *Callback* tại cửa sổ *Command* khi chúng được gọi. Để biết thêm về lệnh `set_param` và *Callback Routines* bạn đọc gọi lệnh `help set_param`. Ngoài ra có thể xem chi tiết trong *User's Guide* hay *Online Help* từ của sổ MATLAB.

6.6.4 Ví dụ

Ví dụ sau đây so sánh tín hiệu ra của hai khối *Constant* và *Repeating Sequence* với nhau. Các tín hiệu ra được cất riêng rẽ vào môi trường *Workspace* và hiển thị bằng đồ họa MATLAB. Tham số của các khối được khai báo trong *script* có tên *Fig06_19_ini.m*.

Một *script* khác với nhiệm vụ vẽ đồ thị có tên *Fig06_19_plot.m*. Cả hai *script* có nội dung như dưới đây. Mô hình SIMULINK *Fig06_19.mdl* được giới thiệu ở hình 6.19.



Hình 6.19 *Ví dụ minh họa việc tham số hóa mô hình SIMULINK từ script*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                       %
% File:       Fig06_19_ini.m                                            %
% Comment:    Initialization file for SIMULINK model                    %
%             Fig06_19.mdl                                              %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input
Const = 1.15;
Time = [0:15:135];
Repeat = [0 0 5 5 10 10 20 20 10 10];
% Gain
Gain = 10;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                       %
% File:       Fig06_19_plot.m                                           %
% Comment:    Print file to plot signals of SIMULINK                    %
%             model Fig06_19.mdl                                        %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(2,1,1);
plot(tout, Gain_Const, tout, Repeat);
grid on;
set(gca,'Fontsize',13);
title('Constant and Repeating Sequence');

subplot(2,1,2);
plot(tout, Result);
grid on;
set(gca,'Ylim',[0 2],'Fontsize',13);
title('Result of Comparison');
```

Ví dụ được thực hiện theo ba bước sau:

- Bước 1: Gọi lệnh

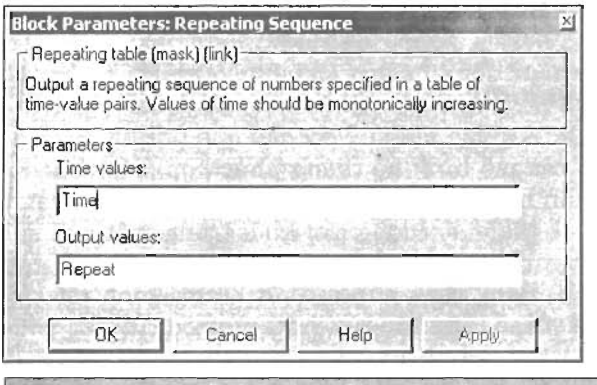
```
>> Fig06_19_ini
```

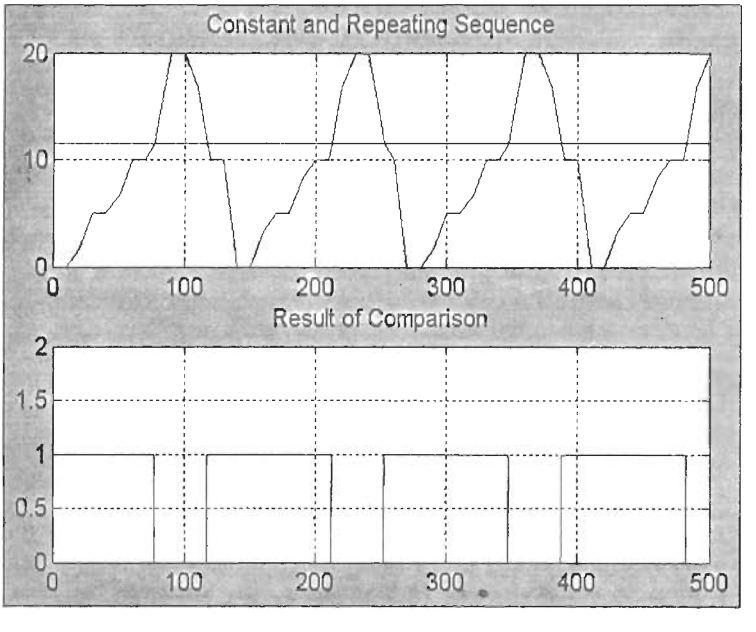
để khai báo tham số với môi trường *Workspace*.

- Bước 2: *Start* mô hình *Fig06_19.mdl*
- Bước 3: Gọi lệnh

```
>> Fig06_19_plot
```

Kết quả mô phỏng được giới thiệu ở hình 6.20, hình dưới. Trong hình đó, các tín hiệu *Gain_Const*, *Repeat* và *Result* (cất với định dạng *Array*) được biểu diễn thành hai đồ thị theo thời gian mô phỏng *tout*, do ta đã buộc SIMULINK tự động cất *tout* bằng cách kích hoạt ô *Save to Workspace* / *Time* trong hộp thoại *Simulation Parameters*.





Hình 6.20 Trái: Khai báo tên tham số cho khối `Repeating Sequence`. *Giá trị của Time và Repeat được gán ở Fig06_19_ini.m. Dưới: Kết quả mô phỏng mô hình Fig06_19.mdl*

6.6.5 In mô hình SIMULINK

Cũng giống như đồ họa MATLAB, ta có thể xuất mô hình SIMULINK dưới các dạng khác nhau. Bằng lệnh `print -s`*model* ta sẽ xuất mô hình có tên *model* ra máy in. Tuy nhiên, nếu in qua menue *File / Print* ta sẽ có nhiều khả năng khai báo tham số in hơn, ví dụ: Chỉ in một *tầng mô hình*[1] nhất định. Trước khi in ta nên chuyển tham số *Paper type* về khổ *A4*, vì một vài máy in có vấn đề khi in theo khổ *usletter*. Có thể làm điều đó từ cửa sổ *Command* của MATLAB.

```
set(gcf,'PaperType','A4')
```

Việc in mô hình SIMULINK thành *File* được thực hiện tương tự như đồ họa MATLAB:

```
print -smodel;                  % Xuất model.mdl ra máy in
print -smodel -dmeta model;     % Xuất ra file model.emf
print -smodel -deps model;      % Xuất ra file model.eps
```

6.7 Hệ thống con (Subsystem)

Để có thể bao quát tốt hơn các mô hình hệ thống phức hợp, SIMULINK tạo điều kiện để người sử dụng phân một hệ thống lớn thành các hệ thống con (mô hình con). Bên cạnh ưu điểm là giảm số lượng các khối trong một cửa sổ mô phỏng, ta có thể gom các khối có liên quan chức năng với nhau thành các hệ thống con có chức năng độc lập. Bằng cách đó, ta thiết kế mô hình mô phỏng theo một cấu trúc có phân tầng với số tầng sâu tùy ý.

6.7.1 Tạo hệ thống con

Có hai cách tạo các hệ thống con:

- Cách 1: Dùng chuột đánh dấu tất cả các khối (thuộc mô hình hệ thống lớn) mà ta muốn gom lại với nhau. Cần chú ý đánh dấu cả các đường tín hiệu kèm theo. Sau đó chọn *Create Subsystem* thuộc menue *Edit*. Các khối bị đánh dấu sẽ được SIMULINK thay thế bởi một khối *Subsystem*. Khi nháy chuột kép vào khối mới, cửa sổ có tên của khối mới sẽ mở ra. Các tín hiệu vào / ra của hệ con sẽ được tự động ghép với hệ thống mẹ bởi các khối *Inport* và *Outport*.
- Cách 2: Dùng khối *Subsystem* có sẵn của thư viện *Signals & Systems*. Sau khi gắp khối đó sang mô hình hệ thống đang mở, ta nháy chuột kép vào khối để mở cửa sổ (đang còn trống rỗng) của khối và lần lượt gắp các khối cần thiết để tạo thành hệ thống con. Việc ghép nối với hệ thống mẹ

[1] Hệ thống phức hợp có thể được phân thành các hệ thống con (Subsystems) có chiều sâu tùy ý, tạo thành các tầng

(hệ thống tầng cấp trên trực tiếp) phải được chủ động thực hiện bằng tay nhờ các khối *Inport* và *Outport*. Đây là cách đi ngược với cách 1: Ta lần lượt tạo các hệ thống con (bắt đầu từ tầng thấp nhất), sau đó nối các hệ thống con để tạo thành hệ thống mẹ (tầng cấp trên trực tiếp).

6.7.2 Thư viện Signals & Subsystems

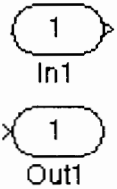
Subsystem



Khối *Subsystem* được sử dụng để tạo hệ thống con trong khuôn khổ của một mô hình SIMULINK. Việc ghép với mô hình thuộc các tầng cấp trên được thực hiện nhờ khối *Inport* (cho tín hiệu vào) và *Outport* (cho tín hiệu ra). Số lượng đầu vào / ra của khối *Subsystem* phụ thuộc số lượng khối *Inport* và *Outport*.

Đầu vào / ra của khối *Subsystem* sẽ được đặt theo tên mặc định của các khối *Inport* và *Outport*. Nếu chọn *Format / Hide Port Labels* trên menue của cửa sổ khối *Subsystem*, ta có thể ngăn chặn được cách đặt tên kể trên và chủ động đặt cho *Inport* và *Outport* các tên phù hợp hơn với ý nghĩa vật lý của chúng.

Inport và Outport



Inport và *Outport* là các khối đầu vào, đầu ra của một mô hình mô phỏng. Tại hộp thoại *Block Parameters* ta có thể điền vào ô *Port number* số thứ tự của khối. SIMULINK tự động đánh số các khối *Inport* và *Outport* một cách độc lập với nhau, bắt đầu từ 1. Khi ta bổ sung thêm khối *Inport* hay *Outpqrt*, khối mới sẽ nhận số thứ tự kế tiếp. Khi xóa một khối nào đó, các khối còn lại sẽ được tự động đánh số mới. Trong hộp thoại *Block Parameters* của *Inport*, ta còn có ô *Port with* dùng để khai báo bề rộng của tín hiệu vào. Khi ghép một tín hiệu có bề rộng lớn hoặc bé hơn bề rộng đã khai báo với *Inport*, ngay lập tức SIMULINK báo lỗi.

Cần lưu tâm đến một vài tham số quan trọng khác của khối *Outport*. Ví dụ, *Output when disabled* cho hệ thống biết cần xử lý tín hiệu ra như thế nào khi hệ thống mô phỏng đang ngừng không chạy (xóa về không hay giữ nguyên giá trị cuối cùng). *Initial Output* cho biết giá trị cần phải lập cho đầu ra.

Thông qua các khối *Inport* và *Outport* thuộc tầng trên cùng (chứ không phải thuộc các hệ thống con), ta có thể cất vào hay lấy số liệu ra khỏi môi trường *Workspace*. Để làm điều đó ta phải kích hoạt các ô *Input* và *Output* ở trang *Workspace I/O* của hộp thoại *Simulation Parameters* và khai báo (ở ô điền chữ bên cạnh) tên của các biến cần lấy số liệu vào, hay tên của các biến mà ta sẽ gửi số liệu tới.

Enable và Trigger


Enable


Trigger

Hai phần tử *Enable* và *Trigger* nhằm mục đích tạo cho các hệ con *Subsystem* khả năng khởi động có điều kiện. Trong một hệ con chỉ có thể sử dụng một khối *Enable* và *Trigger*. Khi được gán một trong hai khối đó, tại khối *Subsystem* sẽ xuất hiện thêm một đầu vào điều khiển đặc biệt, nơi mà tín hiệu *Enable* (cho phép kích hoạt) hay *Trigger* (xung kích hoạt) được đưa tới.

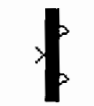
Các hệ con có khối *Enable* được gọi là hệ (đã được) cho phép. Hệ con đó sẽ được kích hoạt tại những bước tích phân có phát ra tín hiệu *Enable* với giá trị dương. Tham số *States when enabling* cho biết cần đặt giá trị ban đầu cho biến trạng thái như thế nào (xóa về không hay giữ nguyên giá trị cuối cùng) trước khi được kích hoạt. Tham số *Show output port* gán cho khối *Enable* thêm một đầu ra, tạo điều kiện xử lý hay sử dụng tiếp tín hiệu *Enable*.

Các hệ con có khối *Trigger* gọi là hệ được kích hoạt bằng xung. Việc kích hoạt xảy ra tại sườn dương (*Trigger type*: *rising*), hay sườn âm (*Trigger type*: *falling*), hay cả hai sườn (*either*) của xung kích hoạt. Nếu *Trigger type* được chọn là *function-call*, ta có cơ hội chủ động tạo xung kích hoạt nhờ một *S-function* (hàm S) do ta tự viết.

Các khối *Enable* và *Trigger* là khối ảo có điều kiện.

Mux và Demux

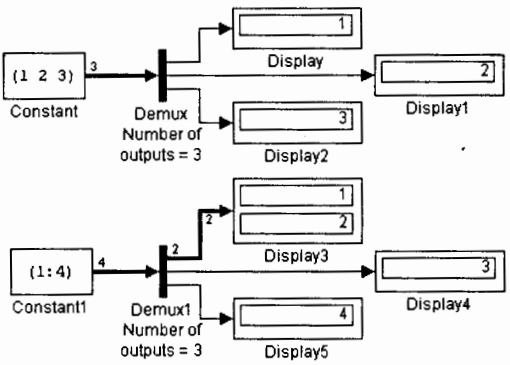

Mux


Demux

Khối *Mux* có tác dụng giống như một bộ chập kênh (*multiplexer*), có tác dụng chập các tín hiệu 1-D riêng rẽ (tức là: các tín hiệu scalar hay vector) thành một vector tín hiệu mới. Nếu như một trong số các tín hiệu riêng rẽ là 2-D (ma trận tín hiệu), khi ấy ta chỉ có thể tập hợp các tín hiệu riêng rẽ thành Bus tín hiệu (mục 6.3.1). Tại ô tham số *Number of inputs* ta có thể khai báo tên, kích cỡ và số lượng tín hiệu vào. Ví dụ: Viết [4 3 –1] nghĩa là có tất cả 3 đầu vào, đầu vào thứ nhất có bề rộng là 4, đầu vào thứ 2 bề rộng là 3, còn đầu vào thứ 3 chưa được xác định vì giá trị khai là -1.

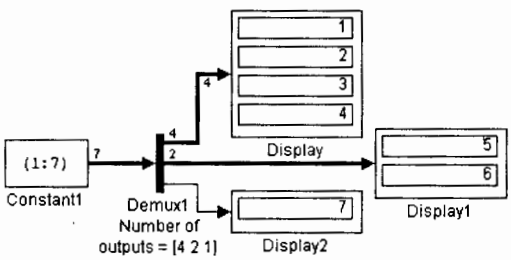
Khối *Demux* có tác dụng ngược với *Mux*: Tách cac tín hiệu được chập từ nhiều tín hiệu riêng rẽ trở lại thành các tín hiệu riêng rẽ mới. Khối *Demux* làm việc hoặc theo chế độ vector (*Bus selection mode* = *off*) hoặc theo chế độ chọn Bus (*Bus selection mode* = *on*). Ở chế độ vector, *Demux* chỉ chấp nhận tín hiệu 1-D ở đầu vào và sẽ tách tín hiệu 1-D đó thành các tín hiệu riêng rẽ như đã khai báo tại *Number of outputs*. Tham số *Number of outputs* có thể được khai báo dưới dạng một số nguyên > 1 hay dưới dạng một vector hàng. Khi khai báo là vector hàng, việc tách các phần tử của tín hiệu vào và phân chia các phần tử đó thành các tín hiệu ra hoàn toàn phụ thuộc vào bề rộng tín hiệu vào, số lượng và bề rộng của tín hiệu ra mà ta khai báo. Khi chọn chế độ *bus selection*, *Demux* chỉ chấp nhận Bus tín hiệu ở đầu vào của khối.

Mux và *Demux* luôn luôn là ảo. Các ví dụ sau đây minh họa rõ thêm cách sử dụng khối *Demux*.



Hình 6.21 *Khối Demux*

Trên: Vector tín hiệu với 3 phần tử được tách thành 3 tín hiệu scalar.
Dưới: Vector tín hiệu với 4 phần tử được tách thành 3 tín hiệu riêng rẽ (tự động tách).



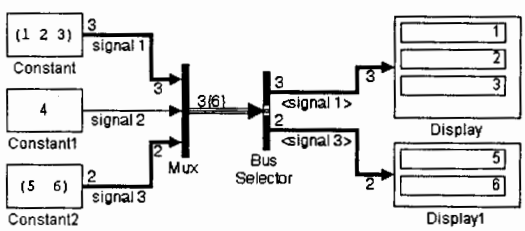
Hình 6.22 *Khối Demux*

Vector tín hiệu với 7 phần tử được tách thành 3 tín hiệu riêng rẽ (người sử dụng tách).

Bus Selector và Selector



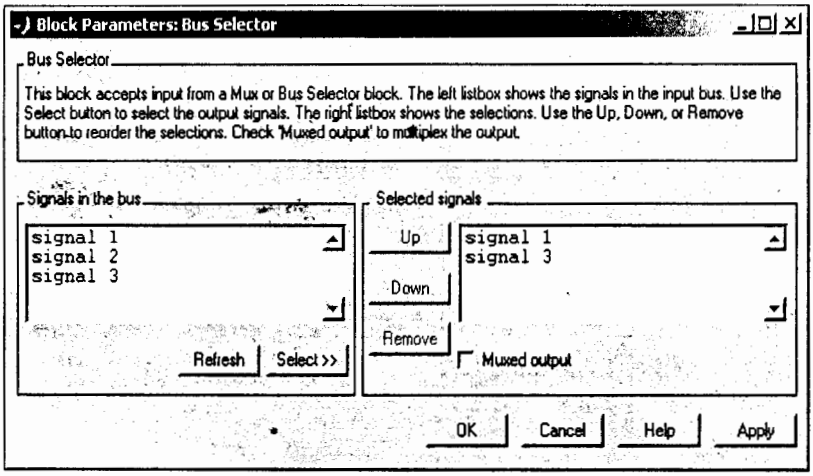
Các tín hiệu do khối *Mux* chập lại, có thể được tách ra không chỉ bằng khối *Demux*. Ta có thể sử dụng khối *Bus Selector* để tái tạo lại các tín hiệu từ một Bus tín hiệu, đồng thời gom chúng lại thành các tín hiệu riêng rẽ ban đầu. Trong ví dụ ở hình 6.23, 3 tín hiệu riêng rẽ của 3 khối *Constant* đã được khối *Mux* chập lại thành Bus tín hiệu (hỗn hợp cả tín hiệu 1-D và 2-D).



Hình 6.23 *Khối Bus Selector*

Tách hai tín hiệu signal 1 và signal 3 từ Bus tín hiệu ở đầu ra của khối Mux.

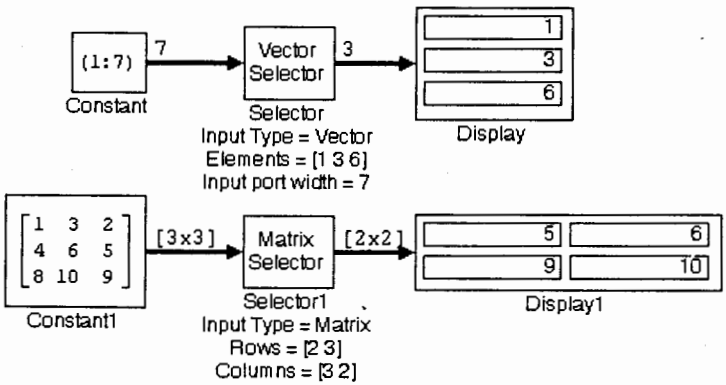
Tại hộp thoại *Block Parameters* của khối *Bus Selector* (hình 6.23b) trong ô *Signals in the bus* ta có thể thấy danh sách liệt kê tất cả các tín hiệu nằm trong Bus. Nhấn nút *Select* >> ta có thể chọn những tín hiệu mà ta cần tách ra khỏi Bus và gom lại như ban đầu.



Hình 6.23b *Hộp thoại Block Parameters của khối Bus Selector*

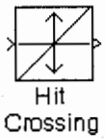
Trong ví dụ ở hình 6.23 ta đã chọn hai tín hiệu *signal*1 và *signal*3 để tách ra khỏi Bus (xem ô *Selected signals* của hình 6.23b). Sau khi chọn, đầu ra của *Bus Selector* có hai tín hiệu. Nếu như ta kích hoạt ô *Muxed output*, khi ấy tín hiệu ra sẽ chỉ là 1.

Khối *Selèctor* cho ta khả năng lựa chọn còn linh hoạt hơn *Bus Selector*: Khả năng tách ra khỏi Bus tín hiệu 1-D (vectơr) hay 2-D (ma trận) các phần tử riêng lẻ để rồi sau đó gom chúng lại thành một tín hiệu 1-D hay 2-D mới. Ví dụ ở hình 6.24 cho ta thấy rõ sự khác biệt ấy. Khối *Bus Selector* luôn luôn là ảo, khối *Selector* là ảo có điều kiện.



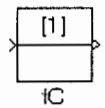
Hình 6.24 *Khối Selector cho phép tách từng phần tử riêng lẻ ra khỏi vector hay ma trận tín hiệu*

Hit Crossing

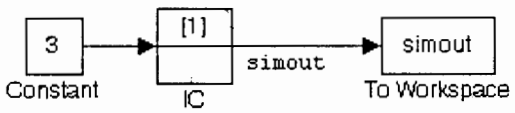


Khối *Hit Crossing* có tác dụng phát hiện thời điểm tín hiệu đầu vào đi qua giá trị khai tại *Hit crossing offset* theo hướng khai tại *Hit crossing direction*. Nếu ta chọn *Show output port*, tại thời điểm Crossing đầu ra sẽ nhận giá trị 1, còn lại là 0. Nếu tại trang *Advanced* của hộp thoại *Simulation Parameters* ta đặt *Boolean logic signals* = *off*, tín hiệu ra sẽ là *double*, ngoài ra là *boolean*.

IC

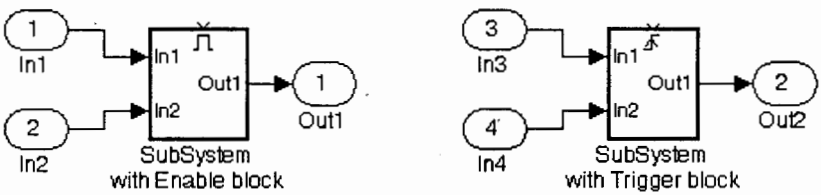


Khối IC có tác dụng gán cho tín hiệu ra của khối một điều kiện ban đầu nhất định. Trong ví dụ ở hình dưới đây, tín hiệu `simout` có giá trị 1 tại thời điểm $t = 0$, ngoài ra là 3.



6.7.3 Kích hoạt có điều kiện các hệ thống con

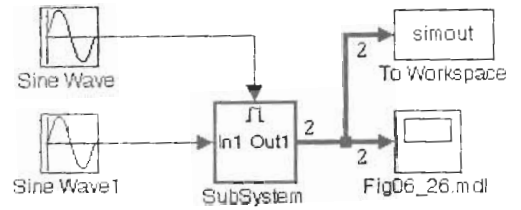
Các hệ thống con có chứa khối *Enable* hay *Trigger*[1] gọi là các hệ (đã được) cho phép kích hoạt hay hệ kích hoạt bằng xung. Việc kích hoạt hệ con hoàn toàn do tín hiệu điều khiển tương ứng xác định. Như ví dụ dưới đây (hình 6.25) cho thấy, khi gán khối *Enable* hay *Trigger* cho một hệ con, hệ đó sẽ tự động có thêm một đầu vào giành cho tín hiệu điều khiển.

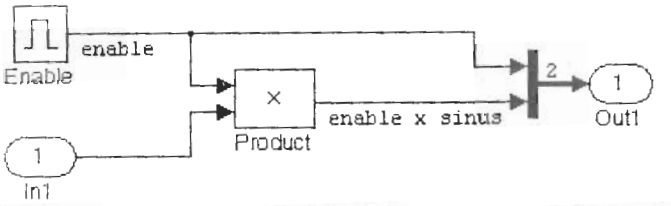


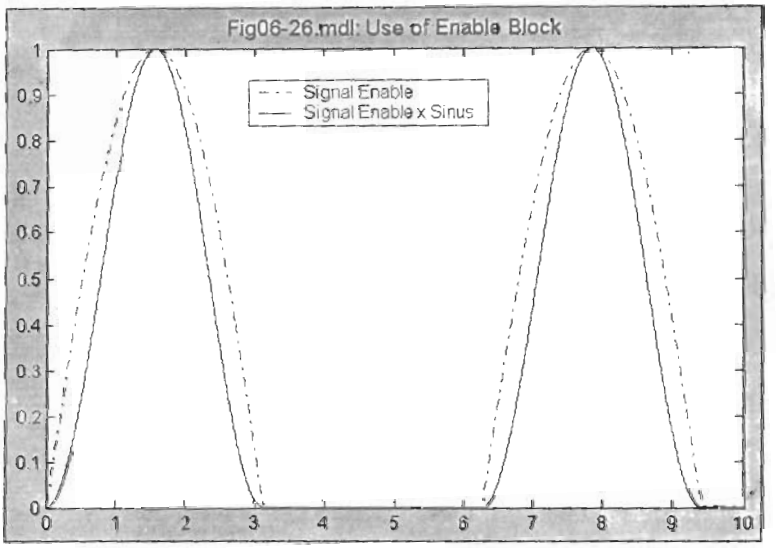
Hình 6.25 *Ví dụ minh họa các hệ thống con được kích hoạt theo điều kiện*

Ví dụ tiếp theo (hình 6.26) giải thích rõ hơn nữa cách sử dụng khối *Enable*. Hai tín hiệu hình sin có cùng biên độ và tần số được đưa tới một *Subsystem*. Tín hiệu sin thứ nhất được đưa tới đầu vào *Enable*, tín hiệu sin thứ hai được đưa tới đầu vào *In1* của hệ con. Bên trong hệ con, tín hiệu của *In1* được nhân (khối *Product*) với tín hiệu ra của khối *Enable*. Tín hiệu đầu ra của khối *Product* và của khối *Enable* được chập kênh, đưa tới khối *Scope* và cất vào *Workspace* dưới định dạng *Array* để sau đó `plot` thành đồ thị như hình 6.26 (dưới).

[1] Mỗi hệ thống con chỉ được phép chứa một khối *Enable* hay *Trigger*







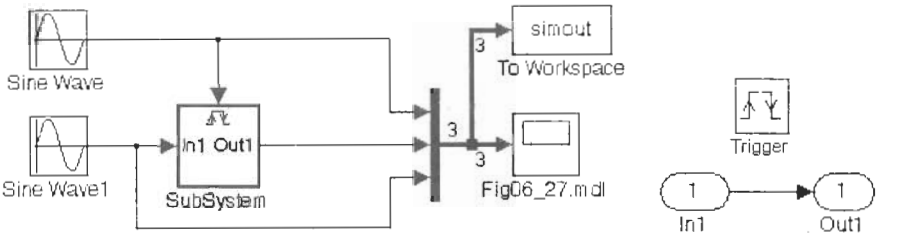
Hình 6.26 Trên: *Ví dụ về hệ con kích hoạt có điều kiện.* ***Giữa:*** *Hệ con có khối* ***Enable.*** ***Dưới:*** *Đồ thị của hai tín hiệu ra cất vào Workspace*

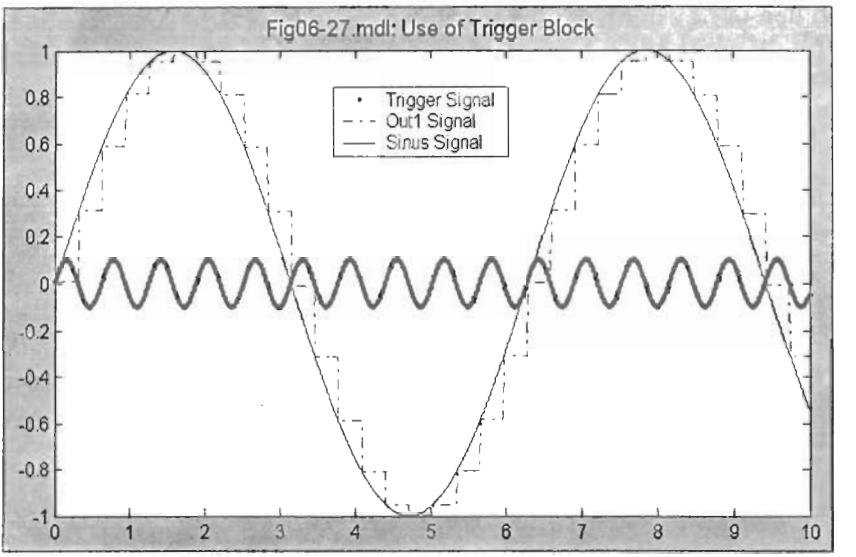
Sau khi chạy Fig06_26.mdl, để thu được đồ thị trên ta cần thực hiện các dòng lệnh sau:

```
>> plot(tout,simout(:,1),'r-.',tout,simout(:,2),'g-');
>> title('Fig06-26.mdl:Use of Enable Block','FontSize',12);
>> legend('Signal Enable','Signal Enable x Sinus');
```

Ví dụ tiếp theo (hình 6.27) sẽ minh họa tác dụng của khối *Trigger*. Trong sơ đồ mô phỏng: Hai tín hiệu hình sin có tần số khác nhau được đưa tới một hệ con. Khối *Sine Wave* thứ nhất cung cấp tín hiệu *Trigger* cho hệ con. Khối *Sine Wave* thứ hai chỉ được đưa tượng trưng qua hệ con. Ba tín hiệu *Trigger*, tín hiệu ra của *Subsystem* và *Sine Wave* thứ hai được đưa tới *Scope*. Tương tự ví dụ

trước, cả ba tín hiệu được plot để ta so sánh. Tuy nhiên, tham số của khối *Trigger* đã được đặt vào *either*, nghĩa là: Hệ con được kích hoạt bằng cả hai sườn lên và xuống của xung kích hoạt. Như hình 6.27 (dưới) cho thấy, nhờ đã sử dụng cả hai sườn của xung kích hoạt nên tần số trích mẫu của tín hiệu *Sine Wave1* có tần số gấp đôi so với tần số của tín hiệu *Trigger*.





Hình 6.27 Trên (trái): Ví dụ về hệ con được kích hoạt bằng xung. Trên (phải): Hệ con có khối Trigger. Dưới: Đồ thị của ba tín hiệu được cất vào Workspace

Sau khi chạy *Fig06_27.mdl*, tức là khi: Kết quả mô phỏng đã được cất vào *Workspace*, ta thực hiện chùm lệnh dưới đây để thu được đồ thị như hình 6.27 (dưới).

```
>> plot (tout,simout(:,1),'r*',tout,simout(:,2),'b-.', ...
        tout,simout(:,3),'k-');
>> title ('Fig06-27.mdl: Use of Trigger Block', ...
         'FontSize',12);
>> legend ('Trigger Signal','Out1 Signal','Sinus Signal');
```

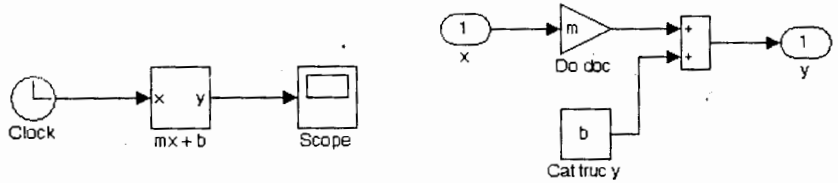
6.7.4 Đánh dấu các hệ con (Mask Subsystems)

Khả năng đánh dấu các hệ con cho phép làm đơn giản việc tham số hóa hệ thống con rất nhiều. Nhờ đánh dấu, tất cả tham số cần thiết sẽ được gom lại trong một hộp thoại *Block Parameters* duy nhất. Nhờ đánh dấu, một hệ con phức hợp sẽ được thu lại chỉ còn là một khối với các đầu vào / ra tương ứng. Đây chính là cách để tạo ra một thư viện con mới.

Các biến sử dụng trong hệ con đã đánh dấu (*Subsystem*) được gán giá trị cụ thể tại hộp thoại *Block Parameters*. Ví dụ: Hai hệ số *m* và *b* của biểu thức *mx+b* trong hình 6.28. Như vậy, các biến đó chỉ có ý nghĩa cục bộ (*local*) và không thể bị truy cập và xử lý qua môi trường *Workspace*. Điều này bảo đảm tính độc lập của khối đã đánh dấu (giống như các khối chuẩn) và do đó ta có thể sử dụng chúng lặp lại nhiều lần trong cùng một mô hình mô phỏng.

Ví dụ minh họa

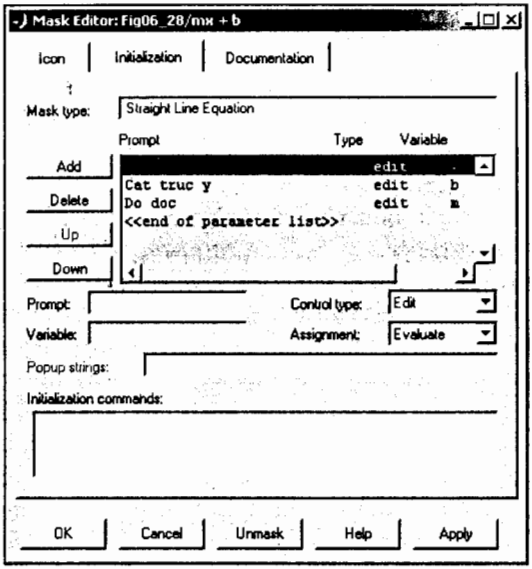
Để minh họa ta hãy đánh dấu hệ con *mx+b* trong hình 6.28 (trái). Hệ đó tính phương trình đường thẳng $y = mx+b$ có độ dốc *m* và giao điểm với trục tung tại *b*, là hai tham số có thể được đặt giá trị bất kỳ từ hộp thoại *Block Parameters* của hệ (sau khi đã đánh dấu), chứ không phải vào tận khối *Gain* hay *Constant* (hình 6.28, bên phải: khối *Do doc* và *Cat truc y*) để khai báo giá trị nữa.

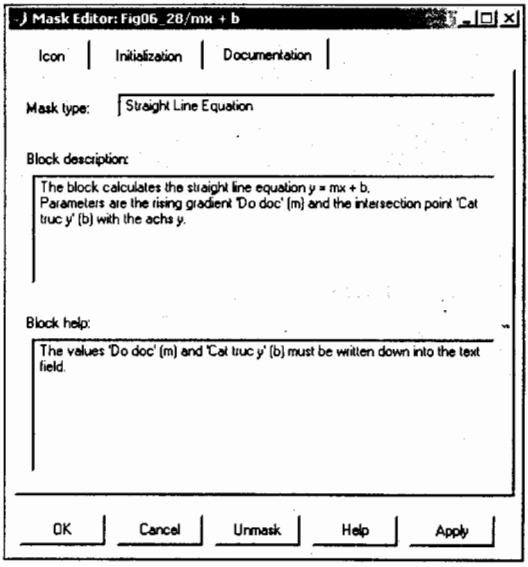


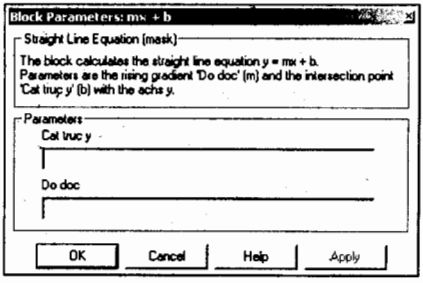
Hình 6.28 *Ví dụ đánh dấu hệ con.* ***Trái:*** *Mô hình SIMULINK có Subsystem.* ***Phải:*** *Sơ đồ của Subsystem mx + b*

Khi ta đánh dấu hệ con *mx+b* và gọi *Mask Subsystem* ở menue *Edit*, hộp thoại *Mask Editor* sẽ mở ra. Trang *Icon* có chứa các khả năng trình bầy *Icon* (trang trí ngoại hình) của khối mới. Trang *Initialization* giành cho việc định nghĩa các tham số, trang *Documentation* cho phép ta viết một đoạn *Text* ngắn mô tả chức năng và nội dung *Help* (phần *Text* mở ra khi nhấn nút *Help*) của khối mới.

Tại ô *Prompt* của trang *Initialization* (hình 6.29, phía trên, bên trái) ta khai báo tên của tham số, tại ô *Variable* khai ký hiệu của biến đặc trưng cho tham số đó. Cần chú ý rằng, đó chính là tên của tham số và tên biến sau này sẽ xuất hiện tại hộp thoại *Block Parameters* của khối mới. Nháy chuột vào nút *Add*, hai tên (tham số và biến đặc trưng cho tham số) sẽ được nhập vào danh sách (*parameter list*). Ô *Control Type* được đặt là *Edit*, có nghĩa là: Phía dưới tên của tham số phải dành một dòng trống để ta viết (khai báo) giá trị của tham số. Giá trị đó sau này sẽ được MATLAB (do chọn *Evaluate* tại ô *Assignment*) đọc vào và gán cho biến tương ứng (trong ví dụ là *m* và *b*).







Hình 6.29 *Đánh dấu hệ con 'mx+b'. Trái: trang Initialization. Dưới (bên trái): trang Documentation. Dưới (bên phải): hộp thoại Block Parameters của khối mới xuất hiện sau khi đánh dấu*

Tại ô *Mask type* ta điền ký hiệu mà ta muốn gán cho khối mới. Khi đã xong tất cả các khai báo, ta nhấn nút *Apply* để kết thúc, mọi khai báo đã được khẳng định. Về sau, khi nháy chuột kép vào khối (thay thế hệ con) trong ví dụ *Fig06_28.mdl* ta sẽ không thu được cửa sổ của hệ con nữa, mà là hộp thoại *Block Parameters* như hình 6.29 (hình dưới, bên phải).

Tại ô *Block description* (hình 6.29, dưới, bên trái) của trang *Documentation* ta nhập đoạn ký tự mô tả chức năng của khối mới, là đoạn ký tự sẽ xuất hiện tại hộp thoại *Block Parameters* (hình 6.29, dưới, bên trái).

Tại trang *Icon* ta có thể tiến hành tạo dáng và trang trí biểu tượng của khối. Tại đó có một số lệnh phục vụ việc biểu diễn lời văn, đường nét cũng như hàm truyền đạt. Ta có thể gán cả hình ảnh với định dạng TIFF. Ví dụ đối với hệ con *mx+b* ta sử dụng các lệnh `plot([0 1],[0.1 0.1]) plot([0.1 0.1],[0 1])` để vẽ một hệ tọa độ *xy* và `plot([0 2],[0.3 3])` để vẽ một đoạn thăng với $m > 0$ và $b > 0$. Tại ô *Drawing coordinates* ta chọn *Normalized* (chuẩn hóa: góc bên trái phía dưới có tọa độ (0,0); góc bên phải phía trên có tọa độ (1,1)). Bằng các tham số *Icon frame*, *Icon transparency* và *Icon rotation* ta có thể khai báo thêm các đặc điểm của khung (đặc hay trong suốt) và khả năng xoay đồ họa trang trí. Nếu thay cho đồ họa hay hình ảnh, ta muốn gán lời văn thì có thể sử dụng lệnh vẽ (*Drawing commands*) như `disp('Text')` hoặc `disp(`*Variable name*`)`. Cần lưu ý: *Variable name* phải là tên của biến đã định nghĩa tại ô *Prompts* của trang *Initialization*. Khi ấy, trong khuôn hình của khối sẽ luôn xuất hiện giá trị được gán cho biến (với điều kiện ta đã khai báo giá trị cho biến đó). Khi gọi `disp('Text')` ta có thể sử dụng lệnh `\n` để cưỡng *Text* xuống dòng.

Một khi đã khai báo xong xuôi khối mới mà ta muốn thay đổi, ta có thể gọi *Mask Editor* bằng cách đánh dấu khối, sau đó chọn menue *Edit / Edit Mask*. Khi chọn menue *Edit / Look Under Mask*, cửa sổ của hệ con sẽ mở ra hệt như khi ta nháy chuột kép vào hệ con (*Subsystem*) chưa qua đánh dấu.

6.8 Tóm tắt nội dung chương 6

Sau khi đã nghiên cứu kỹ chương 6, người đọc cần nắm vững các nội dung sau đây:

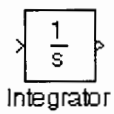
1. Xây dựng và biên soạn lưu đồ cấu trúc cho mô hìn SIMULINK
2. Sử dụng các khối của các thư viện con *Sources*, *Sinks*, *Math* và *Signals & Systems*
3. Hiểu biết cơ sở về các tham số của hộp thoại *Simulation Parameters*
4. Cơ sở về giám sát sai số và nhận biết các điểm không liên tục
5. Các khả năng gọi ra hoặc cất số liệu vào môi trường MATLAB *Workspace*
6. Sử dụng lệnh `sim` và `set_param` để điều khiển quá trình mô phỏng từ MATLAB
7. Gom các tham số của khối trong một *script*
8. In mô hình SIMULINK ra máy in hoặc ra *File* ảnh
9. Xây dựng và kích hoạt có điều kiện các hệ con
10. Hiểu biết cơ sở về đánh dấu hệ con

7 Các hệ thống tuyến tính và phi tuyến

Trên cơ sở các hiểu biết ban đầu về SIMULINK ở chương 6, trong chương 7 ta bổ xung thêm các khối đặc trưng cho động học của các hệ liên tục về thời gian (lấy từ thư viện con *Continuous*), các phần tử phi tuyến trong hệ như khối bão hòa hay khâu trễ (thư viện con *Nonlinear*), các hàm khả trình của MATLAB và các "bảng tra" của thư viện con *Function & Tables*. Chương 7 cũng đề cập tới các vấn đề về "vòng lặp đại số" và lập trình hàm S (*S Functions*).

7.1 Thư viện Continuous

Integrator



Khối *Integrator* lấy tích phân tín hiệu đầu vào của khối. Giá trị ban đầu được khai báo hoặc trực tiếp tại hộp thoại *Block Parameters*, hoặc thông qua chọn giá trị *internal* tại ô *Initial condition source* để sau đó điền giá trị ban đầu vào dòng viết của ô *Initial condition*. Nếu *Initial condition source* được chọn là *external*, trên biểu tượng của khối sẽ xuất hiện thêm một đầu vào thứ hai giành cho giá trị ban đầu lấy từ nguồn bên ngoài khối.

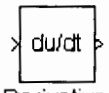
Đầu ra của khối *Integrator* có thể được một tín hiệu bên ngoài lập về (*reset*) một giá trị ban đầu. Tại ô *External reset* ta có thể chọn dạng của tín hiệu *reset* (ví dụ: *rising* có nghĩa là sườn lên). Khi chọn cho *External reset* một trong các giá trị *rising*, *falling*, *either* hay *level*, khối *Integrator* sẽ tự động có thêm một đầu vào giành riêng cho tín hiệu *reset*.

Nếu cần chặn biên độ tín hiệu ra, ta kích hoạt ô *Limit output* và khai báo giá trị tối đa tại dòng (*Upper* hay *Lower*) *saturation limit*. Khi kích hoạt ô *Show saturation port* ta có thể lấy được tín hiệu bão hòa tương ứng: 1 tại đầu ra khi có bão hòa dương, -1 khi bão hòa âm, còn 0 cho các giá trị lưng chừng giữa hai ngưỡng bão hòa.

Nếu kích hoạt ô *Show state port*, trên biểu tượng của khối sẽ xuất hiện thêm một đầu ra (*state port*) cho phép trích tín hiệu trạng thái của khối *Integrator*. Biến trạng thái của khối thực chất đồng nhất về giá trị với biến đầu ra, tuy nhiên, SIMULINK tính hai biến đó (biến ra và biến trạng thái) tại những thời điểm ít nhiều có khác nhau. Trong một vài trường hợp việc sử dụng biến trạng thái là cần thiết: Giả sử tín hiệu ra được đưa hồi tiếp trở về đầu vào xóa hay lập trị ban đầu, sẽ xuất hiện vòng lặp, nếu đưa về đầu vào lập

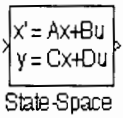
trị ban đầu, thậm chí còn là vòng lặp đại số. Có thể phòng ngừa điều đó bằng cách hồi tiếp biến trạng thái, thay cho biến đầu ra.
Nếu mô hình SIMULINK chứa các biến trạng thái rất chênh lệch nhau về kích cỡ giá trị, khi ấy nên khai báo tham số *Absolute tolerance* riêng rẽ thêm cho từng khối *Integrator* của mô hình, mặc dù ta đã khai *Absolute tolerance* chung tại hộp thoại *Simulation Parameters*. Việc khai báo thêm sẽ buộc SIMULINK sẽ bảo đảm đúng giá trị sai số yêu cầu đối với từng khối.

Derivative



Phép tính đạo hàm tín hiệu đầu vào được thực hiện nhờ khối *Derivative*. Tín hiệu tìm được ở đầu ra có dạng $\Delta u / \Delta t$, với Δ là biến thiên của đại lượng cần tính kể từ bước tích phân liền trước đó. Giá trị ban đầu của biến ra là 0.

State-Space

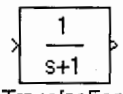


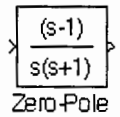
Khối *State-Space* là mô hình trạng thái của một hệ tuyến tính, được mô tả bởi hệ phương trình sau đây (xem Control System Toolbox, mục 3.1.3):

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \end{aligned} \tag{7.1}$$

Bậc của hệ do kích cỡ của ma trận trạng thái **A** quyết định. Nếu *Nx* là số biến trạng thái, *Nu* là số biến vào và *Ny* là số biến ra, các tham số của hệ sẽ phải có kích cỡ như sau: **A**(*Nx* × *Nx*), **B**(*Nx* × *Nu*), **C**(*Ny* × *Nx*) và **D**(*Ny* × *Nu*). Khai báo như vậy ta đã quyết định *Nu* là bề rộng của vector tín hiệu vào và *Ny* là bề rộng của vector tín hiệu ra.

Transfer Function và Zero-Pole





Nhờ khối *Transfer Fcn* ta có thể mô hình hóa hàm truyền đạt của một hệ tuyến tính. Khối *Transfer Fcn* có đặc điểm hoàn toàn tương đương với lệnh `tf(num,den)` của *Control System Toolbox*. Tham số của khối là các hệ số của đa thức tử số (*Numerator*) và mẫu số (*Denominator*), khai báo theo thứ tự số mũ của *s* giảm dần. Bậc của mẫu số phải lớn hơn hoặc bằng bậc của tử số. Ví dụ: Nếu nhập cho *Numerator* [4 3 0 1] và *Denominator* [1 0 7 3 1], khối sẽ tạo ra hàm truyền đạt:

$$H(s) = \frac{y(s)}{u(s)} = \frac{4s^3 + 3s^2 + 1}{s^4 + 7s^2 + 3s + 1} \tag{7.2}$$

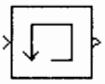
Tại ô *Numerator* ta cũng có thể khai báo ma trận. Bề rộng của vector tín hiệu ra sẽ tùy thuộc vào số dòng của ma trận đã được khai. Các tín hiệu vào chỉ được chấp nhận là scalar.

Ngược lại với *Transfer Fcn*, khối *Zero-Pole* sẽ tạo nên từ các tham số *Zeros*, *Poles* và *Gain* một hàm truyền đạt dưới dạng hệ số hóa theo điểm không, điểm cực (m = số lượng điểm không, n = số lượng điểm cực). Ví dụ:

$$H(s)=K\frac{\mathbf{Z}(s)}{\mathbf{P}(s)}=K\frac{(s-Z(1))(s-Z(2))\ldots(s-Z(m))}{(s-P(1))(s-P(2))\ldots(s-P(n))} \quad (7.3)$$

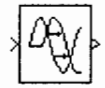
$\mathbf{Z}(s)$ có thể là một vector hay ma trận chứa các giá trị điểm không của hệ, $\mathbf{P}(s)$ là vector các giá trị điểm cực. Đại lượng scalar (hay vector) K có chứa hệ số (hay các hệ số) khuếch đại của hệ thống. Số lượng phần tử của K ứng với số dòng của $\mathbf{Z}(s)$. Số lượng điểm cực phải lớn hơn hoặc bằng số lượng điểm không. Các điểm không, điểm cực phức phải là phức liên hợp. Khối *Zero-Pole* hoàn toàn tương đương với lệnh `zpk(z,p,k)` của *Control System Toolbox*.

Memory



Khối *Memory* xuất ở đầu ra giá trị của đầu vào thuộc bước tính phân vừa qua. Như vậy, khối *Memory* có đặc điểm của một khâu giữ chậm bậc 0, lưu giữ tín hiệu trong khoảng thời gian một bước tính. Khi mô phỏng với *ode15s* và *ode113* (cả hai đều là phương pháp tích phân đa bước) ta không nên sử dụng khối *Memory*.

Transport Delay và Variable Transport Delay





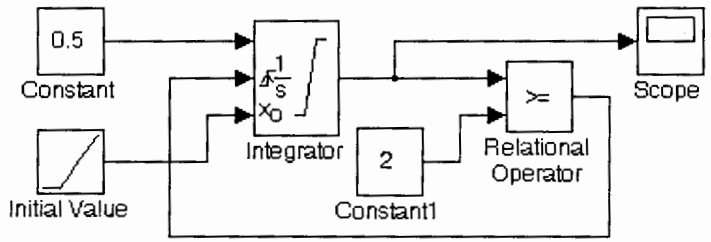
Khối *Transport Delay* làm trễ tín hiệu vào khoảng thời gian ≥ 0 khai báo tại ô *Time delay* trước khi xuất tới đầu ra. Chỉ đến khi thời gian mô phỏng bắt đầu vượt quá thời gian trễ (so với lúc bắt đầu mô phỏng), khối *Transport Delay* mới xuất giá trị khai tại *Initial input* tới đầu ra. Để tạm cất số liệu, SIMULINK sử dụng một vùng nhớ đệm với kích cỡ do *Initial buffer size* quyết định. Nếu số liệu tạm cất vượt quá khả năng vùng nhớ đệm đã khai, SIMULINK sẽ tự động bổ sung thêm dung lượng cần thiết nhưng sẽ làm quá trình mô phỏng chậm đi đáng kể.

Bằng khối *Variable Transport Delay* ta có thể điều khiển trễ tín hiệu một cách rất linh hoạt: Tín hiệu chứa thời gian trễ được đưa tới đầu vào thứ hai (đầu vào phía dưới) của khối. Tại ô *Maximum delay* ta phải khai một giá trị trễ tối đa, có tác dụng giới hạn (chặn trên) giá trị của tín hiệu điều khiển thời gian trễ. Các tham số *Initial input* và *Initial buffer size* có chức năng giống như đã mô tả khối *Transport Delay*.

Cả hai khối đều có tham số *Pade order*, nơi ta khai báo một số nguyên ≥ 0, xác định bậc của phép xấp xỉ Pade mà hai khối sử dụng. Tham số này rất có ý nghĩa nếu ta định tuyến tính hóa một sơ đồ SIMULINK. Giá trị mặc định *Pade order* = 0 cho biết: Khi tuyến tính hóa, hai khối được thay bởi hệ số khuếch đại = 1.

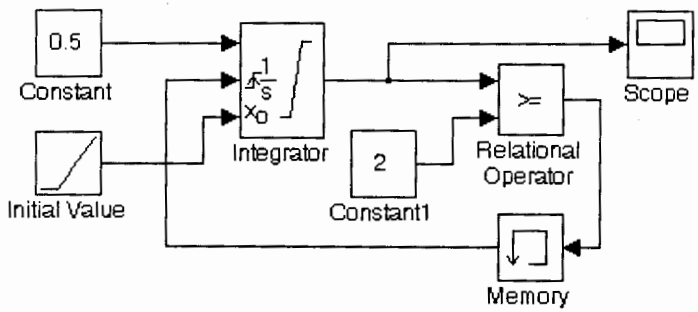
Ví dụ

Ví dụ này chủ yếu lý giải vấn đề *reset* khối *Integrator*. Một khối *Integrator* tính tích phân giá trị 0,5 của khối *Constant*. Mỗi khi tín hiệu ra của khối *Integrator* đạt tới giá trị 2, đầu ra của khối lại được *reset* (*External reset* = *rising*) và chân x_0 nhận giá trị do khối *Ramp* (có tên *Initial Value*: hình 7.1) đưa tới làm giá trị ban đầu. Cấu trúc (hình 7.1) như vậy đã tạo nên một vòng quẩn, bởi vì đầu ra của khối *Relational Operator* (tức là tín hiệu *reset* của chính khối *Integrator*) lại phụ thuộc trực tiếp đầu ra của khối *Integrator*.



Hình 7.1 *Ví dụ reset khối Integrator: Tại đây xuất hiện vòng quẩn*

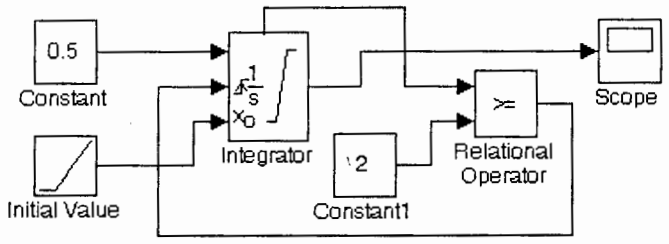
Để khắc phục vấn đề nẩy sinh ở hình 7.1 ta bổ sung thêm khối *Memory* ở nhánh dẫn tín hiệu *reset* như hình 7.2. Tuy nhiên, khi sử dụng *Solvers* có bước linh hoạt *Variable step* ta dễ dàng nhận thấy nhược điểm trễ tín hiệu *reset* (làm sai lạc đặc tính thực của hệ thống) do khối *Memory* gây nên (hình 7.4, bên trái).



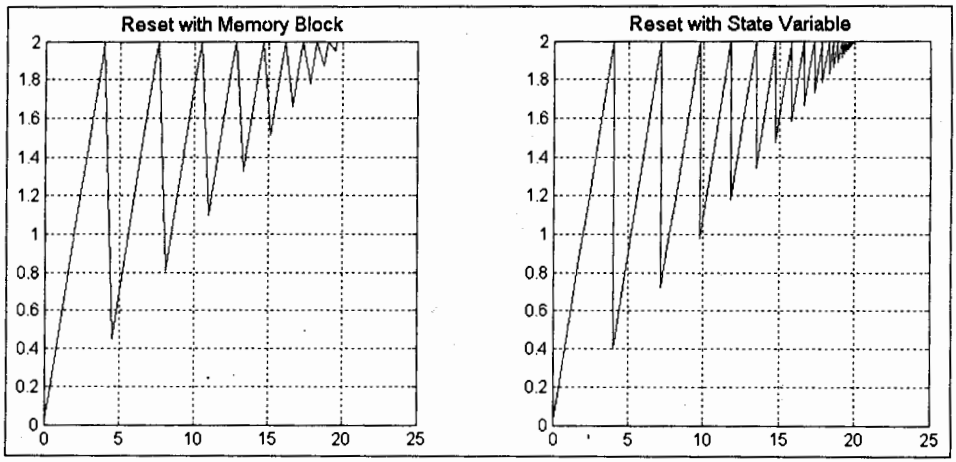
Hình 7.2 *Ví dụ reset khối Integrator: Ngăn ngừa vòng quẩn bằng khối Memory*

Giải pháp thông minh hơn là giải pháp ở hình 7.3: Thay vì sử dụng tín hiệu ra, tín hiệu trạng thái của khối *Integrator* được sử dụng để tạo tín hiệu *reset* (kích hoạt ô *Show state port*). Kết quả được giới thiệu ở hình 7.4 (bên phải).

Khi tín hiệu độ dốc ở đầu ra của khối *Ramp* tăng lớn hơn 2, tín hiệu ra của khối *Integrator* sẽ tăng tỷ lệ thuận với giá trị ban đầu (đặt tại x_0). Để tránh điều đó ta đã giới hạn đầu ra về giá trị 2 (kích hoạt *Limit output*, *Upper saturation limit* = 2).



Hình 7.3 *Ví dụ reset khối Integrator: Ngăn ngừa vòng quẩn bằngtín hiệu trạng thái*



Hình 7.4 *Số liệu Scope của hình 7.2 (**trái**): Reset bằng tín hiệu ra qua khối Memory, và hình 7.3 (**phải**): Reset trực tiếp bằng tín hiệu trạng thái*

7.2 Tuyến tính hóa

Việc tuyến tính hóa các mô hình SIMULINK có thể được thực hiện nhờ lệnh `linmod` của MATLAB. Hệ đã qua tuyến tính hóa sẽ được xuất ra dưới dạng mô hình trạng thái với các ma trận **A**, **B**, **C** và **D**. Các đầu vào / ra của hệ (thuộc tầng cao nhất khi hệ chứa cả các hệ con) phải được thể hiện rõ trong sơ đồ SIMULINK bằng các khối *Inport* và *Outport*. Các khối chức năng của thư viện con *Sources* và *Sinks* không được SIMULINK coi là đầu vào, đầu ra. Tuy nhiên, việc sử dụng hai kiểu khối đó không loại trừ lẫn nhau, ví dụ: Có thể sử dụng các khối *Inport* song song với các khối thuộc *Sources*, nếu đầu ra của chúng cùng đưa tới khối tính tổng *Sum*. Lệnh `linmod` được sử dụng như sau:

$$[A,B,C,D] = \texttt{linmod}(\texttt{'}sys\texttt{'},x,u,pert,xpert,upert)$$

Trong lệnh trên, *sys* là tên của mô hình SIMULINK. Thông qua vector biến trạng thái x và vector biến vào u ta có thể khai báo một điểm làm việc mà ta cần tuyến tính hóa mô hình xung quanh đó. Các giá trị mặc định của chúng là 0. Bằng tham số *pert* ta xác định một hệ số nhiễu (*perturbation*), đặc trưng cho nhiễu nội tại đối với vector biến trạng thái và vector biến vào khi tìm mô hình trạng thái của sơ đồ SIMULINK. Nếu không nhận được một giá trị cụ thể, *pert* sẽ được đặt mặc định là 10^{-5}. Bằng *xpert* và *upert* ta có thể khai báo hệ số nhiễu riêng rẽ cho vector các biến trạng thái và vector các biến vào, khi ấy *pert* sẽ bị bỏ qua.

Bên cạnh lệnh `linmod` còn có lệnh `linmod2` với cách viết (*syntax*) cũng hệt như vậy. Điểm khác duy nhất là `linmod2` có kết quả tính cao hơn, do sử dụng một thuật toán khác, cho phép giảm thiểu sai số làm tròn. Khi tính ma trận hệ thống, mỗi biến trạng thái sẽ nhận được một hệ số nhiễu riêng và tham số *pert* chỉ còn mang ý nghĩa là giá trị giới hạn dưới của các hệ số đó. Giá trị mặc định là 10^{-8}. Do `linmod2` chính xác hơn nên thời gian tính thường cũng kéo dài hơn.

Nếu mô hình cần tuyến tính hóa có chứa khối *Derivative* hoặc *Transport Delay*, trước khi tuyến tính hóa bằng `linmod` nên tìm cách thay thế bởi các khối tương ứng của *Control System Toolbox*, hay bởi khối *Switched derivative for linearization* hoặc *Switched transport delay for linearization* thuộc thư viện con *Extras / Linearization* của SIMULINK. Có thể tới thư viện con *Extras / Linearization* bằng cách nháy chuột kép vào ký hiệu *Blocksets & Toolboxes* trong cửa sổ thư viện chính (hình 6.1, *Library*: *simulink3*) hay vào nhánh *Simulink Extras* của thư mục truy cập (hình 6.2, *Simulink Library Browser*).

Nếu sơ đồ khối SIMULINK đã có sẵn dưới dạng mô hình trạng thái (với các ma trận **A**, **B**, **C** và **D**), ta còn có thể sử dụng thêm một số lệnh khác của *Control System Toolbox* (mục 3.1). Ví dụ với:

```
sys = ss(A,B,C,D)
```

ta chuyển sơ đồ khối SIMULINK sang dạng đối tượng LTI (*Linear Time Invariant*), để rồi áp dụng lệnh

```
bode(sys)
```

vẽ đồ thị BODE của đối tượng LTI là *sys* và biểu diễn trong MATLAB *Figure*.

Với lệnh:

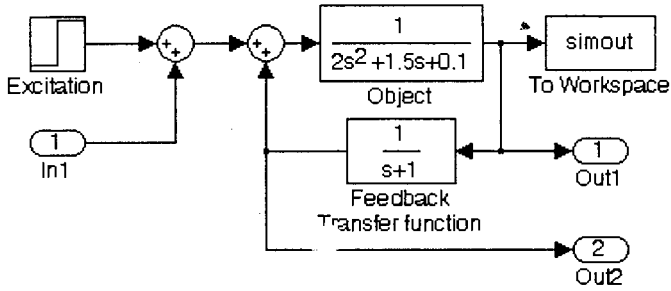
```
[den,num] = ss2tf(A,B,C,D)
```

ta buộc MATLAB xuất các hệ số của đa thức mẫu số *den* và tử số *num* (theo trình tự số mũ giảm dần) của hàm truyền đạt.

Ngoài ra, để tuyến tính hóa các hệ trích mẫu ta sử dụng lệnh `dlinmod` (mục 8.4 sẽ quay lại với lệnh này).

Ví dụ

Hình 7.5 minh họa một mô hình quán tính bậc 2, có đầu ra được đưa hồi tiếp trở lại đầu vào qua một khâu quán tính bậc nhất. Các đầu ra của hệ chính là đầu ra của hai khối *Transfer Fcn*, đầu vào chính là điểm tác động của tín hiệu kích thích.



Hình 7.5 Ví dụ về tuyến tính hóa hệ thống: Mô hình SIMULINK có tên Fig07_05.mdl

Sau khi đã xây dựng mô hình SIMULINK như hình 7.5 và cất thành *File* tên *Fig07_05.mdl*, ta gọi:

```
>> [A,B,C,D]=linmod('fig07_05')
A =
   -0.7500   -0.0500    1.0000
    1.0000         0         0
         0    0.5000   -1.0000
B =
    1.0000
         0
         0
C =
         0    0.5000         0
         0         0    1.0000
D =
     0
     0
```

Kết quả sau lệnh `linmod` là mô hình trạng thái với các ma trận hệ thống.

7.3 Xác định điểm cân bằng

Bằng lệnh trim của MATLAB ta có thể xác định điểm cân bằng (trạng thái xác lập, $\dot{\mathbf{x}} = \mathbf{0}$) của mô hình SIMULINK. Lệnh:

```
[x,u,y] = trim('sys',x0,u0,y0)
```

sẽ tìm thấy điểm cân bằng của mô hình sys, nằm gần điểm $x0$, $u0$, $y0$ nhất. Các hệ phi tuyến có thể có nhiều điểm cân bằng. Tuy nhiên, `trim` luôn chỉ tìm ra một điểm gần với bộ giá trị ban đầu nhất. Nếu bộ giá trị ban đầu không cho trước, SIMULINK sẽ tìm điểm gần với $x0 = 0$ nhất. Nếu không tồn tại điểm cân bằng, SIMULINK sẽ xuất ra kết quả điểm có đạo hàm của **x** là tối thiểu.

Ta hãy thử tìm điểm cân bằng cho mô hình SIMULINK trong hình 7.5.

```
>> x0 = [1; 1; 1]; u0 = [3];
>> [x,y,z,dx] = trim('fig07_05',x0,u0)
x =
   -0.0000
   -1.3793
   -0.6897
y =
    0.6207
z =
   -0.6897
   -0.6897
dx =
  1.0e-016 *
    0.6939
   -0.0000
         0
```

Bằng việc khai báo thêm biến dx trong dòng lệnh, ta đã buộc MATLAB phải cho ta biết thông tin về độ dư sai số (so với 0) đạo hàm của các biến trạng thái.

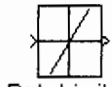
Để biết thông tin về trật tự của các trạng thái tương ứng với các khối trong sơ đồ SIMULINK ta gọi (tham số xstord):

```
>> [size,x0,xstord] = fig07_05
size =
     3
     0
     2
     1
     0
     1
     1
x0 =
     0
     0
     0
xstord =
    'Fig07_05/Object'
    'Fig07_05/Object'
    'Fig07_05/Feedback
     Transfer function'
```

Tham số xstord cho ta biết: Vị trí của ba biến (ba phần tử) thuộc vector trạng thái của mô hình trạng thái thu từ sơ đồ ***Fig07_05.mdl***.

7.4 Thư viện Nonlinear

Rate Limiter và Saturation

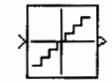


Khối *Rate Limiter* giới hạn đạo hàm bậc nhất của tín hiệu đầu vào tại các tham số *Rising slew rate* (giới hạn khi tín hiệu vào tăng) và *Falling slew rate* (giới hạn khi tín hiệu vào giảm).



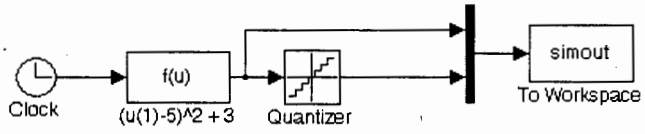
Khối *Saturation* giới hạn giá trị tối đa của tín hiệu vào về phía dương (*Upper limit*) và phía âm (*Lower limit*).

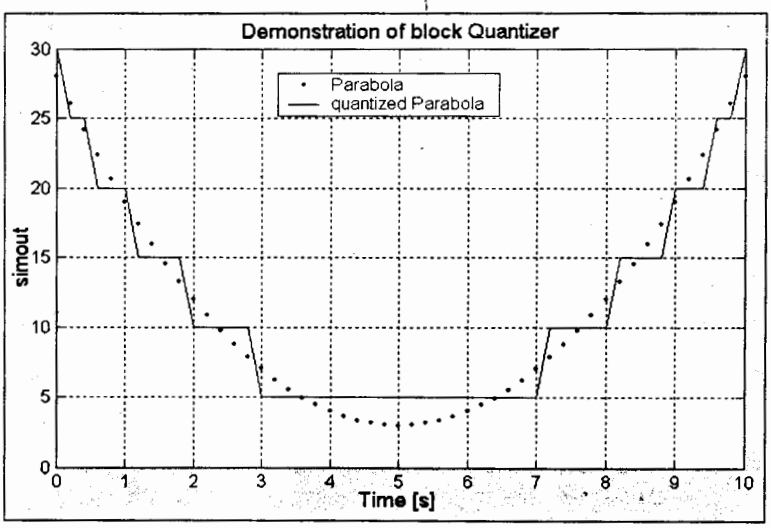
Quantizer



Khối *Quantizer* chuyển tín hiệu ở đầu vào của khối thành tín hi có dạng bậc thang. Chiều cao của bậc do tham số *Quantizatio interval* quyết định. Tại thời điểm tích phân thứ i, tín hiệu ra $y(i)$ được tính từ tham số *Quantization interval* q và tín hiệu vào $u(i)$, đồng thời sử dụng hàm làm tròn round của MATLAB theo:

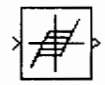
$$y(i) = q \cdot \text{round}\left(\frac{u(i)}{q}\right) \tag{7.4}$$



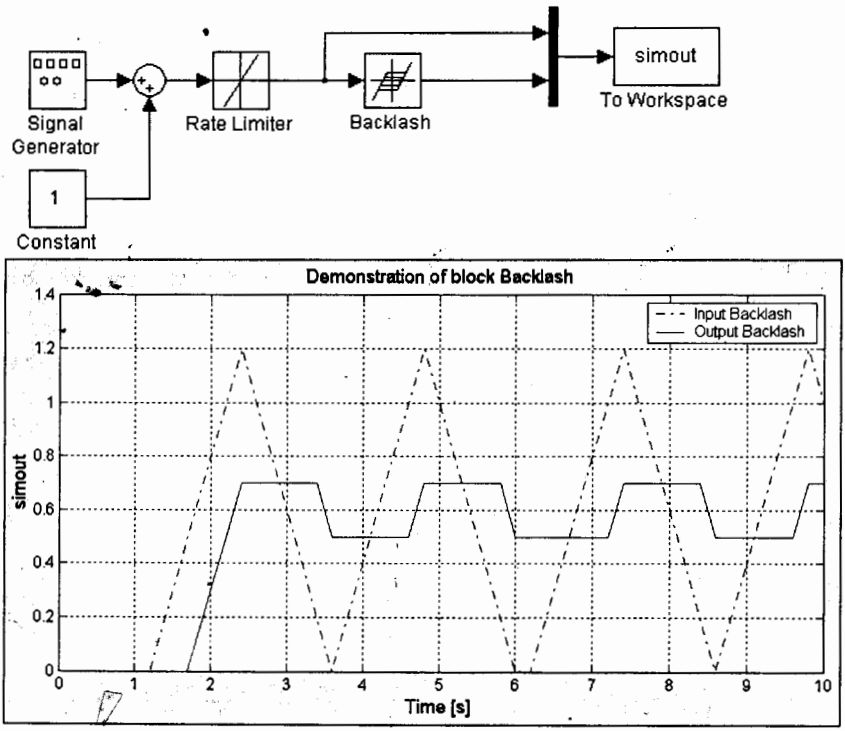


Hình 7.6 *Ví dụ về chức năng của khối Quantizer*

Backlash



Khối *Backlash* phỏng lại đặc tính của một hệ thống có độ dơ. Độ dơ thường xuất hiện trong các kết cấu cơ khí có chứa hộp số, đó chính là độ dơ lắc giữa hai bánh răng khớp nhau của hộp số. Độ dơ lắc thường xuất hiện do gia công thiếu chính xác hay vì lý dơ hao mòn vật liệu. Tham số *Deadband width* khai báo bề rộng của độ dơ (đối xứng ở đầu ra).



Hình 7.7 *Ví dụ về chức năng của khối Backlash*

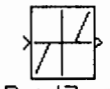
Khối *Backlash* có thể có một trong ba trạng thái hoạt động sau:

- *Kịch dơ theo chiều dương*: Tín hiệu ra tăng tỷ lệ với tín hiệu vào, $y(i) = u(i) - (1/2) \cdot$ *Deadband width*. Ví dụ hộp số: Bánh răng đang trong trạng thái hoạt động, tốc độ quay dương phía động cơ truyền động tạo nên tốc độ quay dương phía tải.
- *Cách ly*: Nếu tín hiệu vào đảo dấu, phải quét qua toàn bộ bề rộng của độ dơ cho tới khi kịch dơ theo chiều âm. Trong quá trình đó, tín hiệu ra hoàn toàn cách ly với tín hiệu vào. Ví dụ hộp số: Khi phía truyền động đảo chiều, hai bánh răng sẽ rời nhau ra và lúc ấy không thể dùng động cơ truyền động để điều khiển tốc độ quay của phía tải.

- *Kịch dơ theo chiều âm*: Tín hiệu ra giảm tỷ lệ với tín hiệu vào, $y(i) = u(i) + (1/2) \cdot$ *Deadband width*. Ví dụ hộp số: Hai bánh răng lại trong trạng thái hoạt động (lần này áp nhau tại sườn đối diện của răng), tốc độ quay âm phía động cơ truyền động gây nên tốc độ quay âm phía tải.

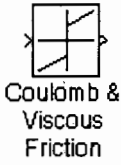
Ví dụ ở hình 7.7 minh họa hoạt động của khối *Backlash*: Bằng khối *Rate Limiter* ta biến chuỗi xung hình chữ nhật (biên độ 1, tần số 0,4Hz) của *Signal Generator* thành chuỗi xung răng cưa có giá trị độ dốc sườn lên và sườn xuống giống nhau. Chuỗi xung răng cưa là tín hiệu vào của khối *Backlash*, với *Initial output* = 0 và *Deadband width* = 1. Từ đồ thị ở hình 7.7 (hình dưới) ta có thể nhận thấy rất rõ ba trạng thái hoạt động của khối *Backlash*. Tại sườn dương của tín hiệu vào u, tín hiệu ra y chỉ tăng chậm một khoảng là 0,5 so với u (trạng thái kịch dơ sườn dương). Khi u đảo chiều, y vẫn giữ nguyên giá trị (trạng thái cách ly) cho đến khi toàn bộ bề rộng của khoảng dơ lắc đã qua, lúc đó (trạng thái kịch dơ sườn âm) y bắt đầu giảm cũng chậm một khoảng là 0,5 so với u.

Dead Zone



Bằng khối *Dead Zone* ta có thể mô phỏng các đối tượng với đặc điểm: Tín hiệu ra có giá trị 0 trong một khoảng (khoảng liệt) nhất định của tín hiệu vào. Giới hạn dưới và trên của khoảng liệt do các tham số *Start of dead zone* và *End of dead zone* quyết định. Nếu tín hiệu vào có giá trị nằm trong khoảng bị chặn bởi hai giới hạn đó, tín hiệu ra sẽ nhận giá trị 0. Khi tín hiệu vào $u \leq$ *Start of dead zone*, tín hiệu ra $y = u$ - *Start of dead zone*. Khi tín hiệu vào $u \geq$ *End of dead zone*, tín hiệu ra $y = u$ - *End of dead zone*.

Coulomb & Viscous Friction



Khối *Coulomb & Viscous Friction* mô phỏng một hệ có ma sát dính và ma sát trượt. Các dạng ma sát này tồn tại trong các hệ thống cơ có chuyển động tịnh tiến hay chuyển động quay tròn. Do tồn tại ma sát dính, khi tín hiệu vào đi qua điểm không (ví dụ: vận tốc góc của chuyển động quay tròn), tại đó sẽ xuất hiện một điểm gián đoạn. Chỉ khi mômen tạo chuyển động quay thắng được mômen cản của ma sát dính, khi ấy mới bắt đầu có chuyển động quay của tải. Nếu mômen tạo chuyển động giảm xuống bé hơn mômen cản, tải sẽ ngừng quay. Khi tốc độ quay > 0, xuất hiện thành phần mômen ma sát cản tăng tuyến tính so với tốc độ, được gọi là mômen trượt (ma sát trượt).

Giá trị của mômen dính được xác định bởi tham số *Coulomb friction value* (Offset). Độ dốc của mômen trượt do *Coefficient of viscous friction* (Gain) quyết định.

Relay (khối Hysteresis)



Tùy theo tín hiệu vào (tín hiệu điều khiển), khối *Relay* có tác dụng chuyển đổi tín hiệu ra giữa hai giá trị *Output when on* và *Output when off*. Relay sẽ đóng mạch (trạng thái 'on'), nếu tín hiệu vào $u \geq$ *Switch on point*, và tín hiệu ra y = *Output when on*. Relay giữ nguyên trạng thái, nếu tín hiệu vào $u \leq$ *Switch off point*, và tín hiệu ra y = *Output when off*.

Giá trị *Switch on point* luôn phải được chọn lớn hơn hoặc bằng giá trị *Switch off point*. Khi *Switch on point* > *Switch off point*, khối *Relay* mô phỏng đường đặc tính Hysteresis. Khi *Switch on point* = *Switch off point*, khối *Relay* mô phỏng một công tắc chuyển mạch.

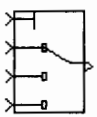
Switch,Manual Switch và Multiport Switch



Khối *Switch* có tác dụng chuyển mạch, đưa tín hiệu từ đầu vào 1 hoặc 3 tới đầu ra. Tín hiệu điều khiển chuyển mạch được đưa tới đầu vào 2 (đầu vào ở giữa). Ngưỡng giá trị điều khiển chuyển mạch được khai báo bằng tham số *Threshold*. Khi tín hiệu điều khiển ≥ *Threshold*, đầu ra được nối với đầu vào 1. Khi tín hiệu điều khiển < *Threshold*, đầu ra được nối với đầu vào 3.



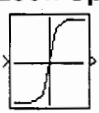
Đối với khối *Manual Switch*, việc lựa chọn đầu vào để nối với đầu ra được thực hiện bằng cách nháy kép chuột trái tại đầu vào cần nối.



Khối *Multiport Switch* có đầu vào điều khiển nằm trên cùng. Đầu vào được chọn phụ thuộc giá trị làm tròn của tín hiệu điều khiển. Ví dụ: Đầu vào điều khiển có giá trị là 3,3 (làm tròn là 3), vậy đầu vào thứ 3 (đếm từ trên xuống, không kể đầu vào điều khiển) được nối với đầu ra. Số lượng đầu vào được quyết định bởi giá trị khai báo tại *Number of inputs*.

7.5 Thư viện Function & Tables

Look-Up Table và Look-Up Table (2-D)



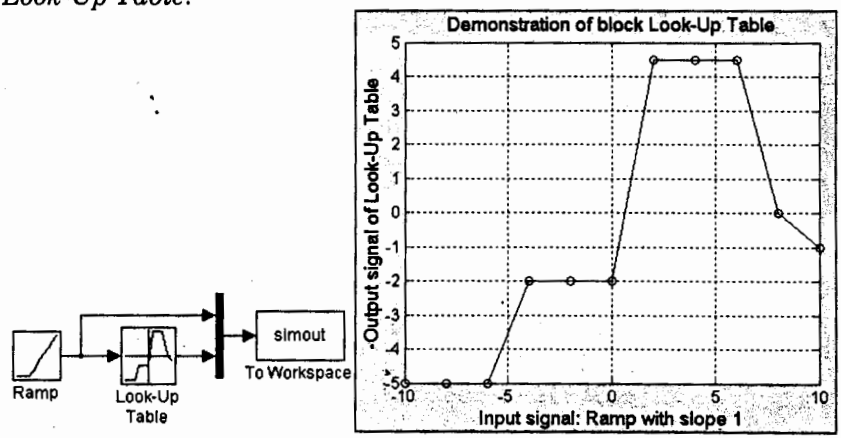
Khối *Look-Up Table* tạo tín hiệu ra từ tín hiệu vào trên cơ sở thông tin cất trong một bảng tra (*Vector of input values* × *Vector of output values*). Nếu giá trị hiện tại của tín hiệu vào trùng với một giá trị thuộc *Vector of input values*, giá trị tương ứng trong bảng thuộc *Vector of output values* sẽ được đưa tới đầu ra. Nếu giá trị của tín hiệu vào nằm giữa hai giá trị thuộc *Vector of input values*, SIMULINK thực hiện nội suy hai giá trị tương ứng của *Vector of output values*. Nếu giá trị của tín hiệu vào bé hơn / lớn hơn giá trị đầu tiên / giá trị cuối cùng của *Vector of input values*, SIMULINK sẽ thực hiện ngoại suy hai giá trị đầu tiên / cuối cùng của *Vector of output values*. *Vector of input values* có thể là một vector hàng hay một vector cột.



Trong ví dụ sau, *Look-Up Table* có chứa các giá trị:

```
% Vector of input values
>> vecin = [-10:2:10];
% Vector of output values
>> vecout = [-5 -5 -5 -2 -2 -2 4.5 4.5 4.5 0 -1];
```

Tín hiệu vào của khối *Look-Up Table* có dạng dốc tuyến tính (*Ramp*) với độ dốc 1, do đó tín hiệu sẽ quét toàn bộ dải giá trị của bảng. Tại đầu ra sẽ xuất hiện tín hiệu có dạng bậc thang cất trong *Look-Up Table*.

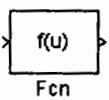


Hình 7.8 *Ví dụ về chức năng của khối Look-Up Table*

Khối *Look-Up Table (2-D)* cho phép ta tạo nên một bảng tra 2 chiều. Bằng tham số *Table* ta khai báo một ma trận cất các giá trị đầu ra. Muốn tìm được giá trị đưa tới đầu ra, ta cần biết *Row* để tìm hàng, và *Column* để tìm cột của ô giá trị trong ma trận. Nghĩa là: Bằng *Row* và *Column* ta định nghĩa các điểm trên mặt phẳng *xy*, còn *Table* là các giá trị *z* tương ứng.

Tín hiệu đặt ở đầu vào phía trên được so với *Row*, còn tín hiệu đặt ở đầu vào phía dưới được so với *Column*. Nếu tồn tại trong *Table* một ô ứng với cặp giá trị đầu vào, giá trị của ô đó được đưa tới đầu ra. Trong các trường hợp còn lại, SIMULINK sẽ tự động nội (hay ngoại) suy các giá trị của bảng để tìm giá trị đưa tới đầu ra, tương tự như đối với *Look-Up Table*.

Function



Bằng khối *Fcn* ta có thể khai báo một hàm của biến vào, dưới dạng một biểu thức viết theo phong cách của ngôn ngữ lập trình C. Nếu viết *u*, đó là tín hiệu vào scalar hay chỉ là phần tử đầu tiên của vector tín hiệu vào. Nếu viết *u*(*i*) hay *u*[*i*], đó là phần tử thứ *i* của tín hiệu vào dạng vector (tín hiệu 1-D).

Biểu thức toán được phép chứa số, các hàm toán (mục 1.1), các toán tử số học, toán tử logic hay toán tử so sánh (mục 1.5), các dấu ngoặc vuông, ngoặc đơn, và cả các biến đã định nghĩa trong môi trường MATLAB *Workspace*. Trình tự ưu tiên của các toán tử trong khi tính được SIMULINK tuân thủ theo các nguyên tắc của ngôn ngữ lập trình C. Các phép tính ma trận không được *Fcn* hỗ trợ.

Ví dụ: Biểu thức

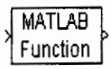
```
(u[1]/m - g*sin(u[3])*cos(u[3]) + l* ...
power(u[2],2)*sin(u[3]))/(M/m + power(sin(u[3]),2))
```

trong khối *Fcn* thực hiện hàm toán dưới đây:

$$y = \frac{\frac{u[1]}{m} - g \cdot \sin(u[3]) \cdot \cos(u[3]) + l \cdot u[2]^2 \cdot \sin(u[3])}{\frac{M}{m} + \sin^2(u[3])} \tag{7.5}$$

với y là biến ra scalar, còn `m`, `M`, `l` và `g` là các biến được định nghĩa trong MATLAB *Workspace*.

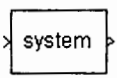
Matlab Function



Khối *MATLAB Fcn* là một dạng mở rộng của khối *Fcn*. Tại ô *MATLAB function* ta có thể khai báo một biểu thức toán hay một hàm MATLAB (viết dưới dạng *m-File*: mục 1.7.1) của biến đầu vào. Điều quan trọng cần chú ý: Bề rộng của tín hiệu do hàm MATLAB xuất ra phải tương ứng với *Output width* của khối *MATLAB Fcn*. Giá trị mặc định –1 sẽ gán cho đầu ra của khối bề rộng của đầu vào. Tại ô *Output signal type* ta khai loại (*auto*, *real* hay *complex*) cần có của tín hiệu ra.

Việc tính toán khối *MATLAB Fcn* tốn nhiều thời gian hơn khối *Fcn* rất nhiều, bởi vì cứ mỗi bước tích phân SIMULINK lại phải gọi MATLAB *parser* để phân tích cú pháp (*Syntax*) của *m-File*. Vì vậy, để tiết kiệm thời gian tính, đối với các hàm đơn giản nên sử dụng khối *Fcn* hay *Math Function*. Các hàm phức tạp nên được viết dưới dạng hàm S (*S-Function*).

S-Function



Khối *S-Function* tạo điều kiện để ta ghép hàm S (*S-Function*), hoặc viết dưới dạng MATLAB *Script* (*m-File*), hoặc dưới dạng MATLAB *Executable File* (*C mex file*, ngôn ngữ C), vào một sơ đồ khối SIMULINK. Tên của *S-Function* được khai báo tại ô *S-function name*. Ngoài ra, ta có thể chuyển cho *S-Function* thêm một vài tham số (viết cách nhau bởi dấu phẩy) tại ô *S-function parameters*, ví dụ: Các biểu thức MATLAB hay các biến. Nếu hàm *S-Function* có nhiều biến vào, *S-Function* sẽ lần lượt gọi từ phần tử đầu tiên của tín hiệu vào (dạng 1-D) và đánh số thứ tự 1 vv... Ta sẽ quay lại vấn đề *S-Function* ở mục 7.7.

7.6 Vòng quẩn đại số

Vòng quẩn đại số có thể xuất hiện trong các mô hình có sử dụng khối với đặc điểm *direct feedthrough*[1]. Trong những khối này, tín hiệu đầu ra của khối phụ thuộc trực tiếp đầu vào tại cùng một thời điểm. Khối có đặc điểm *direct feedthrough* ví dụ là: *Sum*, *Gain*, *Product*, *State Space* có ma trận liên thông **D≠0**, *Integrator* (liên quan tới giá trị ban đầu; vòng lặp – xuất hiện do hồi tiếp tín hiệu ra để lập trị ban đầu – không được phép là vòng quẩn đại số), cũng như *Transfer Function* và *Zero-Pole* (nếu bậc của tử và mẫu số giống nhau).

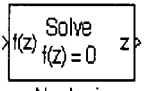
Nếu đầu ra của khối *direct feedthrough* được hồi tiếp về đầu vào của chính khối đó nhờ một khối *direct feedthrough* khác, tín hiệu đầu vào hiện tại của khối sẽ phụ thuộc trực tiếp vào đầu ra của chính nó tại cùng một thời điểm, mà đầu ra vốn lại phụ thuộc đầu vào cũng tại thời điểm ấy. Một vòng quẩn đại số đã xuất hiện.

Đối với khối *Integrator*, vòng quẩn đại số sẽ xuất hiện nếu tín hiệu ra của khối được dùng trực tiếp (hay thông qua một khối *direct feedthrough*) để điều khiển giá trị ban đầu từ bên ngoài *Integrator*. Vấn đề này đã được giải quyết thông qua các ví dụ ở hình 7.1 – 7.4: Giải pháp tốt nhất là sử dụng biến trạng thái lấy từ đầu ra *state port* của khối *Integrator* để điều khiển lập trị ban đầu. Cửa *state port* là một đầu ra phụ, được SIMULINK tự động tạo thêm khi ta kích hoạt ô *Show state port*.

Nếu một mô hình có chứa vòng quẩn đại số, cứ mỗi bước tích phân SIMULINK lại phải gọi một chương trình con có chức năng giải bài toán đó theo phương thức *iterative*[2] (phương pháp tích phân lặp Newton). Để phép tính hội tụ, nhiều khi ta phải khai báo một giá trị ban đầu cho phép tích phân lặp. Có thể làm điều đó bằng cách sử dụng khối *Initial Condition* (*IC*, thư viện con *Signals & Systems*).

Một khả năng khác tương đối tiện lợi để giải quyết vấn đề vòng quẩn đại số là sử dụng khối *Algebraic Constraint*. Giá trị khởi đầu của phép tích phân lặp sẽ do tham số *Initial guess* quyết định.

Algebraic Constraint



Khối *Algebraic Constraint* cưỡng tín hiệu vào của khối về 0 và xuất ở đầu ra của khối giá trị của biến *z* (là giá trị ứng với khi đầu vào là 0). Tuy nhiên, biến ra phải có tác động ngược trở lại đầu vào thông qua một vòng hồi tiếp thích hợp.

Tại ô giành cho tham số *Initial guess* ta có thể khai báo một giá trị khởi đầu cho thuật toán giải vòng quẩn đại số. Bằng cách lựa chọn khéo léo *Initial guess* ta có thể cải thiện độ chính xác, thậm chí là cách làm duy nhất trong một vài trường hợp gay cấn (*critical case*).

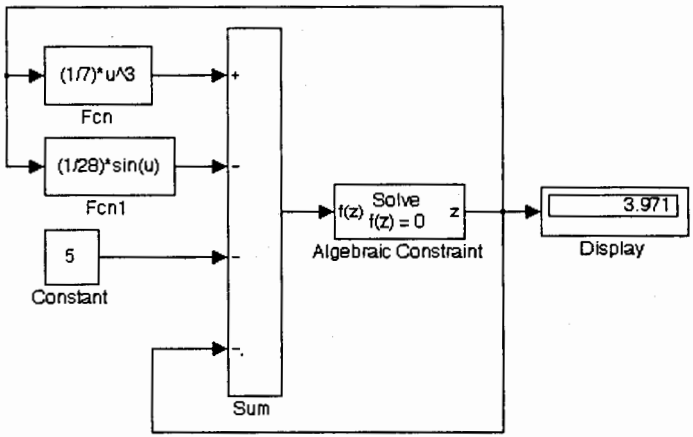
---

[1] Tạm dịch: Nối thông trực tiếp

[2] Tích phân theo bước lặp

Ví dụ dưới đây sẽ minh họa rõ hơn chức năng của khối *Algebraic Constraint*. Ta cần tìm nghiệm của phương trình sau:

$$f(z)=\frac{1}{7}z^3-\frac{1}{28}\sin z-z-5=0 \qquad (7.6)$$



Hình 7.9 *Ví dụ về chức năng của khối Algebraic Constraint*

Như hình 7.9 cho thấy, sau khi khối *Sum* tính tổng của các số hạng (tạo từ các khối *Fcn* và *Constant*), khối *Algebraic Constraint* sẽ cưỡng giá trị ở đầu vào của khối về 0 và xuất ở đầu ra giá trị z (ứng với $f(z) = 0$, hiển thị tại khối *Display*). Đối với ví dụ trên, thuật toán giải đã chỉ có thể tìm được nghiệm $z = 3{,}971$ nếu ta chọn giá trị khởi đầu *Initial guess* ≥ 2. Trên cửa sổ Command của MATLAB sẽ xuất hiện lời cảnh báo: MATLAB đã phát hiện ra vòng quẩn đại số.

```
Warning: Block diagram 'Fig07_09' contains 1
algebraic loop(s).
Found algebraic loop containing block(s):
  'Fig07_09/Algebraic Constraint/Initial Guess'
  'Fig07_09/Fcn'
  'Fig07_09/Fcn1'
  'Fig07_09/Sum'
  'Fig07_09/Algebraic Constraint/Sum' (algebraic
variable)
```

7.7 Hàm S (S-Functions)

Hàm S là các hàm có thể kích hoạt trong sơ đồ SIMULINK nhờ sử dụng khối *S-Function* (thư viện con *Functions & Tables*). Hàm S có thể có dạng:

- MATLAB *Scripts*, tức là *m-File*,
- File chạy (MATLAB *Executable Files*) viết bằng ngôn ngữ C, còn được gọi là C *mex-Files*, hoặc
- File chạy viết bằng ngôn ngữ bậc cao C++, Fortran hay Ada, cũng được gọi là C *mex-Files*.

Việc lập trình trên MATLAB về nguyên tắc dễ học hơn, vì số lượng các thủ tục (chương trình con) cần gọi là ít. Tuy nhiên, thuật toán của hàm S càng phức tạp ta lại càng nên sử dụng một ngôn ngữ bậc cao (ví dụ C) để lập trình. Các hàm S dạng *mex-File* do người sử dụng dịch (*compile*) sẵn từ trước thành dạng File chạy và có thể gọi khi cần. Ngược lại, hàm S dạng *m-File* sẽ buộc SIMULINK quét và phân tích từng dòng mỗi khi gọi nên khá chậm.

Điều quan trọng nhất khi lập trình hàm S là: Tuân thủ nghiêm ngặt các "*điều kiện khung*" (gọi chương trình con, dựng cờ *Flags* vv...), giúp cho SIMULINK có thể đối thoại một cách chuẩn xác với hàm S trong quá trình mô phỏng. Ngoài ra, người lập trình hoàn toàn tự do với sức sáng tạo của mình. Nhờ khả năng lập trình hàm S, ta có thể mở rộng thư viện chuẩn có sẵn của SIMULINK thêm các hàm bất kỳ.

Nếu so sánh với việc xây dựng sơ đồ khối SIMULINK, việc lập trình hàm S đương nhiên là phức tạp hơn nhiều, do ta phải bảo đảm chính xác các điều kiện khung. Tại mục này bạn đọc sẽ chỉ trên cơ sở một ví dụ đơn giản (hàm S dạng *m-File*) làm quen với lập trình hàm S. Trong phần nội dung ứng dụng của sách, ta sẽ có các ví dụ lập trình bằng C. Bạn đọc nào quan tâm buộc sẽ phải đọc kỹ tài liệu gốc trong thư mục *Help/PDF_DOC* của MATLAB, tài liệu *Writing S-Functions* (File *sfunctions.pdf*).

Giống như mọi khối chuẩn của SIMULINK, hàm S cũng phân chia các tín hiệu thành vector biến vào **u**, vector biến trạng thái **x** và vector biến ra **y**. Trong hàm S, các biến trạng thái liên tục và gián đoạn được xử lý tách biệt, nghĩa là, SIMULINK sẽ gọi các chương trình con khác nhau để tính $\dot{\mathbf{x}}$ và $\mathbf{x}_{k+1}$.

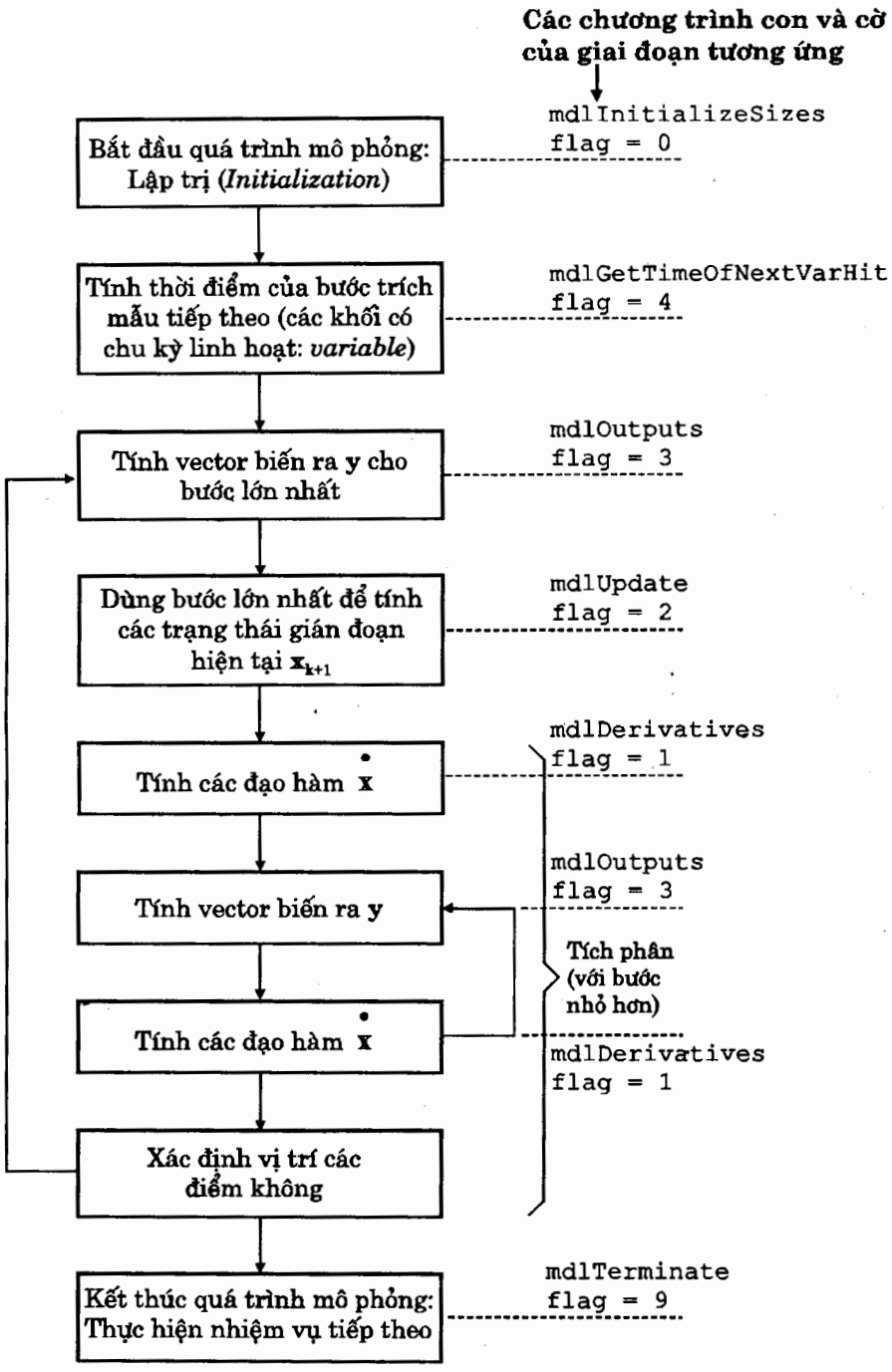
Hình 7.10 minh họa các bước của một quá trình SIMULINK tính mô phỏng và các chương trình con (được hàm S gọi) tương ứng của từng bước. Trong hàm S ta còn cần đến tham số *flag*[1]. Tham số *flag* có nhiệm vụ trao đổi thông tin với chương trình con, được SIMULINK chuyển giao cho hàm S, qua đó báo cho hàm S biết: Quá trình mô phỏng đang ở giai đoạn nào.

Khi hàm S được viết dưới dạng *mex-File*, các chương trình con được SIMULINK trực tiếp gọi, và do đó không cần đến *flag*. Ngược với hàm S dạng *m-File*, hàm S dạng *mex-File* còn có nhiều chương trình con khác, ngoài các chương trình con đã thể hiện ở hình 7.10. Khi gọi:

```
mex name_of_C_mex_SFunction.c
```

MATLAB sẽ thông dịch (*compile*) hàm S có tên*name_of_C_mex_SFunction.c*.

[1] Được gọi là cờ

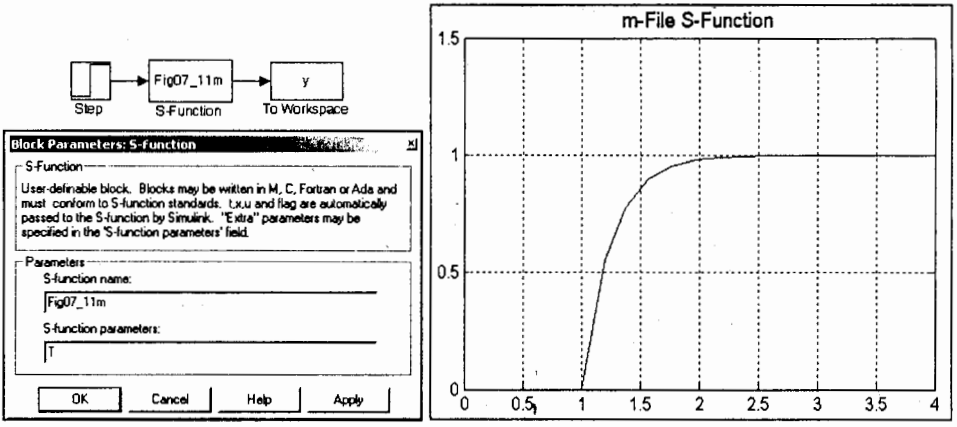


Hình 7.10 Các bước mô phỏng và các chương trình con tương ứng (do hàm S gọi)

Để hỗ trợ người sử dụng khi viết chương trình tạo hàm S, MATLAB đã chuẩn bị sẵn các *Files* mẫu *sfuntmpl.m* và *sfuntmpl_basic.c* / *sfuntmpl_doc.c*, cất tại hai thư mục *$MATLABPATH / toolbox / simulink / blocks* và *$MATLABPATH / simulink / src*. Các Files mẫu đó đã chứa rất đầy đủ các "điều kiện khung" mà ta cần tuân thủ.

Ví dụ: Khâu tỷ lệ có quán tính bậc 1 (khâu PT_1)

Ví dụ ở hình 7.11 dưới đây minh họa cách sử dụng hàm S dạng *m-File*. Hàm S có tên *Fig07_11m.m*, được khai báo tạo ô *Block Parameters* của khối *S-Function* thuộc sơ đồ SIMULINK *Fig07_11.mdl*. Hàm S mô phỏng khâu tỷ lệ có quán tính bậc nhất PT_1 với hằng số thời gian *T*. Hình 7.11 (bên phải) mô tả đáp ứng bước nhảy của khâu PT_1.



Hình 7.11 *S-Function viết dưới dạng m-File: Khâu PT_1*

Hàm *Fig07_11m.m* có mã nguồn dưới đây, được viết theo "điều kiện khung" quy định ở hình 7.7:

```
% Source code of the M-File S-Function Fig07_11m.m
%
function [sys,x0,str,ts] = Fig07_11(t,x,u,flag,T)
%
% FIG07_11m mô phỏng đặc tính khâu PT1. Hằng số thời gian T
% được coi là tham số lấy từ khối S-Function và xuất hiện
% tại cửa sổ Block Parameters của khối.

% Định nghĩa mô hình trạng thái:
A = [-1/T];     % Ma trận hệ thống (tại đây: scalar)
B = [ 1/T];     % Ma trận đầu vào (tại đây: scalar)
C = [ 1  ];     % Ma trận đầu ra (tại đây: scalar)
D = [ 0  ];     % Ma trận liên thông (tại đây: scalar và =0)
```

```
switch flag,
  case 0,                  % Initialization: Lập trị ban đầu
    [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D);
  case 1,                  % Derivatives: Tính đạo hàm
    sys=mdlDerivatives(t,x,u,A,B,C,D);
  case 3,                  % Outputs: Tính biến ra
    sys=mdlOutputs(t,x,u,A,B,C,D);
  case {2, 4, 9},          % Các cờ không sử dụng đến
    sys=[];
  otherwise                % Các trường hợp lỗi
    error(['Unhandled flag = ',num2str(flag)]);
end                        % end fig07_11
%
%=============================================================
% Chương trình con: mdlInitializeSizes
% Return the sizes, initial conditions, and sample times
% for the S-function.
%=============================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D)
%
sizes = simsizes;
sizes.NumContStates = size(A,1);
                           % Kích cỡ A quyết định số
                           % lượng biến trạng thái
sizes.NumDiscStates = 0;   % Không có trạng thái discrete
sizes.NumOutputs = size(C,1);
                           % Kích cỡ C quyết định số
                           % lượng biến ra
sizes.NumInputs      = size(B,1);
                           % Kích cỡ B quyết định số
                           % lượng biến vào
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;  % at least one sample
                           % time is needed
sys = simsizes(sizes);

% Khai báo điều kiện ban đầu
x0  = zeros(size(A,1),1);  % Trạng thái ban đầu: x0 = 0
str = [];                  % str luôn là ma trận rỗng

% Khai báo dãy chu kỳ trích mẫu (initialize the array of
% sample times). Trong ví dụ này ta có mô hình liên tục,
% vậy ts và Offset được đặt về 0.
ts  = [0 0];
% end mdlInitializeSizes
```

```
%=============================================================
% Chương trình con: mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================
%
function sys=mdlDerivatives(t,x,u,A,B,C,D)
%
sys = A*x + B*u;              % Tính các đạo hàm xpoint
% end mdlDerivatives

%=============================================================
% Chương trình con: mdlOutputs
% Return the block outputs.
%=============================================================
%
function sys=mdlOutputs(t,x,u,A,B,C,D)
%
sys = C*x + D*u;              % Tính vector biến ra y
% end mdlOutputs
```

Ta dễ dàng nhận thấy: Vì khâu PT1 là một đối tượng liên tục, ta chỉ cần sử dụng các chương trình con `mdlInitializeSizes`, `mdlDerivatives` và `mdlOutputs`, còn `mdlUpdate` và `mdlGetTimeOfNextVarHit` đã không được dùng đến. Ngoài ra, vì không xuất hiện đòi hỏi tính toán mới (sau khi đã kết thúc một vòng tính) ta cũng không cần hàm con `mdlTerminate`.

Hàm con `mdlInitializeSizes` có tác dụng xác định những đặc điểm cơ bản của khối *S-Function*: Chu kỳ trích mẫu, giá trị ban đầu của các biến trạng thái, cũng như kích cỡ (*sizes*) của *array*. SIMULINK cần các thông tin đó để nhận biết *S-Function*. Lệnh `sizes = simsizes` có tác dụng gán cho biến *sizes* một cấu trúc gồm các mảng như sau:

`NumContStates`	Số lượng biến trạng thái liên tục
`NumDiscStates`	Số lượng biến trạng thái gián đoạn
`NumOutputs`	Số lượng biến ra
`NumInputs`	Số lượng biến vào
`DirFeedthrough`	Cờ giành cho *direct feedthrough*
`NumSampleTimes`	Số lượng chu kỳ trích mẫu

Các mảng trên có giá trị mặc định là 0, người sử dụng sẽ phải tự mình điền các giá trị thích hợp. Khi gọi hàm `simsizes` lần 2, `sys = simsizes(sizes);` các thông tin chứa trong cấu trúc *sizes* được chuyển sang cho biến *sys* dưới dạng *Array*, và sẽ được SIMULINK hỏi đến trong giai đoạn lập trị (khi `flag = 0`).

Các hàm con `mdlDerivatives` và `mdlOutputs` có nhiệm vụ tính đạo hàm của vector trạng thái và vector biến ra. Cả hai vector đó đều là scalar trong ví dụ trên.

Dưới đây, bạn đọc có cả phần mã nguồn của *S-Function* dưới dạng *C-mex-File* mang tên *Fig07_12c.c*, mô phỏng khâu PT_1 trong sơ đồ *Fig07_12.mdl*.

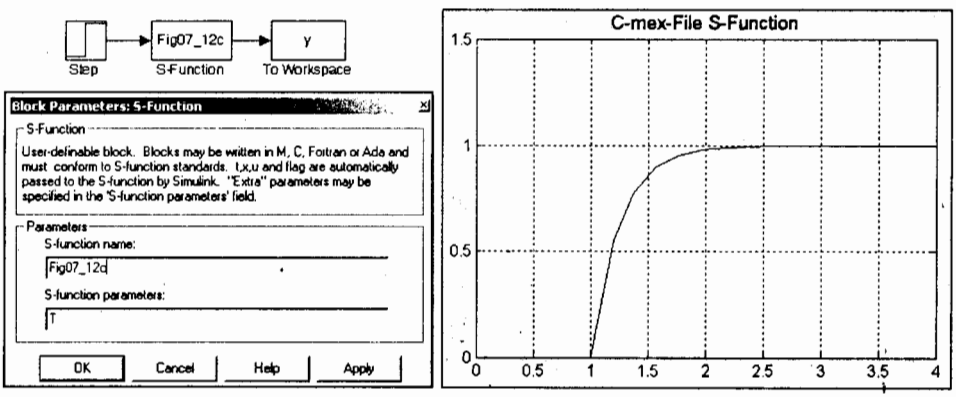
Trước khi mô phỏng ta gọi:

```
>> mex Fig07_12c.c
```

Kết quả thông dịch thu được là *File* có tên *Fig07_12c.dll*, được khai báo sử dụng tại cửa sổ *Block Parameters* của khối *S-Function* (hình 7.12). Tại đây cần lưu ý bạn đọc: Để dịch `Fig07_12c.c` như trên, ta có thể sử dụng các *C-Compiler* khác nhau (điều kiện: đã cài đặt chúng trên PC). Bản thân MATLAB cũng có *C-Compiler* riêng. Việc lựa chọn được thực hiện nhờ lệnh:

```
>>mex -setup
```

Sau khi gọi, MATLAB sẽ liệt kê các *C-Compiler* đã cài để ta lựa chọn.



***Hình 7.12** S-Function viết dưới dạng C-mex-File: Khâu PT_1*

Mã nguồn của C-mex-File có tên Fig07_12c.dll là như sau:

```
/* Source code of C MEX S-Function Fig07_12.c */
/* Used by SIMULINK-File Fig07_12.mdl in the S-Function
   block and simulating the dynamical behaviour of one PT1
   object. */

#define S_FUNCTION_NAME  Fig07_12c
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element])     /* Pointer to
                                            Input Port0 */
#define TIMECONSTANT_IDX 0
#define TIMECONSTANT_PARAM(S)
        ssGetSFcnParam(S,TIMECONSTANT_IDX)
#define NPARAMS 1
```

```
/* S-function methods */
#define MDL_CHECK_PARAMETERS
#if defined(MDL_CHECK_PARAMETERS) &&
    defined(MATLAB_MEX_FILE)
  /* Function: mdlCheckParameters
   * Abstract:
   * Validate our parameters to verify they are okay */
  static void mdlCheckParameters(SimStruct *S)
  { /* Check 1st parameter: time constant */
    { if (mxGetNumberOfElements(TIMECONSTANT_PARAM(S)) > 1)
      {
        ssSetErrorStatus(S,"The parameter TIMECONSTANT"
                           "must be typed in and scalar");
        return;
      }
    }
  }
#endif /* MDL_CHECK_PARAMETERS */

/* Function: mdlInitializeSizes
 * Abstract:
 * The sizes information is used by Simulink to determine
 * the S-function block's characteristics (number of
 * inputs, outputs, states, etc.) */

static void mdlInitializeSizes(SimStruct *S)
{ ssSetNumSFcnParams(S, NPARAMS);  /* Number of expected
                                      parameters */
  #if defined(MATLAB_MEX_FILE)
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S))
         { mdlCheckParameters(S);
           if (ssGetErrorStatus(S) != NULL)
           { return; } }
    else { return; /* Parameter mismatch will be reported
                      by Simulink */ }
  #endif

    ssSetNumContStates(S, 1);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 1);
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
```

```
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOptions(S, 0);
}

/* Function: mdlInitializeSampleTimes
 * Abstract:
 * S-function is comprised of only continuous sample time
 * elements */
static void mdlInitializeSampleTimes(SimStruct *S)
{  ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
   ssSetOffsetTime(S, 0, 0.0); }

#define MDL_INITIALIZE_CONDITIONS   /* Change to #undef to
                                       remove function */
/* Function: mdlInitializeConditions
 * Abstract: Initialize continuous states to zero */
static void mdlInitializeConditions(SimStruct *S)
{  int_T i;
   real_T *x0 = ssGetContStates(S);
   int_T contstates = ssGetNumContStates(S);
   for(i = 0; i < contstates; i++){
   *x0++ = 0.0; } }

/* Function: mdlOutputs
 * Abstract:
 * In this function, you compute the outputs of your
 * S-function block. Generally outputs are placed in the
 * output vector, ssGetY(S). */
static void mdlOutputs(SimStruct *S, int_T tid)
{  real_T*y = ssGetOutputPortRealSignal(S,0);
   real_T*x = ssGetContStates(S);
   InputRealPtrsType uPtrs =
                        ssGetInputPortRealSignalPtrs(S,0);
   y[0] = x[0];
}

#define MDL_DERIVATIVES     /* Change to #undef to remove
                               function */
/* Function: mdlDerivatives
 * Abstract: Calculate state-space derivatives */
static void mdlDerivatives(SimStruct*S)
{  real_T *dx = ssGetdX(S);
```

```
    real_T * x  = ssGetContStates(S);
    InputRealPtrsType uPtrs =
                          ssGetInputPortRealSignalPtrs(S,0);
    real_T * tcpr = mxGetPr(TIMECONSTANT_PARAM(S));
    dx[0] = (-1/tcpr[0]) * x[0] + (1/tcpr[0]) * U(0);
}


/* Function: mdlTerminate
 * Abstract: No termination needed, but we are required to
 * have this routine. */
static void mdlTerminate(SimStruct *S)
{
}


/* Required S-function trailer */
#ifdef  MATLAB_MEX_FILE  /* Is this file being compiled
                            as a MEX-file? */
#include "simulink.c"    /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"     /* Code generation registration
                            function */
#endif
```

Trong phần giới thiệu các ứng dụng mô phỏng, ta sẽ còn quay lại vấn đề *S-Function* dưới dạng *C-mex-File*. Tại đây cần lưu ý rằng: Lợi thế lớn nhất của *S-Function* viết bằng ngôn ngữ C là khả năng truy cập vào phần cứng, điều không thể thực hiện được với *m-File S-Function*. Thêm vào đó, *S-Function* viết bằng *C-mex-File* sẽ tăng tốc độ tính toán lên rất nhiều, cho phép ta rút ngắn thời gian khi mô phỏng các hệ thống lớn - phức hợp.

7.8 Tóm tắt nội dung chương 7

Sau khi đã nghiên cứu kỹ chương 7, người đọc cần nắm vững các nội dung sau đây:

1. Sử dụng các khối của thư viện con *Continous*, *Nonlinear* và *Functions & Tables*.
2. Kiến thức cơ sở khi tuyến tính hóa mô hình SIMULINK và việc xác định điểm cân bằng.
3. Xử lý vòng quẩn đại số.
4. Cơ sở lập trình hàm S (*S-Functions*).

8 Các hệ thống trích mẫu (hệ gián đoạn)

Trong điều khiển tự động ta không chỉ quan tâm tới các hệ liên tục (trên miền ảnh Laplace), ta còn quan tâm tới cả các hệ gián đoạn về thời gian (trên miền ảnh z). Đó là các hệ thống hoạt động theo nguyên tắc trích mẫu tại các thời điểm gián đoạn cách đều, và trước hết, đó là các hệ được điều khiển bằng vi xử lý, bằng máy tính. SIMULINK hỗ trợ nghiên cứu, khảo sát các hệ thống như vậy bằng một thư viện riêng: Thư viện *Discrete*.

8.1 Các khái niệm tổng quan

Thời gian (chu kỳ) trích mẫu

Đối với hệ gián đoạn ta phải khai báo cho mỗi khối một chu kỳ trích mẫu (tính bằng giây) của khối đó. Tất cả các khối gián đoạn (về thời gian) đều được trang bị ở đầu vào một khâu trích mẫu và ở đầu ra một khâu giữ chậm bậc 0. Nghĩa là: Tín hiệu vào được trích mẫu tại mọi thời điểm $t = kT$, tín hiệu ra là hằng giữa hai thời điểm trích mẫu kế tiếp nhau. Bằng cách đó, SIMULINK có thể kết hợp nhiều khối với chu kỳ trích mẫu khác nhau, hay thậm chí lai tạp hệ thống giữa nhiều khối liên tục và gián đoạn (trừ khối co bước tích phân cố định: mục 8.4).

Khi chọn chu kỳ trích mẫu cần chú ý:

- **Offset** (độ chênh lệch): Nếu khai báo chu kỳ trích mẫu (tham số *sample time*) là một vector với chiều dài là 2, ví dụ: [T *offset*], số đầu tiên được coi là chu kỳ trích mẫu còn số thứ 2 được coi là độ chênh lệch. Nghĩa là: việc trích mẫu được thực hiện tại các thời điểm cách đều $t = offset + kT$.
- **Kế thừa**: Chu kỳ trích mẫu (và độ chênh lệch) đều có thể chuyển giao mang tính kế thừa cho các khối khác bằng cách: Tại các khối đó ta nhập giá trị –1 cho chu kỳ trích mẫu.

Biểu diễn bằng đồ thị

Để vẽ đồ thị các đường đặc tính gián đoạn về thời gian, ta sử dụng lệnh:

```
stairs(x_value,y_value[,plotstyle])
```

tương tự như lệnh `plot`. Đồ thị thu được sẽ có dạng bậc thang.

Đánh dấu bằng mầu

Tham số tùy chọn *Format / Sample time color* tạo điều kiện cho người sử dụng vẽ có phân biệt mầu sắc giữa các khối trong một sơ đồ SIMULINK lai tạp.

Ta có thể phân biệt giữa những khối có chu kỳ trích mẫu khác nhau, giữa khối liên tục và khối gián đoạn, hay giữa các hệ con (*Subsystems*). Có thể kể ra một số quy ước mầu có sẵn: khối liên tục có mầu đen, hệ con (và cả những khối xử lý tín hiệu với chu kỳ trích mẫu khác nhau như *Mux* và *Demux*) có mầu vàng, khối với chu kỳ trích mẫu bé nhất có mầu đỏ, khối với chu kỳ trích mẫu bé thứ hai có mầu xanh lá cây vv...

8.2 Tham số mô phỏng

Nếu trong sơ đồ SIMULINK chỉ gồm thuần túy các khối gián đoạn và không hề có khối liên tục, ta nên qua menue *Simulation / Parameters* để chọn thuật toán tích phân là *discrete* (không có trạng thái liên tục, *no continuous states*). Các tham số *Solver Options* còn lại sẽ không có hiệu lực.

Tuy nhiên, khi chọn *Solver* để mô phỏng một cấu trúc SIMULINK gián đoạn cần phải rất chú ý: Liệu có đúng đó là một hệ thuần túy gián đoạn, hay đó thực ra là một hệ lai (*hybrid systems*) giữa gián đoạn và liên tục. Ngoài ra còn cần phải chú ý, các khối gián đoạn sử dụng 1 hay nhiều chu kỳ trích mẫu khác nhau (mục 8.4). Một số nguyên tắc dưới đây nhằm giúp ta lựa chọn *Solver* một cách đúng đắn.

- **Hệ gián đoạn thuần túy**
  - **Các khối gián đoạn có chu kỳ trích mẫu giống nhau:**

    Mô phỏng với bước cố định: Có thể sử dụng *discrete* (*no continuous states*) hoặc các *Solver* khác có bước cố định (*Fixed step size*). Nếu chọn *auto* và *Fixed step size*, bước tích phân sẽ tự động được chọn thích hợp với chu kỳ trích mẫu. Ngoài ra, ta sẽ phải chọn *Fixed step size*, sao cho chu kỳ trích mẫu gấp đúng một số nguyên lần của *Fixed step size*. Ví dụ: Khi *sample time* có giá trị $T = 0.7$, *Fixed step size* có thể được chọn là 0,1 hay 0,05. Khi không chú ý nguyên tắc này sẽ có cảnh báo của MATLAB.

    Mô phỏng với bước linh hoạt: Có thể sử dụng *discrete* (*no continuous states*) hoặc các *Solver* khác có bước linh hoạt (*Variable step*). Nếu chọn *auto* và *Max step size*, bước tích phân sẽ tự động được chọn cố định thích hợp với chu kỳ trích mẫu. Nếu *Max step size* có giá trị lớn hơn chu kỳ trích mẫu, giá trị tối đa của bước sẽ tự động được hạ xuống đúng bằng chu kỳ trích mẫu, đồng thời xuất hiện cảnh báo của MATLAB.
  - **Các khối gián đoạn có chu kỳ trích mẫu khác nhau:**

    Mô phỏng với bước cố định: Có thể sử dụng *discrete* (*no continuous states*) hoặc các *Solver* khác có bước cố định (*Fixed step size*). Tham số *Mode* phải được đặt về *Single Tasking* khi các khối (có chu kỳ trích mẫu không giống nhau) không nối với nhau qua *Zero-Order Hold* hay

Unit Delay (mục 8.4.1). Nếu chọn *auto* và *Fixed step size*, bước tích phân sẽ nhận giá trị ứng với ước số chung lớn nhất của tất cả các chu kỳ trích mẫu (*fundamental sample time*). Ví dụ: Một mô hình có hai chu kỳ trích mẫu là 0,75 và 0,5 sẽ được mô phỏng với bước tích phân là 0,25. Nếu tự chọn *Fixed step size* (không dùng *auto*) ta cũng phải chú ý chọn ước số chung lớn nhất như vậy.

Mô phỏng với bước linh hoạt: Có thể sử dụng *discrete* (*no continuous states*) hoặc các *Solver* khác có bước linh hoạt (*Variable step*). Khi chọn *Solver* như vậy, không phụ thuộc vào việc *Max step size* có giá trị *auto* hay một giá trị nào khác, bước tích phân sẽ được chọn sao cho tất cả các khối được tính phù hợp với chu kỳ trích mẫu riêng của nó. Nếu *Max step size* có giá trị lớn hơn chu kỳ trích mẫu, giá trị tối đa của bước sẽ tự động được hạ xuống đúng bằng chu kỳ trích mẫu, đồng thời ta nhận được cảnh báo của MATLAB.

- **Hệ lai**:

Chọn *Solver* loại *discrete* (*no continuous states*) cho bước cố định hay bước linh hoạt đều không thể sử dụng được nữa, vì hệ thống có cả khối liên tục lẫn khối gián đoạn. Khi ấy nên sử dụng *Solver* loại *ode23* hay *ode45* (phương pháp Runge-Kutta bậc linh hoạt). Do tồn tại các đặc tính không liên tục (ví dụ: có thể xuất hiện trong các khối gián đoạn do trích mẫu và giữ chậm), không nên sử dụng *Solver* loại *ode15s* và *ode113*.

- **Các khối gián đoạn có chu kỳ trích mẫu giống nhau:**

Mô phỏng với bước cố định: Khi chọn *auto* và *Fixed step size*, bước tích phân sẽ tự động được chọn thích hợp với chu kỳ trích mẫu. Ngoài ra, ta sẽ phải chọn *Fixed step size*, sao cho chu kỳ trích mẫu gấp đúng một số nguyên lần của *Fixed step size*. Khi không chú ý nguyên tắc này sẽ có cảnh báo của MATLAB.

Mô phỏng với bước linh hoạt: Nếu *Max step size* có giá trị lớn hơn chu kỳ trích mẫu, giá trị tối đa của bước sẽ tự động được hạ xuống đúng bằng chu kỳ trích mẫu, đồng thời xuất hiện cảnh báo của MATLAB.

- **Các khối gián đoạn có chu kỳ trích mẫu khác nhau:**

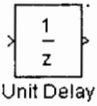
Mô phỏng với bước cố định: Tham số *Mode* phải được đặt về *Single Tasking* khi các khối (có chu kỳ trích mẫu không giống nhau) không nối với nhau qua *Zero-Order Hold* hay *Unit Delay* (mục 8.4.1). Nếu chọn *auto* và *Fixed step size*, bước tích phân sẽ nhận giá trị ứng với ước số chung lớn nhất của tất cả các chu kỳ trích mẫu (*fundamental sample time*). Nếu tự chọn (không dùng *auto*) ta cũng phải chú ý chọn ước số chung lớn nhất như vậy.

Mô phỏng với bước linh hoạt: Nếu *Max step size* có giá trị lớn hơn chu kỳ trích mẫu, giá trị tối đa của bước sẽ tự động được hạ xuống đúng bằng chu kỳ trích mẫu, đồng thời xuất hiện cảnh báo của MATLAB.

8.3 Thư viện Discrete

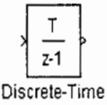
Thư viện *Discrete* phục vụ mô phỏng động học các hệ thống gián đoạn về thời gian. Các khối phục vụ nhập, xuất số liệu, các hàm mô tả ở chương 6 (là những khối không có động học riêng), được sử dụng một cách bình thường trong hệ gián đoạn. Các khối này sẽ tự động nhận thừa kế chu kỳ trích mẫu của các khối nối phía trước chúng. Tất cả các khối thuộc thư viện đều có một khâu trích mẫu ở đầu vào.

Unit Delay



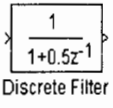
Khối *Unit Delay* có tác dụng trích mẫu tín hiệu vào và cất giữ giá trị thu được trong một chu kỳ trích mẫu. Vì vậy, khối có đặc điểm như một phần tử cơ bản của các hệ gián đoạn. Khối có thể được sử dụng như một khâu quá độ từ tần số trích mẫu thấp sang tần số trích mẫu cao.

Discrete-Time Integrator



Khối *Discrete-Time Integrator* (tích phân gián đoạn) về cơ bản cũng giống như khối *Integrator* (tích phân) liên tục. Bên cạnh chu kỳ trích mẫu ta còn phải chọn cho mỗi khối thuật toán tích phân (tích phân Euler tiến, tích phân Euler lùi hay tích phân hình thang). Sau khi đã chọn thuật toán tích phân, biểu tượng (*Icon*) của khối lại thay đổi tương ứng.

Discrete Filter (scalar)

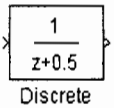


Khối *Discrete Filter* mô tả một khâu lọc số có hàm truyền đạt như sau:

$$H\left(z^{-1}\right)=\frac{y\left(z^{-1}\right)}{x\left(z^{-1}\right)}=\frac{B\left(z^{-1}\right)}{A\left(z^{-1}\right)}=\frac{b_1+b_2z^{-1}+b_3z^{-2}+\cdots+b_{m+1}z^{-m}}{a_1+a_2z^{-1}+a_3z^{-2}+\cdots+a_{n+1}z^{-n}} \tag{8.1}$$

Các hệ số của đa thức tử số và mẫu số được khai báo theo trình tự số mũ của z giảm dần, bắt đầu từ hệ số của z^0. Bằng khối *Discrete Filter* ta có thể cài đặt một cách rất dễ dàng các khâu lọc *digital* đã mô tả ở mục 4.4.

Discrete Transfer Function (scalar)



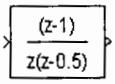
Khối *Discrete Transfer Function* có đặc điểm giống khối *Discrete Filter* và được mô tả bởi hàm truyền đạt sau:

$$H\left(z\right)=\frac{B\left(z\right)}{A\left(z\right)}=\frac{b_1z^m+b_2z^{m-1}+b_3z^{m-2}+\cdots+b_{m+1}}{a_1z^n+a_2z^{n-1}+a_3z^{n-2}+\cdots+a_{n+1}} \tag{8.2}$$

Các hệ số của hai đa thức tử số và mẫu số được khai báo theo trình tự

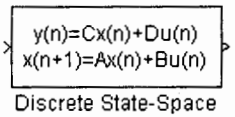
số mũ của z giảm dần, bắt đầu từ m (tử số) và n (mẫu số). Nếu ta khai báo chiều dài của hai vector hệ số tử và mẫu số như nhau bằng cách, bổ sung vào vector ngắn hơn các hệ số với giá trị 0, khi ấy khối *Discrete Transfer Function* sẽ hoạt động giống như khối *Discrete Filter*.

Discrete Zero-Pole (scalar)



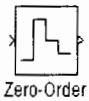
Trong khối *Discrete Zero-Pole*, thay vì phải khai báo các hệ số, ta khai báo điểm cực - điểm không của hàm truyền đạt (8.2) và một hệ số khuếch đại.

Discrete State Space



Khối *Discrete State Space* mô tả một hệ thống gián đoạn bằng mô hình trạng thái. Khối có đặc điểm sử dụng giống như khối *State Space* của các hệ liên tục.

Zero-Order Hold



Khối *Zero-Order Hold* trích mẫu tín hiệu đầu vào và giữ giá trị thu được đến thời điểm trích mẫu tiếp theo. Nên sử dụng khối *Zero-Order Hold* trong các hệ trích mẫu chưa có một trong các khối gián đoạn đã được mô tả ở trên (tức là những khối có sẵn khâu giữ chậm bậc 0). Khi chọn bước tích phân cứng, có thể sử dụng khối *Zero-Order Hold* tại các vị trí chuyển từ tần số trích mẫu cao sang tần số trích mẫu thấp hơn.

8.4 Hệ có chu kỳ trích mẫu hỗn hợp và hệ lai

Một hệ thống số kỹ thuật thường sử dụng nhiều chu kỳ trích mẫu khác nhau (gọi là hệ có chu kỳ hỗn hợp), và cần phải được lưu ý đặc biệt khi mô phỏng. Hệ lai là các hệ có chứa cả hai thành phần liên tục và gián đoạn.

8.4.1 Hệ có chu kỳ hỗn hợp

Khi hệ là hỗn hợp, SIMULINK sẽ tính từng khối với chu kỳ trích mẫu của chính nó, nếu trên menue *Simulation* / *Parameters* ta đã chọn *Solver* với *Variable-step*.

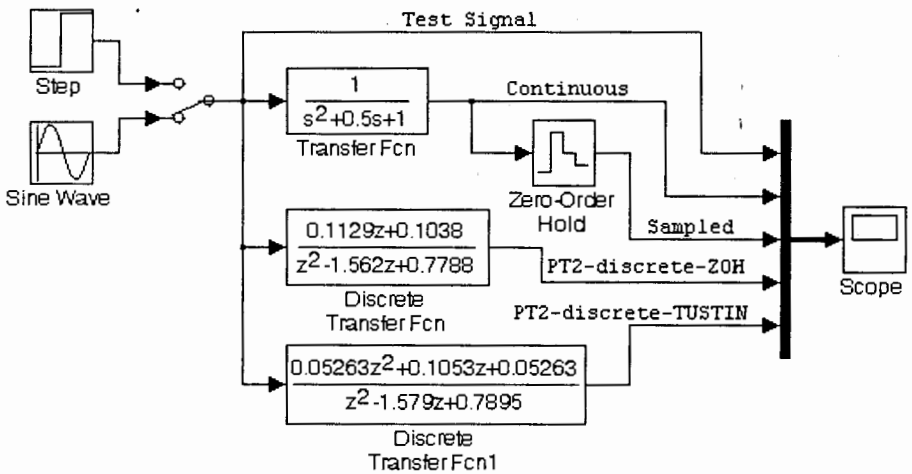
Khi chọn *Solver* với *Fixed-step* ta cần chú ý sao cho: Mọi chu kỳ trích mẫu đều nằm trong khung thời gian do bước tích phân (bước tính mô phỏng) tạo nên. Nguyên tắc này phải được bảo đảm cả đối với những hệ sử dụng chu kỳ trích mẫu bị dịch đi bởi *Offset* và những điểm quá độ giữa các chu kỳ.

Nếu tại ô *Mode* (chế độ) ta chọn *SingleTasking* (ô này sẽ chỉ hiện lênvới *Fixed-step*), SIMULINK sẽ tự động thực hiện kết hợp tại những điểm chuyển tiếp giữa các chu kỳ trích mẫu. Ngược lại, chế độ *MultiTasking* lại có yêu cầu khắt khe hơn: Ta phải sử dụng khối *Zero-Order Hold* tại điểm chuyển từ tần số trích mẫu cao xuống thấp, khối *Unit Delay* tại điểm chuyển từ tần số trích mẫu thấp lên cao. Trong cả hai trường hợp, khối *Zero-Order Hold* và *Unit Delay* đều làm việc với tần số trích mẫu thấp hơn (trong số hai tần số tại điểm chuyển tiếp). Nếu chọn chế độ (đặt *Mode* về) *Auto*, SIMULINK sẽ tự động quyết định *SingleTasking*, khi tất cả các khối của sơ đồ có chu kỳ trích mẫu giống nhau, hoặc *MultiTasking*, khi sơ đồ có nhiều chu kỳ trích mẫu khác nhau.

8.4.2 Hệ lai

Đối với hệ lai ta phải khai báo cho các phần tử liên tục một phương pháp tích phân, và nên ưu tiên chọn một trong hai thuật toán *ode45* hay *ode23* (menue *Simulation / Parameters*).

Khi sử dụng bước mô phỏng cố định (*Fixed-step Solver*), ta phải tuân thủ các nguyên tắc như đối với hệ có chu kỳ hỗn hợp (mục 8.4.1). Khi ấy, các phần tử liên tục sẽ được SIMULINK đối xử như các khối gián đoạn có chu kỳ trích mẫu bé nhất (chính bằng bước mô phỏng).

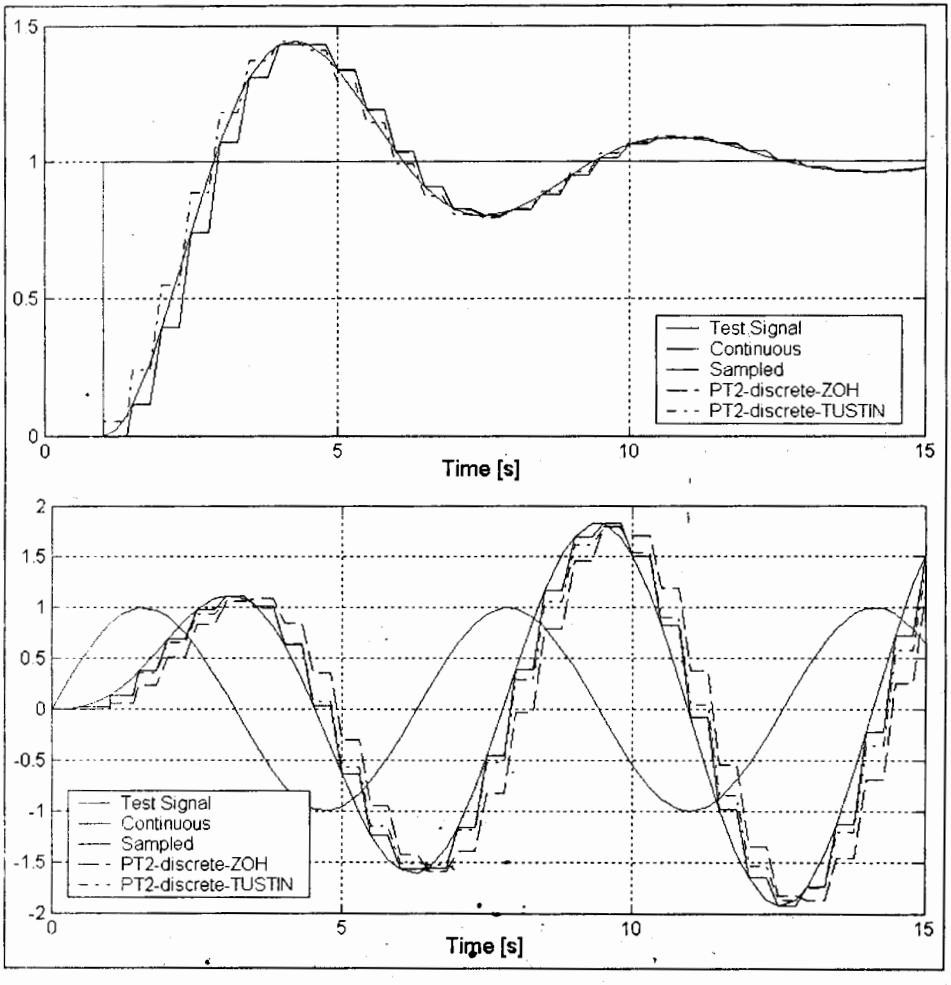


Hình 8.1 *Mô hình SIMULINK (hệ lai) có tên Fig08_01.mdl: Phần hệ liên tục (trên) và phần hệ gián đoạn với chu kỳ trích mẫu 0,5s (dưới)*

Hình 8.1 mô tả ở phía trên là phần hệ thống liên tục (có và không có trích mẫu), phía dưới là phần hệ thống gián đoạn theo hai phương pháp khác nhau *Euler explicit* (*Zero-Order Hold*: ZOH) và xấp xỉ *Tustin*. Có thể dễ dàng tạo nên các phần hệ thống đó nhờ sử dụng các lệnh quen biết của *Control System Toolbox* (chương 3) dưới đây:

```
>> pt2                  = tf([1],[1 0.5 1]);
>> pt2_discrete_zoh     = c2d(pt2,0.5,'zoh');
>> pt2_discrete_tustin = c2d(pt2,0.5,'tustin');
```

Bằng lệnh thứ nhất ta tạo ra một khâu tỷ lệ quán tính bậc 2 có tên là pt2 (phần trên của hình 8.1). Mô hình gián đoạn theo phương pháp ZOH của khâu pt2 đó được tạo bởi lệnh thứ 2, theo phương pháp TUSTIN bởi lệnh thứ 3 (phần giữa và dưới của hình 8.1), cả hai mô hình gián đoạn đều được tạo với chu kỳ trích mẫu 0,5s.



Hình 8.2 Kết quả mô phỏng với sơ đồ SIMULINK ở hình 8.1

Giả sử muốn tạo ra một hệ trích mẫu từ một sơ đồ SIMULINK (liên tục, gián đoạn) có sẵn, ta có thể (tương tự mục 7.2) dùng lệnh:

`dlinmod('modelname',T[,x,u,pert])`

với cú pháp gần giống như `linmod`. Tham số thêm là chu kỳ trích mẫu T. Lệnh `dlinmod` chỉ sử dụng phương pháp *Euler explicit* (ZOH) để gián đoạn hóa. Chính vì vậy, hàm truyền đạt của khối *Discrete Transfer Fcn* (hình 8.1, giữa) còn có thể được tạo ra bởi các lệnh dưới đây (với điều kiện: Khối *Transfer Fcn* phải được đặt vào giữa hai khối *Inport* và *Outport*, là hai khối không xuất hiện ở hình 8.1).

```
>> [A,B,C,D] = dlinmod('Fig08_01',0.5);
>> pt2_discrete_zoh = tf(ss(A,B,C,D,0.5));
```

Kết quả mô phỏng được giới thiệu ở hình 8.2. Trong hình bên trái dường như hai đồ thị "Sampled" và "PT2-discrete-ZOH" trùng với nhau. Tuy nhiên, việc trùng hoàn toàn là không có và cũng không thể có: Các giá trị của "Sampled" phụ thuộc vào tín hiệu liên tục và do đó phụ thuộc vào *Solver* đã chọn, các giá trị của "PT2-discrete-ZOH" (tín hiệu ra của hàm truyền đạt gián đoạn) lại phụ thuộc vào phương trình sai phân là hàm xấp xỉ (*Euler explicit*) của ảnh z chính xác.

8.5 Tóm tắt nội dung chương 8

Sau khi đã nghiên cứu kỹ chương 8, người đọc cần nắm vững các nội dung sau đây:

1. Sử dụng các khối của thư viện con *Discrete*.
2. Tự chọn được *Solver* (thuật toán tích phân) phù hợp cho các hệ gián đoạn thuần túy, hay hệ lai, có chu kỳ trích mẫu thống nhất hay hỗn hợp.
3. Kiến thức cơ sở về mô phỏng các hệ *Multitasking*.

9 Phân tích và tổng hợp vòng điều chỉnh

Chương này sẽ sử dụng kết hợp các kiến thức thu được từ mục 3.3, chương 6 và chương 7 vào mục đích phân tích và tổng hợp (thiết kế) vòng điều chỉnh (ĐC). Trên cơ sở ví dụ đơn giản là hệ thống điều khiển động cơ một chiều (ĐCMC) kích thích độc lập, ta sẽ từng bước xây dựng vòng ĐC trên nền SIMULINK, tính toán tham số của bộ điều khiển (ĐK) và thiết kế khâu quan sát (QS) cần thiết.

9.1 Động cơ một chiều kích thích độc lập

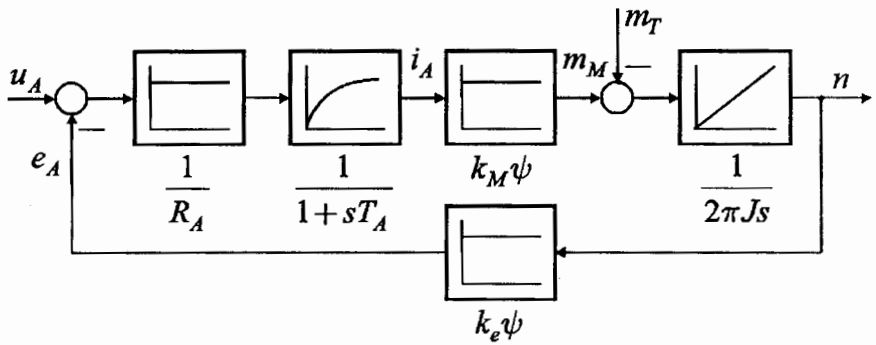
Đối tượng ĐK là ĐCMC cần phải được ĐC dòng (qua đó ĐK mômen quay của động cơ) và ĐC tốc độ quay. Đối tượng được mô tả bởi hệ các phương trình dưới đây (để đơn giản ta chỉ xét trường hợp từ thông là hằng và có giá trị danh định[1]):

- *Điện áp phần ứng*: $$u_A = e_A + R_A i_A + L_A \frac{di_A}{dt} \quad (9.1)$$
- *Sức từ động cảm ứng*: $$e_A = k_e \psi n \quad (9.2)$$
- *Tốc độ quay*: $$\frac{dn}{dt} = \frac{1}{2\pi J}(m_M - m_T) \quad (9.3)$$
- *Mômen quay*: $$m_M = k_M \psi i_A \quad (9.4)$$
- *Hằng số động cơ*: $$k_e = 2\pi k_M \quad (9.5)$$
- *Hằng số thời gian phần ứng*: $$T_A = \frac{L_A}{R_A} \quad (9.6)$$

Sơ đồ cấu trúc ĐK của đối tượng ĐCMC được giới thiệu ở hình 9.1. Động cơ có các tham số sau đây:

Điện trở phần ứng:	R_A = 250mΩ	*Mômen quán tính*:	J = 0,012kgm^2
Điện cảm phần ứng:	L_A = 4mH	*Hằng số động cơ*:	k_e = 236,8
Từ thông danh định:	ψ_R = 0,04Vs		k_M = 38,2

[1] Giá trị danh định: Rated value ψ_R



Hình 9.1 *Sơ đồ cấu trúc của ĐCMC kích thích độc lập (từ thông hằng)*

9.1.1 Khai báo lập trị ban đầu

Tham số của ĐCMC được khai báo trong một *File* có tên là *Fig09_01i.m* và ta sẽ phải gọi *File* từ cửa sổ *Command* của MATLAB trước khi mô phỏng. Ngoài ra ta còn định nghĩa một vài tham số mô phỏng như thời gian ngừng `Tstop` và bước tích phân tối đa `step_max` (hình 9.3). Đồng thời, ta xác định tín hiệu đầu vào dưới dạng bước nhảy điện áp tại thời điểm `Tstep` lên giá trị `UAref`:

```
%%  Initialization file for Fig09_02.mdl

%% General simulation data
Tstop    = 0.2 ;        % Stopp the Simulation
step_max = 0.0001 ;     % Maximal step size

%% Voltage reference step
Tstep = 0 ;             % Step time [ s ]
UAref = 50 ;            % Reference voltage       [ V ]

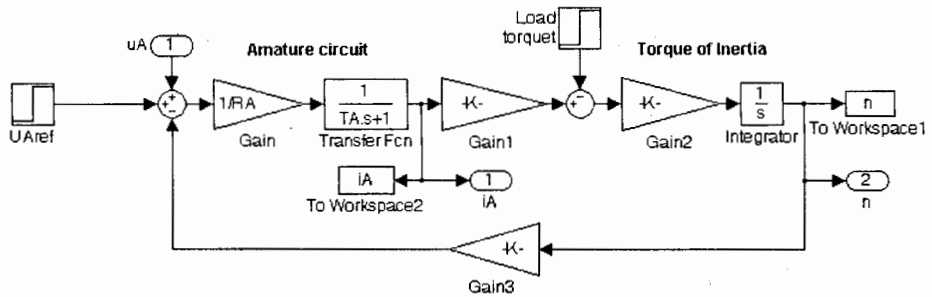
%% Load torque step
T_MW = 0 ;              % Step time [ s ]
MW   = 0 ;              % Step amplitude          [ Nm ]

%% Data of DC motor
RA   = 0.250 ;          % Armature resistor       [ Ohm ]
LA   = 0.004 ;          % Armature inductance     [ H ]
TA   = LA / RA ;        % Armature time constant  [ s ]
PsiR = 0.04 ;           % Rated fluxs             [ Vs ]
J    = 0.012 ;          % Torque of inertia       [ kg m^2 ]
kM   = 38.2 ;           % Motor constants
kE   = 2*pi*kM ;
```

9.1.2 Mô hình SIMULINK

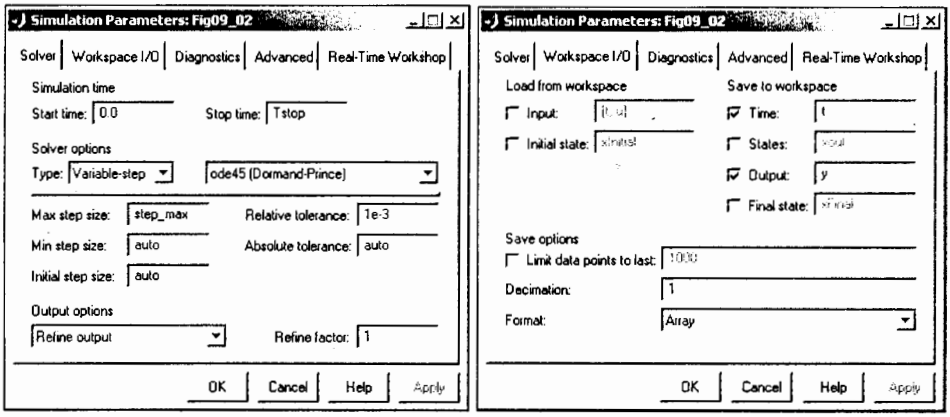
Với sơ đồ cấu trúc ở hình 9.1, các biến và tham số khai báo tại *Fig09_01i.m*, ta dễ dàng xây dựng được mô hình SIMULINK có tên *Fig09_02.mdl* (hình 9.2). Sơ đồ có sử dụng các khối sau đây:

Thư viện con	Khối	Thư viện con	Khối
Sources:	*Step*	*Continuous*:	*Integrator, Transfer Fcn*
Sinks:	*To Workspace*	*Math*:	*Sum, Gain*
Signal&Systems:	*Inport, Outport*		



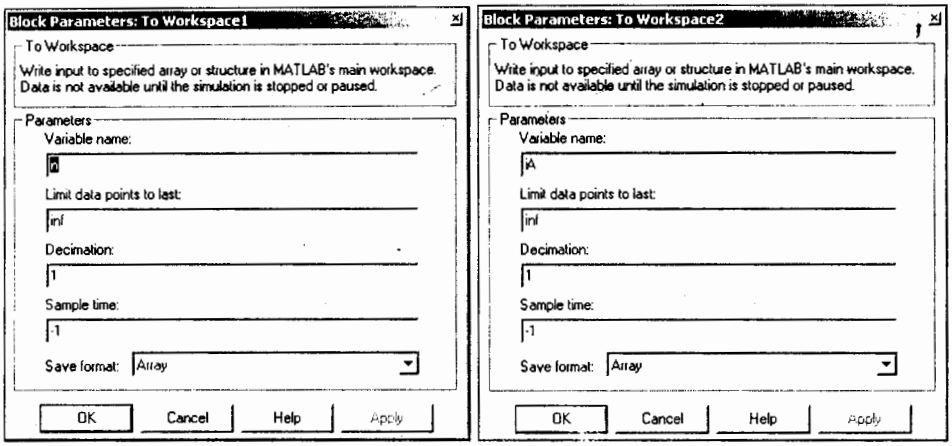
Hình 9.2 *Mô hình SIMULINK có tên Fig09_02.mdl của ĐCMC kích thích độc lập*

Hình 9.3 cho ta thấy tham số khai báo tại các trang *Solver* và *Workspace I/O*. Tại trang *Workspace I/O* ta đặt tên *t* cho biến thời gian tại *Save to Workspace* và *y* cho đầu ra *Output*.



Hình 9.3 *Hộp thoại Simulation Parameters: trang Solver và Workspace I/O*

Bằng các khối To Workspace ta cất tốc độ quay dưới tên biến *n* và dòng phần ứng i_A dưới tên biến iA (hình 9.4).



Hình 9.4 *Trang khai báo Block Parameters: Các khối To Workspace cất tốc độ quay n và dòng phần ứng i_A*

Khối *Inport* uA, các khối *Outport* n và iA được sử dụng nhằm mục đích tạo mô hình trạng thái và mô hình truyền đạt từ đầu vào uA tới các đầu ra n, iA: Các mô hình đó sẽ được sử dụng để khảo sát đối tượng bằng các phương pháp giới thiệu ở chương 3.

9.2 Khảo sát động học của đối tượng

9.2.1 Khảo sát bằng SIMULINK

Để khảo sát ta hãy bắt đầu bằng việc đưa tới đầu vào của ĐCMC một điện áp có dạng bước nhẩy. Để làm điều đó, ta sử dụng ở đầu vào một khối *Step*. Khối *Step* có hai tham số *Step time*, *Final Value* nhận hai giá trị Tstep, UAref được khai báo trong *File* lập trị ban đầu *Fig09_01i.m*. Ta lần lượt thực hiện các bước:

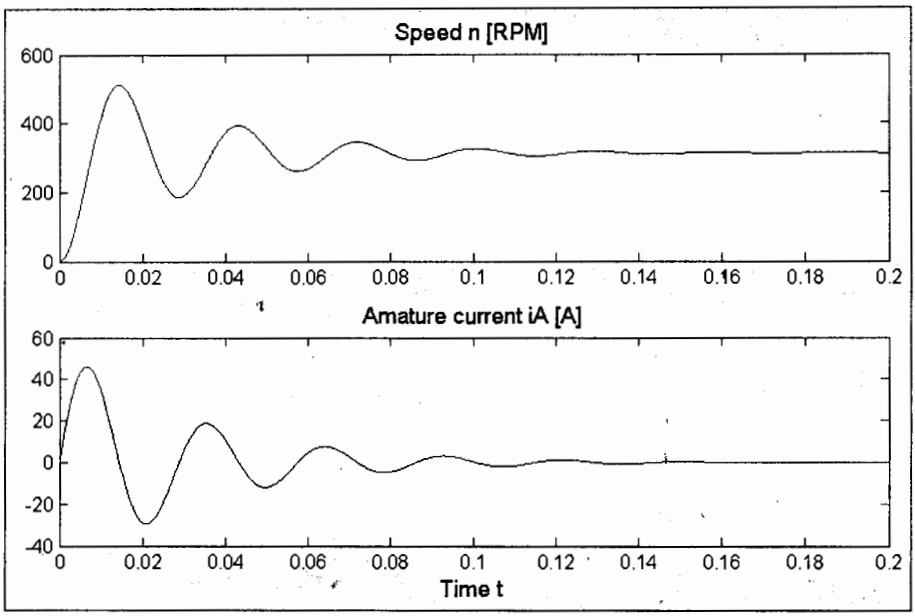
- Bước 1: Lập trị ban đầu và khai báo tham số ĐCMC bằng cách gọi:
  ```
  >> Fig09_01i;
  ```
- Bước 2: Khởi động mô phỏng mô hình *Fig09_02.mdl*
- Bước 3: Vẽ đồ thị bằng cách gọi:
  ```
  >> Fig09_05plot;
  ```

Kết quả thu được chính là hình 9.5. Chuỗi lệnh vẽ đồ thị được tập hợp trong *script* có tên *Fig09_05plot.m* với nội dung sau đây:

```
%%  Plot file of Fig09_02.mdl
```

```
figure
    subplot(211)
    plot(t,n*60)
    title('Speed n [RPM]','FontSize',12)
    subplot(212)
    plot(t,iA)
    title('Amature current iA [A]','FontSize',12)
    xlabel('Time t','FontSize',12)

if exist('print') == 1 , print -depsc p_gnm_nia.eps , end
if exist('print') == 1 , print -deps p_gnm_nia_sw.eps , end
```



Hình 9.5 *Đáp ứng tốc độ quay n và dòng phần ứng i_A khi có bước nhẩy điện áp u_A*

9.2.2 Khảo sát bằng MATLAB sử dụng mô hình tuyến tính hóa

Để có thể sử dụng các công cụ (mà *Control System Toolbox* cung cấp) vào mục đích khảo sát một mô hình SIMULINK, trước hết ta phải chuyển mô hình SIMULINK sang dạng mô hình LTI. Để chuyển dạng mô hình, bạn đọc đã biết đến các công cụ của MATLAB như các lệnh `linmod`, `linmod2`[1] và `dlinmod`, hay của *Control System Toolbox* như `ss` (xem mục 7.2).

[1] Lệnh `linmod2` sử dụng một thuật toán đã cải thiện so với `linmod`, cho phép giảm hữu hiệu sai số làm tròn khi tính toán.

Bước đầu tiên phải làm là việc tạo từ phần mô hình SIMULINK (cần được khảo sát, nội tại được mô tả bởi một hệ phương trình vi phân thường *Ordinary Differential Equations*: ODE) một mô hình trạng thái tuyến tính – dừng[1]:

```
[A,B,C,D] = linmod('sys'[,x,u])
[A,B,C,D] = linmod2('sys'[,x,u])
[A,B,C,D] = dlinmod('sys',Ts[,x,u])
```

Trong các lệnh trên, *sys* là tên của mô hình SIMULINK, *Ts* là chu kỳ trích mẫu của hệ gián đoạn. Với các tham số tùy chọn *x* và *u* ta có thể khai báo giá trị của biến trạng thái và biến vào, tức là: Khai báo điểm làm việc ma ta muốn tuyến tính hóa mô hình quanh đó. Điểm làm việc mặc định sẽ là điểm 0.

Các đầu vào / ra phải được tạo ra ở tầng mô hình trên cùng[2] của sơ đồ SIMULINK nhờ các khối *Inport* và *Outport*. Trong mục 9.1.2 (hình 9.2) ta đã chuẩn bị sẵn điều này với các biến `uA`, `n` và `iA`. Cuối cùng, *A*, *B*, *C* và *D* chính là các ma trận quen biết của mô hình trạng thái.

Mô hình LTI dạng SS được tạo ra từ các ma trận của mô hình trạng thái bằng các lệnh đã biết ở mục 3.1.3.

```
sys  = ss(A,B,C,D)
sysd = ss(A,B,C,D,Ts)
```

Việc áp dụng các thao tác cần thiết vừa mô tả cho đối tượng ĐCMC của ví dụ ở hình 9.2 được tập hợp lại trong *script* có tên *Fig09_06.m* với nội dung dưới đây.

```
%%  Fig09_06.m
%%  Using linmod to generate a LTI model from Fig09_02.mdl
[A,B,C,D] = linmod('Fig09_02')          % Linearization of
                                        % simulink model
sysDCMotor = ss (A,B,C,D) ;             % Generating the LTI
                                        % model sysDCMotor
sysDCMotor.InputName  = 'uA' ;          % Set the LTI input
                                        % variable
sysDCMotor.OutputName = {'iA' 'n'} ;    % Set the LTI output
                                        % variables
sysDCMotor                              % Output and show the
                                        % LTI model
```

Sau đó ta gọi *script* vừa mới viết từ *Command Windows* của MATLAB và thu được kết quả.

```
>> Fig09_06
A =
  1.0e+003 *
   -0.0625   -0.0384
    1.2666         0
```

[1] Hệ dừng là hệ có hệ số hằng

[2] Tức là: không được phép ở trong các Subsystems

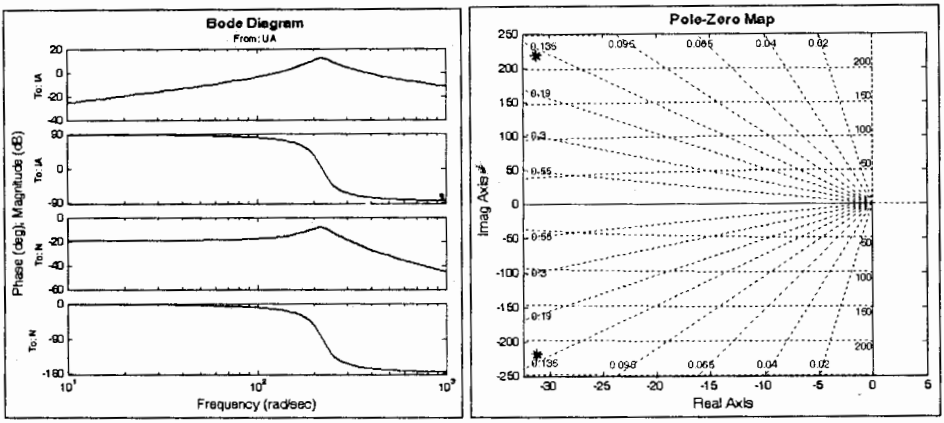
```
B =
    4.0000
         0
C =
   62.5000         0
         0    1.0000
D =
     0
     0

a =
                      x1          x2
           x1        -62.5     -38.403
           x2       1266.6           0
b =
                      UA
           x1              4
           x2              0
c =
                      x1          x2
           IA         62.5           0
            N            0           1
d =
                      UA
           IA             0
            N             0
Continuous-time model.
```

Với kết quả tính của *Fig09_06.m* ta thu được mô hình đã tuyến tính hóa của sơ đồ SIMULINK (xung quanh điểm làm việc được khai báo, đặc trưng bởi các ma trận A, B, C và D) và mô hình LTI dạng SS (có tên sysDCMotor, đặc trưng bởi các ma trận a, b, c và d). Lúc này ta đã có thể sử dụng các công cụ quen biết của chương 3 để khảo sát ĐCMC. Ví dụ: Ta dễ dàng thu được tần số riêng ω_n, hệ số tắt dần D và các điểm cực của ĐCMC bằng cách sử dụng lệnh damp (mục 3.3.2):

```
>> damp(sysDCMotor)
        Eigenvalue              Damping      Freq. (rad/s)
 -3.13e+001 + 2.18e+002i      1.42e-001       2.21e+002
 -3.13e+001 - 2.18e+002i      1.42e-001       2.21e+002
>> bode(sysDCMotor)
>> pzmap(sysDCMotor)
>> sgrid
```

Đồ thị BODE và biểu đồ vị trí 2 điểm cực của ĐCMC được tạo bởi các lệnh bode và pzmap được minh họa tại hình 9.6.



Hình 9.6 *Đồ thị BODE (trái) và biểu đồ phân bố điểm cực (phải) của đối tượng ĐCMC dưới dạng mô hình LTI-SS có tên sysDCMotor*

Tạo mô hình LTI từ mô hình SIMULINK

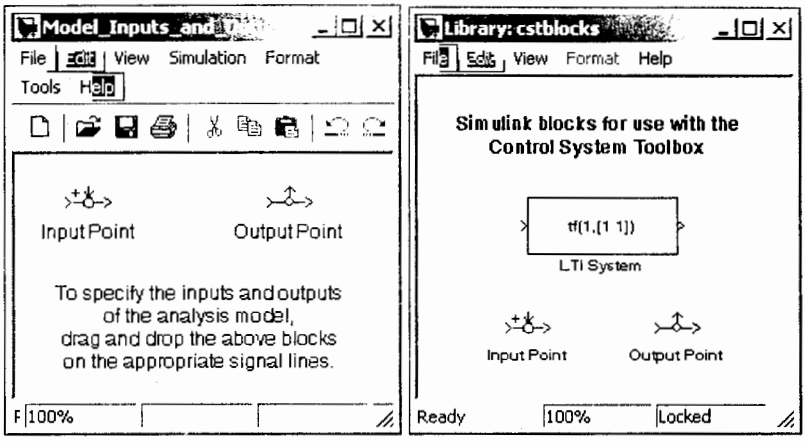
`linmod('sys'[,x,u])`	Tạo mô hình trạng thái liên tục (tuyến tính hóa), từ mô hình SIMULINK có tên *sys*
`linmod2('sys'[,x,u])`	
`dlinmod('sys',Ts[,x,u])`	Tạo mô hình trạng thái gián đoạn (tuyến tính hóa) với chu kỳ trích mẫu *Ts*, từ mô hình SIMULINK có tên *sys*
`ss(A,B,C,D[,Ts])`	Tạo mô hình LTI (dạng SS) liên tục hoặc gián đoạn từ mô hình trạng thái với *A*, *B*, *C* và *D* cho trước

9.2.3 Khảo sát mô hình LTI qua đối thoại với LTI-Viewer

Ngoài hai khả năng giới thiệu ở mục 9.2.1 và 9.2.2, SIMULINK còn cung cấp cho ta công cụ *LTI-Viewer*, cho phép khảo sát hệ thống nhanh và đơn giản mà không cần đến lệnh của MATLAB.

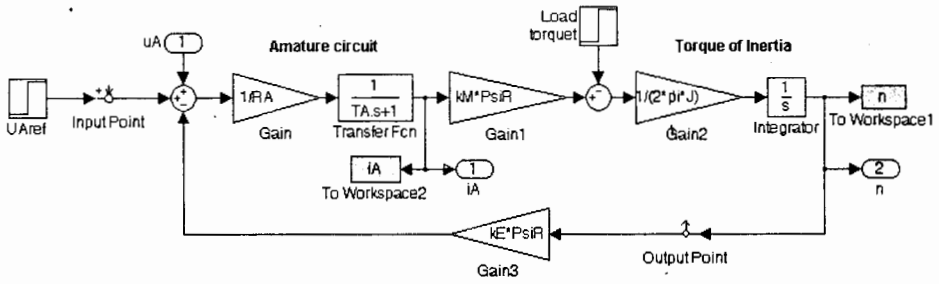
Để gọi *LTI-Viewer* ta vào menue *Tools* / *Linear Analysis* tại cửa sổ của mô hình SIMULINK (mà ta đang muốn khảo sát): Sẽ xuất hiện cửa sổ của thư viện *Model_Inputs_and_Outputs* (hình 9.7, bên trái) và giao diện của *LTI-Viewer* (hình 9.9).

Để sử dụng *LTI-Viewer*, sơ đồ SIMULINK sẽ phải có ít nhất một khối *Input Point* và một khối *Output Point*, được gắp từ thư viện *Model_Inputs_and_Outputs* đưa sang cửa sổ của sơ đồ SIMULINK và thả (thao tác *Drag & Drop*) vào đường tín hiệu mà ta quan tâm (hình 9.8). Sau này, khi đã khảo sát xong, bằng cách chọn *Remove all Input/Output Points* tại menue *Simulink* của *LTI-Viewer*, ta lại gỡ bỏ toàn bộ các khối mới gắp sang và sơ đồ SIMULINK trở lại nguyên như ban đầu.



Hình 9.7 *Thư viện Model_Inputs_and_Outputs và cstblocks*

Control System Toolbox có một thư viện riêng mang tên `cstblocks`, khi gọi từ cửa sổ *Command* của MATLAB (ngoài hai khối *Input Point* và *Output Point*) có thêm khối *LTI System* (hình 9.7, bên phải) phục vụ mục đích tạo mô hình LTI trực tiếp trong sơ đồ SIMULINK.

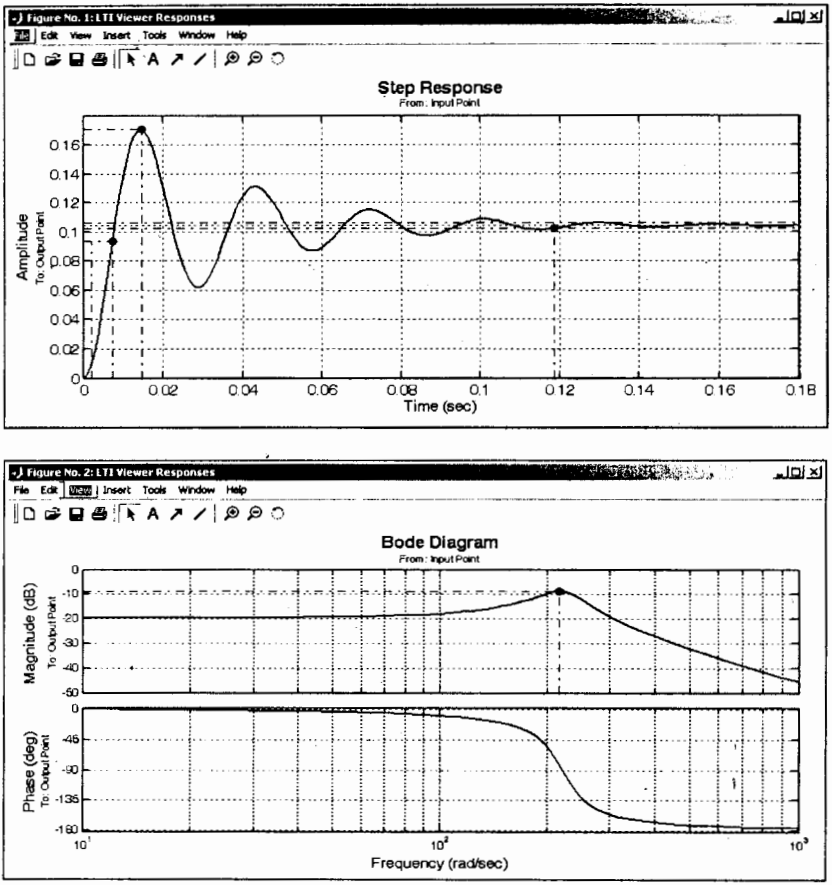


Hình 9.8 *Mô hình SIMULINK (có tên Fig09_08.mdl) của ĐCMC, đã bổ sung thêm hai khối Input Point và Output Point*

Hình 9.8 cho ta thấy sơ đồ SIMULINK của ĐCMC đã được chuẩn bị (bổ sung hai khối *Input Point* dành cho điện áp vào uA và *Output Point* cho tốc độ quay n) để khảo sát bằng *LTI-Viewer*.

Hình 9.9 giới thiệu đáp ứng bước nhảy và đồ thị BODE của tốc độ quay. Để bắt đầu, bạn đọc hãy gọi menue *Simulink / Get Linearized Model*. Với *Simulink LTI-Viewer* ta có thể khảo sát đối tượng bằng các phương pháp khác nhau, như vẽ các đồ thị BODE, NYQUIST, NICHOLS hay biểu đồ phân bố điểm cực. Để chọn phương pháp, ta chỉ cần nháy chuột phải vào cửa sổ *LTI-Viewer*, một menue với các khả năng chọn lựa sẽ mở ra.

Để có thể biết sâu thêm về cách sử dụng *LTI-Viewer*, các bạn đọc quan tâm sẽ buộc phải tự mình tìm đọc *User's Guide* của MATLAB và SIMULINK.



Hình 9.9 LTI Viewer: Đáp ứng bước nhảy (trên) và đồ thị BODE (dưới)

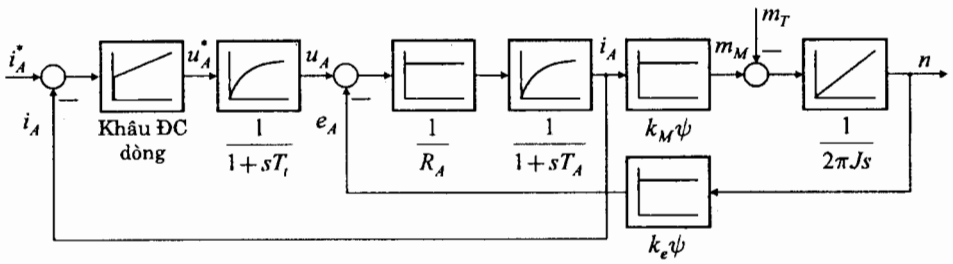
9.3 Điều chỉnh với nhiều vòng phân cấp (Cascade Control)

Mục tiêu tiếp theo là mô phỏng hệ thống ĐC tốc độ quay cho ĐCMC. Cấu trúc kinh điển của một hệ truyền động như vậy là cấu trúc có nhiều vòng phân cấp với một khâu ĐC tốc độ quay n ở vòng ngoài và khâu ĐC dòng phần ứng i_A ở vòng trong. Ta đều biết: Vòng ĐC i_A phía trong có tác dụng áp nhanh mômen quay m_M (do từ thông là hằng, mômen chỉ phụ thuộc i_A, công thức (9.4)).

Do trọng tâm hiện tại của ta là xây dựng mô hình và mô phỏng hệ thống bằng MATLAB, SIMULINK, chứ không phải là thiết kế (tổng hợp) các khâu ĐC của cả 2 vòng. Về vấn đề này bạn đọc sẽ phải tra cứu trong tài liệu chuyên khác.

9.3.1 Điều chỉnh dòng phần ứng

Để thiết kế khâu ĐC dòng, thường ta sử dụng hàm truyền đạt của mạch điện phần ứng và bỏ qua sức từ động cảm ứng e_A. Vòng ĐC dòng có cấu trúc như hình 9.10 dưới đây.



Hình 9.10 Vòng điều chỉnh dòng của ĐCMC với hàm truyền đạt bỏ qua sức từ động e_A

Nếu coi gần đúng thiết bị chỉnh lưu có ĐK là khâu tỷ lệ với quán tính bậc nhất (khâu PT_1, hằng số thời gian T_t), hàm truyền đạt của mạch phần ứng là:

$$G_I(s)=\frac{i_A(s)}{u_A^*(s)}=\frac{1}{1+sT_t}\frac{1}{R_A}\frac{1}{1+sT_A} \tag{9.7}$$

Hằng số thời gian của mạch phần ứng và hệ số khuếch đại ta đã biết:

$$T_1=T_A=\frac{L_A}{R_A} \qquad V=\frac{1}{R_A} \tag{9.8}$$

Hằng số thời gian của khâu chỉnh lưu là:

$$T_\sigma=T_t=100\mu s \tag{9.9}$$

Khâu ĐC dòng giả sử được chọn là khâu PI, thiết kế theo tiêu chuẩn tối ưu module với công thức sau:

$$T_{RI}=T_1=T_A \qquad V_{RI}=\frac{T_1}{2T_\sigma V}=\frac{R_AT_A}{2T_t}=\frac{L_A}{2T_t} \tag{9.10}$$

Từ đó ta thu được hàm truyền đạt của khâu ĐC dòng kiểu PI như sau:

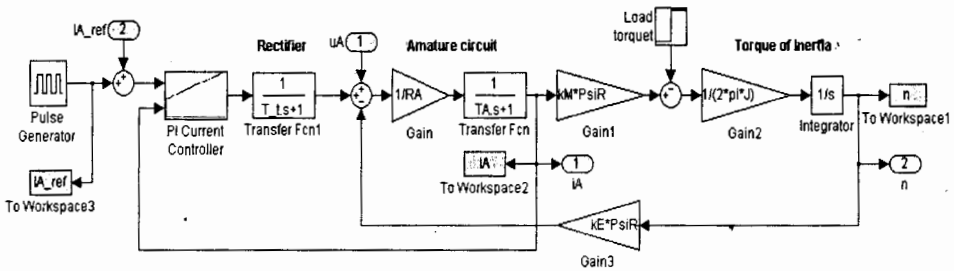
$$G_{RI}(s)=\frac{u_A^*(s)}{i_A^*(s)-i_A(s)}=V_{RI}\left(1+\frac{1}{sT_{RI}}\right)=\frac{L_A}{2T_t}\left(1+\frac{1}{sT_A}\right) \tag{9.11}$$

Sơ đồ cấu trúc SIMULINK có điều chỉnh dòng phần ứng của ĐCMC được giới thiệu ở hình 9.11, xây dựng trên cơ sở sơ đồ ở hình 9.2. Sơ đồ SIMULINK ở hình 9.11 có tên là *Fig09_11.mdl*, sử dụng *script Fig09_11i.m* để khai báo tham số ban đầu của vòng ĐC, và đương nhiên sử dụng tham số của ĐCMC cất trong *Fig09_01i.m*:

```
%% Fig09_11i.m
%% Initialization file for Fig09_11.mdl
```

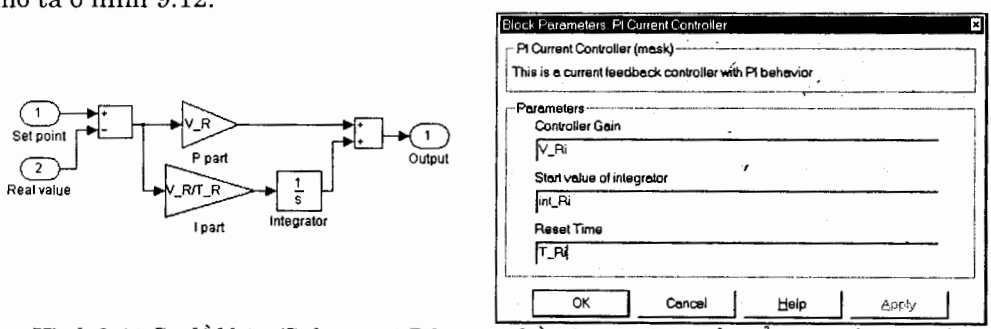
```
%% Motor and general simulation parameters
Fig09_01i                % Parameters of DC motor
%% Time constant of controlable rectifier
T_t = 0.0001 ;           % Time constant
%% Current blocks
T_IAref = 0.05 ;         % Width of period    [ s ]
IA_ref  = 10 ;           % Set point for current [ A ]
%% PI current feedback controller
V_Ri    = LA / (2*T_t) ; % Gain     [ Ohm ]
T_Ri    = TA ;           % Time constant of controller [ s ]
int_Ri = 0 ;             % Start value of integration part
```

Tương tự mục 9.2.1, để mô phỏng trước hết ta gọi *Fig09_01i.m* trên *Command Windows* của MATLAB để khai báo các biến và tham số với *Workspace*, sau đó ta chạy sơ đồ *Fig09_11.mdl* ở hình 9.11 dưới đây.



Hình 9.11 *Mô hình vòng ĐC dòng phần ứng của ĐCMC (có tên Fig09_11.mdl)*

Hình 9.11 có khâu ĐC dòng (*PI Current Controller*) là một *Subsystem* được mô tả ở hình 9.12.

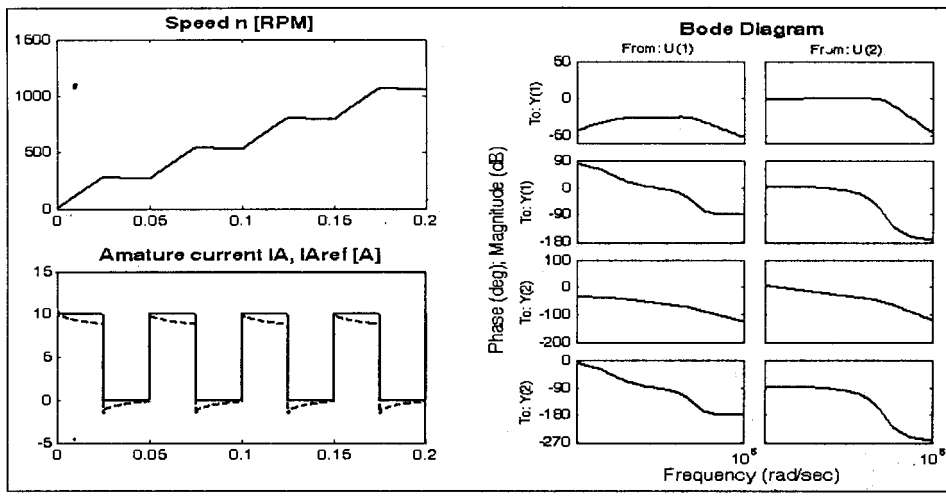


Hình 9.12 *Sơ đồ khâu (Subsystem) ĐC dòng phần ứng (trái) và cửa sổ nhập số liệu (phải)*

Sau khi chạy ta thực hiện chùm lệnh dưới đây để vẽ hình 9.13.

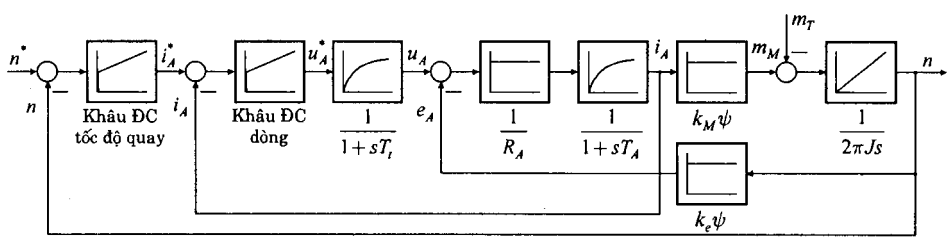
```
>> Fig09_11i
>> subplot(221)
>> plot(t,60*n,'k-')
```

```
>> title('Speed n [RPM]','FontSize',10)
>> subplot(223)
>> plot(t,IA_ref,'k-',t,iA,'k--')
>> title('Amature current IA, IAref [A]','FontSize',10)
>> [A,B,C,D] = linmod('Fig09_11');
>> sysDCMotor = ss (A,B,C,D);
>> subplot(122)
>> bode(sysDCMotor)
```



Hình 9.13 *Trái: Đáp ứng tốc độ quay n và dòng i_A khi có ĐC. Phải: Đồ thị BODE, phản ánh quan hệ vào ra giữa các đầu vào U(1), U(2) và các đầu ra Y(1), Y(2)*

9.3.2 Điều chỉnh tốc độ quay



Hình 9.14 *Hệ thống điều chỉnh tốc độ quay của ĐCMC với hai vòng có phân cấp*

Đối với vòng ĐC tốc độ quay, vòng ĐC dòng vừa thiết kế là vòng cấp dưới, có thể được gom lại thành một phần của đối tượng ĐK với hàm truyền đạt như sau (*Iscl* = vòng ĐC dòng thay thế, *Substitutional Control Loop*):

$$G_{Iscl}(s)=\frac{i_A(s)}{i_A^*(s)}=\frac{1}{1+2T_t s+2T_t^2 s^2}\approx\frac{1}{1+2T_t s} \tag{9.12}$$

Cùng với hàm truyền đạt của phần cơ, đối tượng ĐK tổng quát lúc này sẽ có mô hình truyền đạt như sau:

$$G_N(s)=\frac{n(s)}{i_A^*(s)}=\frac{k_M\psi}{2\pi Js}\frac{1}{1+2T_t s}=\frac{1}{T_M s}\frac{1}{1+T_\sigma s} \tag{9.13}$$

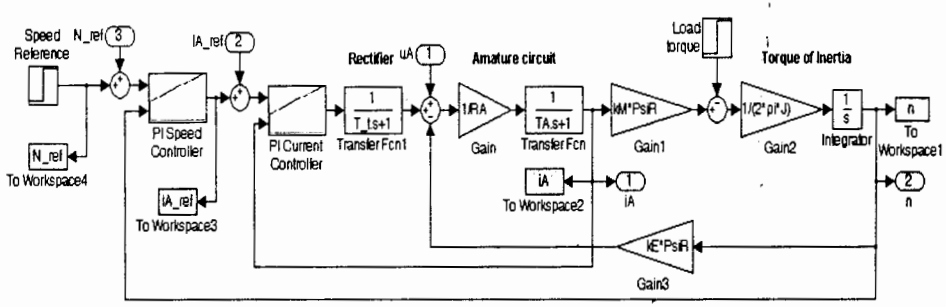
Với hai tham số của đối tượng là hằng số thời gian cơ T_M và hằng số thời gian thay thế của vòng trong T_σ, ta dễ dàng áp dụng phương pháp thiết kế theo tiêu chuẩn tối ưu đối xứng và thu được các tham số của khâu ĐC:

$$T_{RN}=4T_\sigma=8T_t \qquad V_{RN}=\frac{T_M}{2T_\sigma}=\frac{1}{4T_t}\frac{2\pi J}{k_M\psi} \tag{9.14}$$

Với các tham số trên ta thu được hàm truyền đạt tổng quát (thay thế: *Substitutional Control Loop*) của vòng ĐC tốc độ quay, thiết kế theo tiêu chuẩn tối ưu đối xứng:

$$G_{Nscl}(s)=\frac{n(s)}{n^*(s)}=\frac{1+8T_t s}{1+8T_t s+16T_t^2 s^2+16T_t^3 s^3} \tag{9.15}$$

Sơ đồ SIMULINK *Fig09_15.mdl* chính là sơ đồ ở hình 9.11 có hồi tiếp tốc độ quay, được mở rộng thêm vài khối *To Workspace*, *Input* và *Output* để thuận tiện cho việc khảo sát.



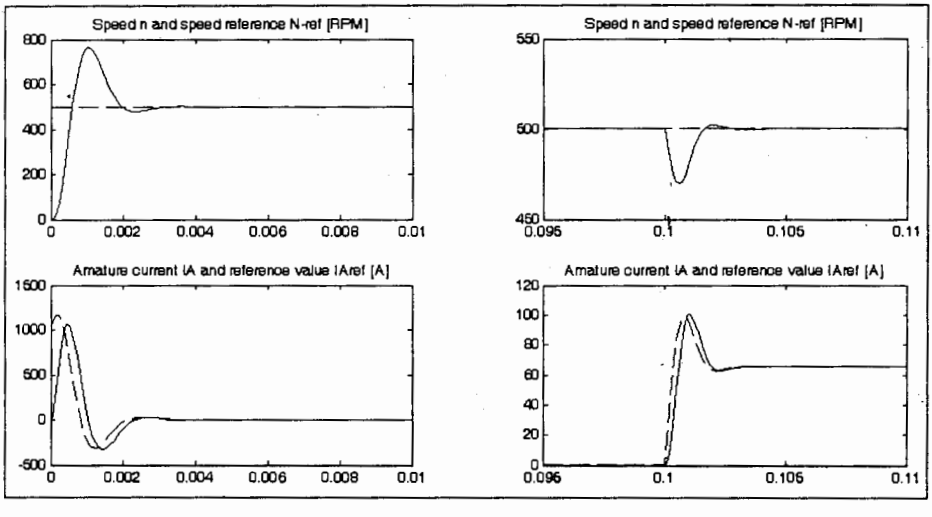
Hình 9.15 *Cấu trúc hệ thống ĐC tốc độ quay của ĐCMC kích thích độc lập với hai vòng ĐC có phân cấp*

Giống như các lần khảo sát trước, lần này các tham số của động cơ và điều kiện mô phỏng cũng được khai báo lập trị trong một *File* có tên là *Fig09_15i.m*. Trong lần khảo sát này, ngoài đột biến bước nhảy của giá trị đặt cho tốc độ quay `N_ref`, ta còn kiểm tra đặc tính của hệ bằng cách tạo đột biến phụ tải `MW`. *Script* lập trị có nội dung như đoạn mã dưới đây. Sau khi chạy *Fig09_15i.m*, ta khởi động *Fig09_15.mdl* để mô phỏng. Cuối cùng, các kết quả khảo sát được biểu diễn bằng đồ thị ở hình 9.16:

```
%% Fig09_15i.m
%% Initialization file for Fig09_15.mdl

%% Motor and general simulation data
Fig09_01i                          % DC Motor
Fig09_11i                          % Current control loop
                                   % + Rectifier
step_max = 0.0001 ;                % Maximal step size
%% Reference value of speed
T_Nref = 0 ;                       % Width of a period   [ s ]
N_ref  = 500/60 ;                  % Set point of speed  [1/s]
%% Step of load torque
T_MW = 0.10 ;                      % Step time           [s]
MW   = 100 ;                       % Step amplitude      [Nm]

%% Speed controller: PI-Behavior, symmetrical optimum
V_Rn = 2*pi*J/(kM*PsiR*2*2*T_t); % Controller gain    [Vs]
T_Rn = 4*2*T_t ;                   % Reset time          [s]
int_regn = 0 ;                     % Start value of integrator
```



Hình 9.16 *Đáp ứng bước nhảy giá trị đặt tốc độ (trái) và đáp ứng nhiễu phụ tải (phải). Hàng trên là tốc độ quay, hàng dưới là dòng phần ứng*

Đồ thị ở hình 9.16 thu được nhờ chùm lệnh dưới đây:

```
>> Fig09_15i          % Lập trị ban đầu và chuẩn bị mô phỏng
```

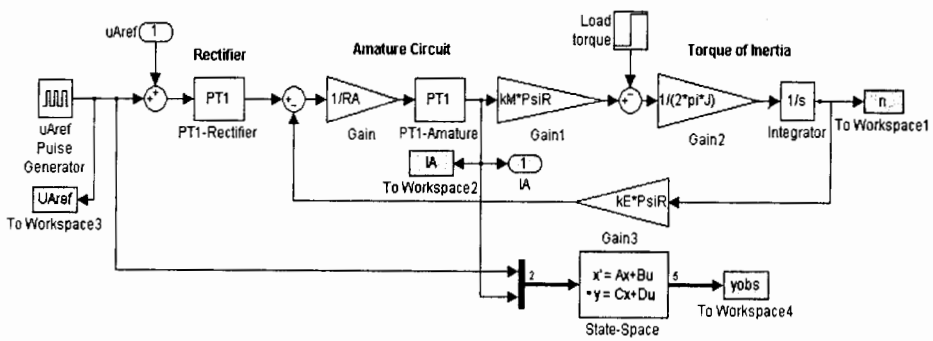
(Sau khi gọi *Fig09_15i.m*, cho chạy *Fig09_15.mdl*)

```
>> subplot(221)
>> plot(t,60*N_ref,'k--',t,60*n,'k-')
>> axis([0 0.01 0 800])
```

```
>> title('Speed n and speed reference N-ref [RPM]', ...
        'FontSize',10)
>> subplot(223)
>> plot(t,IA_ref,'k--',t,iA,'k-')
>> axis([0 0.01 -500 1500])
>> title('Amature current iA and reference value ...
         IAref [A]','FontSize',10)
>> subplot(222)
>> plot(t,60*N_ref,'k--',t,60*n,'k-')
>> axis([0.095 0.11 450 550])
>> title('Speed n and speed reference ...
         N-ref [RPM]','FontSize',10)
>> subplot(224)
>> plot(t,IA_ref,'k--',t,iA,'k-')
>> axis([0.095 0.11 0 120])
>> title('Amature current iA and reference value ...
         IAref [A]','FontSize',10)
```

9.4 Quan sát trạng thái

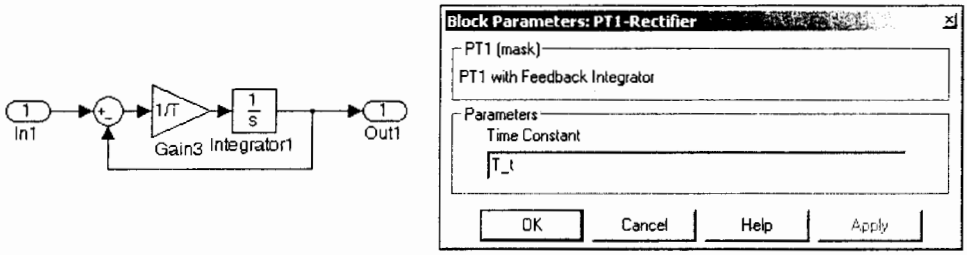
Vì các điều kiện công nghệ nhất định, hay cũng có thể vì lý do muốn giảm chi phí thiết bị, ta có ý định tiết kiệm khâu đo tốc độ quay nhưng lại vẫn muốn ĐC tốc độ quay của ĐCMC. Đại lượng ra duy nhất có thể đo chỉ còn là dòng phần ứng i_A. Vậy là ta sẽ phải thiết kế một khâu QS, phục vụ mục đích tính (chứ không đo) tốc độ quay.



***Hình 9.17** Mô hình SIMULINK của ĐCMC với khâu QS trạng thái (Fig09_17.mdl)*

Mô hình mô phỏng được giới thiệu ở hình 9.17: Bao gồm ĐCMC có thêm thiết bị chỉnh lưu (thay thế xấp xỉ bởi khâu *PT1-Rectifier*) và khối *State-Space* (đặc trưng cho khâu QS và được gán các ma trận trạng thái của ĐCMC làm tham số). Thêm vào đó, mạch điện phần ứng của ĐCMC (xem lại hình 9.2) vốn được mô phỏng bằng khối *Transfer Fcn*,. nay được thay bằng khối *PT1* (bản chất là hệ

con được đánh dấu có cửa sổ nhập tham số; xem mục 6.7.4). Việc thay thế đó nhằm bảo đảm tín hiệu ra của khối (dòng phần ứng i_A) đồng thời là một biến trạng thái của mô hình trạng thái thay thế ĐCMC.



Hình 9.18 *Khối PT1: Dưới dạng khâu tích phân có hồi tiếp và cửa sổ nhập tham số*

Để lập trị ban đầu, khai báo tham số mô phỏng và tạo mô hình LTI kiểu SS có tên *sysDCMotor*, ta soạn thảo *script* mang tên *Fig09_17i.m* với nội dung dưới đây:

```
%% Fig09_17i.m
%% Initialization file for zu Fig09_17.mdl

%% General simulation data
Tstop = 0.8 ;              % Stop time of simulation
step_max = 0.00001 ;       % Maximal step size
%% Rectifier data
T_t = 0.0001 ;             % Dead-time
%% Amature voltage puls
T_uAref = 0.4 ;            % Puls width              [s]
uA_ref  = 500/60 ;         % Reference voltage       [V]
%% Load torque step
T_MW = 0.40 ;              % Step time               [s]
MW   = 10  ;               % Step height             [1/s]
%% Data of DC motor
RA   = 0.250 ;             % Armature resistor       [Ohm]
LA   = 0.004 ;             % Armature inductance     [H]
TA   = LA / RA ;           % Armature time constant  [s]
PsiR = 0.04 ;              % Rated fluxs             [Vs]
J    = 0.012 ;             % Torque of inertia       [kg m^2]
kM   = 38.2 ;              % Motor constants
kE   = 2*pi*kM ;

%% Observer basic settings
obs = ss([],[0 0],[],[0 0]);    % Initialization of observer
obsx0 = [ 0 ];                  % Initial states of observer
[A,B,C,D]= linmod('Fig09_17'); % Extracting system matrices
sysDCMotor = ss(A,B,C,D);       % Generating SS-LTI-model
```

Để hiểu rõ trình tự thiết kế khâu QS ta sẽ phải phân tích ý nghĩa vài lệnh sau. Trước hết, muốn tuyến tính hóa mô hình SIMULINK bằng lệnh `linmod`, tất cả các biến và tham số phải có sẵn một giá trị cụ thể. Kể cả khối *State-Space*, là khối mà đến thời điểm này ta chưa hề tìm các ma trận trạng thái của ĐCMC. Chính vì vậy ta phải khai báo lập trị ban đầu cho các ma trận của `obs`, đồng thời định nghĩa một vector trạng thái ban đầu `obsx0`.

Để có thể xác định được các điểm cực của khâu QS cũng như tham số của ma trận phản hồi **L**, ta cần đến mô tả trạng thái của đối tượng với các biến vào **u**, các trạng thái **x** và các đầu ra **y**. Để tìm mô tả đó (đặc trưng bởi các ma trận `A`, `B`, `C` và `D`) ta dùng lệnh `linmod('Fig09_17')` và sau đó từ bốn ma trận `A`, `B`, `C` và `D` ta chuyển sang thành mô hình LTI dạng SS có tên `sysDCMotor`.

Tín hiệu vào của hệ thống là giá trị đặt cho tốc độ quay (dưới dạng điện áp) `uAref`; các trạng thái là dòng phần ứng `IA`, đầu ra (không tên) của khâu chỉnh lưu PT1 và tốc độ quay `n`; tín hiệu ra của hệ đồng thời là tín hiệu trạng thái `IA`.

Trình tự mô phỏng vẫn là: Trước hết gọi *Fig09_17i.m* tại *Command Windows* của MATLAB, sau đó chạy mô phỏng *Fig09_17.mdl*. Cần nhắc lại là ta phải lưu ý trình tự của các biến trạng thái sau khi tạo mô hình LTI-SS bằng `sysDCMotor = ss(A,B,C,D)`. Để kiểm tra trình tự ta có thể sử dụng lệnh `[size,x0,xstord] = modelname` (mục 6.6, đoạn nói về *Workspace I/O* và mục 7.3). Trong lệnh đó, `xstord` là tham số cất các biến trạng thái theo trình tự mà SIMULINK đã chọn. Áp dụng lệnh vào ví dụ trên ta thu được kết quả:

```
>> [obs,obsx0,xstord] = Fig09_17;
>> xstord
xstord =
    'Fig09_17/PT1-Amature/Integrator'
    'Fig09_17/Integrator'
    'Fig09_17/PT1-Rectifier/Integrator1'
```

Các biến trạng thái vậy là có trình tự: 1. Đầu ra của khâu tích phân trong khối *PT1-Amature* (dòng phần ứng i_A); 2. Đầu ra của khâu tích phân (tốc độ quay n); 3. Đầu ra của khâu tích phân trong khối *PT1-Rectifier* (điện áp đặt lên động cơ).

Vì khâu QS còn có thể được tìm thấy nhờ lệnh `estim`, khi ấy vector biến vào của khâu QS sẽ có hai thành phần: Biến vào của hệ thống `uAref` và biến ra của hệ thống dòng phần ứng `IA`. Vector biến ra của khâu QS là `yobs`, chứa các giá trị tính được của biến ra và các biến trạng thái của hệ (hình 3.28).

9.4.1 Khâu quan sát Luenberger

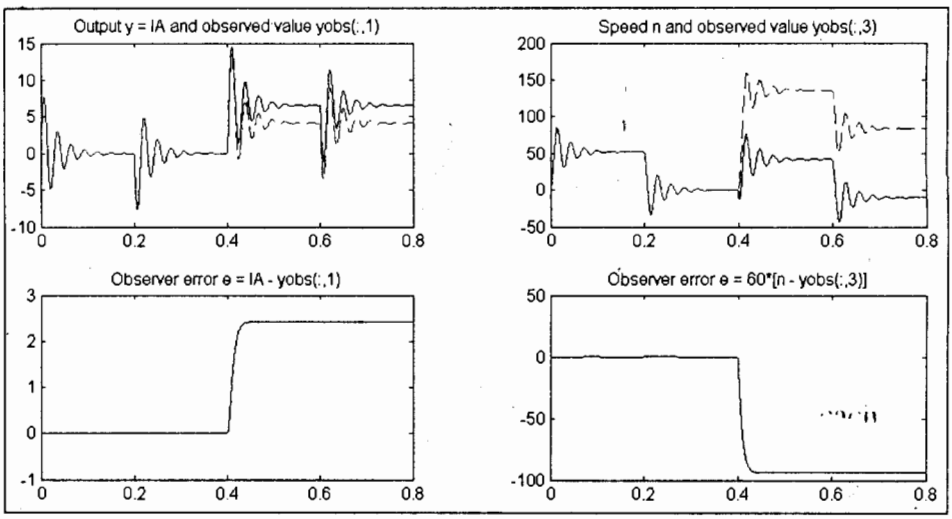
Để thiết kế khâu QS Luenberger ta sử dụng các phương pháp và lệnh đã được giới thiệu ở mục 3.4.3. Để tiện theo dõi ta viết lại lần nữa tại đây phương trình của khâu QS Luenberger đã có sai số QS (hình 3.27):

$$\dot{\hat{\mathbf{x}}} = (\mathbf{A} - \mathbf{LC})\hat{\mathbf{x}} + [\mathbf{B} - \mathbf{LD} \quad \mathbf{L}]\begin{bmatrix}\mathbf{u}\\ \mathbf{y}\end{bmatrix} \tag{9.16}$$

Việc tìm ma trận phản hồi **L** được thực hiện nhờ lệnh `place` tương tự ví dụ ở mục 3.4.4 (hình 3.31). Để chuẩn bị cho quá trình mô phỏng, ta bổ sung thêm cho *Fig09_17i.m* ba dòng lệnh dưới đây và cất thành *File* mới có tên *Fig09_19i.m*.

```
%% Luenberger observer
polo = 5*real(pole(sysDCMotor))+imag(pole(sysDCMotor))/5*i;
                        % Pole of observer
L = place(sysDCMotor.a',sysDCMotor.c',polo).';
                        % Estimating the feedback vector
obs = estim(sysDCMotor,L,1,1);
                        % Observer for LTI-SS-model
```

Trong đoạn lệnh bổ sung trên, trước hết ta xác định các điểm cực `polo` của khâu QS, sau đó giao `polo` cùng với các ma trận chuyển vị `sysDCMotor.a'` và `sysDCMotor.c'` cho lệnh `place` để tính `L`. Cuối cùng, lệnh `estim` sẽ trên cơ sở mô hình LTI-SS `sysDCMotor` và `L` để tìm khâu QS `obs`. Kết quả quan sát bằng khâu Luenberger `obs` được giới thiệu ở hình 9.19.



Hình 9.19 *Khâu QS Luenberger: **(trên)** Giá trị đo (nét liền) và giá trị tính được (nét đứt) của dòng và tốc độ quay; **(dưới)** Sai số quan sát.*

Sau khi mô phỏng, hình 9.19 đã được vẽ bởi chùm lệnh MATLAB dưới đây:

```
>> subplot(221)
```

```
>> plot(t,IA,'k-',t,yobs(:,1),'b--')
>> title('Output y = IA and observed value ...
         yobs(:,1)','FontSize',10)
>> subplot(223)
>> plot(t,IA-yobs(:,1),'k-')
>> title('Observer error e = IA - yobs(:,1)','FontSize',10)
>> subplot(222)
>> plot(t,60*n,'k-',t,60*yobs(:,3),'b--')
>> title('Speed n and observed value ...
         yobs(:,3)','FontSize',10)
>> subplot(224)
>> plot(t,60*(n-yobs(:,3)),'k-')
>> title('Observer error e = 60*[n - ...
         yobs(:,3)]','FontSize',10)
```

Qua sai số QS ở hình 9.19 (hai hình dưới) ta dễ dàng nhận thấy rằng: Ban đầu (khi ĐCMC làm việc ở chế độ không tải), giá trị thực (nét liền) và giá trị tính được (nét đứt) hoàn toàn trùng với nhau. Tuy nhiên, sau khi có thay đổi phụ tải (đột biến mômen cản), đã xuất hiện và tồn tại một sai số QS đáng kể.

Điều này có xuất sứ từ cấu trúc của khâu QS: Một mặt, mô hình LTI-SS sử dụng để thiết kế đã bỏ qua, coi phụ tải (được coi là nhiễu) là 0. Mặt khác, vector sai số QS lại chỉ được phản hồi tỷ lệ qua **L** và do đó khâu QS sẽ không có khả năng khử sai số QS tĩnh. Để giải quyết vấn đề ta sẽ phải sử dụng khâu QS nhiễu ở mục tiếp theo.

9.4.2 Khâu quan sát nhiễu (QS phụ tải)

Để triệt tiêu sai số QS tĩnh, ta phải mở rộng khâu QS (**D** = **0**, vì ĐCMC có đặc điểm khâu quán tính) thêm nhánh phản hồi như ở hình 9.20.

Sai số quan sát $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$ được đưa qua ma trận điều khiển **K**, khâu tích phân và ma trận đầu vào nhiễu **F** tới cửa $\dot{\hat{\mathbf{x}}}$. Khâu tích phân có tác dụng tích lũy sai lệch tĩnh (tác dụng hiệu chỉnh) tới chừng nào sai lệch đó bị triệt tiêu. Sau khi thế sai lệch $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$, phương trình trạng thái của khâu quan sát nhiễu sẽ có dạng:

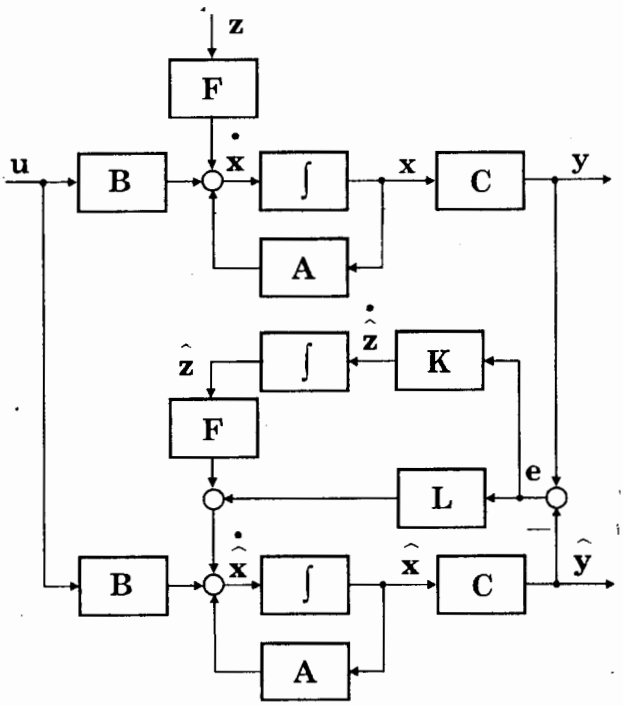
$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}\mathbf{e} + \mathbf{F}\hat{\mathbf{z}} = (\mathbf{A} - \mathbf{L}\mathbf{C})\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}\mathbf{y} + \mathbf{F}\hat{\mathbf{z}} \quad (9.17)$$

$$\dot{\hat{\mathbf{z}}} = \mathbf{K}\mathbf{e} = -\mathbf{K}\mathbf{C}\hat{\mathbf{x}} + \mathbf{K}\mathbf{y} \quad (9.18)$$

$$\hat{\mathbf{y}} = \mathbf{C}\hat{\mathbf{x}} \quad (9.19)$$

Viết lại các phương trình trạng thái trên dưới dạng mô hình ma trận quen biết ta có:

$$\begin{bmatrix} \dot{\hat{\mathbf{x}}} \\ \dot{\hat{\mathbf{z}}} \end{bmatrix} = \begin{bmatrix} \mathbf{A}-\mathbf{LC} & \mathbf{F} \\ -\mathbf{KC} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}} \\ \hat{\mathbf{z}} \end{bmatrix} + \begin{bmatrix} \mathbf{B} & \mathbf{L} \\ \mathbf{0} & \mathbf{K} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{y} \end{bmatrix}$$
$$\hat{\mathbf{y}} = \begin{bmatrix} \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}} \\ \hat{\mathbf{z}} \end{bmatrix} \tag{9.20}$$



Hình 9.20 Khâu quan sát nhiễu

Để mô phỏng ta sẽ biến đổi vector biến ra QS được $\hat{\mathbf{y}}$, sao cho đồng thời cả các biến trạng thái $\hat{\mathbf{x}}$ và các biến nhiễu $\hat{\mathbf{z}}$ đều được cất trong yobs:

$$\hat{\mathbf{y}}_{obs} = \begin{bmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}} \\ \hat{\mathbf{z}} \end{bmatrix} \tag{9.21}$$

Như các ví dụ trước, ta thực hiện khai báo lập trị ban đầu và tính các ma trận của khâu QS bằng *script* có tên ***Fig09_21i.m*** với nội dung sau đây:

```
%% Initialization file for simulation
%% with disturbance observer

Fig09_17i;                 % Initialization of motor
                           % and LTI-model data
F = [ 0 -1/(2*pi*J) 0 ]'; % Disturbance matrix

%% Luenberger observer: Placement of observer pole
polo = 5*real(pole(sysDCMotor))+imag(pole(sysDCMotor))/5*i;
%% Estimating the state feedback
L = place(sysDCMotor.a',sysDCMotor.c',polo).';
%% Feedback coefficient
K = 100;

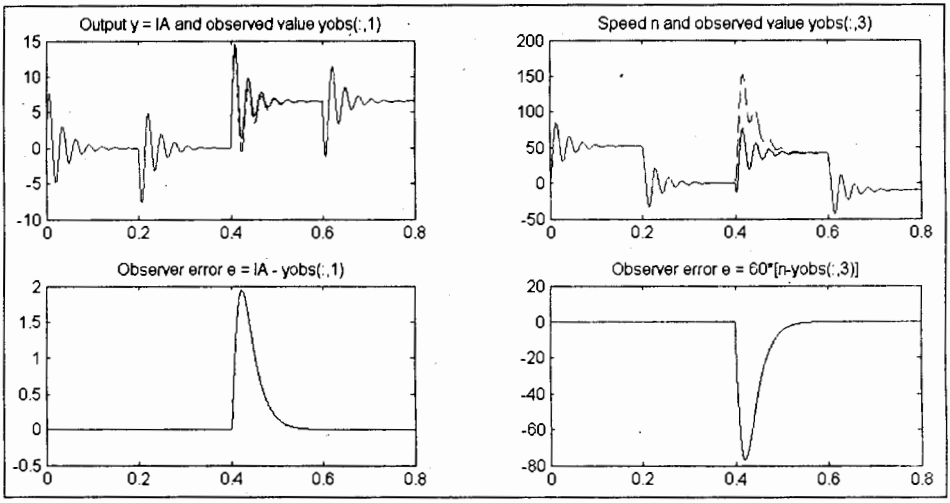
%% Computing the matrices of disturbance observer
clear obs
obs.a = [ sysDCMotor.a-L*sysDCMotor.c F ; ...
          -K*sysDCMotor.c 0 ];
obs.b = [ sysDCMotor.b L ; 0 K ];
obs.c = [ sysDCMotor.c 0 ; eye(size(obs.a)) ];
obs.d = zeros(size(obs.c*obs.b));
%% LTI-SS-Model as observer.
obs = ss(obs.a,obs.b,obs.c,obs.d);
```

Dễ dàng thấy rằng, *File Fig09_21i.m* trên chuẩn bị số liệu cho việc mô phỏng bằng sơ đồ SIMULINK ở hình 9.17: Sau khi sử dụng *Fig09_17i.m* để khai báo tham số của ĐCMC và tìm mô hình `sysDCMotor` (mô hình LTI-SS) của động cơ, ta xác định cực `polo` để tìm vector phản hồi trạng thái `L` của khâu Luenberger thông thường. Để mô phỏng với khâu QS nhiễu: Ta khai báo thêm ma trận đầu vào của nhiễu `F`, hệ số hồi tiếp `K` và thay thế các ma trận của khâu QS Luenberger (khối *State-Space*: hình 9.17) bởi các ma trận mới tính theo các công thức (9.20), (9.21).

Tại dòng khai ma trận đầu vào của phụ tải `F = [ 0 -1/(2*pi*J) 0 ]'` bạn đọc cần lưu ý: Khi khảo sát sơ đồ ở hình 9.17 (trang 308) ta đã hỏi trình tự của các biến trạng thái bằng lệnh:

```
>> [obs,obsx0,xstord] = Fig09_17;
>> xstord
xstord =
    'Fig09_17/PT1-Amature/Integrator'
    'Fig09_17/Integrator'
    'Fig09_17/PT1-Rectifier/Integrator1'
```

và nhờ đó ta biết: Tốc độ quay (nơi tín hiệu phụ tải can thiệp vào hệ thống) được SIMULINK đặt ở vị trí thứ hai. Vì vậy, ta phải khai điểm can thiệp đó ở vị trí thứ hai của vector nhiễu `F`. Kết quả mô phỏng được giới thiệu ở hình 9.21.

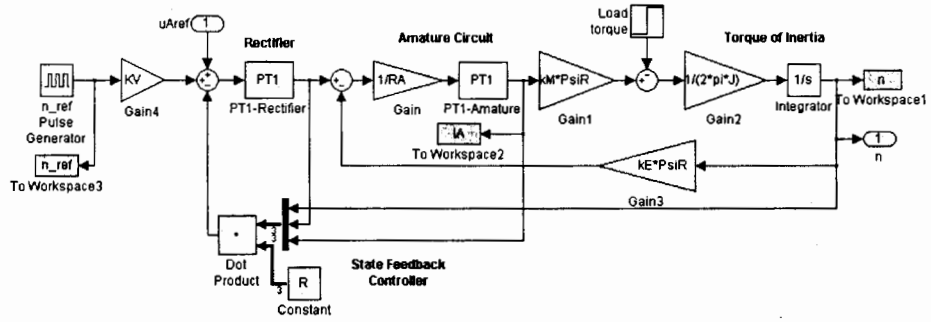


Hình 9.21 *Khâu QS nhiễu: Giá trị đo được (nét liền) và tính được (nét đứt), sai số QS (hàng dưới). Giá trị dòng đo, dòng tính được và sai số (cột trái).. Giá trị tốc độ quay đo, dòng tính được và sai số (cột trái).*

Khác hẳn với kết quả mô phỏng ở hình 9.19, hình 9.21 cho ta thấy rất rõ tác dụng của thành phần tích phân I[1] trong khâu QS nhiễu: Sai số QS đã được triệt tiêu hoàn toàn (hình 9.21, hàng dưới) sau khi xuất hiện nhiễu (đóng mômen tải) chừng 0,1s.

9.5 Điều khiển trạng thái sử dụng khâu quan sát trạng thái

Trước hết ta bắt đầu bằng việc thiết kế và thực hiện hệ thống ĐC tốc độ quay cho ĐCMC trên không gian trạng thái như hình 9.22.



Hình 9.22 *Mô hình ĐK tốc độ quay của ĐCMC trên không gian trạng thái*

[1] Thành phần I trong nhánh hồi tiếp sai số QS cũng có thể được tính ghép vào **L** từ trước

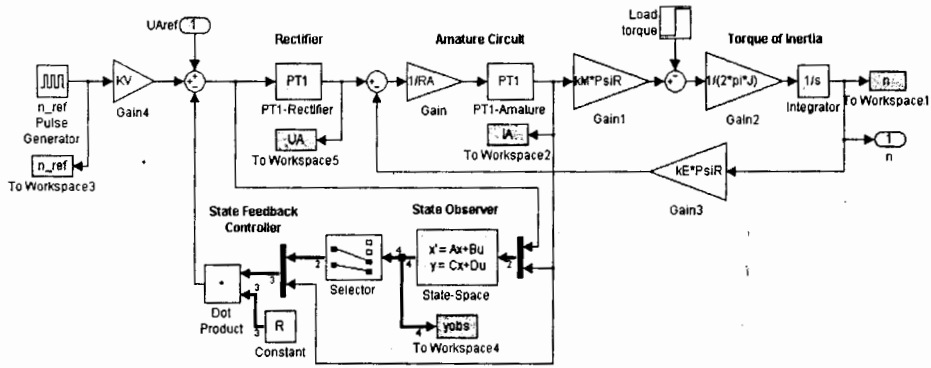
Đặc điểm chính của sơ đồ ĐK trạng thái ở hình 9.22 là: Việc phản hồi các trạng thái n, UA và IA (ký hiệu của sơ đồ SIMULINK) qua vector phản hồi R và việc phối hợp đại lượng chủ đạo (giá trị đặt) qua hệ số KV. Vị trí các điểm cực của hệ có phản hồi trạng thái cần được chọn với độ tắt dần hợp lý. Ví dụ: Dịch nhiều lần sang trái để tăng động học của hệ có phản hồi so với khi chưa có phản hồi. Hệ số phối hợp giá trị chủ đạo được tính theo:

$$K_V = -\frac{1}{\mathbf{C}(\mathbf{A}-\mathbf{B}\mathbf{R})^{-1}\mathbf{B}} \tag{9.22}$$

Hệ số K_V là kết quả thu được từ luật ĐK sau đây (**w** là vector giá trị đặt, giá trị chủ đạo):

$$\mathbf{u} = K_V\mathbf{w} - \mathbf{R}\hat{\mathbf{x}} \tag{9.23}$$

Trong ví dụ mô phỏng tiếp theo (hình 9.23), phần khai báo các tham số mô phỏng chung được cất trong *script* có tên *Fig09_23i.m*. Các tham số của ĐCMC đã được cất trong *Fig09_01i.m*.



***Hình 9.23** Hệ thống ổn định tốc độ quay của ĐCMC sử dụng khâu QS và ĐC trạng thái*

Trước khi mô phỏng hệ thống ĐK có sử dụng khâu QS trạng thái (khâu QS Luenberger thông thường hay khâu QS nhiễu) ta sẽ phải gọi *script* *Fig09_23LOi.m* hoặc *Fig09_23DOi.m* để tính các ma trận của khâu QS đã chọn. Hai *scripts* này bao giờ cũng bắt đầu bằng việc gọi hai *scripts* con khác là *Fig09_01i.m*, *Fig09_23i.m*. Trong *Fig09_23i.m*, ta quan tâm trước hết tới các dòng lệnh dưới đây: Giống như ví dụ ở mục 9.4, ta phải dùng lệnh `linmod` tuyến tính hóa mô hình SIMULINK ở hình 9.17 để thu được mô hình LTI-SS của đối tượng có tên `sysObs`, làm cơ sở thiết kế khâu QS `obs`:

```
%% Generating the SS-LTI model for obeserver design
obs       = ss([],[0 0],[],0) ; % Initial observer states
obsx0     = [ 0 ] ;             % Observer start values
[A,B,C,D] = linmod('Fig09_17'); % System matrices
sysObs = ss(A,B,C,D);           % SS-LTI model
```

Tiếp theo, để có thể thiết kế vector phản hồi trạng thái **R** và hệ số phối hợp đầu vào K_V, ta tuyến tính hóa sơ đồ ở hình 9.22 bằng lệnh `linmod` và thu được mô hình LTI-SS có tên `sysCon`:

```
%% Generating the SS-LTI model for state controller design
R          = [ 0 0 0 ] ;          % Switch-off the feedback
KV         = 1 ;                  % Prefactor start values
[A,B,C,D] = linmod('Fig09_22'); % System matrices
sysCon     = ss(A,B,C,D);         % SS-LTI model
```

Đến đây, rất có thể nẩy sinh câu hỏi: Tại sao để thiết kế khâu QS ta lại sử dụng sơ đồ ở hình 9.17, còn thiết kế khâu ĐC trạng thái lại phải dùng hình 9.22. Quan sát kỹ ta thấy: Hình 9.17 chỉ khai báo với SIMULINK 1 *Inport* là điện áp `uAref` và 1 *Outport* là dòng phần ứng `IA`, còn hình 9.22 có 1 *Inport* là điện áp `uAref` và 1 *Outport* là tốc độ quay `n`. Tại mục 7.2 ta đã biết vai trò của các khối *Inport* và *Outport* khi sử dụng lệnh `linmod`, mô hình LTI-SS của cùng một đối tượng được tạo ra hoàn toàn phụ thuộc vào vị trí (tức là vị trí đầu vào, đầu ra của hệ) của hai khối loại đó. Khi thiết kế khâu QS ta xuất phát từ giả thiết: Không đo tốc độ quay, chỉ đo dòng phần ứng, điều này phù hợp với cấu trúc ở hình 9.17. Khi thiết kế khâu ĐC ta lại có xuất phát điểm khác: Đại lượng mà ta muốn ĐC ổn định chính là tốc độ quay `n` (không cần biết tín hiệu `n` từ đâu tới, từ khâu QS hay khâu đo) và hình 9.22 sẽ là cấu trúc phù hợp để tìm mô hình đối tượng.

Điều vừa diễn giải cũng minh họa thêm *tính độc lập*[1] *giữa hai quá trình thiết kế khâu QS và khâu ĐC*. Chính vì vậy, ngoài hai đoạn mã lệnh trên, *script Fig09_23i.m* còn có đoạn thiết kế khâu ĐC trạng thái dưới đây (chung cho cả hai trường hợp sử dụng khâu QS Luenberger hay khâu QS nhiễu):

```
% State controller design: Placement controller poles
polc = 5*real(pole(sysCon))+imag(pole(sysCon))/5*i
% Computing the feedback vector
R    = place(sysCon.a,sysCon.b,polc)
KV    = -1/(sysCon.c*inv(sysCon.a-sysCon.b*R)*sysCon.b)
```

Cách sử dụng lệnh `place` để gán cực mong muốn và từ đó thiết kế vector `R` phản hồi trạng thái đã được đề cập đến rất kỹ tại mục 3.4.4. Trong cấu trúc ĐK ở hình 9.23, khối *Selector* có nhiệm vụ tách ra khỏi vector các giá trị QS `yobs` hai tín hiệu trạng thái `n` và `UA`, tín hiệu trạng thái `IA` còn lại là tín hiệu đo được, vì vậy ta lấy `IA` trực tiếp từ mô hình đối tượng:

```
sel_out = [3 4] ;          % 2 Selector output
sel_in  = size(obs,1) ;    % Number of selector inputs
```

[1] *Separation Theorem*: Định lý về tính độc lập giữa hai quá trình thiết kế khâu QS và khâu ĐK trạng thái.

Cuối cùng, để mô phỏng sơ đồ SIMULINK ở hình 9.23 (ĐC ổn định tốc độ quay trên không gian trạng thái, sử dụng khâu QS Luenberger hay khâu QS nhiễu) ta còn phải tiến hành thiết kế khâu QS bằng cách gọi 1 trong 2 *scripts* dưới đây:

- Khâu QS Luenberger: *Fig09_23LOi.m*

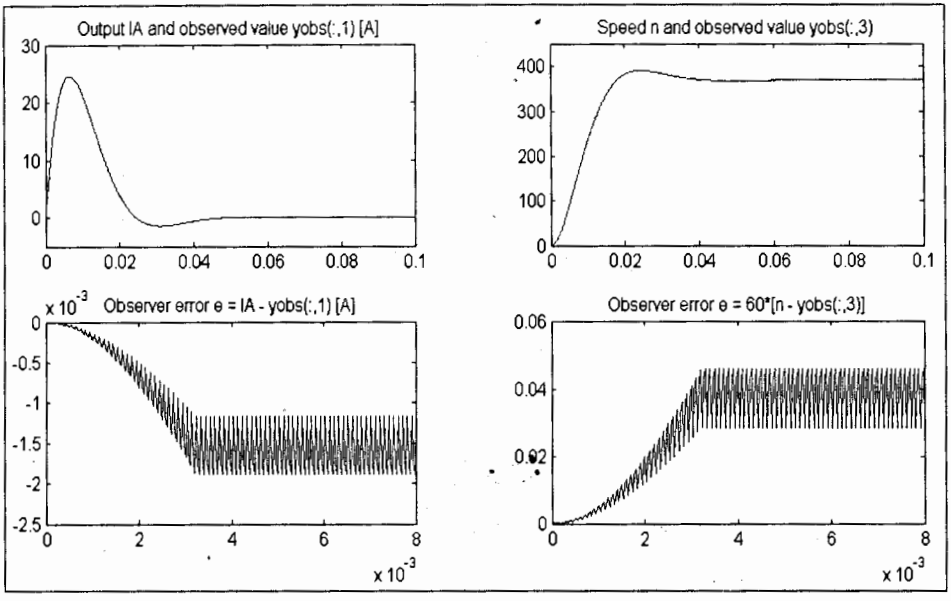
```
%% Simulation of Fig09_23 with Luenberger observer
%% General motor and simulation data
Fig09_01i;
Fig09_23i;
%% Design of Luenberger observer
polo = 100*real(pole(sysObs))+imag(pole(sysObs))/100*i
                              % Observer poles
L    = place(sysObs.a',sysObs.c',polo).'
                              % Computing the feedback vector
obs  = estim(sysObs,L,1,1) % Observer based on SS-LTI model
```

- Khâu QS nhiễu (phụ tải): *Fig09_23DOi.m*

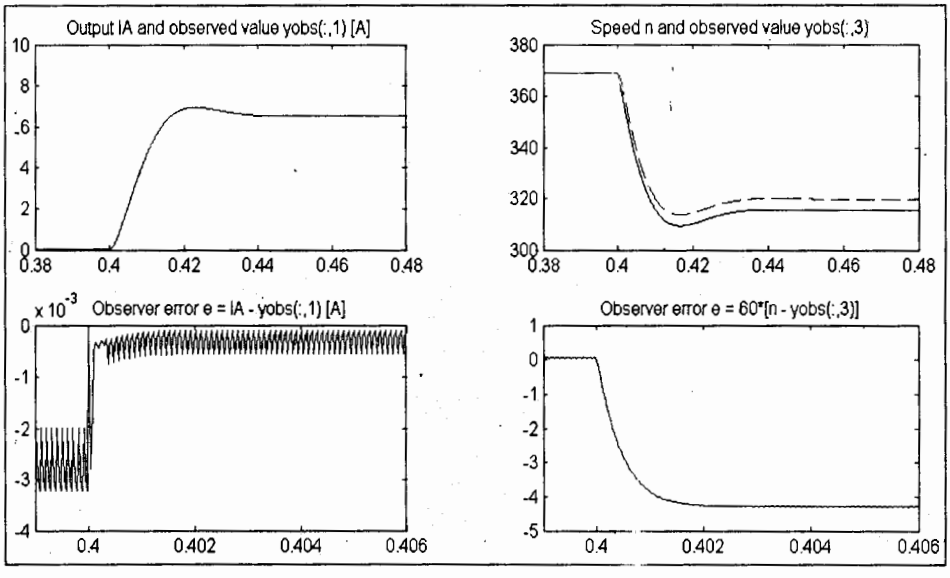
```
%% Simulation of Fig09_23 with disturbance observer
%% General motor and simulation data
Fig09_01i;
Fig09_23i;
%% Design of disturbance observer
F = [ 0 -1/(2*pi*J) 0 ]' ; % Disturbance matrix
polo = 60*real(pole(sysObs))+imag(pole(sysObs))/60*i
                              % Observer poles
L = place(sysObs.a',sysObs.c',polo).'
                              % Computing of feedback vector
K = 50 ;                      % Feedback coefficient
%% Computing of observer matrices
clear obs
obs.a = [ sysObs.a-L*sysObs.c F ; -K*sysObs.c 0 ] ;
obs.b = [ sysObs.b L ; 0 K ] ;
obs.c = [ sysObs.c 0 ; eye(size(obs.a)) ] ;
obs.d = zeros(size(obs.c*obs.b)) ;
obs   = ss(obs.a,obs.b,obs.c,obs.d)
                              % Observer based on SS-LTI model
```

Hình 9.24 và 9.25 giới thiệu kết quả mô phỏng, gồm có đáp ứng dòng IA và tốc độ quay n khi có bước nhẩy giá trị đặt tốc độ ở đầu vào, đáp ứng nhiễu cũng của dòng IA và tốc độ quay n khi có đột biến phụ tải.

Từ hai hình 9.24 và 9.25 ta thấy rất rõ rằng hệ thống ĐC tốc độ quay của ĐCMC trên không gian trạng thái (không đo tốc độ quay mà sử dụng khâu QS) có khả năng hoạt động tốt và không có vấn đề gì lớn. Tuy nhiên, việc chọn thuật toán tích phân *ode45* đã làm cho sai số QS không được triệt tiêu hoàn toàn.



Hình 9.24 *ĐK trạng thái sử dụng khâu QS Luenberger: Đáp ứng bước nhẩy của dòng phần ứng* `IA` *và tốc độ quay* `n` *(hàng trên), sai số QS dòng và tốc độ quay (hàng dưới)*



Hình 9.25 *ĐK trạng thái sử dụng khâu QS Luenberger: Đáp ứng nhiễu phụ tải của dòng phần ứng* `IA` *và tốc độ quay* `n` *(hàng trên), sai số QS dòng và tốc độ quay (hàng dưới)*

Qua hình 9.25 dễ dàng thấy rằng khâu QS Luenberger đã không triệt tiêu sai số QS khi có nhiễu (có phụ tải). Hiện tượng này đã được giải thích kỹ qua ví dụ ở hình 9.21 và cũng tương tự: Có thể sử dụng khâu QS nhiễu (thiết kế bởi *Fig09_23DOi.m*) để khử sai số tĩnh đó.

9.6 Tóm tắt nội dung chương 9

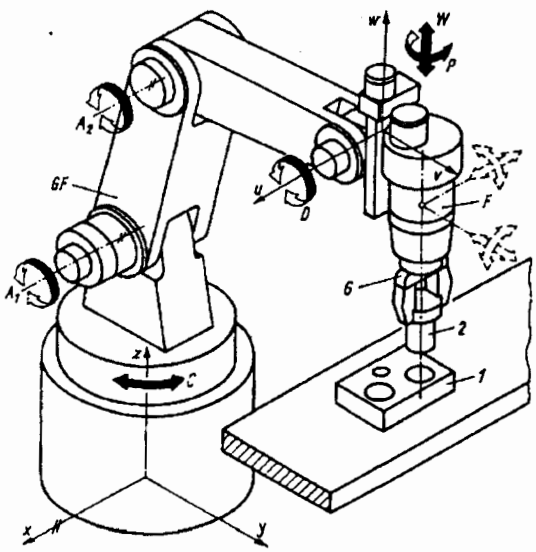
Sau khi đã nghiên cứu kỹ chương 9, người đọc cần nắm vững các nội dung sau đây:

1. Soạn thảo các *scripts* phục vụ khai báo tham số, lập trị ban đầu cho mo hình SIMULINK.
2. Xây dựng mô hình SIMULINK sử dụng các khối có sẵn trong thư viện.
3. Trao đổi (cất vào, lấy ra) dữ liệu, biến với MATLAB *Workspace*.
4. Tuyến tính hóa mô hình SIMULINK bằng lệnh `linmod`.
5. Sắp xếp và sử dụng đúng trật tự các biến trạng thái của mô hình SIMULINK đã tuyến tính hóa.
6. Sử dụng cửa sổ LTI-*Viewer*.

10 Thư viện mô hình máy điện quay

10.1 Khái quát về Motion Control Blockset

Motion Control Blockset là một thư viện con có thể cài đặt xen vào thư viện mẹ SIMULINK, bao gồm các khối hỗ trợ mô phỏng, khảo sát các hệ thống truyền động đơn lẻ cũng như các hệ thống nhiều động cơ truyền động, tạo nên các chuyển động phức hợp. Những hệ thống phức hợp nhiều động cơ truyền động ấy được ta nghiên cứu chế ngự trong một lĩnh vực mới là *điều khiển (các quá trình) chuyển động*[1] (*Motion Control*). Các ví dụ về tay máy nhiều bậc tự do (hình 10.1), hay thiết bị gia công kim loại nhiều trục CNC là những minh chứng rõ ràng nhất về tính phức hợp của các hệ thống ĐK như vậy.



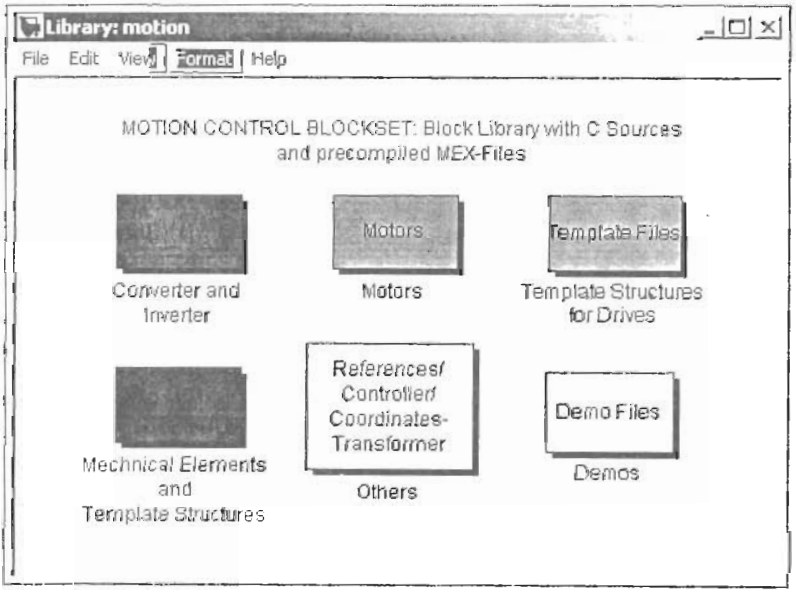
Hình 10.1 *Tay máy công nghiệp: Ví dụ điển hình về một hệ thống điều khiển chuyển động phức hợp*

Cho đến nay, việc nghiên cứu các hệ thống ĐKCĐ đã thường diễn ra một cách chưa đầy đủ và một chiều, ví dụ: Các kỹ sư truyền động điện thường thỏa

[1] Điều khiển chuyển động: ĐKCĐ

mãn với việc mô hình hóa và mô phỏng hệ thống ĐK động cơ, còn các kỹ sư về tay máy nhiều khi chỉ quan tâm tới việc viết phương trình động học mô tả qũy đạo chuyển động và các yếu tố cơ học đi kèm chứ không hề biết lực/mômen từ đâu sinh ra, với đặc tính (quality) và module (quantity) như thế nào. Để việc nghiên cứu, khảo sát các hệ thống chuyển động trở nên đầy đủ, hoàn thiện, cần phải có công cụ hỗ trợ mô hình hóa và mô phỏng hệ thống với trọn vẹn cả hai mảng điện và cơ đó. Đó chính là động cơ đã thôi thúc tác giả nỗ lực xây dựng bộ công cụ *Motion Control Blockset*[1].

Hiện tại công cụ MCB chưa hoàn thiện, nó bao gồm nhiều khối do tác giả và đồng nghiệp (hình 10.2) tự xây dựng để sử dụng riêng trong quá trình nghiên cứu & phát triển lĩnh vực ĐK truyền động/chuyển động. Từ năm 2000 bắt đầu được bổ sung thêm một loạt khối mới (hình 10.3) trong khuôn khổ các đề tài tốt nghiệp đại học, luận văn tốt nghiệp cao học. Chương này giới thiệu với bạn đọc một số khối của MCB và vài ví dụ ứng dụng chúng. Như bạn đọc sẽ thấy, các khối đó hoàn toàn được xây dựng trên cơ sở kiến thức cơ bản đã được giới thiệu ở các chương trước. Vì vậy mục đích của phần C chỉ là: Truyền tới người đọc cảm giác tự tin khi sử dụng các kiến thức cơ bản về MATLAB & SIMULINK để mô hình hóa và mô phỏng hệ thống theo ý tưởng của riêng mình.

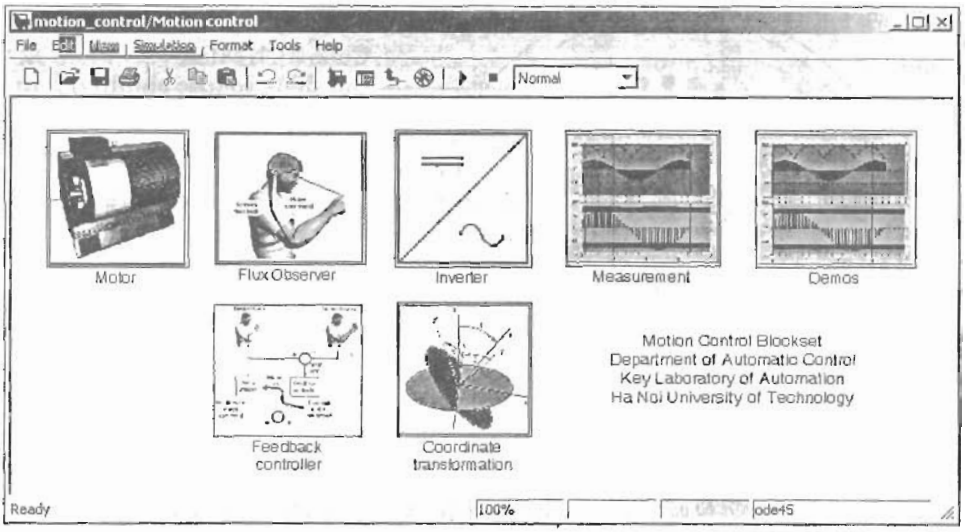


Hình 10.2 Thư viện MCB giai đoạn trước 1999: Thực hiện tại trường Đại học kỹ thuật Dresden (Technische Universität Dresden)

Phần C bao gồm bốn phần chính (ba chương 10-12): Phần **mô hình hóa và xây dựng *các khối máy điện*** (phần tử sản sinh mômen/lực, để tạo nên chuyển

[1] Motion Control Blockset: MCB

động), phần dành cho *các thiết bị biến đổi điện năng* (phần tử điều khiển máy điện), phần dành cho *các thiết bị cơ* (phần tử truyền động cơ học) và cuối cùng là một vài *ví dụ minh họa* (*Demos*).



Hình 10.3 Thư viện MCB giai đoạn từ sau 1999: Thực hiện tại trường Đại học Bách khoa Hà Nội

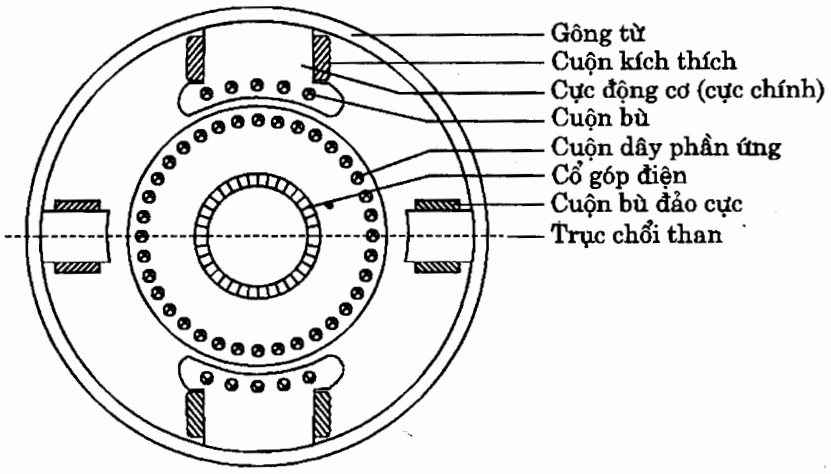
Hy vọng đến một thời điểm nào đó, các kết quả ở hai hình 10.2 và 10.3 đã đủ chín muồi để có thể tập hợp lại thành một *Blockset* thống nhất, cung cấp (dưới hình thức phù hợp) cho các kỹ sư điều khiển tự động đang làm việc trên lĩnh vực *Motion Control*.

10.2 Máy điện một chiều kích thích độc lập

Trước khi tiến hành mô hình hóa máy điện một chiều (MĐMC, xem mục 9.1) ta hãy tóm tắt lại phương thức hoạt động của máy. Về nguyên lý chi tiết, bạn đọc sẽ phải tra cứu tại các tài liệu chuyên dụng. Trong mục này, người viết cố tình viết MĐMC, chứ không viết ĐCMC như mục 9.1. Động cơ chỉ là một trong hai chế độ vận hành (chế độ máy phát hay chế độ động cơ) của máy, vì vậy cách viết MĐMC là chuẩn xác hơn.

Mômen quay của MĐMC được sinh ra trên cơ sở tác động qua lại giữa từ thông Stator (từ thông của *cuộn kích thích*) với từ thông Rotor (từ thông của *cuộn dây phần ứng*). Trên bề mặt Rotor (bề mặt *cuộn dây phần ứng*) có tác động của lực tiếp tuyến gây chuyển động quay của Rotor (của *cuộn dây phần ứng*). Có thể hình dung ra lực đẩy đó khi ta đưa hai cực giống nhau của hai nam châm lại gần nhau. Khi Rotor quay, tại *cổ góp điện* sẽ xảy ra chuyển mạch sang

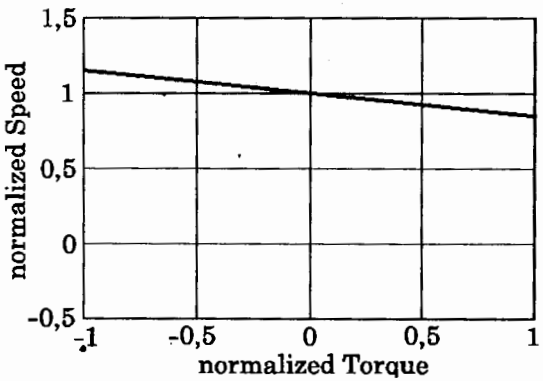
cuộn dây phần ứng tiếp theo, và lại xuất hiện lực đối với cuộn dây phần ứng mới. Quá trình chuyển mạch và tạo lực cứ liên tục tiếp diễn như vậy, tạo nên chuyển động quay của MĐMC (hình 10.4).



Hình 10.4 Nguyên lý cấu tạo của máy điện một chiều kích thích độc lập

Cuộn bù có tác dụng khử méo dạng từ thông phân bố trên bề mặt Rotor do ảnh hưởng của cuộn dây phần ứng. Đối với MĐMC có ĐK, sử dụng trong các hệ truyền động có đòi hỏi chất lượng tốt, cuộn dây phần ứng và cuộn kích thích được nuôi bởi hai nguồn điện (có ĐK) độc lập. Tại đây ta sẽ không đề cập tới loại MĐMC có kích thích nối tiếp.

Đặc tính cơ (đặc tính tĩnh) của MĐMC có thể được mô tả bởi đồ thị tốc độ – mômen (đồ thị $n - m_M$) như ở hình 10.5. Khi tăng phụ tải, tốc độ quay của máy sẽ giảm ở chế độ động cơ nhưng lại tăng ở chế độ máy phát theo một đường thẳng tuyến tính.

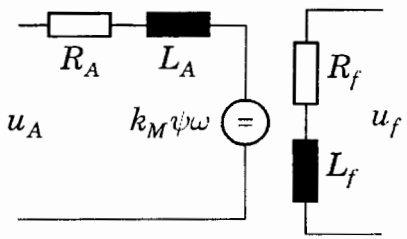


Hình 10.5 Đặc tính cơ của máy điện một chiều kích thích độc lập

Có thể mô tả đặc tính cơ như hình 10.5 bởi phương trình đường thẳng:

$$\omega = \frac{u_A}{k_M \psi} - \frac{R_A}{(k_M \psi)^2} m_M \tag{10.1}$$

Biểu thức đầu tiên của (10.1) ứng với tốc độ quay không tải, biểu thức thứ 2 mô tả quan hệ phụ thuộc bậc nhất của tốc độ quay vào mômen tải. Nếu muốn mô tả đầy đủ đặc điểm động học, ta sẽ phải bổ sung vào mô hình sơ đồ thay thế (hình 10.6) của mạch điện MĐMC. Trong sơ đồ đó ta đã không thể hiện các cuộn bù từ thông đỉnh cực, cuộn bù đảo cực. Tác dụng của hai cuộn dây đó đã được bỏ qua khi ta giả thiết: Hai cuộn dây phần ứng và kích thích không có tác động tương hỗ về từ (độc lập lý tưởng). Ta cũng đã bỏ qua các hiện tượng đánh lửa chổi than, nhiễu cổ góp (đã được hạn chế nhờ cuộn bù đảo cực).



Hình 10.6 *Sơ đồ mạch điện thay thế của MĐMC kích thích độc lập*

Khi mô hình hóa MĐMC bằng sơ đồ SIMULINK ta đã phải chấp nhận các giả thiết dưới đây. Tuy nhiên, cần nhấn mạnh rằng các mô hình đó đã phản ánh đầy đủ hầu hết các chế độ vận hành, và có hiệu lực khi khảo sát cả chế độ động lẫn chế độ tĩnh. Các phương trình mô tả có hiệu lực cả cho chế độ tín hiệu biên độ lớn (ví dụ: Khi xảy ra bước nhảy giá trị đặt).

- Các cuộn dây được coi là bố trí tập trung.
- Tham số của mô hình là hằng (ví dụ: Không phụ thuộc nhiệt độ).
- Bỏ qua hiện tượng bão hòa vật liệu từ. Tuy nhiên, có thể dễ dàng cài đặt khâu bão hòa bằng cách sử dụng hàm xấp xỉ.
- Bỏ qua tổn hao dòng quẩn trong vật liệu từ.
- Bỏ qua tổn hao ma sát (ví dụ: Của quạt làm mát, của ổ bi. Các tổn hao này cần được xét đến trong mô hình phần cơ).

Máy điện được mô tả bởi hệ phương trình đã giới thiệu ở mục 9.1, để tiện theo dõi ta viết lại một lần nữa tại đây:

- *Điện áp phần ứng*: $$u_A = k_M \psi \omega + R_A i_A + L_A \frac{di_A}{dt} \tag{10.2}$$
- *Mômen quay*: $$m_M = k_M \psi i_A \tag{10.3}$$

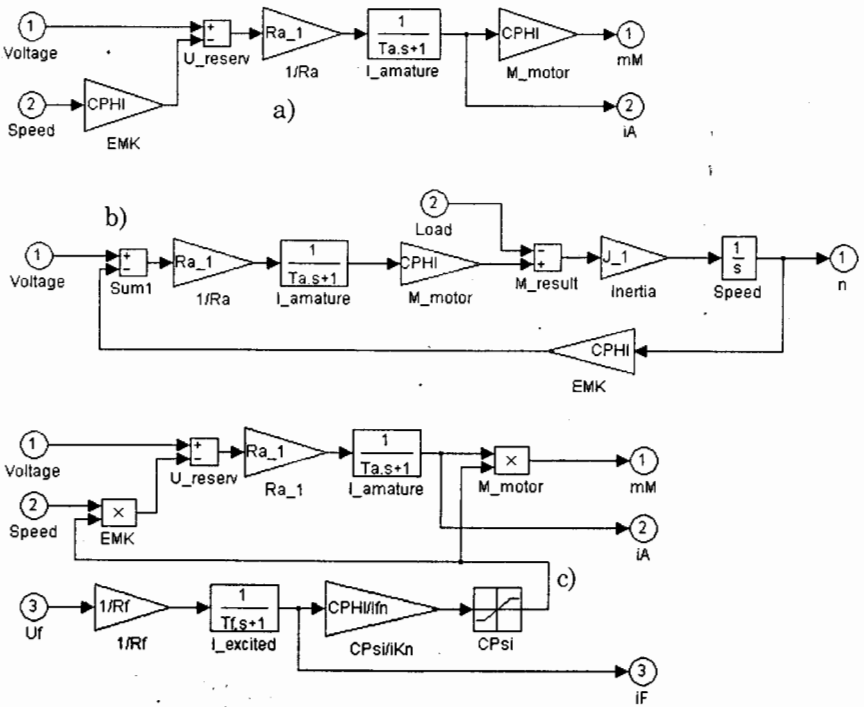
- *Phương trình chuyển động:* $$\frac{dn}{dt} = \frac{1}{2\pi J}(m_M - m_T) \quad (10.4)$$

Ý nghĩa của các ký hiệu sử dụng trong (10.2), (10.3) và (10.4) đã được giải thích ở mục 9.1. Ngoài ra, ta có thêm phương trình mạch kích từ như sau:

$$u_f = R_f i_f + L_f \frac{di_f}{dt} \quad (10.5)$$

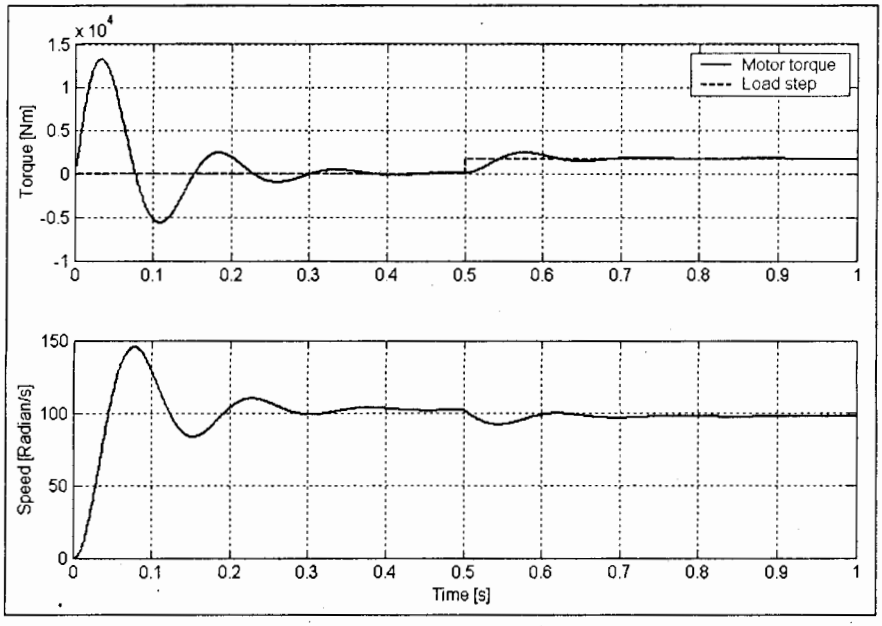
Với u_f và i_f là điện áp và dòng, R_f và L_f là điện trở và điện cảm của mạch điện kích thích. Trong thư viện ta có sẵn ba mô hình như sau:

- Mô hình ở hình 10.7a: Cài đặt hai phương trình (10.2), (10.3), phục vụ mô phỏng các bài toán đơn giản, tiết kiệm thời gian.
- Mô hình ở hình 10.7b: Giống hình 10.7a, nhưng được bổ sung thêm phương trình chuyển động (10.4). Cho phép mô phỏng có bổ sung thêm tác dụng của mômen quán tính nhưng chưa có đến hệ thống cơ được ghép phía phụ tải.
- Mô hình ở hình 10.7c: Giống hình 10.7a, nhưng được bổ sung thêm phương trình kích từ (10.5). Rất có ý nghĩa khi khảo sát các hệ có kích từ thay đổi (ví dụ: Suy giảm từ thông ở dải tốc độ ngoài danh định).



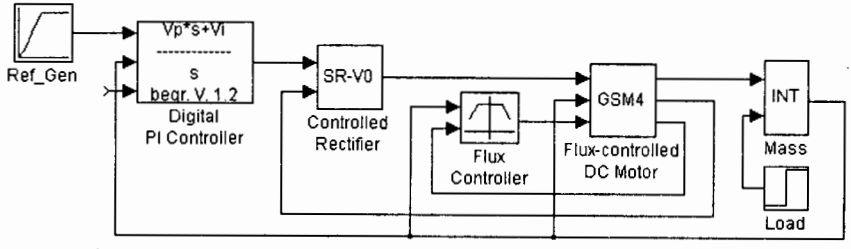
Hình 10.7 *Ba mô hình SIMULINK có sẵn của MĐMC kích thích độc lập trong thư viện Motion Control Blockset*

Hình 10.8 là một ví dụ mô phỏng sử dụng mô hình 10.7b, mô tả quá trình khởi động MĐMC (không có ĐK) khi đóng điện ban đầu và đóng tải sau 0,5 giây.



***Hình 10.8** Đặc tính của ĐCMC kích thích độc lập khi khởi động và khi đóng tải*

Một hệ thống truyền động có ĐK cả hai phía (phần ứng và kích từ) sẽ được thực hiện (ví dụ) bởi sơ đồ như hình 10.9. Điểm đặc biệt của sơ đồ là *khối chỉnh lưu*[1] có ĐK theo phương pháp cắt pha và có hai vòng ĐC dòng, ĐC tốc độ quay. Trong mục 11.2 ta sẽ đề cập tới vấn đề mô phỏng thiết bị CL này. Để gọn mắt dễ nhìn, trong hình dưới đây ta đã sử dụng Subsystem để gom thành các hệ thống con GSM4 (ĐCMC), *Flux Controller* (ĐK từ thông), SR-V0 (cầu chỉnh lưu).

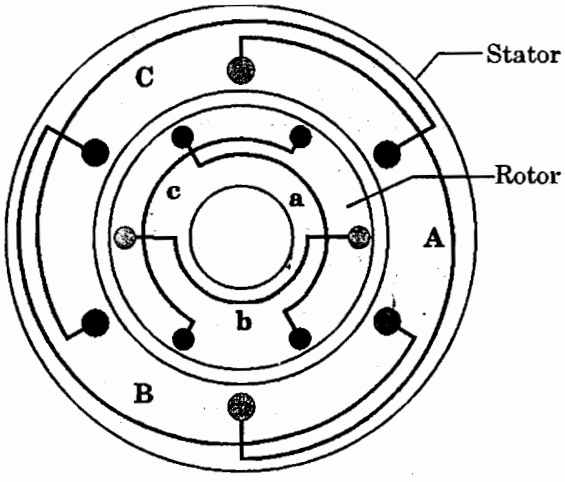


***Hình 10.9** Cấu trúc hệ thống truyền động điện một chiều có ĐK mômen và từ thông kích thích, sử dụng mô hình cầu chỉnh lưu 6 pha (có điều khiển)*

[1] Chỉnh lưu (Rectifier): viết tắt CL

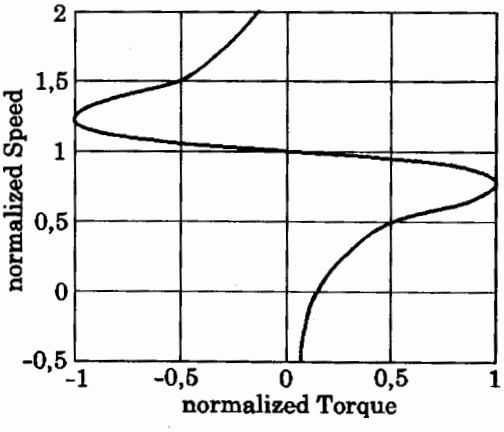
10.3 Máy điện không đồng bộ xoay chiều ba pha[1]

Về nguyên tắc, có thể mô tả kết cấu của MĐDB như hình ảnh đơn giản sau:



Hình 10.10 *Nguyên lý kết cấu (mặt cắt) của máy điện không đồng bộ ba pha*

Khi đặt điện áp xoay chiều ba pha lên hệ thống cuộn dây phía Stator sẽ tạo ra dòng Stator, gây nên điện áp cảm ứng phía Rotor và do đó xuất hiện dòng Rotor. Dòng hai phía Stator, Rotor có tác dụng tạo nên từ thông Stator, Rotor, và đó chính là nguyên nhân sinh ra mômen quay của máy điện. Điều kiện để xảy ra cảm ứng và tạo được mômen là tồn tại một "*sự trượt*" nhất định (không đồng bộ) giữa chuyển động quay của Rotor và của vector từ thông Stator. Trong trường hợp đồng bộ, máy điện sẽ không tạo được mômen quay.



Hình 10.11 *Đặc tính cơ của máy điện không đồng bộ ba pha (Rotor lồng sóc)*

[1] Induction Machines, hay 3-Phase Asynchronous Machines, máy điện dị bộ, viết tắt: MĐDB

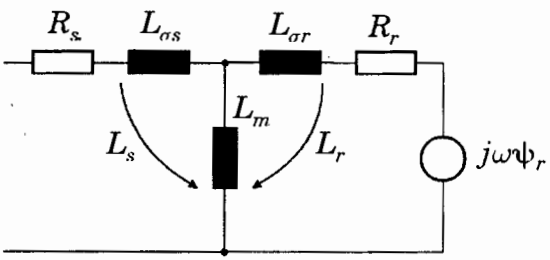
Đặc tính cơ của MĐDB được biểu diễn ở hình 10.11, tính theo công thức Kloss dưới đây:

$$\frac{m_M}{m_K} = \frac{2}{\frac{s}{s_K} + \frac{s_K}{s}}; R_s \approx 0 \tag{10.6}$$

$$m_K = \frac{3}{2} z_p \frac{1}{\sigma L_s} \left(\frac{U_s}{\omega_s} \right)^2; \quad s_K = \frac{R_r}{\omega_s \sigma L_s} \tag{10.7}$$

m_K	Mômen lật	R_r	Điện trở Rotor
s_K	Hệ số trượt khi $m_M = m_K$	L_s	Điện cảm Stator
m_M	Mômen động cơ	z_p	Số đôi cực
U_s	Điện áp Stator	σ	Hệ số tản
ω_s	Tần số Stator (vận tốc góc của các vector phía Stator)		

Công thức (10.6) là công thức gần đúng, xuất phát từ giả thiết điện trở Stator $R_s = 0$. Giả thiết đó có thể coi là đúng đối với máy điện công suất lớn – rất lớn. Để xây dựng hệ phương trình mô tả động học của MĐDB, ta có thể xuất phát từ sơ đồ thay thế một pha dưới đây:



Hình 10.12 *Sơ đồ thay thế của MĐDB*

Từ sơ đồ thay thế ta dễ dàng viết được các phương trình mô tả MĐDB.

- *Phương trình điện áp Stator*

$$\mathbf{u}_s = R_s \mathbf{i}_s + \frac{d\psi_s}{dt} \tag{10.8}$$

- *Phương trình điện áp Rotor*

$$\mathbf{0} = R_r \mathbf{i}_r + \frac{d\psi_r}{dt} - j\omega\psi_r \tag{10.9}$$

- *Phương trình từ thông*

$$\begin{aligned} \psi_s &= L_s \mathbf{i}_s + L_m \mathbf{i}_r \\ \psi_r &= L_m \mathbf{i}_s + L_r \mathbf{i}_r \end{aligned} \tag{10.10}$$

R_s	Điện trở Stator	$\mathbf{u}_s$	Vector điện áp Stator
R_r	Điện trở Rotor	$\mathbf{i}_s$	Vector dòng Stator
L_s	Điện cảm Stator	$\mathbf{i}_r$	Vector dòng Rotor
L_r	Điện cảm Rotor	ψ_s	Vector từ thông Stator
L_m	Hỗ cảm giữa hai cuộn dây	ψ_r	Vector từ thông Rotor

Các phương trình (10.8), (10.9) và (10.10) mô tả MĐDB trên hệ tọa độ $\alpha\beta$ cố định so với Stator. Viết lại hệ phương trình đó dưới dạng thành phần ta thu được mô hình sau đây:

$$\frac{di_{s\alpha}}{dt} = -\left(\frac{1}{\sigma T_s} + \frac{1-\sigma}{\sigma T_r}\right) i_{s\alpha} + \frac{1-\sigma}{\sigma T_r}\psi'_{r\alpha} + \frac{1-\sigma}{\sigma}\omega\psi'_{r\beta} + \frac{1}{\sigma L_s}u_{s\alpha}$$

$$\frac{di_{s\beta}}{dt} = -\left(\frac{1}{\sigma T_s} + \frac{1-\sigma}{\sigma T_r}\right) i_{s\beta} - \frac{1-\sigma}{\sigma}\omega\psi'_{r\alpha} + \frac{1-\sigma}{\sigma T_r}\psi'_{r\beta} + \frac{1}{\sigma L_s}u_{s\beta}$$

$$\frac{d\psi'_{r\alpha}}{dt} = \frac{1}{T_r} i_{s\alpha} - \frac{1}{T_r}\psi'_{r\alpha} - \omega\psi'_{r\beta}$$

$$\frac{d\psi'_{r\beta}}{dt} = \frac{1}{T_r} i_{s\beta} + \omega\psi'_{r\alpha} - \frac{1}{T_r}\psi'_{r\beta} \qquad (10.11)$$

Trong hệ phương trình (10.11) ta đã định nghĩa thêm các tham số và biến mới sau đây:

$$T_s = L_s/R_s ; T_r = L_r/R_r ; \sigma = 1 - L_m^2/L_s L_r$$

$$\psi'_r = \psi_r/L_m = \psi'_{r\alpha} + j\psi'_{r\beta} = \psi_{r\alpha}/L_m + j\,\psi_{r\beta}/L_m$$

Cuối cùng ta còn phải bổ sung thêm phương trình mômen.

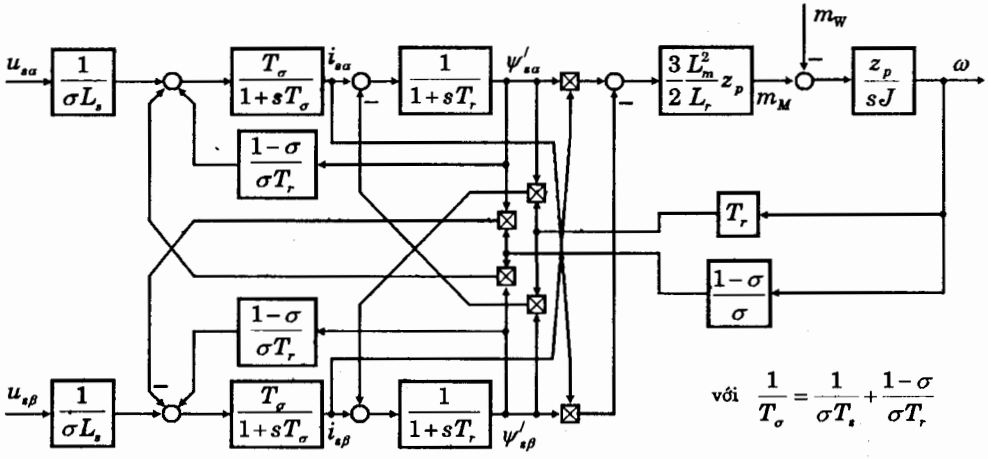
$$m_M = \frac{3}{2} z_p (1-\sigma) L_s \left(\psi'_{r\alpha} i_{s\beta} - \psi'_{r\beta} i_{s\alpha}\right) \qquad (10.12)$$

Hệ phương trình mô tả MĐDB đã được xây dựng trên cơ sở chấp nhận các giả thiết sau:

- Hệ phương trình thu được trên cơ sở hài cơ bản của các đại lượng dòng, áp và từ thông. Hiện tượng móc vòng từ thông giữa Stator và Rotor chỉ xảy ra với hài cơ bản. Mômen hài chưa được quan tâm.
- Hệ chưa xét tới hiện tượng bão hòa vật liệu từ. Tuy nhiên, có thể sử dụng hàm xấp xỉ để bổ sung đặc tính bão hòa một cách rất dễ dàng.
- Chưa xét tới tổn hao dòng quẩn và tổn hao sắt từ.
- Chưa xét tới hiện tượng dãn dòng (xuất hiện đối với hài dòng bậc cao, làm tăng giá trị hiệu dụng của điện trở).

- Stator và Rotor có kết cấu tròn đều đối xứng. Hệ thống Stator có chứa z_p cuộn dây ba pha. Việc đặt điện áp Rotor bằng 0 xuất phát từ kết cấu ngắn mạch của mạch điện phía Rotor. Đối với mô hình của MĐDB Rotor dây quấn sẽ phải bổ sung thêm phương trình cho hệ.
- Tham số của mô hình là hằng (ví dụ: Không phụ thuộc nhiệt độ).
- Bỏ qua tổn hao ma sát (ví dụ: Của quạt làm mát, của ổ bi. Các tổn hao này cần được xét đến trong mô hình phần cơ).

Có thể chuyển mô hình (10.11) sang biểu diễn bằng đồ họa như hình 10.13.



Hình 10.13 *Mô hình MĐDB Rotor lồng sóc trên hệ tọa độ αβ (hệ tọa độ Stator)*

Nếu chuyển mô hình (10.11) sang biểu diễn trên hệ tọa độ dq, là hệ tọa độ có trục thực d trùng với trục của vector từ thông Rotor ψ_r. Nghĩa là, hệ tọa độ dq quay đồng bộ với vector ψ_r và còn được gọi là hệ tọa độ từ thông. Khi ấy mô hình sẽ có dạng.

$$\frac{di_{sd}}{dt}=-\left(\frac{1}{\sigma T_s}+\frac{1-\sigma}{\sigma T_r}\right)i_{sd}+\omega_s i_{sq}+\frac{1-\sigma}{\sigma T_r}\psi'_{rd}+\frac{1-\sigma}{\sigma}\omega\psi'_{rq}+\frac{1}{\sigma L_s}u_{sd}$$

$$\frac{di_{sq}}{dt}=-\omega_s i_{sd}-\left(\frac{1}{\sigma T_s}+\frac{1-\sigma}{\sigma T_r}\right)i_{sq}-\frac{1-\sigma}{\sigma}\omega\psi'_{rd}+\frac{1-\sigma}{\sigma T_r}\psi'_{rq}+\frac{1}{\sigma L_s}u_{sq}$$

$$\frac{d\psi'_{rd}}{dt}=\frac{1}{T_r}i_{sd}-\frac{1}{T_r}\psi'_{rd}+(\omega_s-\omega)\psi'_{rq}$$

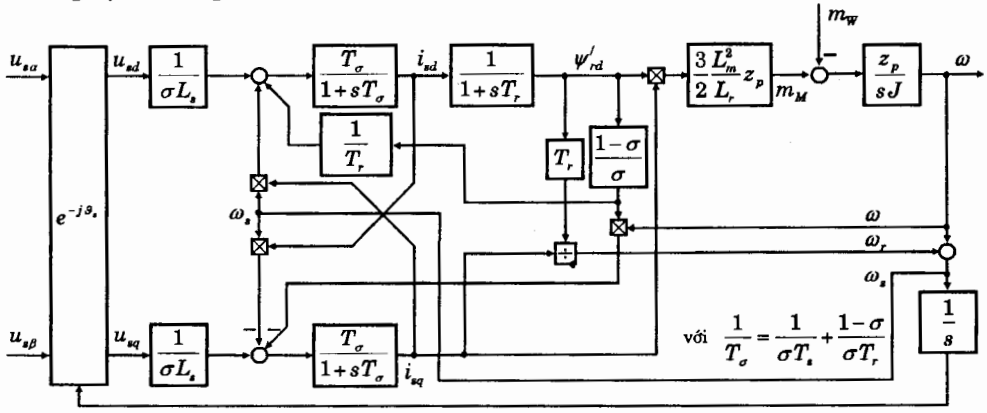
$$\frac{d\psi'_{rq}}{dt}=\frac{1}{T_r}i_{sq}-(\omega_s-\omega)\psi'_{rd}-\frac{1}{T_r}\psi'_{rq}$$

(10.13)

với phương trình mômen như sau:

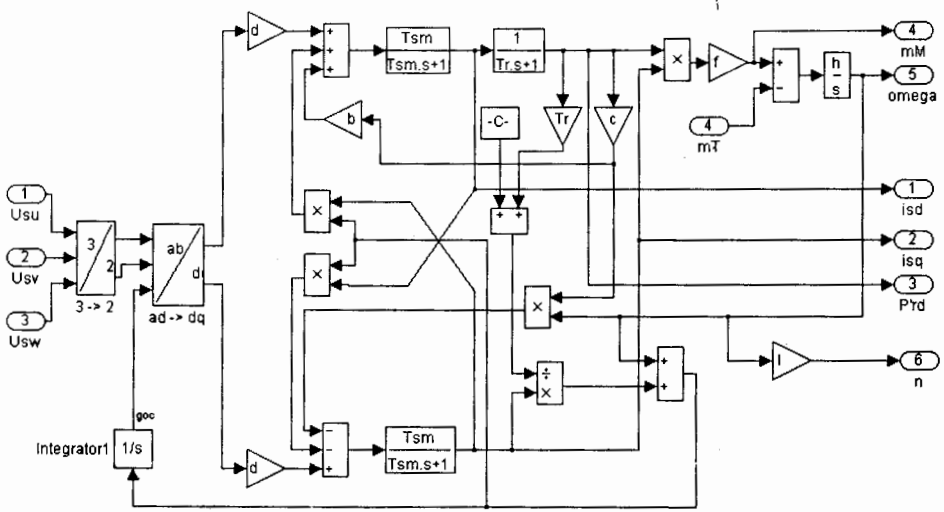
$$m_M = \frac{3}{2} z_p \frac{L_m^2}{L_r} \psi'_{rd} i_{sq} = \frac{3}{2} z_p (1-\sigma) L_s \psi'_{rd} i_{sq} \qquad (10.14)$$

Tương tự, ta cũng có thể biểu diễn mô hình (10.13) như hình 10.14.



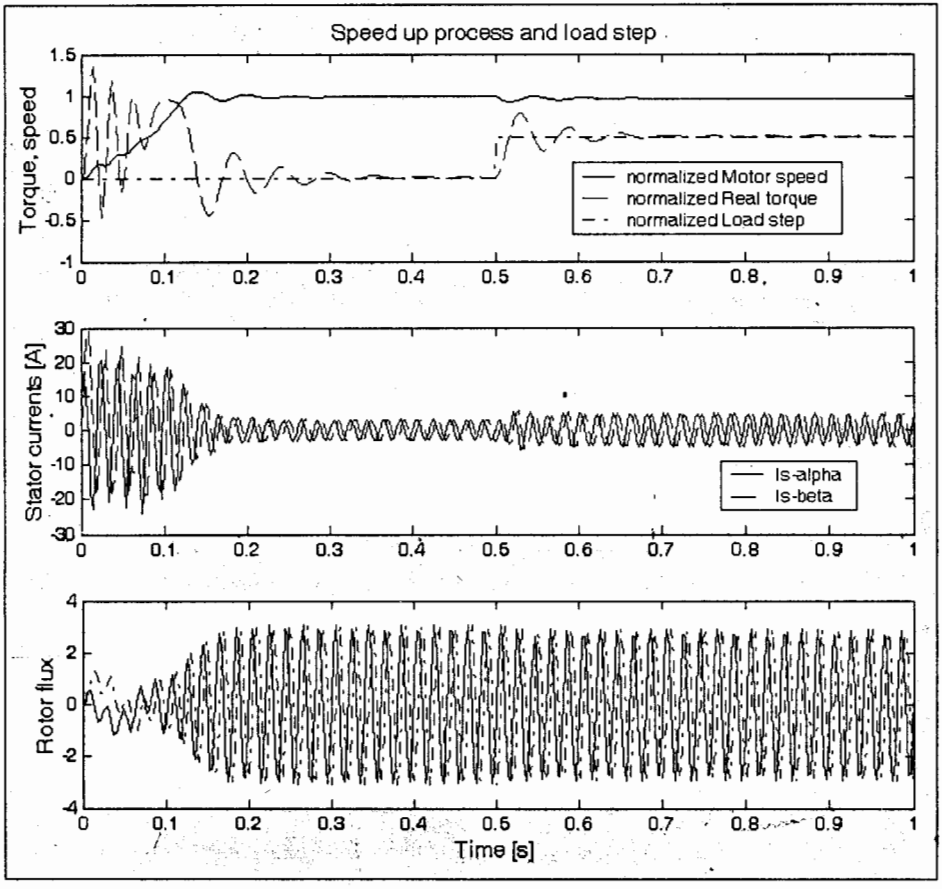
Hình 10.14 *Mô hình MĐDB Rotor lồng sóc trên hệ tọa độ dq (hệ tọa độ từ thông Rotor)*

Việc xây dựng mô hình mô phỏng đến đây không còn là vấn đề khó khăn: Ta chỉ việc sử dụng các khối có sẵn thuộc thư viện SIMULINK để kết hợp lại theo đúng như một trong hai hình 10.13 và 10.14. Mô hình SIMULINK của MĐDB trên hệ tọa độ *dq* ở hình 10.15 là một minh chứng cho cách thực hiện này.



Hình 10.15 *Mô hình mô phỏng MĐDB Rotor lồng sóc trên hệ tọa độ từ thông Rotor, thực hiện bằng các khối có sẵn từ thư viện SIMULINK*

Hình 10.16 giới thiệu vài kết quả mô phỏng MĐDB với mô hình xây dựng trên hệ tọa độ $\alpha\beta$ (hình 10.13): Quá trình khởi động ban đầu không tải và đóng tải sau 0,5s (hình trên). Tải chỉ được đóng sau khi biên độ của từ thông Rotor (hình dưới) đã đạt tới giá trị danh định. Dòng Stator (hình giữa) cho ta thấy rất rõ quá trình khởi động nặng (dù là không tải) của MĐDB vì động cơ còn đang trong quá trình từ hóa: *Dòng khởi động có thể lớn gấp nhiều lần dòng danh định*. Bạn đọc cần lưu ý: Các hình đó mô tả một quá trình khởi động chưa có ĐK, giống như khi ta đóng mạch nối động cơ trực tiếp với lưới điện.



Hình 10.16 *Quá trình khởi động không tải của MĐDB Rotor lồng sóc, và đóng tải sau khi biên độ từ thông đã đủ lớn*

Mặc dù việc mô hình hóa MĐDB lúc này đã trở nên đơn giản và dễ dàng, ta vẫn phải nêu câu hỏi: Mô hình SIMULINK của riêng MĐDB đã phong phú như vậy, nếu ghép vào một hệ thống ĐK truyền động/chuyển động với tất cả các thuật toán chế ngự động cơ, chế ngự chuyển động thì nó sẽ còn trở nên phong phú như thế nào nữa, liệu có vượt quá khả năng của máy tính không?

Đây quả thực là một vấn đề cần được suy nghĩ và giải quyết. Trong thực tế, nhiều khi chỉ một hệ truyền động với một MĐDB Rotor lồng sóc duy nhất, ĐK bằng *phương pháp tựa theo từ thông Rotor*[1], đã đủ để làm PC mất rất nhiều thời gian tính. Chưa nói đến việc mô phỏng các hệ ĐKCĐ nhiều trục phức hợp với đầy đủ mọi phần tử cơ - điện của các trục (hình 10.1). Để giải quyết vấn đề này, thay vì xây dựng mô hình theo cách sử dụng các khối có sẵn từ thư viện SIMULINK, ta phải tự tạo ra các khối mới sử dụng hàm S (mục 7.7: *S-Function*). Để tạo hàm S ta có hai cách:

- Soạn thảo một *script* (*m-File*), lập trình tính toán các phương trình thuộc hệ (10.11) hoặc (10.13), kèm theo phương trình mômen (10.12) hoặc (10.14), có thể có (hoặc không có) phương trình chuyển động (10.4). Khi viết phải tuân thủ các điều kiện khung mô tả ở hình 7.10.
- Soạn thảo chương trình bằng *ngôn ngữ bậc cao* như C, C++, FORTRAN hay Ada (thuận lợi nhất là C), tính toán các phương trình mô hình của máy điện. Sau khi soạn thảo, mã nguồn được dịch sang thành *File* chạy trực tiếp. Với các phiên bản MATLAB cũ, *File* chạy được gọi là *mex-File* (có đuôi *.mex*). Hiện tại, từ MATLAB 6.0 trở về sau, mã nguồn được dịch trực tiếp sang *File* có đuôi *.dll* (*Dynamic Link Library*) rất thuận tiện cho việc sử dụng lặp lại. Tuy nhiên, các *dll-File* vẫn được gọi là *mex-File*, bởi chúng được tạo nên bởi lệnh `mex` của MATLAB.

Để gọi hàm S ta sử dụng khối *S-Function* (mục 7.5: thư viện *Function & Tables*) với cách khai báo tương tự ví dụ ở hình 7.11. Đoạn *script* dưới đây là ví dụ về hàm S, mô hình của MĐDB Rotor lồng sóc trên hệ tọa độ $\alpha\beta$, tính hệ phương trình (10.11):

```
function [sys,x0,str,ts] = ...
         ASMabm(t,x,u,flag,Lr,Ls,Rr,Rs,Lm,pc)
% Motor paramters :
%       Lr   Rotor inductance
%       Ls   Stator inductance
%       Rr   Rotor resistance
%       Rs   Stator resistance
%       Lm   Mutual inductance between stator and rotor
%       pc   Number of pole pair
%
% Computing motor parameters
sigma = 1-(Lm*Lm)/(Lr*Ls);  % Total leakage factor
Tr = Lr/Rr;                 % Rotor time constant
Ts = Ls/Rs;                 % Stator time constant
a  = 1/(sigma*Ts);          % Axiliary variables
b  = 1/Tr;
c  = (1-sigma)/sigma;
d  = 1/(sigma*Ls);
```

[1] Tựa theo từ thông Rotor: Rotor Flux Orientation, viết tắt T$^{\text{d}}$R

```
g = c*b;
e = a+g;
f = (3/2)*pc*Lm*Lm/Lr;

% The flag controls the calls of S- function routines at
% each simulation stage.
switch flag,
  case 0                              % Initialization
    [sys,x0,str,ts] = mdlInitializeSizes(t,x,u);
  case 1                              % Calculate derivatives
    sys = mdlDerivatives(t,x,u,b,c,d,e,g);
  case 3                              % Calculate outputs
    sys = mdlOutputs(t,x,u,f);
  case {2,4,9}                        % Unused flags
sys = [];
otherwise
error(['Unhandled flag=',num2str(flag)]);
end
% End of function ASMabm
%
%=============================================================
% mdlInitializeSizes
% Return the sizes,initial conditions,sample time,state
% ordering strings (str) for the S -function
%=============================================================
function [sys,x0,str,ts] = mdlInitializeSizes(t,x,u)
% Call simsizes for a sizes structure , fill it in and
% convert it to a sizes array
%
sizes = simsizes;
% Load the sizes structure with the initialization
% information
  sizes.NumContStates  = 4;      % isa, isb, Psira', Psirb'
  sizes.NumDiscStates  = 0;
  sizes.NumOutputs     = 4;      % isa, isb, |Psird'|, mM
  sizes.NumInputs      = 3;      % usa, usb, omega
  sizes.DirFeedthrough = 0;
  sizes.NumSampleTimes = 1;
% Load the sys vector with the sizes information
sys = simsizes(sizes);
x0 = zeros(4,1);                  % Initial conditions
str = [];                         % str is an empty matrix
                                  % ( No state ordering )
% Initialize the array of sample time: the sample time is
% continuous, so set ts to 0 and its offset to 0.
ts =[0 0];
% End mdlInitializeSizes
```

```
%=============================================================
% mdlDerivatives:
% Return the derivatives for the continuous states.
% Equations (11.11a, b, c, d)
%=============================================================
function sys = mdlDerivatives(t,x,u,b,c,d,e,g);
  sys(1) = -e*x(1)+g*x(3)+c*u(3)*x(4)+d*u(1);
  sys(2) = -e*x(2)-c*u(3)*x(3)+g*x(4)+d*u(2);
  sys(3) = b*x(1)-b*x(3)-u(3)*x(4);
  sys(4) = b*x(2)+u(3)*x(3)-b*x(4);
% End Derivatives

%=============================================================
% mdloutputs
% Return the block outputs
%=============================================================
function sys = mdlOutputs(t,x,u,f);
  sys(1) = x(1);
  sys(2) = x(2);
  sys(3) = sqrt(x(3)^2+x(4)^2);      % |Psir'|
  sys(4) = f*(x(3)*x(2)-x(4)*x(1));  % Equation (11.12)
%End mdlOutputs
```

Mô hình MĐDB Rotor lồng sóc trên hệ tọa độ từ thông (hệ tọa độ *dq*) được thực hiện bởi đoạn *m-File* dưới đây:

```
function [sys,x0,str,ts] = ...
         ASMdqm(t,x,u,flag,Lr,Ls,Rr,Rs,Lm,pc)
% Motor paramters :
%       Lr   Rotor inductance
%       Ls   Stator inductance
%       Rr   Rotor resistance
%       Rs   Stator resistance
%       Lm   Mutual inductance between stator and rotor
%       pc   Number of pole pair
%
% Computing motor parameters
sigma = 1-(Lm*Lm)/(Lr*Ls);  % Total leakage factor
Tr = Lr/Rr;                 % Rotot time constant
Ts = Ls/Rs;                 % Stator time constant
a = 1/(sigma*Ts);           % Axiliary variables
b = 1/Tr;
c = (1-sigma)/sigma;
d = 1/(sigma*Ls);
g = c*b;
e = a+g;
f = (3/2)*pc*Lm*Lm/Lr;
```

```
% The flag controls the calls of S-function routines at
% each simulation stage.
switch flag,
  case 0                              % Initialization
    [sys,x0,str,ts] = mdlInitializeSizes(t,x,u);
  case 1                              % Calculate derivatives
    sys = mdlDerivatives(t,x,u,b,c,d,e,g);
  case 3                              % Calculate outputs
    sys = mdlOutputs(t,x,u,f,b);
  case {2,4,9}                        % Unused flags
    sys = [];
otherwise
error(['Unhandled flag=',num2str(flag)]);
end
% End of function ASMdqm
%
%=============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, sample time, state
% ordering strings (str) for the S -function
%=============================================================
function [sys,x0,str,ts] = mdlInitializeSizes(t,x,u)
% Call simsizes for a sizes structure, fill it in and
% convert it to a sizes array
%
sizes = simsizes;
% Load the sizes structure with the initialization
% information
  sizes.NumContStates = 3;     % isd, isq, Psird'
  sizes.NumDiscStates = 0;
  sizes.NumOutputs = 5;        % isd, isq, Psird', mM, omega
  sizes.NumInputs = 4;
  sizes.DirFeedthrough = 0;
  sizes.NumSampleTimes = 1;
% Load the sys vector with the sizes information
sys = simsizes(sizes);
x0 = [0;0;.00001];             % Initial conditions
str = [];                      % str is an empty matrix
                               % ( No state ordering )
% Initialize the array of sample time: the sample time is
% continuous, so set ts to 0 and its offset to 0.
ts =[0 0];
% End mdlInitializeSizes
%
%=============================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
```

```
% Equations (11.13a, b, c)
%=============================================================
function sys = mdlDerivatives(t,x,u,b,c,d,e,g)
  sys(1) = -e*x(1) + u(4)*x(2) + g*x(3) + d*u(1);
  sys(2) = -u(4)*x(1) - e*x(2) - c*u(3)*x(3) + d*u(2);
  sys(3) = b*x(1) - b*x(3);
% End Derivatives
%
%=============================================================
% mdloutputs
% Return the block outputs
%=============================================================
function sys = mdlOutputs(t,x,u,f,b)
  sys(1) = x(1);
  sys(2) = x(2);
  sys(3) = x(3);
  sys(4) = f*x(2)*x(3); % (2.7e)
  sys(5) = b*x(2)/x(3); % (2.7d)
%End mdlOutputs
```

Nếu muốn tăng tốc độ mô phỏng thêm nữa, giải pháp tốt nhất vẫn là xây dựng mô hình dưới dạng *C-mex-File*, dịch sang thành *.dll-File* để gọi từ khối *S-Function*. Đoạn mã nguồn C dưới đây là mô hình của MĐDB trên hệ tọa độ $\alpha\beta$ (hệ tọa độ cố định Stator):

```
/*****************************************************
 * ASM1ABX.C: C-MEX-File to simulate a 3-phase IM in  *
 * stator coordinates                                 *
 * Syntax: [sys,x0] = asm1abx(t,x,u,flag,p,xo)        *
 *   Parameters p[Rr,Lr,Lm,Rs,Ls,zp],                 *
 *         Rr  Rotor resistance                       *
 *         Lr  Rotor inductance                       *
 *         Lm  Mutual inductance                      *
 *         Rs  Stator resistance                      *
 *         Ls  Stator inductance                      *
 *         zp  Pole pair                              *
 *   Variables xo[isa0,isb0,Psira0,Psirb0]            *
 *         isa0 Start value of stator current alpha   *
 *         isb0 Start value of stator current beta    *
 *         Psira0 Start value of rotor flux alpha     *
 *         Psirb0 Start value of rotor flux beta      *
 *****************************************************/
/* The following #define is used to specify the name of
 * this S-Function */

#define S_FUNCTION_NAME asm1abx
```

```
/* Need to include simstruc.h for the definition of the
 * SimStruct and its associated macro definitions */
#include "simstruc.h"

/* Input variables: */
#define usa      u[0]          /* Stator voltage usa */
#define usb      u[1]          /* Stator voltage usb */
#define w_mech   u[2]          /* Rotor speed w_mech */

/* Output variables: */
#define mm       y[0]             /* Torque mM */

/* State variables */
#define isax     x[0]          /* Stator current alpha isa */
#define isbx     x[1]          /* stator current beta isb */
#define Psirax   x[2]          /* Rotor flux Psira */
#define Psirbx   x[3]          /* Rotor flux Psirb */

/* Derivatives: */
#define disax    dx[0]
#define disbx    dx[1]
#define dPsirax  dx[2]
#define dPsirbx  dx[3]

/* Defines for easy access of p, xo matrices which are
   passed in */
#define p        ssGetArg(S,0)
#define xo       ssGetArg(S,1)

/* mdlInitializeSizes - initialize the sizes array */
static void mdlInitializeSizes(S)
  SimStruct  *S;
  { ssSetNumContStates(S,4);      /* number of continuous
                                     states */
    ssSetNumDiscStates(S,0);      /* number of discrete
                                     states */
    ssSetNumInputs(S,3);          /* number of inputs */
    ssSetNumOutputs(S,5);         /* number of outputs */
    ssSetDirectFeedThrough(S,0); /* direct feedthrough
                                     flag */
    ssSetNumSampleTimes(S,1);     /* number of sample
                                     times */
    ssSetNumInputArgs(S,2);       /* number of input
                                     arguments */
    ssSetNumRWork(S,0);           /* number of real work
                                     vector elements */
    ssSetNumIWork(S,0);           /* number of integer work
```

```
                                        vector elements */
    ssSetNumPWork(S,0);              /* number of pointer work
                                        vector elements */
  }

/* mdlInitializeSampleTimes - initialize the sample times
 * array */
static void mdlInitializeSampleTimes(S)
  SimStruct  *S;
  { ssSetSampleTimeEvent(S, 0, 0.0);
    ssSetOffsetTimeEvent(S, 0, 0.0);
  }

/* mdlInitializeConditions - initialize the states */
static void mdlInitializeConditions(x0, S)
  double *x0;
  SimStruct *S;
  { inti;
    for(i=0;i<4;i++)
    x0[i] = mxGetPr(xo)[i];
  }

/* mdlOutputs - compute the outputs */
static void mdlOutputs(y, x, u, S, tid)
  double *y, *x, *u;
  SimStruct *S;
  int tid;
  { { double *param;
      double Lr,Lm,zp;
      param = ( double *)mxGetPr(p);
      Lr = param[1];                /* Rotor inductance */
      Lm = param[2];                /* Mutual inductance */
      zp = param[5];                /* Pole pair */
      mm = 3.0/2.0 * zp*Lm/Lr * (Psirax * isbx
                                - Psirbx * isax);
                                    /* Motor torque */
      y[1] = isax;
      y[2] = isbx;
      y[3] = Psirax;
      y[4] = Psirbx;
    }
  }

/* mdlUpdate - perform action at major integration
 * time step */
static void mdlUpdate(x, u, S, tid)
  double *x, *u;
```

```
  SimStruct *S;
  int tid;
  {
  }

/* mdlDerivatives - compute the derivatives */
static void mdlDerivatives(dx, x, u, S, tid)
  double *dx, *x, *u;
  SimStruct *S;
  int tid;
  { double *param;
    double Rr,Lr,Lm,Rs,Ls,zp;
    double Tr, Ts, sigma;
    param = ( double *)mxGetPr(p);
    Rr = param[0];                /* Rotor resistance */
    Lr = param[1];                /* Rotor inductance */
    Lm = param[2];                /* Mutual inductance */
    Rs = param[3];                /* Stator resistance */
    Ls = param[4];                /* Stator inductance */
    zp = param[5];                /* Pole pair */

  /* Computing of auxiliary parameters */
    Tr    = Lr / Rr;              /* Rotor time constant */
    Ts    = Ls / Rs;              /* Stator time constant */
    sigma = 1 - Lm*Lm/(Ls*Lr);    /* Total leakage factor */

  /* State equations */
    disax = -(1/(sigma*Ts)+(1-sigma)/(Tr*sigma))*isax
          + (1-sigma)/(sigma*Tr*Lm)*Psirax
          + (1-sigma)/(sigma*Lm)*zp*w_mech*Psirbx
          + 1/(sigma*Ls)*usa;                  /* isa */
    disbx = - (1/(sigma*Ts)+(1-sigma)/(Tr*sigma))*isbx
          - (1-sigma)/(sigma*Lm)*Psirax*zp*w_mech
          + (1-sigma)/(sigma*Tr*Lm)*Psirbx
          + 1/(sigma*Ls)*usb;                  /* isb */

    dPsirax = (-Psirax + Lm*isax)/Tr
          - Psirbx*zp*w_mech;                  /* Psira */
    dPsirbx = (-Psirbx + Lm*isbx)/Tr
          + Psirax*zp*w_mech;                  /* Psirb */
  }
/* mdlTerminate - called when the simulation is
 * terminated */
static void mdlTerminate(S)
  SimStruct *S;
  {
  }
```

```
#ifdef MATLAB_MEX_FILE    /* Is this file being compiled
                             as a MEX-file? */
#include "simulink.c"     /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"      /* Code generation registration
                             function */
#endif
```

Cho đến nay, các mô hình MĐDB Rotor lồng sóc (trên hệ tọa độ *dq* hay *αβ*) đều xuất phát từ giả thiết: Máy điện được nuôi bởi một thiết bị nguồn áp (ví dụ: *Nghịch lưu nguồn áp*[1] dạng xung băm, hay dạng xung Block). Trong thực tiễn, rất nhiều động cơ với công suất rất lớn được nuôi bởi *nghịch lưu nguồn dòng*[2], có lọc ở *mạch điện một chiều trung gian*[3] là cuộn cảm. Do công suất quá lớn, ta không thể dùng tụ để lọc được nữa. Trong trường hợp đó, mô hình (phần điện) thích hợp của MĐDB sẽ chỉ còn là hai phương trình cuối của hệ (10.11) hay (10.13). Các phương trình mômen quay (10.12), (10.14) và phương trình chuyển động (10.4) vẫn giữ nguyên hiệu lực.

Phần mã nguồn C sau đây là ví dụ về mô hình trên hệ tọa độ *αβ* của MĐDB nuôi bởi nghịch lưu nguồn dòng.

```
/*****************************************************
 * ASM3ABI.C: C-MEX-File to simulate a 3-phase IM in  *
 * stator coordinates fed by a current source inverter *
 * Syntax: [sys,x0] = asm3abi(t,x,u,flag,p)           *
 *   Parameter p[Rr,Lr,Lm,Rs,Ls,zp]                   *
 *         Rr  Rotor resistance                       *
 *         Lr  Rotor inductance                       *
 *         Lm  Mutual inductance                      *
 *         Rs  Stator resistance                      *
 *         Ls  Stator inductance                      *
 *         zp  Pole pair                              *
 *****************************************************/
/* The following #define is used to specify the name of
 * this S-Function */
#define S_FUNCTION_NAME asm3abi

/* Need to include simstruc.h for the definition of the
 * SimStruct and its associated macro definitions */
#include "simstruc.h"

/* Input variables: */
#define isa      u[0]            /* Stator current isa */
```

[1] Nghịch lưu nguồn áp: Voltage-Source Inverter
[2] Nghịch lưu nguồn dòng: Current-Source Inverter
[3] Mạch một chiều trung gian: DC-Link

```
#define isb      u[1]            /* Stator current isb */
#define w_mech   u[2]            /* Rotor speed w_mech */

/* Output variables: */
#define mm       y[0]            /* Torque mM */

/* State variables: */
#define Psirax   x[0]            /* Rotor flux Psira */
#define Psirbx   x[1]            /* Rotor flux Psirb */
#define M_KONST  x[2]            /* Motor constant:
                                    3.0/2.0*zp*Lm/Lr */
#define Tr       x[3]            /* Rotor time constant */

/* Derivatives: */
#define dPsirax  dx[0]
#define dPsirbx  dx[1]

/* Defines for easy access of p, xo matrices which are
 * passed in */
#define p        ssGetArg(S,0)

/* mdlInitializeSizes - initialize the sizes array */
static void mdlInitializeSizes(S)
  SimStruct  *S;
  { ssSetNumContStates(S,2);     /* number of continuous
                                    states */
    ssSetNumDiscStates(S,2);     /* number of discrete
                                    states */
    ssSetNumInputs(S,3);         /* number of inputs */
    ssSetNumOutputs(S,5);        /* number of outputs */
    ssSetDirectFeedThrough(S,0); /* direct feedthrough
                                    flag */
    ssSetNumSampleTimes(S,1);    /* number of sample
                                    times */
    ssSetNumInputArgs(S,1);      /* number of input
                                    arguments */
    ssSetNumRWork(S,0);          /* number of real work
                                    vector elements */
    ssSetNumIWork(S,0);          /* number of integer work
                                    vector elements */
    ssSetNumPWork(S,0);          /* number of pointer work
                                    vector elements */
  }

/* mdlInitializeSampleTimes - initialize the sample times
 * array */
```

```
static void mdlInitializeSampleTimes(S)
  SimStruct  *S;
  { ssSetSampleTimeEvent(S, 0, 0.0);
    ssSetOffsetTimeEvent(S, 0, 0.0);
  }

/* mdlInitializeConditions - initialize the states */
static void mdlInitializeConditions(x0, S)
  double *x0;
  SimStruct *S;
  { inti;
    double Lm,Lr,Rr,zp;
    Rr = mxGetPr(p)[0];              /* Rotor resistance */
    Lr = mxGetPr(p)[1];              /* Rotor inductance */
    Lm = mxGetPr(p)[2];              /* Mutual inductance */
    zp = mxGetPr(p)[3];              /* Pole pair */

    for(i=0;i<2;i++)
      x0[i] = 0;
      x0[2] = 3.0/2.0*zp*Lm/Lr;
      x0[3] = Lr/Rr;
  }

/* mdlOutputs - compute the outputs */
static void mdlOutputs(y, x, u, S, tid)
  double *y, *x, *u;
  SimStruct *S;
  int tid;
  { { mm = M_KONST * (Psirax * isb - Psirbx * isa);
                                     /* Motor torque */
      y[1] = isa;
      y[2] = isb;
      y[3] = Psirax;
      y[4] = Psirbx;
    }
  }

/* mdlUpdate - perform action at major integration
 * time step */
static void mdlUpdate(x, u, S, tid)
  double *x, *u;
  SimStruct *S;
  int tid;
  {
  }
```

```
/* mdlDerivatives - compute the derivatives */
static void mdlDerivatives(dx, x, u, S, tid)
  double *dx, *x, *u;
  SimStruct *S;
  int tid;
  { double Lm,zp,w_el;
    Lm = mxGetPr(p)[2];            /* Mutual inductance */
    zp = mxGetPr(p)[3];            /* Pole pair */

  /* Computing of auxiliary parameter */
    w_el = zp * w_mech;

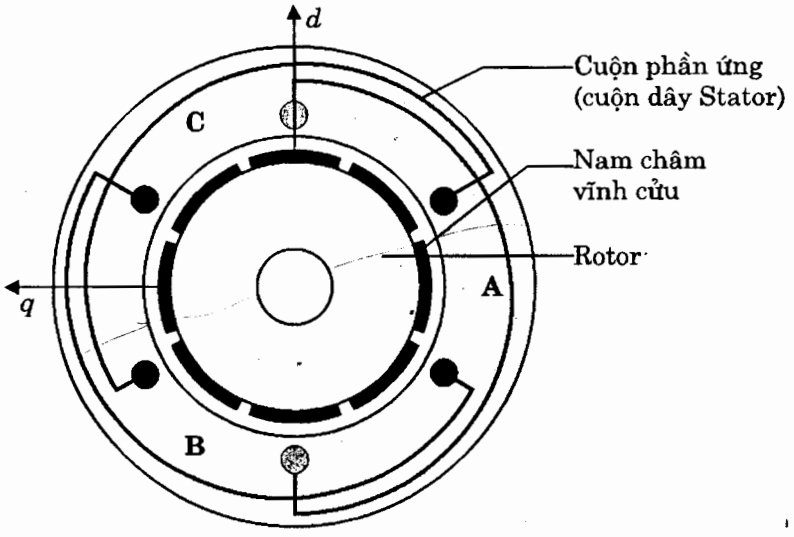
  /* State equations */
    dPsirax = (Lm*isa-Psirax)/Tr-Psirbx*w_el; /* Psira */
    dPsirbx = (Lm*isb-Psirbx)/Tr+Psirax*w_el; /* Psirb */
  }

/* mdlTerminate - called when the simulation is
 * terminated */
static void mdlTerminate(S)
  SimStruct *S;
  {
  }
#ifdef MATLAB_MEX_FILE    /* Is this file being compiled
                             as a MEX-file? */
#include "simulink.c"     /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"      /* Code generation registration
                             function */
#endif
```

10.4 Mô hình máy điện đồng bộ ba pha kích thích vĩnh cửu

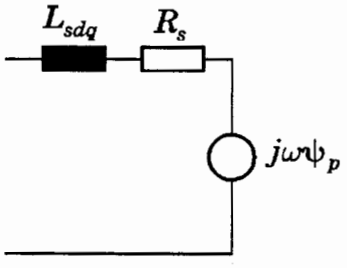
Máy điện đồng bộ ba pha kích thích vĩnh cửu (MĐĐB-KTVC) có kết cấu phía Stator giống như MĐDB: Đó là hệ thống cuộn dây nhận nguồn cấp điện ba pha. Phía Rotor, nơi MĐDB phải dựa vào *sự trượt tốc độ* giữa Rotor và hệ thống vector phía Stator để tạo từ thông Rotor, MĐĐB-KTVC có một hệ thống nam châm vĩnh cửu gắn chặt trên bề mặt. Nghĩa là: Từ thông luôn luôn tồn tại, không còn nhu cầu trượt tốc độ để cảm ứng từ Stator sang Rotor nữa, và máy điện hoạt động hoàn toàn đồng bộ. Kết cấu của MĐĐB-KTVC có thể được mô tả bằng sơ đồ mặt cắt đơn giản như ở hình 10.17.

Phạm vi sử dụng chính của MĐĐB-KTVC là công nghiệp chế tạo máy chế biến (ví dụ: Máy gia công cắt gọt kim loại, máy đóng bao gói, máy gia công chính xác vv...). Trong các ứng dụng đó, MĐĐB-KTVC gần như tuyệt đối chỉ được sử dụng kèm theo thiết bị ĐK (ví dụ: Nghịch lưu có ĐC) chất lượng cao.



Hình 10.17 *Nguyên lý kết cấu (mặt cắt) của máy điện đồng bộ kích thích vĩnh cửu*

Mô hình toán của MĐĐB-KTVC cũng được xây dựng xuất phát từ giả thiết coi máy điện có kết cấu tròn đối xứng. Các đại lượng xoay chiều ba pha đã được biểu diễn dưới dạng vector và chuyển sang hệ tọa độ từ thông ***dq***, đồng thời cũng là hệ tọa độ cố định trên Rotor. Vì vậy có thể xuất phát từ sơ đồ thay thế một pha ở hình 10.18 dưới đây.



Hình 10.18 *Sơ đồ thay thế của MĐĐB-KTVC*

Từ sơ đồ thay thế ta viết được các phương trình sau:

- *Phương trình điện áp Stator*

$$\mathbf{u}_s^f = R_s \mathbf{i}_s^f + \frac{d\psi_s^f}{dt} + j\omega_s \psi_s^f \quad [1] \tag{10.15}$$

- *Phương trình từ thông*

$$\psi_s^f = L_s \mathbf{i}_s^f + \psi_p^f \tag{10.16}$$

Viết lại dưới dạng thành phần và sau vài biến đổi ta có hệ phương trình:

[1] Chỉ số " f " viết trên cao, bên phải: Flux, vector thuộc hệ tọa độ từ thông

$$\frac{di_{sd}}{dt} = -\frac{1}{T_{sd}} i_{sd} + \omega_s \frac{L_{sq}}{L_{sd}} i_{sq} + \frac{1}{L_{sd}} u_{sd}$$
$$\frac{di_{sq}}{dt} = -\omega_s \frac{L_{sd}}{L_{sq}} i_{sd} - \frac{1}{T_{sq}} i_{sq} + \frac{1}{L_{sq}} u_{sq} - \omega_s \frac{\psi_p}{L_{sq}} \quad (10.17)$$

Với phương trình từ thông:

$$\psi_{sd} = L_{sd} i_{sd} + \psi_p$$
$$\psi_{sq} = L_{sq} i_{sq} \quad (10.18)$$

L_{sd} Điện cảm Stator đo ở vị trí đỉnh cực

$T_{sd} = L_{sd} / R_s$ Hằng số thời gian Stator tại vị trí đỉnh cực

L_{sq} Điện cảm Stator đo ở vị trí ngang cực

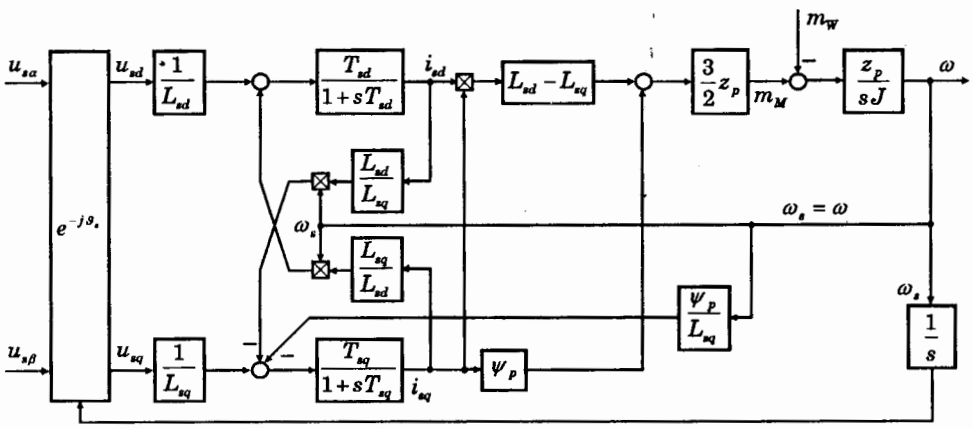
$T_{sq} = L_{sq} / R_s$ Hằng số thời gian Stator tại vị trí đỉnh cực

ψ_p Từ thông cực (vĩnh cửu)

Phương trình mômen của MĐĐB-KTVC có dạng sau đây:

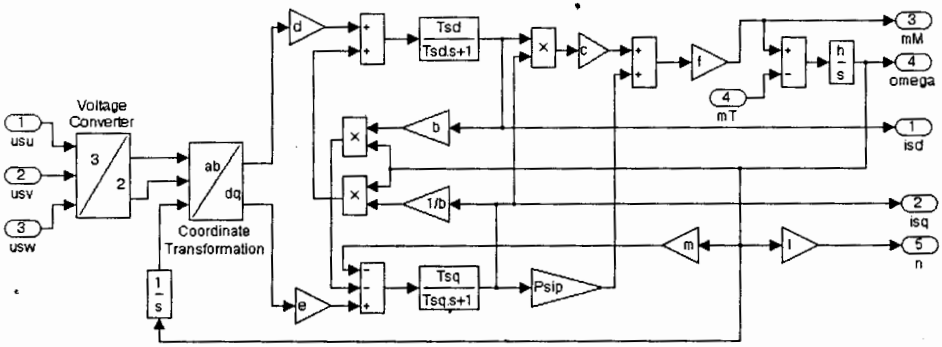
$$m_M = \frac{3}{2} z_p \left(\psi_{sd} i_{sq} - \psi_{sq} i_{sd} \right)$$
$$= \frac{3}{2} z_p \left[\psi_p i_{sq} + i_{sd} i_{sq} \left(L_{sd} - L_{sq} \right) \right] \quad (10.19)$$

Với hệ phương trình (10.17), (10.19) ta xây dựng sơ đồ khối ở hình 10.19.



Hình 10.19 *Mô hình MĐĐB-KTVC trên hệ tọa độ dq (hệ tọa độ từ thông cực)*

Tương tự MĐĐB (hình 10.15), với sơ đồ khối ở hình 10.19 ta đã có thể xây dựng mô hình SIMULINK cho MĐĐB-KTVC một cách dễ dàng nhờ sử dụng các khối có sẵn của thư viện. Mô hình SIMULINK ở hình 10.20 là ví dụ minh chứng cho cách đi này.



Hình 10.20 *Mô hình mô phỏng MĐĐB-KTVC trên hệ tọa độ từ thông cực, thực hiện bằng các khối có sẵn từ thư viện SIMULINK*

Để tăng tốc độ mô phỏng ta cũng có thể trên cơ sở các phương trình (10.17) và (10.19) để soạn thảo một *C-mex-File*, sử dụng dưới dạng hàm S, như đoạn mã nguồn C dưới đây:

```
/*********************************************************
 * PSMDQ.C: C-MEX-File to simulate a 3-phase permanent   *
 * excited synchronous machine in dq-(rotor) coordinates *
 * Syntax:  [sys,x0] = psmdq(t,x,u,Parametervektor)      *
 * Parametervektor = [Rs Lsd Lsq Psi_p zp]               *
 *          Rs    Stator resistance                      *
 *          Lsd   d-axis stator inductance               *
 *          Lsq   q-axis stator inductance               *
 *          Psi_p Permanent rotor flux                   *
 *          zp    Pole pair                              *
 *********************************************************/
/* The following #define is used to specify the name of
   this S-Function */
#define S_FUNCTION_NAME psmdq

/* Need to include simstruc.h for the definition of the
 * SimStruct and its associated macro definitions */
#include "simstruc.h"
#include <math.h>           /* needed for fabs-function */

/* Input variables: */
#define usa      u[0]       /* Stator voltage usa */
#define usb      u[1]       /* Stator voltage usb */
#define w_mech   u[2]       /* Rotor speed w_mech */

/* Output variables: */
#define mm       y[0]       /* Torque mM */
```

```
/* State variables: */
#define isdx    x[0]        /* Stator current alpha isa */
#define isqx    x[1]        /* Stator current alpha isb */
#define phix    x[2]        /* Elektrical pole angle */
#define disdx   dx[0]       /* Derivative of stator current
                               alpha isa */
#define disqx   dx[1]       /* Derivative of stator current
                               beta isb */
#define dphix   dx[2]       /* Derivative of elektrical
                               pole angle */
/* Parameter definition: */
#define Parameter ssGetArg(S,0)    /* Parameter vector */
#define Rs  mxGetPr(Parameter)[0] /* Stator resistance */
#define Lsd mxGetPr(Parameter)[1]   /* d-axis stator
                                       inductance */
#define Lsq mxGetPr(Parameter)[2]   /* q-axis stator
                                       inductance */
#define Psi_p mxGetPr(Parameter)[3] /* Pole flux */
#define zp    mxGetPr(Parameter)[4] /* Pole pair */

/* Mathematical function definition */
#define sign(x) (x==0.0?0.0:((x>0.0)?1.0:-1.0))
                                    /* Sign-Function */
#define TWO_PI 6.28318530717958621724893790087 6

/* mdlInitializeSizes - initialize the sizes array */
static void mdlInitializeSizes(S)
  SimStruct  *S;
  { ssSetNumContStates(S,3);      /* number of continuous
                                     states */
    ssSetNumDiscStates(S,1);      /* number of discrete
                                     states */
    ssSetNumInputs(S,3);          /* number of inputs */
    ssSetNumOutputs(S,3);         /* number of outputs */
    ssSetDirectFeedThrough(S,0); /* direct feedthrough
                                    flag */
    ssSetNumSampleTimes(S,1);     /* number of sample
                                     times */
    ssSetNumInputArgs(S,1);       /* number of input
                                     arguments */
    ssSetNumRWork(S,0);           /* number of real work
                                     vector elements */
    ssSetNumIWork(S,0);           /* number of integer work
                                     vector elements */
    ssSetNumPWork(S,0);           /* number of pointer work
                                     vector elements */
```

```
/* mdlInitializeSampleTimes - initialize the sample
 * times array */
static void mdlInitializeSampleTimes(S)
  SimStruct  *S;
  { ssSetSampleTimeEvent(S, 0, 0.0);
    ssSetOffsetTimeEvent(S, 0, 0.0);
  }

/* mdlInitializeConditions - initialize the states */
static void mdlInitializeConditions(x0, S)
  double *x0;
  SimStruct *S;
  { { /* Start values of state variables: */
      x0[0] = 0.0;           /* isdx */
      x0[1] = 0.0;           /* isqx */
      x0[2] = 0.0;           /* phix */
      x0[3] = 0.0;           /* discrete dummy state */
    }
  }

/* mdlOutputs - compute the outputs */
static void mdlOutputs(y, x, u, S, tid)
  double *y, *x, *u;
  SimStruct *S;
  int tid;
  { mm = 3.0*zp/2.0*isqx*(Psi_p+isdx*(Lsd-Lsq));
    y[1] = isdx;
    y[2] = isqx;
  }

/* mdlUpdate - perform action at major integration
 * time step */
static void mdlUpdate(x, u, S, tid)
  double *x, *u;
  SimStruct *S;
  int tid;
  { { if (fabs(phix) > TWO_PI)
      phix = phix-TWO_PI*sign(phix);
    }
  }

/* mdlDerivatives - compute the derivatives */
static void mdlDerivatives(dx, x, u, S, tid)
  double *dx, *x, *u;
  SimStruct *S;
  int tid;
  { double w_el,usd,usq;
```

```
    w_el = w_mech*zp;
    usd = usa*cos(phix)+usb*sin(phix);
                        /* Coordinate transformation */
    usq = usb*cos(phix)-usa*sin(phix);
                        /* from fixed into dq coordinates */
    dphix = w_el;       /* Calculating pole angle
                           from mechanical speed */
    disdx = usd/Lsd + w_el*Lsq/Lsd*isqx - Rs/Lsd*isdx;
    disqx = usq/Lsq - w_el*Lsd/Lsq*isdx - Rs/Lsq*isqx
                    - w_el*Psi_p/Lsq;
    }

/* mdlTerminate - called when the simulation is
 * terminated */
static void mdlTerminate(S)
  SimStruct *S;
  {
  }

#ifdef MATLAB_MEX_FILE    /* Is this file being compiled
                             as a MEX-file? */
#include "simulink.c"     /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"      /* Code generation registration
                             function */
#endif
```

Để kết thục mục 11.1.3 (mô hình hóa MĐĐB-KTVC), xin cung cấp tiếp cho bạn đọc một mô hình khác, một hàm S dưới dạng *m-File*. Trong đó, tác giả đã cắt bớt các phần thuyết minh bằng tiếng Anh cho gọn:

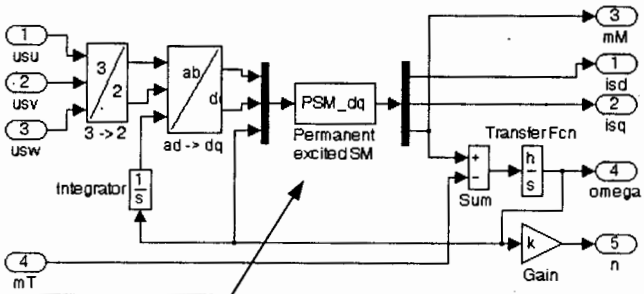
```
function [sys,x0,str,ts] =
                        PSM_dq(t,x,u,flag,Lsd,Lsq,Rs,mN,IN,pc)
Tsd  = Lsd/Rs; Tsq  = Lsq/Rs; IsqN = sqrt(2)*IN;
Psip = (2/3)*(mN/(pc*IsqN));
a = 1/Tsd; b = 1/Tsq; c = Lsq/Lsd; d = 1/c;
e = 1/Lsd; f = 1/Lsq; g = Psip/Lsq; h = Psip/Lsd;
switch flag,
  case 0
    [sys,x0,str,ts] = mdlInitializeSizes;
  case 1                          % Derivative states
    sys = mdlDerivatives(t,x,u,a,b,c,d,e,f,g);
  case 3                          % Return output
    sys = mdlOutputs(t,x,u,pc,h,Lsd,Lsq);
  case {2,4,9}                    % Dummy states
    sys = [];
    otherwise
```

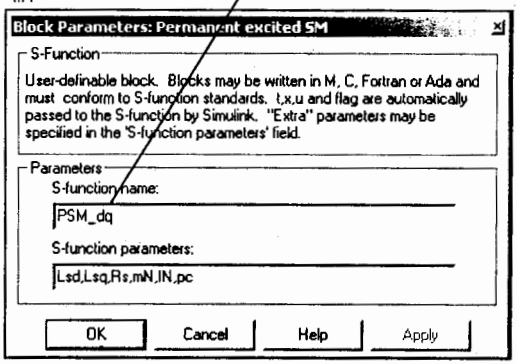
```
    error(['Error,unknow flag=',num2str(flag)]);
end

function [sys,x0,str,ts] = mdlInitializeSizes
sizes = simsizes;
  sizes.NumContStates = 2;                    % isd,isq
  sizes.NumDiscStates = 0;
  sizes.NumOutputs = 3;                       % isd,isq,mM
  sizes.NumInputs = 3;                        % usd,usq,ws
  sizes.DirFeedthrough = 0;
  sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = zeros(2,1);
str = [];
ts  = [0 0];

function sys = mdlDerivatives(t,x,u,a,b,c,d,e,f,g)
sys(1) = -a*x(1)+c*u(3)*x(2)+e*u(1);
sys(2) = -d*u(3)*x(1)-b*x(2)+f*u(2)-g*u(3);

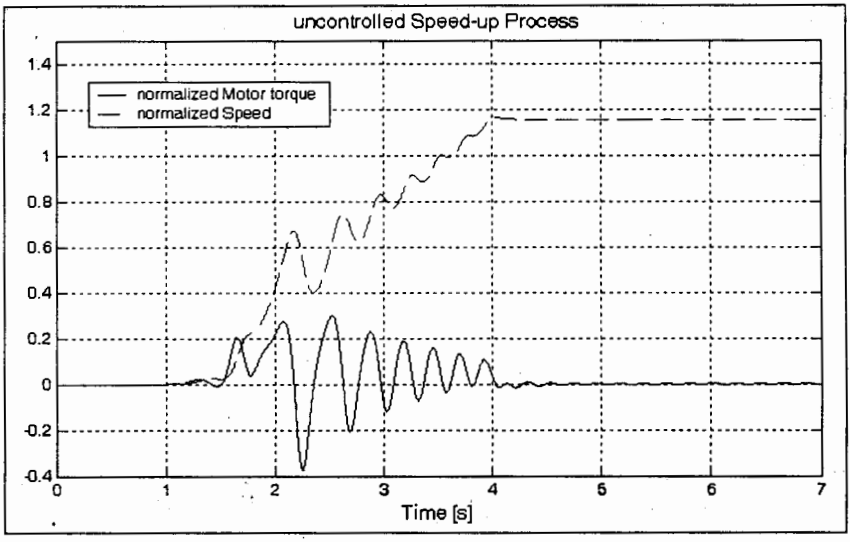
function sys = mdlOutputs(t,x,u,pc,h,Lsd,Lsq)
sys(1) = x(1);
sys(2) = x(2);
sys(3) = (3*pc/2)*((h+x(1))*Lsd*x(2)-Lsq*x(1)*x(2));
```





Hình 10.21 *Khai báo mô hình PSM_dq trong khối S-Function*

Hình 10.21 minh họa cách sử dụng hàm S có tên *PSM_dq* mà ta vừa mới soạn thảo. Trong hàm S đó, bạn đọc dễ dàng thấy rằng ta đã bỏ qua chưa cài đặt phần phương trình chuyển động và chỉ bổ sung thêm bên ngoài bằng các khối có sẵn của SIMULINK. Hình 10.22 minh họa một quá trình khởi động tự nhiên (không có ĐK) của MĐĐB-KTVC.

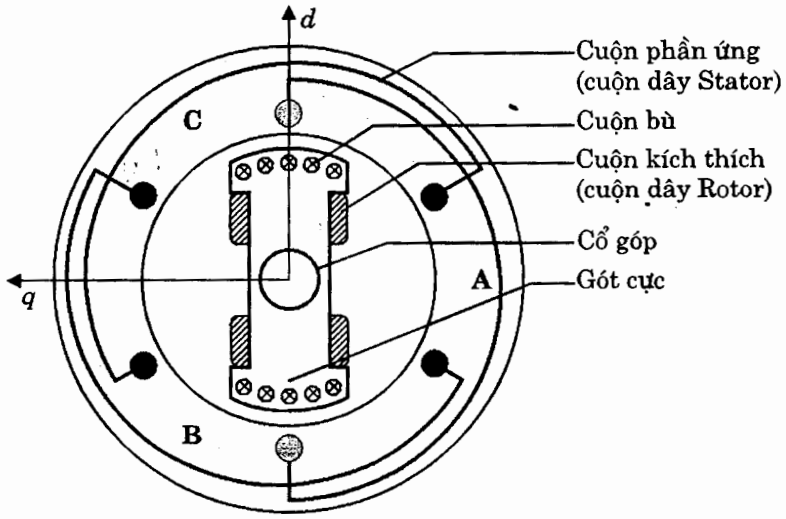


Hình 10.22 *Quá trình khởi động tự nhiên (không có điều khiển) của máy điện đồng bộ ba pha kích thích vĩnh cửu*

10.5 Mô hình máy điện đồng bộ ba pha kích thích độc lập

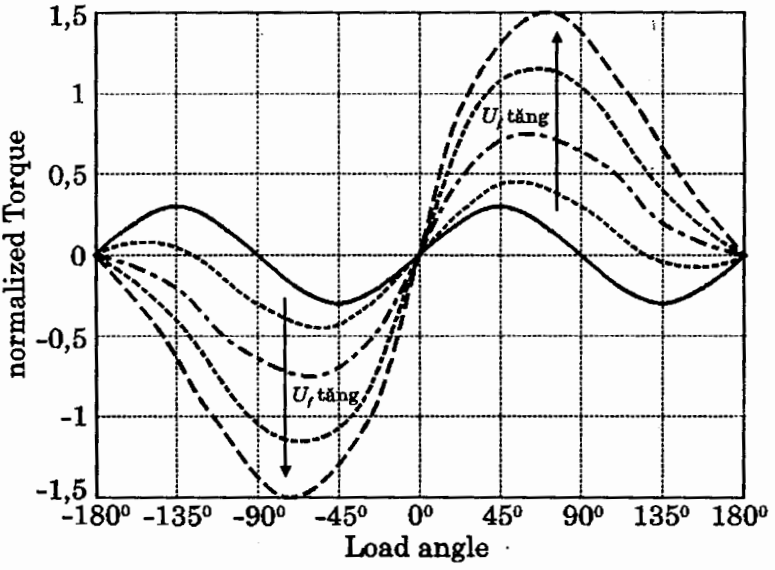
Máy điện đồng bộ kích thích độc lập (MĐĐB-KTĐL) có cấu tạo (mặt cắt) ở hình 10.23. ***Một mặt, so với MĐĐB-KTVC*** (hình 10.17), MĐĐB-KTĐL có hệ thống kích từ cũng nằm trên Rotor, nhưng không cấu tạo bởi nam châm vĩnh cửu mà bởi nam châm điện nuôi bởi nguồn áp một chiều đưa từ ngoài tới[1]. Kết cấu này là cần thiết khi máy điện có công suất rất lớn, không thể chế tạo các nam châm vĩnh cửu có kích cỡ lớn với mật độ từ năng cao. ***Mặt khác, so với MĐMC*** (hình 10.4) ta lại thấy có nhiều điểm giống nhau: Chỉ cần hoán đổi vị trí của hai cuộn dây phần ứng và cuộn dây kích thích của MĐMC là thấy rõ sự giống đó. Cuộn dây kích thích của MĐĐB-KTĐL không nằm trên Stator mà trên Rotor, nhận điện áp kích thích qua hai vành góp điện. Cuộn dây phần ứng của MĐĐB-KTĐL không nằm trên Rotor mà trên Stator, có kết cấu thích hợp để nhận nguồn áp nuôi xoay chiều ba pha chứ không phải áp một chiều.

[1] Vì thế được gọi là kích thích độc lập



Hình 10.23 *Nguyên lý kết cấu (mặt cắt) của MĐĐB ba pha kích thích độc lập*

Cũng giống như các loại động cơ đã qua, tác động tương hỗ giữa từ thông Stator (do nguồn xoay chiều ba pha tạo nên) và từ thông Rotor (do kích thích độc lập tạo nên) sẽ sản sinh ra mômen quay. Mômen đó sẽ gia tốc máy điện tới tốc độ quay đồng bộ, và sau đó tốc độ được giữ ổn định (không phụ thuộc tải) cho tới điểm lật chiều quay.



Hình 10.24 *Mômen quay của MĐĐB-KTĐL phụ thuộc góc mang tải*

Trước khi tới điểm lật, mặc dù tốc độ quay không đổi, góc giữa hai vector từ thông Stator và Rotor tăng tỷ lệ với phụ tải. Góc đó được gọi là *góc phụ tải*[1]. Khi góc phụ tải (hình 10.24) đạt tới giá trị lớn hơn ½ bước cực[2] (hoặc > 90^0 điện), mômen quay bắt đầu giảm. Nếu mômen của động cơ không còn đủ lớn để thắng mômen tải, xuất hiện trạng thái lật tốc độ và động cơ sẽ bị lỡ bước quay. Đây là hiện tượng không chỉ tồn tại với MĐĐB-KTĐL, mà còn xảy ra với cả MĐĐB-KTVC. Trong trường hợp MĐĐB-KTĐL (với kết cấu cực lồi) còn có thêm các hiệu ứng khác, ví dụ: mômen phản kháng với hậu quả đẩy điểm lật về phía các góc nhỏ hơn. Hình 10.24 cho thấy tác dụng tăng mômen khi tăng kích thích của MĐĐB-KTĐL (tăng U_f).

Để mô tả toán học MĐĐB-KTĐL ta sẽ phải xuất phát từ phương trình điện áp của mạch điện Stator (mạch điện phần ứng), mạch điện Rotor (mạch điện kích thích) và của mạch điện cuộn bù. Ta giả thiết Stator có kết cấu tròn đều còn Rotor đối xứng qua hai trục dọc và trục ngang. Tương tự phương pháp mô tả T^4R của MĐĐB-KTVC, ta mô tả tựa theo hướng của từ thông Rotor. Để mô hình hóa ta cần đến một loạt tham số hoặc là khó biết, hoặc là không thể biết. Ví dụ: Hỗ cảm giữa cuộn Stator và cuộn bù, giữa cuộn Rotor và cuộn bù, hay giữa cuộn Stator và cuộn Rotor. Bằng các phương pháp đo đạc rất phong phú và khá phức tạp (đo ngắn mạch và hở mạch) ta có thể thu được các tham số của một sơ đồ thay thế tương ứng. Vấn đề này vượt quá khuôn khổ của cuốn sách và bạn đọc buộc phải tham khảo các tài liệu chuyên sâu. Trong sách này ta sẽ chỉ sử dụng các mô hình toán có sẵn.

Thông thường, để xây dựng mô hình ta phải sử dụng các đại lượng đã qua chuẩn hóa. Bảng sau đây tập hợp các đại lượng chuẩn được sử dụng.

Điện áp pha	$U_0 = \sqrt{2/3}U_n$	U_n	Điện áp danh định
Dòng pha	$I_0 = \sqrt{2}I_n$	I_n	Dòng danh định
Điện áp kích thích	$U_{f0} = I_{f0}R_f$	I_{f0}	Dòng kích không tải
Vận tốc góc danh định	$\omega_0 = 2\pi f_n$	f_n	Tần số danh định
Vận tốc góc cơ	$\omega_{m0} = \omega_0/z_p$	z_p	Số đôi cực
Từ thông	$\psi_0 = U_0/\omega_0$		
Công suất	$S_0 = (3/2)U_0I_0$		
Mômen quay	$M_0 = S_0/\omega_{m0}$		
Kháng phức	$Z_0 = U_0/I_0$		
Điện cảm	$L_0 = \psi_0/I_0$		

[1] Góc phụ tải: Load angle

[2] Bước cực do số đôi cực của hệ thống cuộn dây Stator quyết định

Với các đại lượng chuẩn từ bảng trên, MĐĐB-KTĐL được mô tả bởi hệ phương trình dưới đây.

$$
\begin{aligned}
u_{sd} &= R_s i_{sd} + s\psi_{sd} - \omega\psi_{sq} \\
u_{sq} &= R_s i_{sq} + s\psi_{sq} + \omega\psi_{sd} \\
u_f &= R_f\left(1+sT_{d0}^{'}\right)i_f + \left(X_d - X_d^{'}\right)\frac{U_{f0}}{U_0}sT_{d0}^{'}i_{sd} \\
\psi_{sd} &= \frac{X_d(s)}{\omega_0}i_{sd} + \frac{1/R_f}{1+sT_{d0}^{'}}\frac{U_0}{\omega_0 I_{f0}}u_f \\
\psi_{sq} &= \frac{X_q(s)}{\omega_0}i_{sq} \\
m_e &= \frac{3}{2}z_p\left(\psi_{sd}i_{sq} - \psi_{sq}i_{sd}\right) \\
\omega &= \frac{d\vartheta}{dt};\ \omega_m = \frac{\omega}{z_p}
\end{aligned}
\tag{10.20}
$$

$$
\begin{aligned}
X_d(s) &= \frac{\left(1+sT_d^{'}\right)\left(1+sT_d^{''}\right)}{\left(1+sT_{d0}^{'}\right)\left(1+sT_{d0}^{''}\right)}X_q \\
X_q(s) &= \frac{\left(1+sT_q^{''}\right)}{\left(1+sT_{q0}^{''}\right)}X_d
\end{aligned}
\tag{10.21}
$$

$$
\begin{aligned}
u_{shd} &= \omega_n\psi_{sd} - u_{fh}\frac{U_n}{U_{fn}}\sqrt{\frac{2}{3}} \\
i_{sd} &= u_{shd}\frac{1}{X_d^{''}} - i_{sd}^{''} - i_{sd}^{'} \\
i_{sq} &= \psi_{sq}\frac{\omega_n}{X_q^{''}} - i_{sq}^{''}
\end{aligned}
\tag{10.22}
$$

u_{shd} Điện áp từ trường chính phía Stator (trục d)
$i_{sd}^{'}$ Dòng Stator quá độ (trục d)
$i_{sd}^{''}$ Dòng Stator quá độ bậc 2 (trục d)
$i_{sq}^{''}$ Dòng Stator quá độ bậc 2 (trục q)
u_{fh} Điện áp kích từ thông chỉnh

Trên cơ sở các phương trình mô tả MĐĐB-KTĐL ta viết *C-mex-File* dưới đây:

```
/*********************************************************
 * E_SMdq.C: C-MEX-File to simulate an external excited *
 * synchonous 3-phase machines in rotor coordinates     *
 * Syntax: [sys,x0] =                                   *
 *       e_smdq(t,x,u,RatedValues,Parameter)            *
 *********************************************************/
#define S_FUNCTION_NAME E_SMdq
/* Need to include simstruc.h for the definition of the
   SimStruct and its associated macro definitions */
#include "simstruc.h"
#include <math.h>

#define sign(x) (x==0.0?0.0:((x>0.0)?1.0:-1.0))
                          /* Definition of Sign-Function */
#define TWO_PI 6.28318530717958621724893790087 6

/* Block parameters */
#define RatedValues ssGetArg(S,0)
#define Parameter   ssGetArg(S,1)

/* Input variables */
#define ua      u[0] // Stator voltage alpha
#define ub      u[1] // Stator voltage beta
#define uE      u[2] // Exciting voltage
#define omega_m u[3] // Mechanical speed

/* State variables */
#define theta x[0]   // Stator angle theta
#define psid  x[1]   // Stator flux d-axis
#define psiq  x[2]   // Stator flux q-axis
#define idE   x[3]   // Mutual current
#define id2   x[4]   // Subtransient stator current d-axis
#define iq2   x[5]   // Subtransient stator current q-axis
#define id1   x[6]   // Transient stator current d-axis
#define uEh   x[7]   // Exciting main voltage
#define Xd_1  x[10]  // Factor 1/Xd
#define Ge    x[11]  // 1/Rf
#define U_fac x[12]  // Voltage factor Un/UEn*sqrt(2/3)
#define hfac1 x[13]  // Axiliary factor (Xd-Xd1)*IE0/U0
#define hfac2 x[14]  // Axiliary factor w_n*(1/Xq2-1/Xq)
#define hfac3 x[15]  // Axiliary factor w_n/Xq
#define hfac4 x[16]  // Axiliary factor 1/Xd2-1/Xd1
#define hfac5 x[17]  // Axiliary factor 1/Xd1-1/Xd
#define w_n   x[18]  // Rated frequency
```

```
/* Derivatives of state variables */
#define d_theta dx[0]
#define d_psid  dx[1]
#define d_psiq  dx[2]
#define d_idE   dx[3]
#define d_id2   dx[4]
#define d_iq2   dx[5]
#define d_id1   dx[6]
#define d_uEh   dx[7]

/* mdlInitializeSizes - initialize the sizes array */
static void mdlInitializeSizes(S)
  SimStruct  *S;
  { ssSetNumContStates(S,8);      /* number of continuous
                                     states */
    ssSetNumDiscStates(S,11);     /* number of discrete
                                     states */
    ssSetNumInputs(S,4);          /* number of inputs */
    ssSetNumOutputs(S,6);         /* number of outputs */
    ssSetDirectFeedThrough(S,0); /* direct feedthrough
                                     flag */
    ssSetNumSampleTimes(S,1);     /* number of sample
                                     times */
    ssSetNumInputArgs(S,2);       /* number of input
                                     arguments */
    ssSetNumRWork(S,0);           /* number of real work
                                     vector elements */
    ssSetNumIWork(S,0);           /* number of integer work
                                     vector elements */
    ssSetNumPWork(S,0);           /* number of pointer work
                                     vector elements */
  }

/* mdlInitializeSampleTimes - initialize the sample
 * times array */
static void mdlInitializeSampleTimes(S)
  SimStruct  *S;
  { ssSetSampleTimeEvent(S, 0, 0.0);
    ssSetOffsetTimeEvent(S, 0, 0.0);
  }

/* mdlInitializeConditions - initialize the states */
static void mdlInitializeConditions(x0, S)
  double *x0;
  SimStruct *S;
  { { int i;
      double UE0,IE0,fn,axilvar,Xq2,Xq,Un,Xd,Xd1,Xd2;
```

```
        Un = mxGetPr(RatedValues)[0];
        fn = mxGetPr(RatedValues)[1];
        UE0 = mxGetPr(RatedValues)[2];
        IE0 = mxGetPr(RatedValues)[3];
        Xd = mxGetPr(Parameter)[1];
        Xd1 = mxGetPr(Parameter)[2];
        Xd2 = mxGetPr(Parameter)[3];
        Xq = mxGetPr(Parameter)[4];
        Xq2 = mxGetPr(Parameter)[5];
      for (i=0;i<9;i++)
        x0[i] = 0.0;            /* Initial states: zeros */
        axilvar = sqrt(2.0/3.0) * Un;
        x0[12] = axilvar / UE0;
        x0[18] = TWO_PI * fn;
        x0[11] = IE0/UE0;
        x0[13] = (Xd - Xd1) * IE0 / axilvar;
        x0[14] = x0[18] * (1.0 / Xq2 - 1.0 / Xq);
        x0[15] = x0[18] / Xq;
        x0[16] = 1.0 / Xd2 - 1.0 / Xd1;
        x0[10] = 1.0 / Xd;
        x0[17] = 1.0 / Xd1 - x0[10];
      }
    }

  /* mdlOutputs - compute the outputs */
  static void mdlOutputs(y, x, u, S, tid)
    double *y, *x, *u;
    SimStruct *S;
    int tid;
  { {
    double zp,ushd,id,iq,did1,did2,diq2,sintheta,costheta;
  // Reading block parameter:
    zp =  mxGetPr(RatedValues)[4];
  // Computing axiliary variables:
    ushd = w_n * psid - uEh * U_fac;
    diq2 = (psiq * hfac2 - iq2);
    did2 = (ushd * hfac4 - id2);
    did1 = (ushd * hfac5 - id1);
    id = ushd * Xd_1 + did1 + did2;
    iq = psiq * hfac3 + diq2;
    sintheta = sin(theta);
    costheta = cos(theta);
  // Computing output variables:
    if ((u[0] == 1.23e-7) || (u[1] == 1.23e-7))
    { id = 0.0;
      iq = 0.0;
    }
```

```
    y[0] = 1.5 * zp * (psid * iq - psiq * id);  // Torque
    y[1] = id * costheta - iq * sintheta;       // is_alpha
    y[2] = id * sintheta + iq * costheta;       // is_beta
    y[3] = psid * costheta - psiq * sintheta;   // psi_alpha
    y[4] = psid * sintheta + psiq * costheta;   // psi_beta
    y[5] = uEh * Ge - (id * hfac1 - idE); // Exciting current
    }
}

/* mdlUpdate - perform action at major integration
 * time step */
static void mdlUpdate(x, u, S, tid)
  double *x, *u;
  SimStruct *S;
  int tid;
  { { if (fabs(theta) > TWO_PI)    // Limit: -2PI<theta<2PI
      theta = theta - TWO_PI * sign(theta);
    }
  }

/* mdlDerivatives - compute the derivatives */
static void mdlDerivatives(dx, x, u, S, tid)
  double *dx, *x, *u;
  SimStruct *S;
  int tid;
{ { double ud,uq,Td1,Td2,Tq2,zp,Rs,Td01,omega;
    double ushd,id,iq,did1,did2,diq2;
    double sintheta,costheta;
    sintheta = sin(theta);
    costheta = cos(theta);
// Coordinate transformation into rotor-fixed coordinates
    ud = ua * costheta + ub * sintheta;
    uq = ub * costheta - ua * sintheta;

// Reading block parameters:
    zp =  mxGetPr(RatedValues)[4];
    Rs = mxGetPr(Parameter)[0];
    Td01 = mxGetPr(Parameter)[6];
    Td1 = mxGetPr(Parameter)[7];
    Td2 = mxGetPr(Parameter)[8];
    Tq2 = mxGetPr(Parameter)[9];

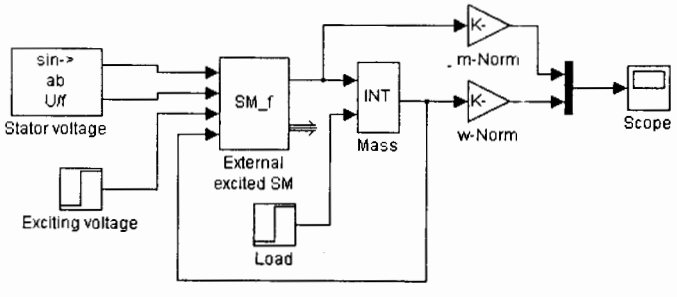
// Computing axiliary variables:
    omega = zp * omega_m; // Computing elektrical frequency
    ushd = w_n * psid - uEh * U_fac;
    diq2 = (psiq * hfac2 - iq2);
    did2 = (ushd * hfac4 - id2);
```

```
    did1 = (ushd * hfac5 - id1);
    id = ushd * Xd_1 + did1 + did2;
    iq = psiq * hfac3 + diq2;

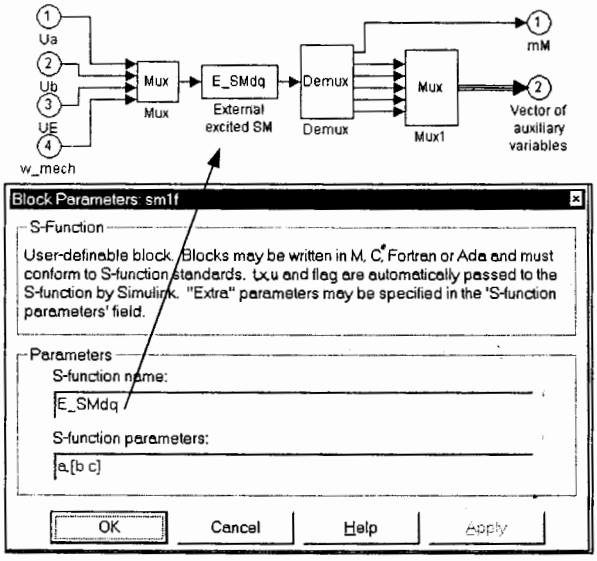
// Computing derivatives:
    d_idE = (id * hfac1 - idE) / Td01;
    d_iq2 = diq2 / Tq2;
    d_id2 = did2 / Td2;
    d_id1 = did1 / Td1;
    d_uEh = (uE - uEh) / Td01;
    d_psiq = uq - Rs * iq - omega * psid;
    d_psid = ud - Rs * id + omega * psiq;
    d_theta =  omega;
  }
}

/* mdlTerminate - called when the simulation is
 * terminated */
static void mdlTerminate(S)
  SimStruct *S;
  {
  }
#ifdef   MATLAB_MEX_FILE /* Is this file being compiled
                            as a MEX-file? */
#include "simulink.c"    /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"     /* Code generation registration
                            function */
#endif
```

Để thử nghiệm ta hãy bổ sung các khối còn thiếu thành sơ đồ MĐĐB-KTĐL nối trực tiếp với lưới điện như hình 10.25 dưới đây. Cách thức khai báo sử dụng *C-mex-File* vừa soạn thảo được nhắc lại ở hình 10.26.

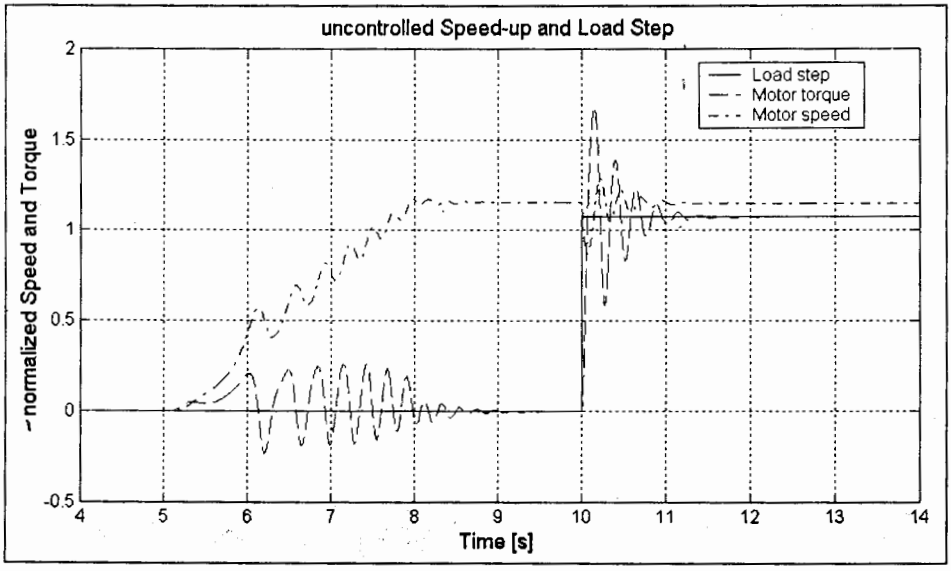


Hình 10.25 *Mô phỏng máy điện đồng bộ kích thích độc lập nối trực tiếp với lưới điện và không sử dụng thiết bị điều khiển*



Hình 10.26 *Khai báo mô hình E_SMdq trong khối S-Function*

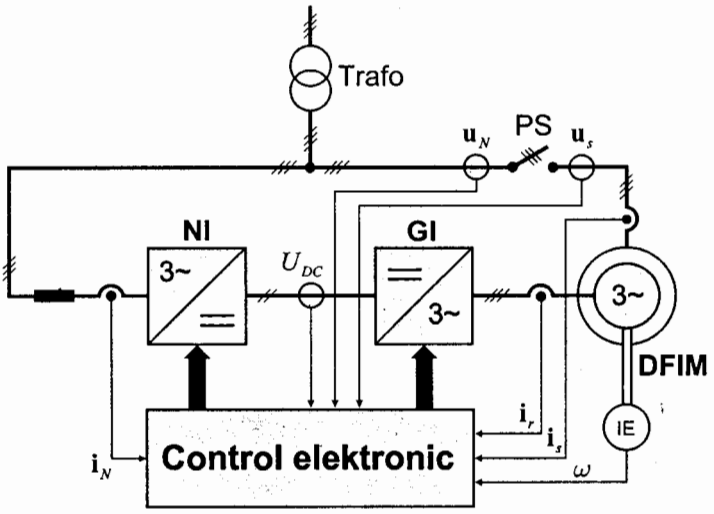
Kết quả mô phỏng kiểm tra *C-mex-File* vừa biên soạn *E_SMdq* theo sơ đồ ở hình 10.25 được giới thiệu ở hình 10.27.



Hình 10.27 *Quá trình khởi động và đóng tải (sau 10s) của một động cơ đồng bộ kích thích độc lập, nối trực tiếp với lưới và không qua thiết bị điều khiển*

10.6 Máy điện không đồng bộ nguồn kép[1]

Đã một thời, nhờ có khả năng cung cấp điện áp đồng thời cả hai phía Stator và Rotor, máy điện dị bộ Rotor dây quấn (MĐDB-RDQ) được sử dụng nhiều trong các ứng dụng truyền động có ĐK tốc độ trong phạm vi hẹp. Hiện tại, do các loại MĐDB Rotor lồng sóc, hay MĐĐB-KTVC chiếm ưu thế tuyệt đối trong các hệ truyền động có ĐK tốc độ, MĐDB-RDQ đã bị đẩy lùi khỏi các ứng dụng truyền động. Tuy nhiên, MĐDB-RDQ ngày càng được sử dụng nhiều trong các hệ thống phát điện chạy sức gió nhờ ưu thế: Thiết bị ĐK sử dụng van bán dẫn công suất lớn (điện tử công suất) được đặt ở phía Rotor (hình 10.28) và vì vậy chỉ cần được thiết kế với công suất cỡ ≈1/3 công suất của máy phát, giá thành của hệ thống được hạ xuống rất nhiều.



Hình 10.28 *Sơ đồ khối hệ thống phát điện chạy sức gió sử dụng MĐDB-RDQ*
Các ký hiệu: u_N, u_s: điện áp lưới, điện áp Stator; U_{DC}: điện áp mạch một chiều trung gian; i_r, i_s, i_N: dòng Rotor, Stator và phía lưới; Trafo: biến thế; PS: Power Switch; NI, GI: net side, generator side inverter; DFIM: Doubly-fed induction motor; IE: incremental encoder

MĐDB-RDQ được mô tả bởi các phương trình trên hệ tọa độ tựa theo hướng của vector điện áp lưới $\mathbf{u}_N$ sau đây:

$$\begin{aligned}\mathbf{u}_s &= R_s\mathbf{i}_s + \frac{d\psi_s}{dt} + j\omega_s\psi_s \\ \mathbf{u}_r &= R_r\mathbf{i}_r + \frac{d\psi_r}{dt} + j\omega_r\psi_r\end{aligned} \tag{10.23}$$

[1] Còn gọi là máy điện dị bộ Rotor dây quấn (MĐDB-RDQ): Doubly-fed Induction Machines

Việc lựa chọn vector điện áp lưới $\mathbf{u}_N$ làm hướng tựa xuất phát từ hai nguyên nhân chính như sau:

- Hệ thống máy phát trước khi phát điện lên lưới cần phải được hòa đồng bộ với lưới, tức là với $\mathbf{u}_N$.
- Sau khi hòa đồng bộ, Stator của máy phát được nối với lưới và vì vậy:

$$\mathbf{u}_N = \mathbf{u}_s \approx \frac{d\boldsymbol{\psi}_s}{dt} \quad \Rightarrow \quad \mathbf{u}_N \approx j\omega_s \boldsymbol{\psi}_s \tag{10.24}$$

Nghĩa là, vector điện áp lưới và vector từ thông Stator của máy phát luôn vuông góc với nhau, rất thuận lợi cho việc mô hình hóa. Mặt khác, thiết bị ĐK được đặt ở phía Rotor và ta có cơ hội để sử dụng dòng Rotor làm biến ĐK trạng thái của đối tượng MĐDB-RDQ. Vì vậy, trong mô hình (10.23) ta sẽ tìm cách thông qua phương trình từ thông:

$$\begin{aligned} \boldsymbol{\psi}_s &= \mathbf{i}_s L_s + \mathbf{i}_r L_m \\ \boldsymbol{\psi}_r &= \mathbf{i}_s L_m + \mathbf{i}_r L_r \end{aligned} \tag{10.25}$$

khử dòng Stator $\mathbf{i}_s$ và từ thông Rotor $\boldsymbol{\psi}_r$, giữ lại dòng Rotor $\mathbf{i}_r$ và từ thông Stator $\boldsymbol{\psi}_s$, đồng thời viết lại dưới dạng phương trình thành phần.

$$\begin{aligned} \frac{di_{rd}}{dt} &= -\frac{1}{\sigma}\left(\frac{1}{T_r} + \frac{1-\sigma}{T_s}\right) i_{rd} + \omega_r i_{rq} + \frac{1-\sigma}{\sigma}\left(\frac{1}{T_s}\psi'_{sd} - \omega\psi'_{sq}\right) \\ &\quad + \frac{1}{\sigma L_r} u_{rd} - \frac{1-\sigma}{\sigma L_m} u_{sd} \\ \frac{di_{rq}}{dt} &= -\omega_r i_{rd} - \frac{1}{\sigma}\left(\frac{1}{T_r} + \frac{1-\sigma}{T_s}\right) i_{rq} + \frac{1-\sigma}{\sigma}\left(\frac{1}{T_s}\psi'_{sq} + \omega\psi'_{sd}\right) \\ &\quad + \frac{1}{\sigma L_r} u_{rq} - \frac{1-\sigma}{\sigma L_m} u_{sq} \\ \frac{d\psi'_{sd}}{dt} &= \frac{1}{T_s} i_{rd} - \frac{1}{T_s}\psi'_{sd} + \omega_s \psi'_{sq} + \frac{1}{L_m} u_{sd} \\ \frac{d\psi'_{sq}}{dt} &= \frac{1}{T_s} i_{rq} - \omega_s \psi'_{sd} - \frac{1}{T_s}\psi'_{sq} + \frac{1}{L_m} u_{sq} \end{aligned} \tag{10.26}$$

Để hoàn thiện mô hình ta bổ sung thêm phương trình mômen (đặc trưng cho công suất hữu công) và phương trình hệ số công suất $\cos\varphi$ (đặc trưng cho công suất vô công) của máy phát:

$$m_M = -\frac{3}{2} z_p \frac{L_m}{L_s} \psi_{sq} i_{rd} = -\frac{3}{2} z_p (1-\sigma) L_r \psi'_{sq} i_{rd} \tag{10.27}$$

$$\cos\varphi = \frac{i_{sd}}{|\mathbf{i}_s|} = \frac{i_{sd}}{\sqrt{i_{sd}^2 + i_{sq}^2}} \tag{10.28}$$

Qua hai công thức (10.27), (10.28) ta có thể thấy rõ: Có thể sử dụng hai thành phần i_{rd}, i_{rq} của vector dòng $\mathbf{i}_r$ làm đại lượng ĐK công suất hữu công và vô công của máy phát một cách rất lợi hại. Hệ phương trình (10.26), (10.27) được sử dụng chủ yếu làm mô hình đối tượng để thiết kế hệ thống ĐK của máy phát.

Cũng có thể dễ dàng thu được mô hình trên hệ tọa độ cố định với Stator (hệ $\alpha\beta$) tương tự như các loại máy điện ở các mục trước.

$$\begin{aligned}
\frac{di_{s\alpha}}{dt} &= -\frac{1}{\sigma T_s} i_{s\alpha} + \frac{L_m}{\sigma L_s T_r} i_{r\alpha} + \omega \frac{1}{\sigma}\left[(1-\sigma) i_{s\beta} + \frac{L_m}{L_s} i_{r\beta}\right] \\
&\quad + \frac{1}{\sigma L_s} u_{s\alpha} - \frac{L_m}{\sigma L_s L_r} u_{r\alpha} \\
\frac{di_{s\beta}}{dt} &= -\frac{1}{\sigma T_s} i_{s\beta} + \frac{L_m}{\sigma L_s T_r} i_{r\beta} - \omega \frac{1}{\sigma}\left[(1-\sigma) i_{s\alpha} + \frac{L_m}{L_s} i_{r\alpha}\right] \\
&\quad + \frac{1}{\sigma L_s} u_{s\beta} - \frac{L_m}{\sigma L_s L_r} u_{r\beta} \\
\frac{di_{r\alpha}}{dt} &= \frac{L_m}{\sigma L_r T_s} i_{s\alpha} - \frac{1}{\sigma T_r} i_{r\alpha} - \omega \frac{1}{\sigma}\left[\frac{L_m}{L_r} i_{s\beta} + i_{r\beta}\right] - \frac{L_m}{\sigma L_s L_r} u_{s\alpha} + \frac{1}{\sigma L_r} u_{r\alpha} \\
\frac{di_{r\beta}}{dt} &= \frac{L_m}{\sigma L_r T_s} i_{s\beta} - \frac{1}{\sigma T_r} i_{r\beta} + \omega \frac{1}{\sigma}\left[\frac{L_m}{L_r} i_{s\alpha} + i_{r\alpha}\right] - \frac{L_m}{\sigma L_s L_r} u_{s\beta} + \frac{1}{\sigma L_r} u_{r\beta}
\end{aligned} \tag{10.29}$$

Chuyển hệ (10.29) sang miền ảnh Laplace (s: toán tử Laplace) ta có:

$$\begin{aligned}
i_{s\alpha} &= \frac{\sigma T_s}{1+s\sigma T_s}\left\{\frac{L_m}{\sigma L_s T_r} i_{r\alpha} + \frac{\omega}{\sigma}\left[(1-\sigma) i_{s\beta} + \frac{L_m}{L_s} i_{r\beta}\right] + \frac{1}{\sigma L_s} u_{s\alpha} - \frac{L_m}{\sigma L_s L_r} u_{r\alpha}\right\} \\
i_{s\beta} &= \frac{\sigma T_s}{1+s\sigma T_s}\left\{\frac{L_m}{\sigma L_s T_r} i_{r\beta} - \frac{\omega}{\sigma}\left[(1-\sigma) i_{s\alpha} + \frac{L_m}{L_s} i_{r\alpha}\right] + \frac{1}{\sigma L_s} u_{s\beta} - \frac{L_m}{\sigma L_s L_r} u_{r\beta}\right\} \\
i_{r\alpha} &= \frac{\sigma T_r}{1+s\sigma T_r}\left[\frac{L_m}{\sigma L_r T_s} i_{s\alpha} - \frac{\omega}{\sigma}\left(\frac{L_m}{L_r} i_{s\beta} + i_{r\beta}\right) - \frac{L_m}{\sigma L_s L_r} u_{s\alpha} + \frac{1}{\sigma L_r} u_{r\alpha}\right] \\
i_{r\beta} &= \frac{\sigma T_r}{1+s\sigma T_r}\left[\frac{L_m}{\sigma L_r T_s} i_{s\beta} + \frac{\omega}{\sigma}\left(\frac{L_m}{L_r} i_{s\alpha} + i_{r\alpha}\right) - \frac{L_m}{\sigma L_s L_r} u_{s\beta} + \frac{1}{\sigma L_r} u_{r\beta}\right]
\end{aligned} \tag{10.30}$$

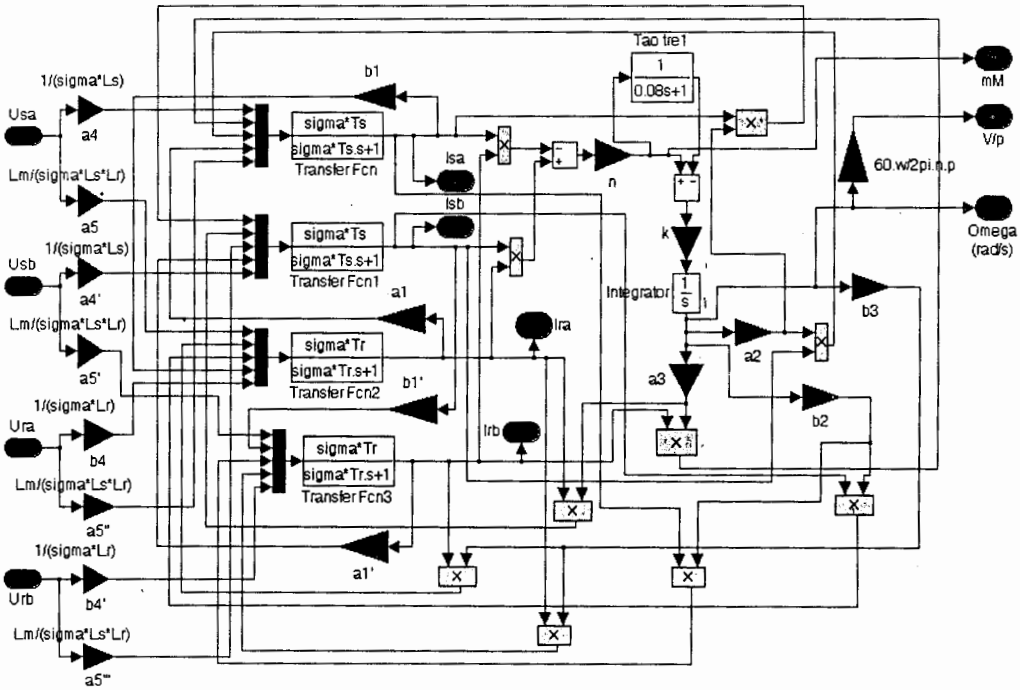
Mô hình đầy đủ trên hệ tọa độ $\alpha\beta$ sẽ bao gồm cả các phương trình sau:

- *Phương trình từ thông*:
$$\begin{cases} \psi_{r\alpha} = L_m i_{s\alpha} + L_r i_{r\alpha} \\ \psi_{r\beta} = L_m i_{s\beta} + L_r i_{r\beta} \end{cases} \quad (10.31)$$

- *Phương trình mômen*:
$$m_M = \frac{3}{2} z_p \frac{L_m}{L_r} \left(\psi_{r\alpha} i_{s\beta} - \psi_{r\beta} i_{s\alpha} \right) \quad (10.32)$$

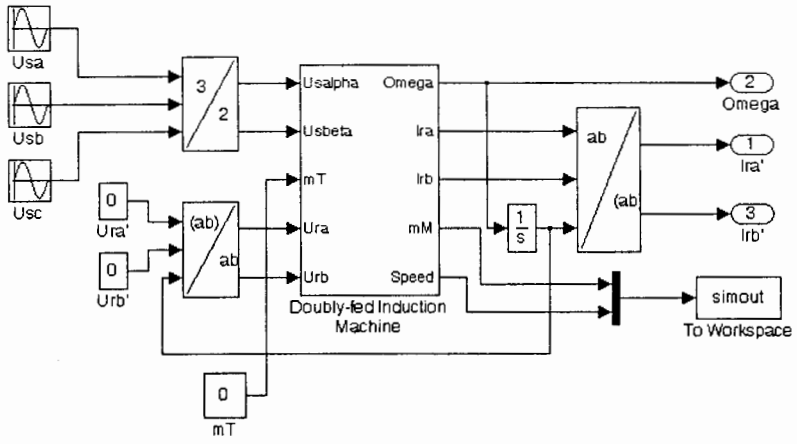
- *Phương trình chuyển động*:
$$m_M = m_T + \frac{J}{z_p} \frac{d\omega}{dt} \quad (10.33)$$

Một *C-mex-File* của mô hình (10.30) sẽ choán rất nhiều trang, nếu được in trọn vẹn ra đây. Vì vậy, ta tạm hài lòng với mô hình mô phỏng xây dựng bởi các khối có sẵn từ thư viện SIMULINK trong hình 10.29 dưới đây.

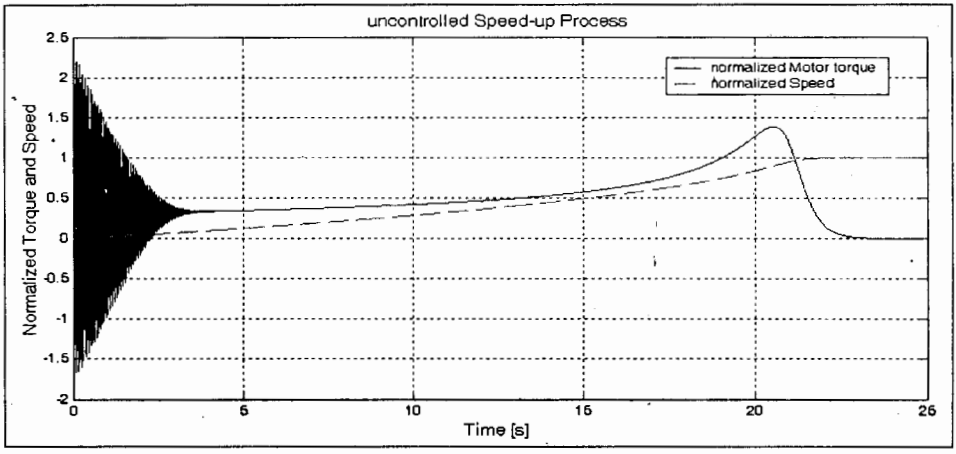


Hình 10.29 *Mô hình SIMULINK mô phỏng máy điện dị bộ nguồn kép trên hệ tọa độ $\alpha\beta$*

Đến đây, để bước đầu kiểm tra mô hình vừa xây dựng, ta gom hình 10.29 lại thành *Subsystem* (hình 10.30) và bổ sung thêm các nguồn nuôi cấp điện cho Stator, phía Rotor ta đặt điện áp bằng 0 (ứng với MĐDB Rotor ngắn mạch). Kết quả mô phỏng quá trình khởi động không có ĐK của một MĐDB-RDQ với công suất 690kW được giới thiệu ở hình 10.31.



Hình 10.30 *Mô hình SIMULINK mô phỏng máy điện dị bộ nguồn kép 620kW (Rotor bị nối ngắn mạch), nối trực tiếp với lưới điện ba pha 690V (không có ĐK)*



Hình 10.31 *Quá trình khởi động MĐDB-RDQ khi nối thẳng với lưới*

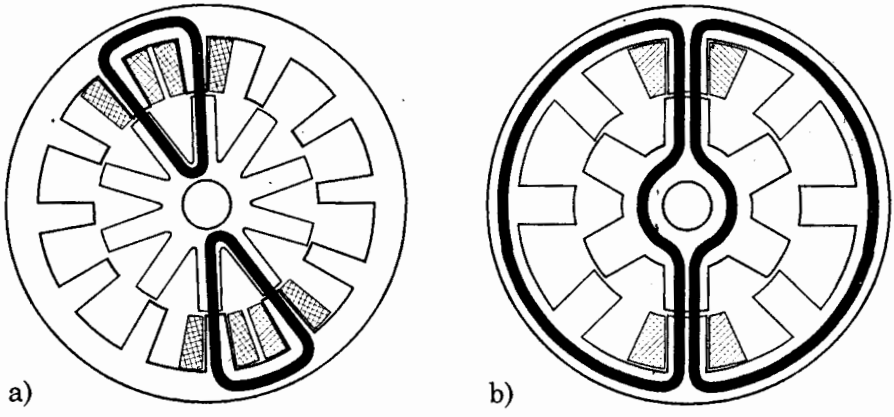
10.7 Máy điện từ kháng kiểu đóng ngắt[1]

Máy điện từ kháng là loại máy điện có nguyên lý hoạt động khác với tất cả các loại ta đã khảo sát ở các mục 10.1 – 10.6: Để tạo mômen quay, MĐTK không cần hệ thống kích từ, mà chỉ cần một hệ thống cảm kháng phân bố không đều trên toàn bộ phạm vi khe hở giữa Stator và Rotor.

[1] Máy điện từ kháng kiểu đóng ngắt (MĐTK): Switched Reluctance Machines

Để hiểu nguyên lý hoạt động của ĐCTK ta chỉ việc quan sát phương trình (10.19), phương trình mômen của ĐCĐB-KTVC. Mômen quay bao gồm hai thành phần: *Thành phần chính* do từ thông cực ψ_p (nam châm vĩnh cửu) gây nên, *thành phần phản kháng* do hiệu số $L_{sd} - L_{sq}$ gây nên. Rõ ràng là: Nếu máy điện hoàn toàn không có kích thích và $\psi_p = 0$, khi ấy máy điện vẫn có khả năng tạo ra mômen, và mômen đó càng lớn nếu chênh lệch từ kháng $L_{sd} - L_{sq}$ càng lớn. Đây chính là cơ sở vật lý của MĐTK, là loại máy điện không có hệ thống kích từ một cách rõ ràng.

Stator của MĐTK có cấu tạo bởi nhiều cực từ chứa các cuộn dây tập trung. Khác với cuộn dây của máy điện 3 pha, là loại máy với cuộn dây có thể phân tán tùy theo số đôi cực. Rotor của MĐTK không chứa cuộn dây và được chế tạo bằng sắt từ có xẻ răng (teeth), với tổng số răng bao giờ cũng ít hơn tổng số cực Stator. Hình 10.32 giới thiệu hai MĐTK, bên trái là *động cơ 3 pha với stator 12 cực và rotor 10 răng* (gọi tắt: loại 12/10), bên phải là *động cơ 4 pha với stator 8 cực và rotor 6 răng* (gọi tắt: loại 8/6). Để tạo chuyển động quay, các cuộn dây pha stator sẽ lần lượt được đóng ngắt nguồn phụ thuộc vị trí của rotor. Để có thông tin về vị trí của rotor, thông thường ta sẽ phải sử dụng khâu đo vị trí tuyệt đối.



Hình 10.32 *Động cơ từ kháng với (a) đường sức từ ngắn, và (b) đường sức từ dài*

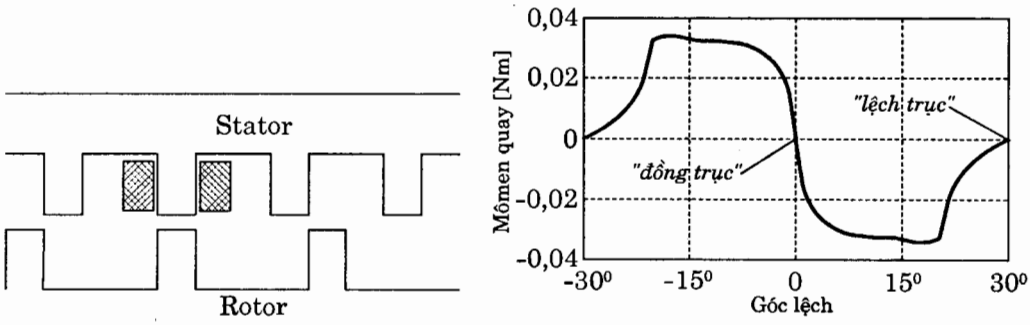
Mômen quay của MĐTK có đường phân bố trên bề mặt rotor lặp lại theo chu kỳ của răng. Trong mỗi chu kỳ đều có hai vị trí: *vị trí đồng trục* (cực có cuộn dây mang dòng - gọi là cực *active* - và răng đồng trục với nhau) và *vị trí lệch trục* (cực *active* ở vị trí giữa 2 răng). Mômen được tính theo:

$$m_M\left(\varphi, i_j\right) = \frac{1}{2} i_j^2 \frac{dL\left(\varphi, i_j\right)}{d\varphi} \tag{10.34}$$

Chỉ số j: Cuộn dây cực thứ j của Stator

Hình 10.33 minh họa vị trí đồng trục của loại động cơ 8/6. ở *vị trí lệch trục*, răng gần nhất với cực *active* sẽ chuyển động về phía cực *active* để đạt được

trạng thái đồng trục. Hình 10.34 minh họa phân bố mômen quay của động cơ 8/6, trong đó góc 0^0 chính là vị trí đồng trục của Stator và Rotor.



Hình 10.33 *Trái: Vị trí "đồng trục" của rotor và cực stator active. Phải: Mômen quay của 1 pha phụ thuộc góc lệch giữa cực và răng*

Xuất phát từ phương trình điện áp pha:

$$u_j = R i_j + \frac{d\psi_j}{dt} \tag{10.35}$$

để đơn giản, ta hãy bỏ qua điện trở R và viết:

$$u_j = L(\varphi)\frac{di_j}{dt} + i_j \frac{dL(\varphi)}{d\varphi}\omega \tag{10.36}$$

Trong (10.35), điện cảm L là một tham số phụ thuộc vị trí φ của rotor. Để tính công suất ta hãy nhân hai vế của (10.35) với dòng i:

$$u_j i_j = L i_j \frac{di_j}{dt} + i_j^2 \frac{dL}{d\varphi}\omega \tag{10.37}$$

hoặc:

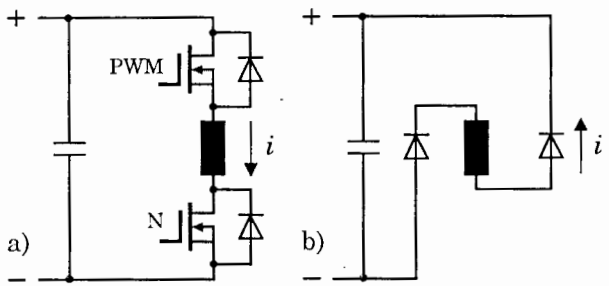
$$P_j = \frac{d}{dt}\left(\frac{1}{2} L i_j^2\right) + \frac{1}{2} i_j^2 \frac{dL}{d\varphi}\omega \tag{10.38}$$

Biểu thức thứ nhất ở vế phải của (10.37) đặc trưng cho thành phần từ năng tích trong cuộn dây pha. Biểu thức thứ hai của (10.37) mô tả cơ năng cung cấp ra trục động cơ. Từ đó ta có công thức tính mômen quay đã cho ở (10.34) và thấy rõ: Dấu của mômen – quyết định chế độ động cơ hay máy phát – hoàn toàn do dấu của $dL/d\varphi$ quyết định. Cũng theo (10.34), dấu của mômen hoàn toàn không phụ thuộc vào dấu của dòng. Dòng i chỉ có tác dụng ĐK biên độ của mômen quay.

Cần lưu ý rằng, dòng và áp trong các phương trình (10.34) - (10.38) là dòng áp của một cuộn dây Stator tích cực, φ là vị trí của Rotor so với cực Stator tích cực (cực *active*: cực có cuộn dây đang được cấp dòng).

Để ĐK biên độ của mômen, MĐTK phải được ĐK nhờ một vòng ĐC dòng. Thiết bị nghịch lưu thường được nuôi bởi nguồn áp một chiều, và đối với MĐTK - theo công thức (10.34) - chỉ cần dòng chảy theo một chiều cũng đủ để vận hành ở cả 4 góc ¼ (chế độ vận hành 4Q).

Nghịch lưu lý tưởng phải có khả năng đóng/ngắt dòng không có trễ. Để có thể ĐC dòng pha, có thể sử dụng hai van (hình 10.34, trái): van N phục vụ chọn pha, van PWM[1] có nhiệm vụ điều chế bề rộng xung áp đặt lên cuộn dây pha và nhờ đó dễ dàng ĐC dòng qua cuộn dây. Nhằm giảm tổn hao đóng/ngắt của van, từ năng tích lũy khi dòng chảy qua cuộn dây phải có khả năng được hoàn nguyên trở lại nguồn (hình 10.34, phải).



Hình 10.34 *Cuộn dây pha a) khi dẫn dòng, và b) khi nạp dòng ngược trở lại nguồn*

Để mô hình hóa MĐTK ta viết lại phương trình (10.35), phương trình điện áp của pha thứ j (pha tích cực) với sự ảnh hưởng của tất cả các pha còn lại:

$$u_j = Ri_j + \frac{d}{dt}\sum_{k=1}^{m}\psi_{kj} \tag{10.39}$$

Trong (10.39), j là chỉ số của cuộn dây đang được ta quan sát, m là số pha của Stator. Từ (10.39) ta thu được:

$$u_j = Ri_j + \frac{d}{dt}\sum_{k=1}^{m}\left[L_{kj}\left(i_k,\varphi\right)i_k\right] \tag{10.40}$$

Công thức (10.40) chỉ rõ: Điện cảm L_{jj} của cuộn dây pha thứ j, hỗ cảm L_{kj} giữa hai cuộn dây pha thứ k, j không chỉ phụ thuộc vào góc φ (đặc điểm phân bố không đều để tạo mômen), L_{jj} còn phụ thuộc vào dòng i_j và L_{kj} vào dòng i_k (đặc điểm bão hòa của cuộn cảm).

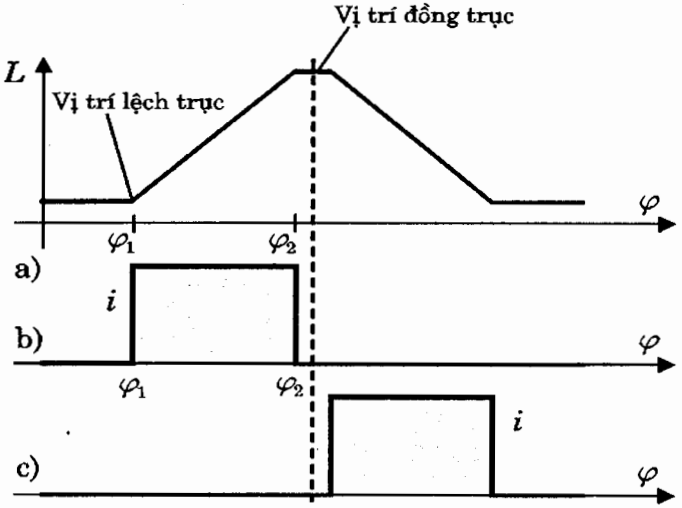
Tiếp tục khai triển (10.40) ta có:

$$u_j = Ri_j + \sum_{k=1}^{m}\left(i_k\frac{\partial L_{kj}}{\partial i_k}\frac{di_k}{dt} + L_{kj}\frac{di_k}{dt} + i_k\frac{\partial L_{kj}}{\partial\varphi}\omega\right) \tag{10.41}$$

Phương trình (10.41) là phương trình điện áp đầy đủ của một cuộn dây pha Stator và là cơ sở để ta mô hình hóa MĐTK. Từ (10.41) ta rút ra các nhận xét:

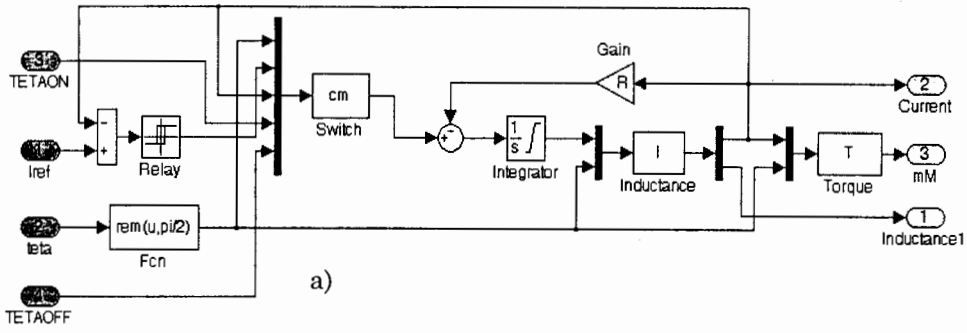
[1] PWM: Puls Width Modulation, điều chế bề rộng xung

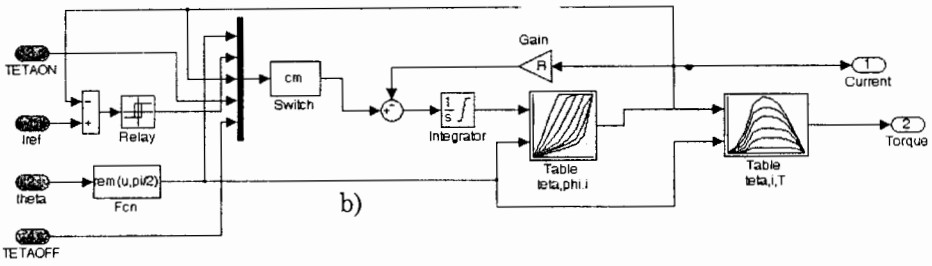
- Phương trình (10.41) là cơ sở để ta xây dựng mô hình SIMULINK cho 1 cuộn dây pha Stator. Nếu MĐTK có m pha, ta sẽ phải sử dụng m mô hình SIMULINK 1 pha giống nhau.
- Việc chuyển mạch giữa m mô hình SIMULINK 1 pha (chuyển mạch giữa m pha) được thực hiện phụ thuộc vào góc φ (tại vị trí có biến thiên điện cảm) để tạo mômen theo chế độ công tác (động cơ hay máy phát). Điều này được thực hiện trên cơ sở (10.34) và minh họa qua hình 10.35.



Hình 10.35 *Điện cảm L của MĐTK: a) Đặc tính L lý tưởng phụ thuộc vị trí rotor φ b) Dòng pha tạo chế độ động cơ; c) Dòng pha tạo chế độ máy phát*

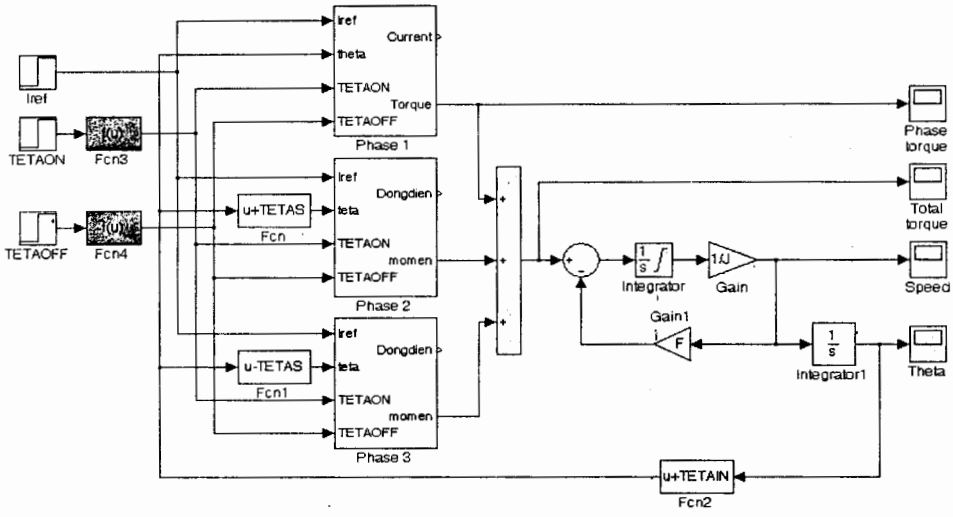
- Biểu thức thứ nhất nằm trong ngoặc của (10.41) mô tả đặc điểm bão hòa từ của máy điện. Nếu tạm coi điện cảm L_{jj} và L_{kj} không phụ thuộc dòng, biểu thức này sẽ bằng 0. Ta có mô hình SIMULINK tuyến tính (hình 10.36a). Nếu biết đường đặc tính từ hóa, ta có thể cất đặc tính vào một bảng tra (*Look-Up Table*, hay *Look-Up Table 2-D*) và xây dựng mô hình SIMULINK phi tuyến (hình 10.36b).





Hình 10.36 *Sơ đồ SIMULINK mô phỏng 1 pha của Stator: a) Mô hình tuyến tính (trang trước); b) Mô hình phi tuyến sử dụng Look-Up Table*

Ví dụ, để mô phỏng một MĐTK loại 6/4, tức là có 6 cực (≅ 3 pha) và 4 răng, ta phải sử dụng ba mô hình 1 pha (lấy từ hình 10.36a hoặc 11.36b) như sau:



Hình 10.37 *Sơ đồ SIMULINK mô phỏng máy điện từ kháng loại 6/4*

Dễ dàng thấy: Trong sơ đồ 10.37 ta có sử dụng khâu đo vị trí Rotor (góc `Theta`) và sau khi so sánh `Theta` với các góc `TETAON` (vị trí đóng mạch pha) `TETAOFF` (vị trí cắt mạch pha), sơ đồ sẽ đưa ra quyết định kích hoạt các pha. Tác dụng của `TETAON` và `TETAOFF` sẽ sáng tỏ thêm nếu ta quan sát kỹ hai phương án ở hai hình 10.36a,b. Trong hai phương án đó, khối `Switch` (chuyển mạch: ***S-Function*** có tên `cm`) sẽ trên cơ sở `Theta`, `TETAON` và `TETAOFF` để chọn biến đưa ra tính dòng i (theo công thức (10.41)) và tính mômen của pha (theo công thức (10.34)). Mômen tổng của MĐTK ở đầu ra sẽ được tính theo:

$$m_{M\Sigma} = \frac{1}{2}\sum_{j=1}^{3}\left[\frac{dL(\varphi, i_j)}{d\varphi} i_j^2\right] \qquad (10.42)$$

Để hiểu rõ hơn chức năng vừa mô tả của khối `Switch`, bạn đọc hãy xem đoạn mã quan trọng nhất của *S-Function* `cm`: Đoạn chuyển mạch chọn đại lượng để ĐC dòng *i* (ĐC theo kiểu 2 điểm, bằng cách chuyển mạch van bán dẫn: khối `Relay`).

```
% This function allow to chose the commutation instants
% of the semeconductor
.
.
%=============================================================
% Function mdlOutputs performs the calculations.
%=============================================================
function sys = mdlOutputs(t,x,u)
global  V;
teta    = u(1);
e       = u(2);
i       = u(3);
TETAON  = u(4);
TETAOFF = u(5);
sys = 0;
if (teta >= TETAON) & (teta <= TETAOFF)
sys = e;
elseif (teta > TETAOFF)
    if (i > 0)
        sys = -V;
    elseif (i <= 0)
        sys = 0;
    end;
end;
% End of mdlOutputs.
```

Trong trường hợp mô hình phi tuyến (hình 10.36b) việc tìm giá trị dòng để rồi tính mômen dựa trên kết quả tra bảng đặc tính từ hóa. Trong trường hợp mô hình tuyến tính (hình 10.36a) ta phải sử dụng hai khối `Inductance` (*S-Function* `L`) để tìm dòng *i*, và khối `Torque` (*S-Function* `T`) để tính mômen của pha. Hai đoạn mã sau đây cũng là hai đoạn quan trọng nhất trong hai *script* đó.

- *S-Function* `L`

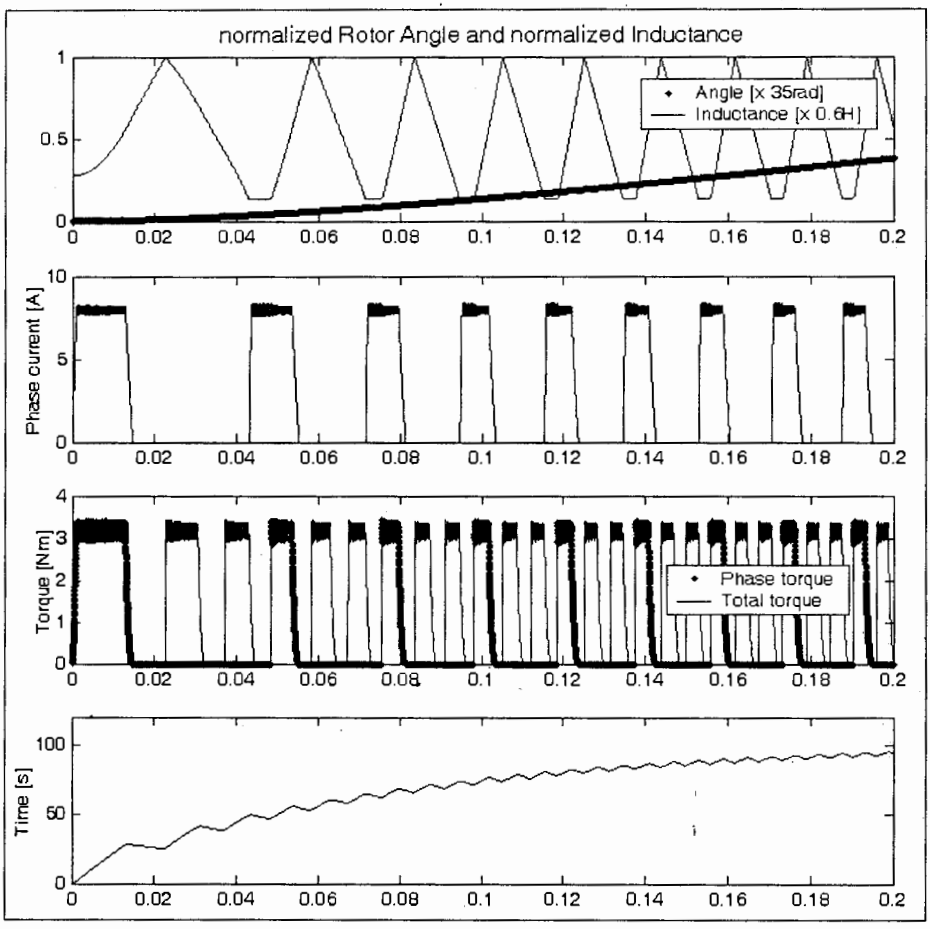
```
% This function has to output the current
% belong to flux and teta
.
.
%=============================================================
% Function mdlOutputs performs the calculations.
%=============================================================
function sys = mdlOutputs(t,x,u)
```

```
global TETAS TETAX TETAY TETAXY TETAON TETAOFF TETAQ
global V AUP BUP ADOWN BDOWN DL A B LMIN LMAX
flux = u(1);
teta = u(2);
if (teta >= 0) & (teta <= TETAX)
    sys = [flux/LMIN, LMIN];
elseif (teta > TETAX) & (teta <= TETAY)
    sys = [flux / (AUP * (teta - TETAX) + LMIN), ...
           (AUP * (teta - TETAX) + LMIN)];
elseif (teta > TETAY) & (teta <= TETAXY)
    sys = [flux / (ADOWN * (teta - TETAY) + LMAX), ...
           (ADOWN * (teta - TETAY) + LMAX)];
elseif (teta > TETAXY)
    sys = [flux / LMIN, LMIN];
end;
% End of mdlOutputs.
```

- *S-Function* T

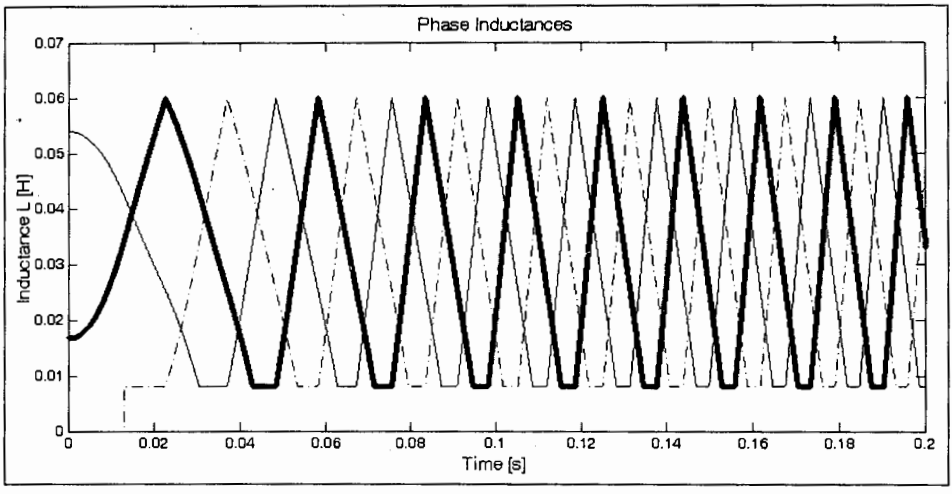
```
% This program computes the output torque in one phase
% with the input current and teta
.
.
%=============================================================
% Function mdlOutputs performs the calculations.
%=============================================================
function sys = mdlOutputs(t,x,u)
global TETAS TETAX TETAY TETAXY TETAON TETAOFF TETAQ
global V AUP BUP ADOWN BDOWN DL A B LMIN LMAX
i    = u(1);
teta = u(2);
if (teta > 0) & (teta <= TETAX)
    sys = 0;
elseif (teta > TETAX) & (teta <= TETAY)
    sys = 0.5 * (DL) * (i*i);
elseif (teta > TETAY) & (teta <= TETAXY)
    sys = -0.5 * (ADOWN) * (i*i);
elseif (teta > TETAXY)
    sys = 0;
end;
% End of mdlOutputs.
```

Các hình ảnh tiếp theo minh họa kết quả mô phỏng mô hình MĐTK theo sơ đồ ở hình 10.37, phương án tuyến tính (sơ đồ 1 pha theo hình 10.36a). Tại đây cũng cần lưu ý bạn đọc: Giống như nhiều ví dụ đã qua, ta còn phải soạn thảo một *script* với tác dụng khai báo các tham số của đối tượng ĐK và chế độ mô phỏng. *Script* đó sẽ được gọi trước khi bắt đầu quá trình mô phỏng.



Hình 10.38 *Kết quả mô phỏng MĐTK loại 6/4 với mô hình tuyến tính (từ trên xuống): 1. Đồ thị vị trí của Rotor và điện cảm tương ứng; 2. Dòng có ĐC (2 điểm) của 1 pha; 3. Mômen của 1 pha (đường đậm) và mômen tổng; 4. Tốc độ quay*

Đồ thị trên cùng của 10.38 cho ta thấy rất rõ đặc tính điện cảm phụ thuộc vị trí của Rotor (góc `Theta`) như hình 10.35 đã minh họa. Đồ thị ở hình tiếp theo là dòng điện chẩy qua cuộn dây pha. Dễ dàng thấy rằng dòng đã được bơm vào cuộn dây hoàn toàn phụ thuộc vào vị trí Rotor, tại sườn tăng của đồ thị điện cảm, nơi mà theo (10.34) mômen quay có dấu dương cần để gia tốc động cơ. Phần băm ở đỉnh xung dòng cho thấy tác dụng ĐK ổn định biên độ dòng (và do đó của mômen quay) của khâu ĐC 2 điểm (khối `Relay`). Phần mômen đóng góp của 1 pha (hình tiếp theo: nét đậm) minh họa thêm: Xung dòng pha chỉ đóng góp vào việc tạo mômen quay trong khoảng (mà theo (10.34)) thực sự thỏa mãn điều kiện $\partial L/\partial\varphi \neq 0$. Hình 10.39 minh họa phân bố điện cảm của ba cuộn dây pha Stator phụ thuộc vị trí Rotor.



Hình 10.39 *Phân bố điện cảm của ba cuộn dây pha (MĐTK loại 6/4) phụ thuộc vào vị trí của Rotor*

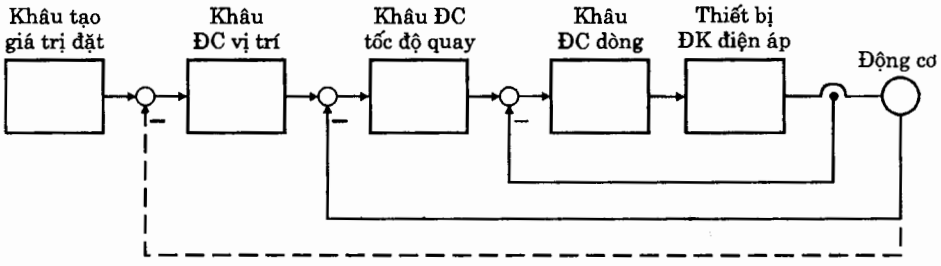
10.8 Tóm tắt nội dung chương 10

Sau khi đã nghiên cứu kỹ chương 10, người đọc cần nắm vững các nội dung sau đây:

1. Xây dựng mô hình toán của các loại máy điện như: Máy điện một chiều kích thích độc lập, máy điện dị bộ Rotor lồng sóc, máy điện dị bộ nguồn kép (còn gọi là máy điện dị bộ Rotor dây quấn), máy điện đồng bộ kích thích vĩnh cửu, máy điện đồng bộ kích thích độc lập và máy điện từ kháng kiểu đóng ngắt.
2. Chuyển mô hình toán của các loại động cơ kể trên sang mô hình trên nền MATLAB và SIMULINK, theo phương thức sơ đồ khối SIMULINK, phương thức *S-Function* kiểu *m-File* hay *C-mex-File* (sau khi thông dịch chuyển thành *.dll-File*). Qua đó rèn luyện kỹ năng tạo *S-Function* để tạo ra những khối SIMULINK mới.

11 Thư viện mô hình thiết bị biến đổi (điện tử công suất)

Từng nhánh của chuyển động phức hợp (hình 10.1) đều do một hệ thống truyền động sử dụng một động cơ điện thực hiện. Cấu trúc ĐK của hệ thống truyền động đơn thường có dạng như hình 11.1 dưới đây.



Hình 11.1 *Cấu trúc một hệ thống truyền động đơn, tạo một chuyển động nhánh*

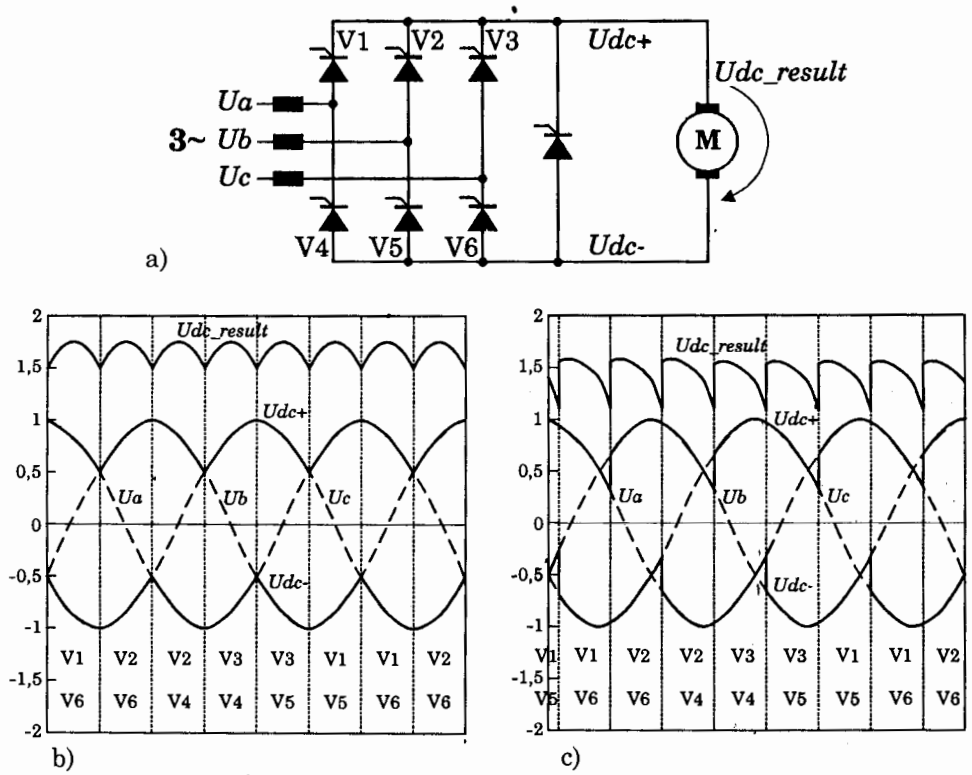
Cấu trúc có nhiều vòng ĐC phân cấp trên là một cấu trúc quen thuộc, với vòng trong cùng là vòng ĐC dòng có động học cao, nhằm mục đích áp đặt mômen quay (hay lực chuyển động) với tốc độ cao nhất. Vòng ngoài tiếp theo là vòng ĐC ổn định tốc độ quay, và ngoài cùng (có thể tồn tại, cũng có thể không) là vòng ĐC vị trí (hay góc). Điều ta cần chú ý trong cấu trúc là: Nằm giữa khâu ĐC dòng và động cơ bao giờ cũng là thiết bị tạo điện áp ĐK, bản chất là thiết bị điện tử công suất sử dụng van bán dẫn có dòng / áp lớn.

Cấu trúc trên cho ta thấy sự liên kết chặt chẽ giữa thiết bị *điện tử công suất*[1] và khâu ĐC dòng. Trong một số trường hợp, thậm chí nguyên lý ĐC dòng còn quyết định dạng của mạch ĐTCS. Chính vì lẽ này, khi tiến hành mô hình hóa thiết bị ĐTCS ta sẽ thực hiện hai phương án: 1. Phương án không có vòng ĐC dòng; 2. Phương án có tích hợp cả vòng ĐC dòng. Nhờ đó, người sử dụng sẽ có nhiều khả năng để lựa chọn khối thích hợp và đỡ mất thời gian để xây dựng mô hình mô phỏng.

[1] Điện tử công suất: ĐTCS, Power Electronics

11.1 Mô hình cầu chỉnh lưu 6 xung có điều khiển cắt pha

Mặc dù có khá nhiều phương án mạch chỉnh lưu (viết tắt: CL), trong mục này ta sẽ chỉ tập trung mô hình hóa cầu 6 xung được điều khiển theo phương pháp cắt pha. Van bán dẫn sử dụng trong sơ đồ thường là Thyristor (hình 11.2a).



Hình 11.2 *a) Sơ đồ cầu chỉnh lưu 6 xung sử dụng Thyristor, điều khiển theo phương pháp cắt pha; b) Đặc tính điện áp của cầu khi góc cắt $\alpha = 0$; c) Đặc tính điện áp của cầu khi góc cắt $\alpha \neq 0$*

Hình 11.2b,c mô tả dạng của điện áp ra, phụ thuộc vào điện áp lưới và góc cắt α, đồng thời cho ta biết các cặp van nào dẫn điện trong từng khoảng thời gian ứng với góc 60^0. Giá trị trung bình của điện áp đặt lên động cơ được tính theo công thức sau:

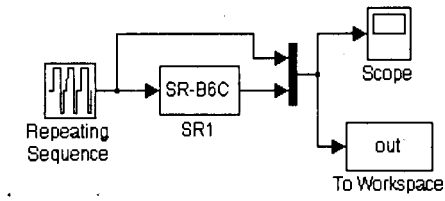
$$U_{d\alpha} = \frac{1}{6f} \int_{-\pi/6+\alpha}^{\pi/6+\alpha} \widehat{U} \cos(2\pi f t)\, dt \qquad \widehat{U} = \sqrt{2} U_N \tag{11.1}$$

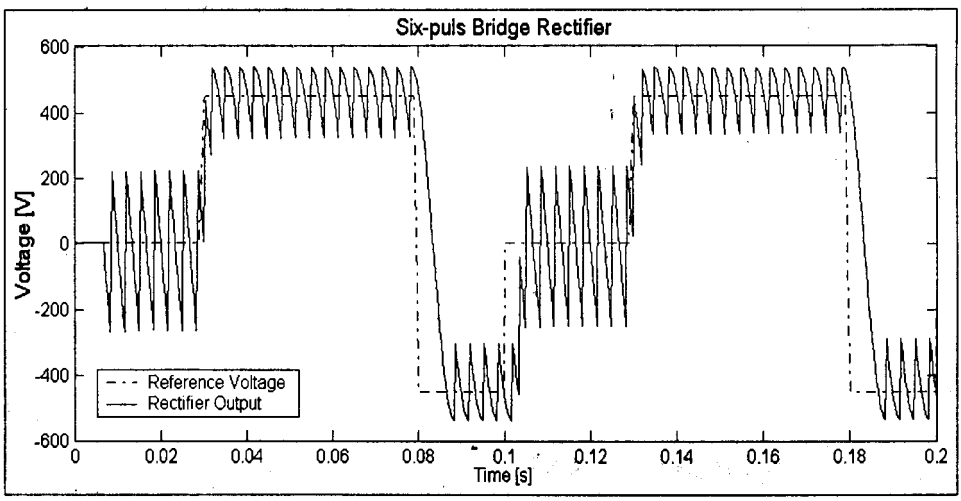
Kết quả tính tích phân của (11.1) sẽ là:

$$U_{d\alpha} = \widehat{U}\frac{\sin\left(\pi/6\right)}{\pi/6}\cos(\alpha) \cong 1,35U_N\cos(\alpha) \qquad (11.2)$$

U_N Điện áp hiệu dụng của lưới

Từ công thức (11.2) ta có thể tính được dễ dàng, khi điện áp lưới là 380V, điện áp đầu ra của cầu chỉnh lưu sẽ là 513V với $\alpha = 0$. Khi xây dựng mô hình cầu chỉnh lưu ta cần lưu ý: Tín hiệu (điện áp) ra của khâu ĐC dòng là giá trị mà cầu CL phải thực hiện. Tuy nhiên, việc thực hiện đó bao giờ cũng bị trễ một khoảng thời gian phụ thuộc vào góc cắt α. Có thể xét tới hiện tượng này bằng cách bổ sung thêm ở đầu vào một khâu quán tính bậc 1 với hằng số thời gian phụ thuộc tần số lưới $(1/6)/f_N$. Khi mô hình hóa cầu CL ta sẽ không còn cơ hội sử dụng các khối có sẵn của thư viện SIMULINK và con đường duy nhất sẽ là: Soạn thảo hàm S tính công thức (11.2) bằng một trong hai cách, hoặc *m-File*, hoặc *C-mex-File*. Hình 11.3 giới thiệu kết quả của cách giải quyết với *C-mex-File*.





Hình 11.3 *Trái: Cầu CL 6 xung (SR1) với giá trị đặt do khối Repeating Sequence tạo. Dưới: Giá trị điện áp đặt và giá trị điện áp thực ở đầu ra của CL*

Một phương pháp mô hình hóa khác cũng có thể được sử dụng một cách rất lợi hại: Đó là sử dụng ***công thức tính phổ của các sóng hài*** (phụ thuộc α) chứa trong điện áp đầu ra của CL. Có thể tìm thấy khá dễ dàng công thức đó trong các sách chuyên ngành về ĐTCS. Đồ thị của dạng điện áp ra chính là tổng của tất cả các thành phần hài.

$$f(t)=\frac{a_0}{2}+\sum_{n=1}^{\infty}\left[a_n\cos(nt)+b_n\sin(nt)\right]$$
$$f(t+2k\pi)=f(t) \qquad (k=0,\pm1,\ldots) \tag{11.3}$$

Các thành phần thuộc (11.3) chỉ khác 0 đối với những hài có tần số gấp ($n\pm1$) lần tần số lưới và được tính cụ thể theo:

$$a_0=\frac{12}{\pi}\sin\frac{\pi}{6}\cos\alpha \tag{11.4}$$

$$a_n=\begin{cases}\dfrac{3}{\pi}\left[\dfrac{\cos(\alpha(n+1))}{n+1}-\dfrac{\cos(\alpha(n-1))}{n-1}\right] & \text{khi} \quad n=6k\\ 0 & \text{khi} \quad n\neq 6k\end{cases} \tag{11.5}$$

$$b_n=\begin{cases}\dfrac{3}{\pi}\left[\dfrac{\sin(\alpha(n+1))}{n+1}-\dfrac{\sin(\alpha(n-1))}{n-1}\right] & \text{khi} \quad n=6k\\ 0 & \text{khi} \quad n\neq 6k\end{cases} \tag{11.6}$$

Kết quả mô phỏng sau khi mô hình hóa thiết bị CL theo các công thức (11.4), (11.5) và (11.6) cũng tương tự như hình 11.3.

Trong lời mở đầu của mục 10.2 ta đã đề cập đến mô hình CL có kèm theo khâu ĐC dòng. Việc thiết kế xấp xỉ *khâu ĐC số*[1] xuất phát từ các tham số của MĐMC (đã định nghĩa ở các mục 9.1 và 10.1.1) và được tiến hành thành hai bước.

- *Bước* 1: Thiết kế khâu ĐC trên miền tần số (miền ảnh Laplace).

$$G_{sPI}(s)=V_P+\frac{V_I}{s} \qquad V_I=\frac{R_A}{2T_\Sigma};\ V_P=V_I\,T_A \tag{11.7}$$

với: $T_\Sigma = 1/6f_N$; $T_A = L_A/R_A$

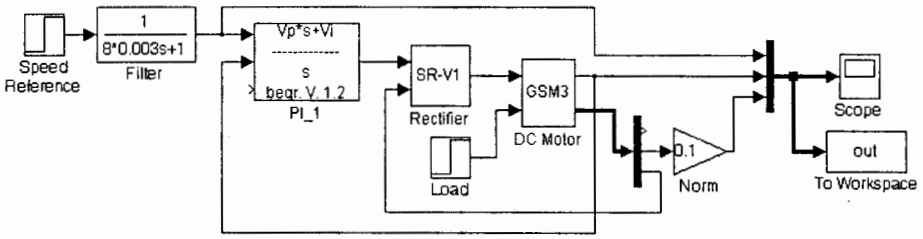
- *Bước* 2: Gián đoạn hóa khâu ĐC (11.7)

$$G_{zPI}(z)=\frac{V_P+(V_I\,T-V_P)z^{-1}}{1-z^{-1}}=V_P\frac{1+d_1z^{-1}}{1-z^{-1}} \tag{11.8}$$

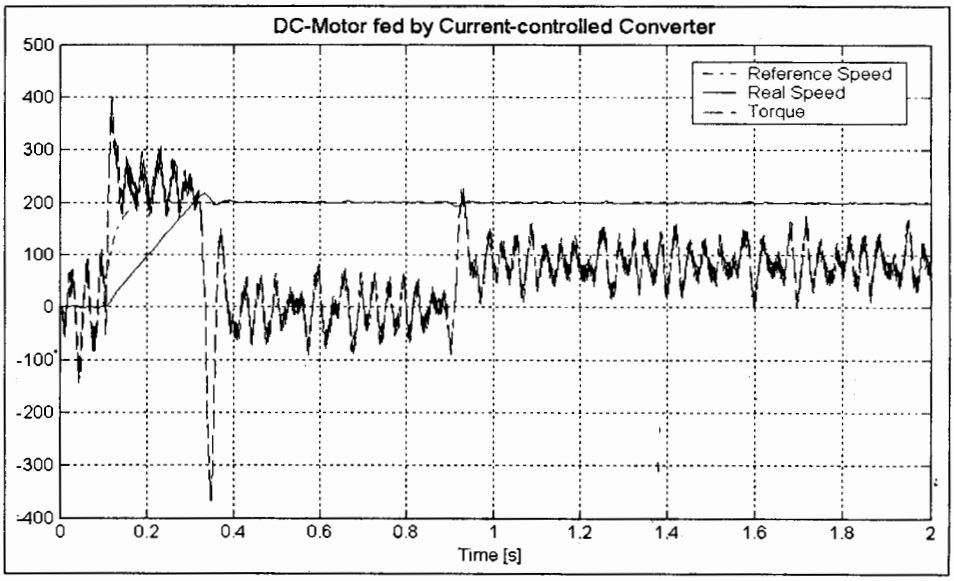
với: T = chu kỳ trích mẫu, $d_1=V_I T/V_P-1$

Hình 11.4 giới thiệu sơ đồ SIMULINK mô phỏng hệ truyền động điện một chiều, sử dụng mô hình cầu CL theo các công thức (11.4) - (11.6) có cài đặt khâu ĐC dòng. Kết quả mô phỏng một quá trình khởi động và đột biến phụ tải được giới thiệu ở hình 11.5.

[1] Khâu ĐC số: Digital Controller



Hình 11.4 Mô hình hệ truyền động điện một chiều sử dụng cầu CL 6 xung (SR-V1) có khâu ĐC dòng phần ứng

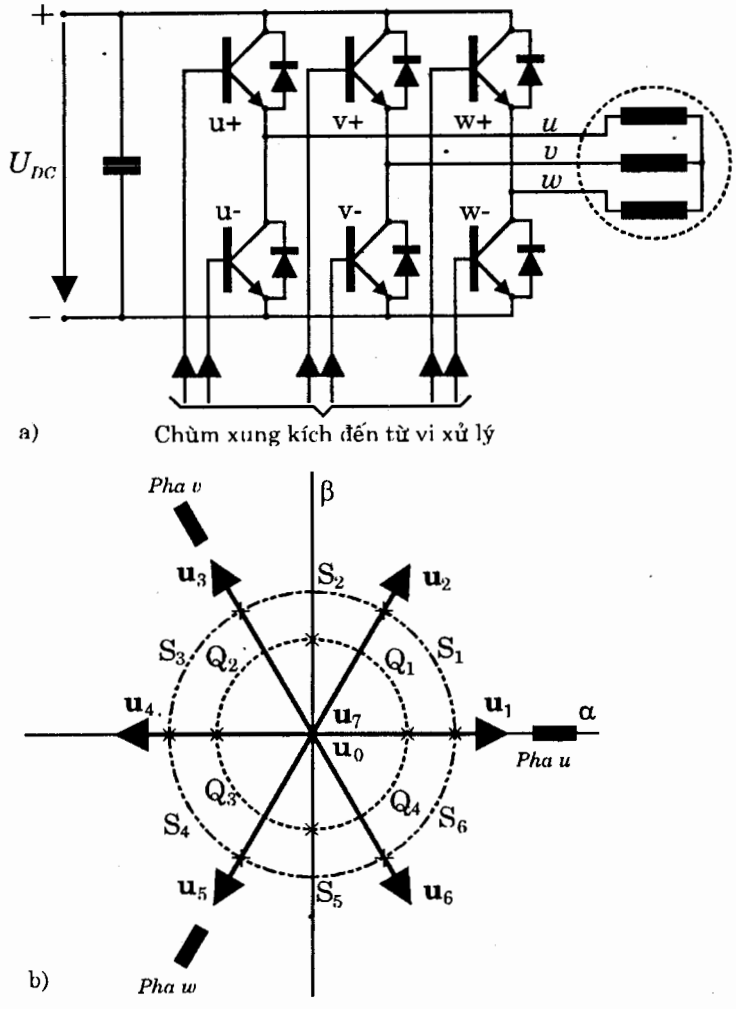


Hình 11.5 Mô phỏng quá trình khởi động và đột biến tải sử dụng sơ đồ ở hình 11.4

11.2 Mô hình nghịch lưu băm xung nguồn áp

Nghịch lưu băm xung nguồn áp[1] là dạng thiết bị ĐTCS rất quan trọng, được sử dụng để ĐK các máy điện xoay chiều ba pha (XCBP). Nghịch lưu (NL) có nhiệm vụ tạo ra một hệ thống nguồn XCBP có biên độ, tần số và góc pha theo như ý muốn. Nghịch lưu nguồn áp (NLNA) bao giờ cũng được nuôi bởi một nguồn điện áp một chiều U_{DC} (hình 11.6).

[1] Pulsed Voltage-Source Inverter: Pulsed VSI



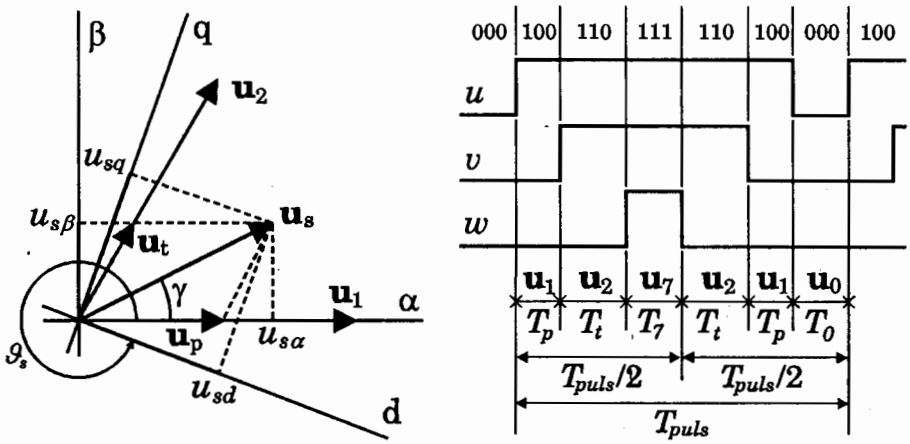
Hình 11.6 *a) Sơ đồ mạch nghịch lưu nguồn áp nuôi máy điện XCBP; b) Các vector điện áp chuẩn* $\mathbf{u}_0$, $\mathbf{u}_1$ ... $\mathbf{u}_7$ *tạo bởi ba nhánh van IGBT (*Q_1 ... Q_4*: góc phần tư;* S_1 ... S_6*: góc phần sáu)*

Bằng ba nhánh van ta có thể tạo ra 8 trạng thái lôgich, ứng với 8 vector điện áp chuẩn $\mathbf{u}_0$, $\mathbf{u}_1$ ... $\mathbf{u}_7$. Trong đó, hai vector $\mathbf{u}_0$ – *cả ba cuộn dây nối với cực âm* – và $\mathbf{u}_7$ – *cả ba cuộn dây nối với cực dương* – là các vector có module bằng không. Vị trí tương đối của các vector chuẩn so với các trục α, β được minh họa ở hình 11.6b. Vector chuẩn chia không gian vector thành các góc phần sáu S_1 ... S_6 (*Sector*) và góc phần tư Q_1 ... Q_4 (*Quadrant*). Trạng thái lôgich của các nhánh van được tập hợp trong bảng sau.

Pha	$\mathbf{u}_0$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{u}_4$	$\mathbf{u}_5$	$\mathbf{u}_6$	$\mathbf{u}_7$
u	0	1	1	0	0	0	1	1
v	0	0	1	1	1	0	0	1
w	0	0	0	0	1	1	1	1

Để thực hiện một vector điện áp bất kỳ, ví dụ: $\mathbf{u}_s$ đang nằm trong S_1 (hình 11.7, trái), ta tách $\mathbf{u}_s$ theo hướng của hai vector chuẩn thuộc *Sector* đó thành hai vector biên phải $\mathbf{u}_p$ và biên trái $\mathbf{u}_t$. Từ đó ta quy vấn đề thực hiện $\mathbf{u}_s$ thành việc thực hiện hai vector chuẩn (trong ví dụ) $\mathbf{u}_1$ và $\mathbf{u}_2$ trong khoảng thời gian T_p và T_t (hình 11.7, phải) ứng với:

$$T_p = \left(\frac{T_{puls}}{2}\right)\frac{|\mathbf{u}_p|}{|\mathbf{u}_s|_{\max}}; T_t = \left(\frac{T_{puls}}{2}\right)\frac{|\mathbf{u}_t|}{|\mathbf{u}_s|_{\max}}; \quad |\mathbf{u}_s|_{\max} = \frac{2}{3}U_{DC} \quad (11.9)$$



Hình 11.7 Trái: Thực hiện vector điện áp bất kỳ bằng hai vector biên; Phải: Mẫu xung của vector điện áp thuộc góc S_1

Dễ dàng thấy rằng: Để tính module của hai vector biên ta sẽ phải sử dụng các công thức khác nhau, được tập hợp trong bảng ở trang tiếp theo. Mặc dù bảng đó có chứa nhiều công thức, chúng vẫn có thể quy về được chỉ ba biểu thức sau:

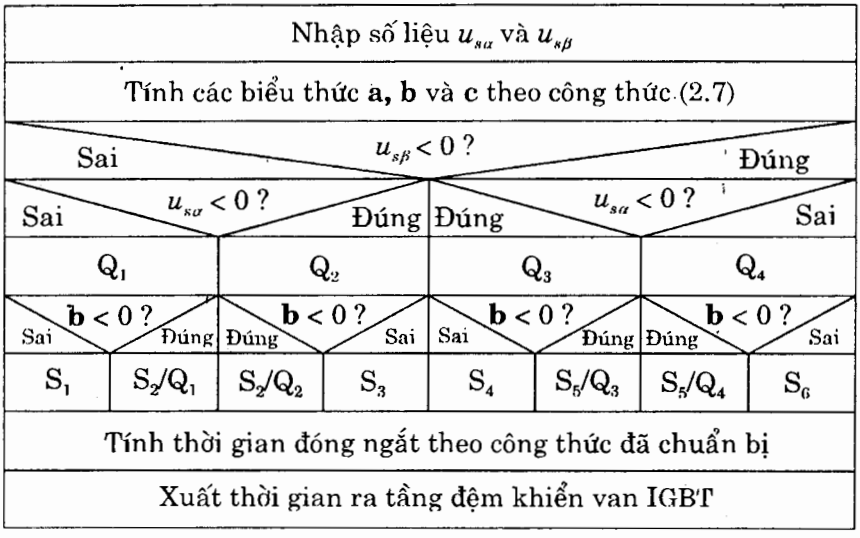
$$a = |u_{s\alpha}| + \frac{1}{\sqrt{3}}|u_{s\beta}|; b = |u_{s\alpha}| - \frac{1}{\sqrt{3}}|u_{s\beta}|; c = \frac{2}{\sqrt{3}}|u_{s\beta}| \quad (11.10)$$

Ba biểu thức thuộc (11.10) là xuất phát điểm để ta điều chế điện áp $\mathbf{u}_s$, ĐK các nhánh van của NLNA, khi ta đã biết vector $\mathbf{u}_s$ nằm ở *Sector* nào (vị trí của $\mathbf{u}_s$). Để xác định vị trí ta thực hiện các bước:

- Xét dấu của $u_{s\alpha}$ và $u_{s\beta}$ để tìm góc phần tư của $\mathbf{u}_s$.
- Biểu thức b đổi dấu khi $\mathbf{u}_s$ đi qua biên giới giữa hai góc phần sáu. Xét dấu của b để xác định góc phần sáu của $\mathbf{u}_s$.

		$\lvert\mathbf{u}_p\rvert$	$\lvert\mathbf{u}_t\rvert$
S_1	Q_1	$\lvert u_{s\alpha}\rvert - \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$	$\frac{2}{\sqrt{3}}\lvert u_{s\beta}\rvert$
S_2	Q_1	$\lvert u_{s\alpha}\rvert + \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$	$-\lvert u_{s\alpha}\rvert + \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$
	Q_2	$-\lvert u_{s\alpha}\rvert + \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$	$\lvert u_{s\alpha}\rvert + \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$
S_3	Q_2	$\frac{2}{\sqrt{3}}\lvert u_{s\beta}\rvert$	$\lvert u_{s\alpha}\rvert - \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$
S_4	Q_3	$\lvert u_{s\alpha}\rvert - \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$	$\frac{2}{\sqrt{3}}\lvert u_{s\beta}\rvert$
S_5	Q_3	$\lvert u_{s\alpha}\rvert + \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$	$-\lvert u_{s\alpha}\rvert + \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$
	Q_4	$-\lvert u_{s\alpha}\rvert + \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$	$\lvert u_{s\alpha}\rvert + \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$
S_6	Q_4	$\frac{2}{\sqrt{3}}\lvert u_{s\beta}\rvert$	$\lvert u_{s\alpha}\rvert - \frac{1}{\sqrt{3}}\lvert u_{s\beta}\rvert$

Thuật toán điều chế vector bao gồm hai bước chính: 1. Xác định vị trí của $\mathbf{u}_s$ để chọn công thức tính module của hai vector biên; 2. Tính thời gian đóng ngắt các van IGBT theo (11.9). Lưu đồ thuật toán được giới thiệu ở hình 11.8.



***Hình 11.8** Lưu đồ thuật toán điều chế vector điện áp ba pha*

Với thuật toán 10.47 ta có thể điều chế, tạo rất chính xác vector điện áp, chuyển thành điện áp XCBP đặt lên động cơ. Tuy nhiên, khi mô hình hóa NLNA ta sẽ luôn phải trả lời câu hỏi: NLNA đó được ĐK theo phương pháp nào? Bởi vì, điều chế vector đâu có phải là phương pháp ĐK duy nhất.

Mặc dù, về lý thuyết ta vẫn có thể tạo được mô hình NLNA mang đặc điểm đóng ngắt thuần túy. Trong trường hợp điều chế vector, NLNA chỉ còn mang đặc điểm của khâu trễ một chu kỳ trích mẫu (khâu z^{-1}) và luôn gắn liền với khâu ĐC dòng phía trước. Vì vậy, để thuận tiện ta có thể ghép luôn thuật toán ở hình 11.8 hoặc vào mô hình của khâu ĐC dòng ở phía trước, hoặc của máy điện XCBP ở phía sau theo dòng tín hiệu của cấu trúc hệ thống.

Phương pháp tốt nhất vẫn là tạo hàm S theo một trong hai cách: *script m-File* hoặc *C-mex-File*. Dưới đây bạn đọc có thể tham khảo hai trích đoạn quan trọng nhất trong *C-mex-File* của MĐDB soạn theo mô hình (10.11), đi kèm mô hình NLNA theo thuật toán ở hình 11.8:

```
/*********************************************************
 * ASMABPWM.C: C-MEX-File to simulate the 3-phase        *
 * induction motor in stator-fixed coordinates inclusive *
 * voltage-source inverter, pulsed by vector modulation  *
 * algorithm.                                            *
 * Motor paramters :                                     *
 *      Lr  Rotor inductance                             *
 *      Ls  Stator inductance                            *
 *      Rr  Rotor resistance                             *
 *      Rs  Stator resistance                            *
 *      Lm  Mutual inductance                            *
 *      pc  Number of pole pair                          *
 *********************************************************/

// Function for computing the 3-phase IM
static
void ASM_Model(x,Derivative,we,Usa,Usb)
     double *x,*Derivative,we,Usa,Usb;

/* Đoạn tính mô hình trạng thái MĐDB */
/*.Arguments: States x for IM-states current and flux,
  derivative of diff. equations, we-electrical speed,
  Usa, Usb-components of stator voltage */
{ Derivative[0] = - Tsig_1 * isax + Tr_sig_1_Lm * Psirax
                  + Tr_sig1_1_Lm * Psirbx * we
                  + Lsig_1 * Usa;
  Derivative[1] = - Tsig_1 * isbx + Tr_sig_1_Lm * Psirbx
                  - Tr_sig1_1_Lm * Psirax * we
                  + Lsig_1 * Usb;
  Derivative[2] =   Tr_1_Lm * isax - Tr_1 * Psirax
                  - Psirbx * we;
  Derivative[3] =   Tr_1_Lm * isbx - Tr_1 * Psirbx
                  + Psirax * we;
}
// Ende ASM-Funktion
```

```
/* Phương pháp tích phân hai bước */
// Function to implement the double-step integration method
// First step
static
void PreCor1(Derivative,State,T_integration)
     double *Derivative,*State,T_integration;
     { int i;
       for (i=0;i<4;i++)
       State[i] = State[i] + Derivative[i] * T_integration;
     }
// End of the first step

// Second step
static
void PreCor2(Derivative1,Derivative2,State,T_integration)
     double *Derivative1,*Derivative2,*State,T_integration;
     { int i;
       for (i=0;i<4;i++)
       State[i] = State[i] + (Derivative1[i]
                + Derivative2[i]) * 0.5 * T_integration;
     }
// End of the second step

/* mdlUpdate: Thuật toán điều chế vector */
/* mdlUpdate - perform action at major integration
 * time step */
static void mdlUpdate(x, u, S, tid)
  double *x, *u;
  SimStruct *S;
  int tid;
{ { int State_vec;
    double usa_l,usb_l,usa_r,usb_r,u_r,u_l;
    double usa_abs,usb_abs,u_vec;
    double w_el,T_r,T_l,T_zeros,Time_ab1[4],Time_ab2[4];
    double State_memory[4],aux;
// Computing the modulation algorithm
    w_el = w_mech * zp;       // Computing electrical speed
    usa_abs = fabs(usa);      // Axiliary variables
    usb_abs = fabs(usb) * one_divides_sqrt3;
    u_vec = usa_abs - usb_abs;
    State_vec = ((usb < 0.0) << 2) + ((usa < 0.0) << 1)
                                   + (u_vec < 0.0);

/* Tính module vector biên phải, biên trái: u_r, u_l */
// Determining current sector
  switch(State_vec)
  { case 0:              // 0 degree <= angle < 60 degree
```

```
{ u_r = usa_abs - usb_abs;
  u_l = usb_abs + usb_abs;
  usa_r = udc;
  usa_l = udc_divides_2;
  usb_r = 0.0;
  usb_l = udc_mult_sqrt3_divides_2;
}
break;
case 1:                  // 60 degree <= angle < 90 degree
{ u_r = usa_abs + usb_abs;
  u_l = -usa_abs + usb_abs;
  usa_r = udc_divides_2;
  usa_l = -usa_r;
  usb_r = udc_mult_sqrt3_divides_2;
  usb_l = usb_r;
}
break;
case 3:                  // 90 degree <= angle < 120 degree
{ u_r = -usa_abs + usb_abs;
  u_l = usa_abs + usb_abs;
  usa_r = udc_divides_2;
  usa_l = -usa_r;
  usb_r = udc_mult_sqrt3_divides_2;
  usb_l = usb_r;
}
break;
case 2:                  // 120 degree <= angle < 180 degree
{ u_r = usb_abs + usb_abs;
  u_l = usa_abs - usb_abs;
  usa_r = -udc_divides_2;
  usa_l = -udc;
  usb_r = udc_mult_sqrt3_divides_2;
  usb_l = 0.0;
}
break;
case 6:                  // 180 degree <= angle < 240 degree
{ u_r = usa_abs - usb_abs;
  u_l = usb_abs + usb_abs;
  usa_r = -udc;
  usa_l = -udc_divides_2;
  usb_r = 0.0;
  usb_l = -udc_mult_sqrt3_divides_2;
}
break;
case 7:                  // 240 degree <= angle < 270 degree
{ u_r = usa_abs + usb_abs;
  u_l = -usa_abs + usb_abs;
```

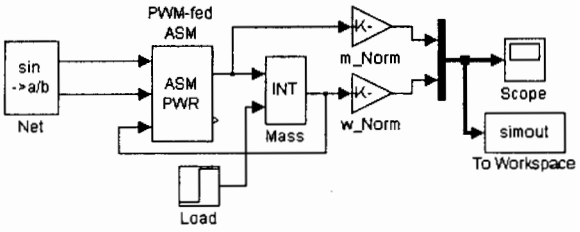
```
      usa_r = -udc_divides_2;
      usa_l = -usa_r;
      usb_r = -udc_mult_sqrt3_divides_2;
      usb_l = usb_r;
    }
    break;
    case 5:              // 270 degree <= angle < 300 degree
    { u_r = -usa_abs + usb_abs;
      u_l = usa + usb_abs;
      usa_r = -udc_divides_2;
      usa_l = -usa_r;
      usb_r = -udc_mult_sqrt3_divides_2;
      usb_l = usb_r;
    }
    break;
    case 4:              // 300 degree <= angle < 360 degree
    { u_r = usb_abs + usb_abs;
      u_l = usa_abs - usb_abs;
      usa_r = udc_divides_2;
      usa_l = udc;
      usb_r = -udc_mult_sqrt3_divides_2;
      usb_l = 0.0;
    }
    break;
  }
/* Tính thời gian đóng ngắt vector biên phải, trái */
    T_r = Tpuls_half * u_r / udc;  // Switch-time ri-vector
    T_l = Tpuls_half * u_l / udc;  // Switch-time le-vector
    T_zeros = Tpuls_half-T_r-T_l;
                                   // Switch-time zero vector
    if (T_zeros < 0.0)             // Activating the limits
    { T_zeros = 0.0;
      aux = Tpuls_half / (T_r + T_l);
      T_r = T_r * aux;
      T_l = T_l * aux;
    }

// Nuôi ASM bằng các vector biên mới tính được
// Tpuls/2 thứ 1 (hình 11.7, phải): Right - Left - Zero
// ASM-Model with right margin vector
    State_memory[0] = isax;
    State_memory[1] = isbx;
    State_memory[2] = Psirax;
    State_memory[3] = Psirbx;
    ASM_Model(&x[0],&Time_ab1,w_el,usa_r,usb_r);
    PreCor1(&Time_ab1,&x[11],T_r);
    ASM_Model(&x[0],&Time_ab2,w_el,usa_r,usb_r);
```

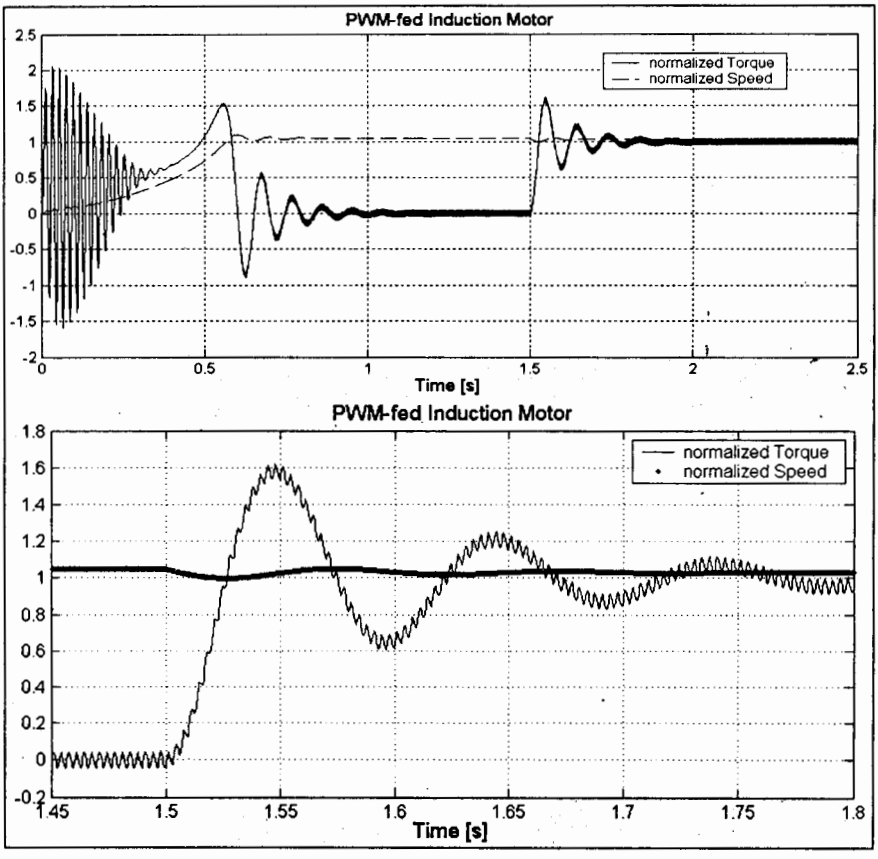
```
    PreCor2(&Time_ab1,&Time_ab2,&State_memory,T_r);
    isax = State_memory[0];
    isbx = State_memory[1];
    Psirax = State_memory[2];
    Psirbx = State_memory[3];
// ASM-Model with left margin vector
    ASM_Model(&x[0],&Time_ab1,w_el,usa_l,usb_l);
    PreCor1(&Time_ab1,&x[11],T_l);
    ASM_Model(&x[0],&Time_ab2,w_el,usa_l,usb_l);
    PreCor2(&Time_ab1,&Time_ab2,&State_memory,T_l);
    isax = State_memory[0];
    isbx = State_memory[1];
    Psirax = State_memory[2];
    Psirbx = State_memory[3];
// ASM-Model with zero vector
    ASM_Model(&x[0],&Time_ab1,w_el,0.0,0.0);
    PreCor1(&Time_ab1,&x[11],T_zeros);
    ASM_Model(&x[0],&Time_ab2,w_el,0.0,0.0);
    PreCor2(&Time_ab1,&Time_ab2,&State_memory,T_zeros);
    isax = State_memory[0];
    isbx = State_memory[1];
    Psirax = State_memory[2];
    Psirbx = State_memory[3];

// Tpuls/2 thứ 2 (hình 11.7, phải): Left - Right - Zero
// ASM-Model with left margin vector
    ASM_Model(&x[0],&Time_ab1,w_el,usa_l,usb_l);
    PreCor1(&Time_ab1,&x[11],T_l);
    ASM_Model(&x[0],&Time_ab2,w_el,usa_l,usb_l);
    PreCor2(&Time_ab1,&Time_ab2,&State_memory,T_l);
    isax = State_memory[0];
    isbx = State_memory[1];
    Psirax = State_memory[2];
    Psirbx = State_memory[3];
// ASM-Model with right margin vector
    ASM_Model(&x[0],&Time_ab1,w_el,usa_r,usb_r);
    PreCor1(&Time_ab1,&x[11],T_r);
    ASM_Model(&x[0],&Time_ab2,w_el,usa_r,usb_r);
    PreCor2(&Time_ab1,&Time_ab2,&State_memory,T_r);
    isax = State_memory[0];
    isbx = State_memory[1];
    Psirax = State_memory[2];
    Psirbx = State_memory[3];
// ASM-Model with zero vector
    ASM_Model(&x[0],&Time_ab1,w_el,0.0,0.0);
    PreCor1(&Time_ab1,&x[11],T_zeros);
    ASM_Model(&x[0],&Time_ab2,w_el,0.0,0.0);
```

```
        PreCor2(&Time_ab1,&Time_ab2,&State_memory,T_zeros);
        isax = State_memory[0];
        isbx = State_memory[1];
        Psirax = State_memory[2];
        Psirbx = State_memory[3];
    }
}
```



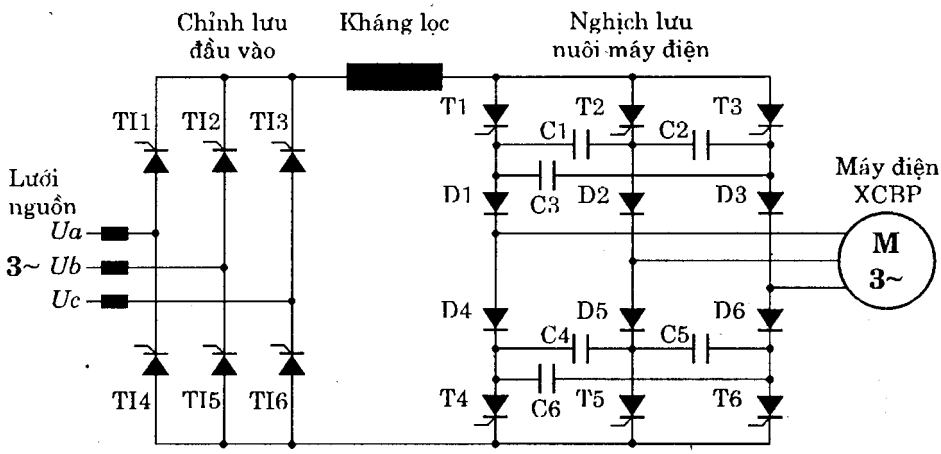
Hình 11.9 giới thiệu quá trình khởi động tự nhiên của MĐDB. Điều đáng chú ý ở đây là ảnh hưởng của việc băm xung áp tới thành phần hài của mômen quay (hình 11.9, dưới).



Hình 11.9 *Trên: Sơ đồ mô phỏng NLNA tích hợp sẵn vào mô hình MĐDB; Giữa: Khởi động tự nhiên (không có ĐK) và bước nhảy phụ tải; Dưới: Việc băm xung điện áp đã gây nên hài bậc cao trong mômen quay.*

11.3 Mô hình nghịch lưu nguồn dòng nuôi MĐDB

Kỹ thuật ĐK nghịch lưu theo phương pháp băm xung sử dụng NLNA mô tả ở mục 10.2.2 là bước phát triển cao trong ĐK máy điện XCBP. Kỹ thuật đó xuất hiện đồng thời với sự ra đời của các linh kiện bán dẫn công suất loại IGBT[1], hoặc GTO[2]. Trong các ứng dụng đòi hỏi chất lượng thấp hơn, rẻ hơn mà công suất lại lớn, thường ta sử dụng giải pháp *nghịch lưu nguồn dòng*[3] (NLND). Các mạch NLND – kể từ giai đoạn mới phát triển – chủ yếu sử dụng Thyristor (hình 11.10), là loại linh kiện có tốc độ đóng ngắt chậm.



Hình 11.10 *Sơ đồ thiết bị nghịch lưu nguồn dòng sử dụng Thyristor*

Khó khăn chủ yếu khi sử dụng van Thyristor là đặc điểm có thể chủ động mở van (kích hoạt dẫn dòng), nhưng lại không chủ động ngắt van được (ngừng hoạt động dẫn dòng). Mạch cầu 6 xung (hình 11.2) cũng sử dụng Thyristor và phải nhờ vào thế năng của lưới để chuyển mạch van. Những mạch hoạt động độc lập như NLND (phần mạch bên phải hình 11.10) nuôi máy điện XCBP phải sử dụng một mạch dập để chuyển mạch.

Thiết bị NLND bao giờ cũng có ba thành phần (hình 11.10):

- *Phần chỉnh lưu đầu vào* với nhiệm vụ ĐC ổn định dòng i_{DC} của mạch một chiều trung gian. Giá trị của i_{DC} quyết định biên độ của dòng máy điện (có tác dụng *điều khiển mômen quay*). CL sử dụng điện áp lưới để ĐC dòng i_{DC} (ĐC mômen).
- *Phần mạch một chiều trung gian* sử dụng cuộn kháng lọc với nhiệm vụ hạn chế hài dòng của i_{DC}.

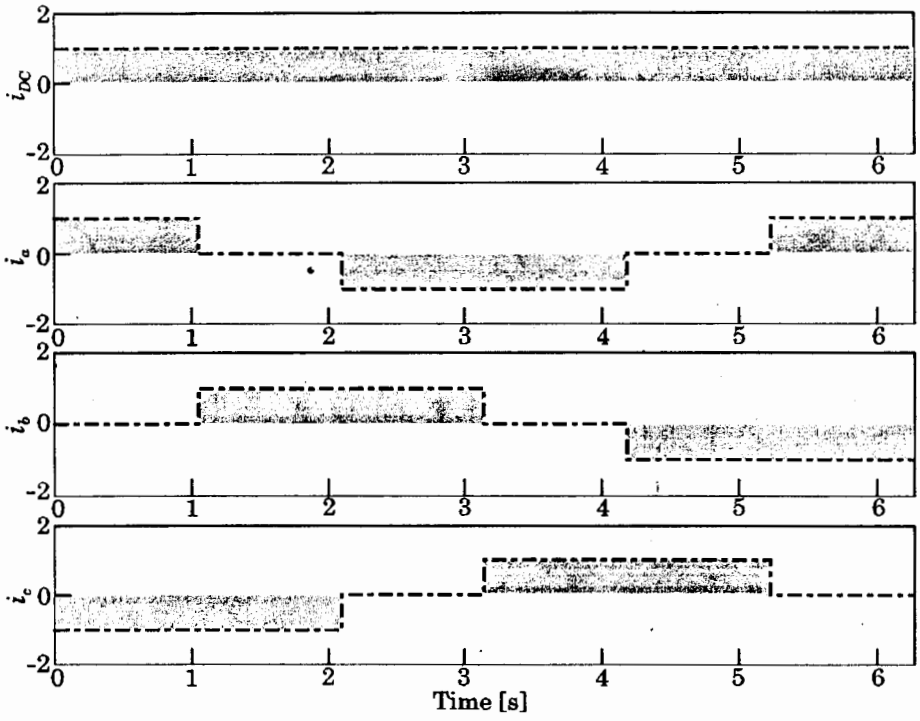
[1] IGBT: Insulated Gate Bipolartransistor

[2] GTO: Gate Turn-off Thyristor

[3] Current-Source Inverter: CSI; nghịch lưu nguồn dòng: NLND

- *Phần nghịch lưu nuôi máy điện XCBP* chỉ có nhiệm vụ chuyển mạch dòng, tức là tạo tần số của trường quay và qua đó *điều khiển tốc độ quay* của máy điện.

Vậy là: Máy điện XCBP được vận hành theo chế độ cấp dòng dạng Block (hình 11.11), từ thời điểm mở tới thời điểm ngắt van sẽ chẩy qua cuộn dây của máy một dòng không đổi (chứ không phải dạng sin như khi sử dụng NLNA). Giá trị của kháng lọc (mạch một chiều trung gian) được thiết kế từ số liệu của máy điện và từ yêu cầu về phổ hài của dòng. Để tìm hiểu vấn đề này bạn đọc sẽ phải tra cứu thêm các tài liệu chuyên sâu.



Hình 11.11 *Dạng dòng mạch một chiều trung gian và các dòng pha của máy điện*

Thông qua các lý giải ngắn gọn trên về nguyên lý của NLND ta đã có thể đưa ra một số yêu cầu khi mô hình hóa.

- *Mô hình phải có một vòng ĐC dòng* với nhiệm vụ ĐK dòng chẩy qua mạch một chiều trung gian. Giá trị dòng được tính theo:

$$I_s = I_\mu \sqrt{1 + T_r^2 \omega_r^2} \tag{11.11}$$

T_r Hằng số thời gian Rotor I_s Module dòng Stator
ω_r Tần số mạch Rotor I_μ Dòng từ hóa của máy điện XCBP

Xuất phát từ giả thiết MĐDB được vận hành với từ thông hằng, dòng Stator sẽ bao gồm thành phần dòng từ hóa (tạo từ thông) và thành phần tạo mômen (thường từ khâu ĐC tốc độ quay tới). Để phân tách được rõ ràng hai thành phần đó, vòng ĐC dòng sẽ phải được cài đặt trên hệ tọa độ *dq* (hệ tọa độ từ thông, xem mục 10.1.2)

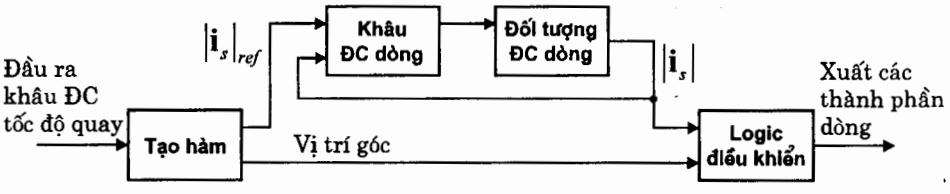
- Vị trí góc của hệ tọa độ *dq* được tính theo công thức dưới đây (rút ra từ (11.11))

$$\vartheta_s = \text{arctg}\left(T_r\omega_r\right) \tag{11.12}$$

Góc tìm được theo (11.12) sẽ (tùy theo chiều quay) được chọn gán cho một trong các vector dòng chuẩn thuộc bảng dưới đây.

Pha *a*	Pha *b*	Pha *c*	Góc của vector dòng
I_s	0	$-I_s$	30^0
0	I_s	$-I_s$	90^0
$-I_s$	I_s	0	150^0
$-I_s$	0	I_s	210^0
0	$-I_s$	I_s	270^0
I_s	$-I_s$	0	330^0

Ví dụ: Khi tính được $\vartheta_s = 78^0$ và MĐDB đang quay theo chiều dương, vậy ta chọn vector dòng chuẩn ứng với góc 90^0, và (theo sơ đồ ở hình 11.10) hai van T2 và T6 là hai van dẫn dòng. Hay với $\vartheta_s = 250^0$ và MĐDB đang quay theo chiều âm, ta chọn vector chuẩn ứng với góc 210^0 và hai van T3, T5. Việc lựa chọn vector nào (trên cơ sở thông tin về góc) do khâu Logic điều khiển (hình 11.12) đảm nhiệm.



Hình 11.12 *Khái quát nguyên lý mô hình hóa nghịch lưu nguồn dòng*

Khi sử dụng khối NLND (tạo theo theo nguyên tắc mô tả ở hình 11.12) ta phải nhớ: Khối NLND sẽ trao cho mô hình MĐDB dòng điện dưới dạng khối (chứ không phải là điện áp), vì vậy mô hình MĐDB sẽ phải là mô hình nguồn dòng. Tại mục 10.1.2 ta đã nhắc đến vấn đề này: Mô hình sẽ chỉ còn bao gồm hai phương trình thứ 3 và 4 của hệ **Error! Reference source not found.**, tức là hạn chế về chỉ còn hai phương trình từ thông, dòng Stator đã được NLND (chứ hoàn toàn không do động cơ) quyết định.

Do cấu trúc NLND có kèm theo vòng ĐC nên trở thành khá phức hợp, ta sẽ thực hiện mô hình bằng *C-mex-File* như đoạn mã dưới đây:

```
/*********************************************************
 * CSI.C: C-MEX-File to simulate the current-source      *
 * inverter feeding a 3-phase induction motor in alpha-  *
 * beta-coordinates                                      *
 * Syntax:   [sys,x0] = csi(t,x,u,flag,p)                *
 * Parameter p[Rr,Lr]                                    *
 *           Rr Rotor resistance                         *
 *           Lr Rotor inductance                         *
 *********************************************************/
#define  S_FUNCTION_NAME csi
#include "simstruc.h"
#include <math.h>

/* Input variables: */
#define I_s   u[0]    /* Reference stator current */
#define w_r   u[1]    /* Slip frequency (Output of
                         speed controller) */
#define Psira u[4]    /* Rotor flux Psira of IM (State) */
#define Psirb u[5]    /* Rotor flux Psirb of IM (State) */

/* State variables: */
#define isa   x[0]    /* Stator current alpha */
#define isb   x[1]    /* Stator current beta */
#define Tr    x[2]    /* Rotor time constant */

/* Auxiliary variables: */
#define TWO_PI  6.28318530718
#define sign(x) (x==0.0?0.0:((x>0.0)?1.0:-1.0))
                      /* Definition of Sign-Function */

/* The current inverter was made under the following
 * acceptances:
 * 1. The 6 standard current vectors have to be switched
 *    with 2/3 pi switch-on
 * 2. The commutation events are not noticed
 * 3. The dead-time of the signal processing will be one
 *    integration step
 * 4. The netside-converter is a current-source, feeding
 *    the IM with a delay of 6ms (because of stator time
 *    constant, DC-link inductance and sum time constant)
 * 5. The current controller is a PI-controller, designed
 *    by magnitude optimum
 * 6. The motorside-inverter switchs on the possible
 *    vectors, chosen by the load angle (=arctan(Tr*wr)
 *********************************************************/
/* Defines for easy access of p, xo matrices which are
   passed in */
```

```
#define p ssGetArg(S,0)
/* mdlInitializeSizes - initialize the sizes array */
static void mdlInitializeSizes(S)
SimStruct  *S;
{ ssSetNumContStates(S,0);  /* Number of contin. states */
  ssSetNumDiscStates(S,3);  /* Number of discrete states */
  ssSetNumInputs(S,DYNAMICALLY_SIZED);
                            /* Number of inputs */
  ssSetNumOutputs(S,2);     /* Number of outputs */
  ssSetDirectFeedThrough(S,0);
                            /* Direct feedthrough flag */
  ssSetNumSampleTimes(S,1); /* Number of sample times */
  ssSetNumInputArgs(S,1);   /* Number of input arguments */
  ssSetNumRWork(S,0);       /* Number of real work vector
                               elements */
  ssSetNumIWork(S,0);       /* Number of integer work
                               vector elements */
  ssSetNumPWork(S,0);       /* Number of pointer work
                               vector elements */
}
/* mdlInitializeSampleTimes - initialize the sample times
 * array */
static void mdlInitializeSampleTimes(S)
SimStruct  *S;
{ ssSetSampleTimeEvent(S, 0, 0.0);
  ssSetOffsetTimeEvent(S, 0, 0.0);
}

/* mdlInitializeConditions - initialize the states */
static void mdlInitializeConditions(x0, S)
double *x0;
SimStruct *S;
{ double Lr,Rr;
  Lr    = mxGetPr(p)[1];    /* Rotor inductance */
  Rr    = mxGetPr(p)[0];    /* Rotor resistance */
  x0[0] = 0.0;
  x0[1] = 0.0;
  x0[2] = Lr / Rr;
}
/* mdlOutputs - compute the outputs */
static void mdlOutputs(y, x, u, S, tid)
double *y, *x, *u;
SimStruct *S;
int tid;
{ y[0] = isa;
  y[1] = isb;
}
```

```
/* mdlUpdate - perform action at major integration time
 * step */
static void mdlUpdate(x, u, S, tid)
double *x, *u;
SimStruct *S;
int tid;
{ int i, point;                          /* Remember */
  double T[7] = {0.0,1.04719755,2.0943951,3.141592654,
                 4.1887902,5.2359878,TWO_PI};
  double S1[6] = {0.866,0.0,-0.866,-0.866,0.0,0.866};
  double S2[6] = {0.5,1.0,0.5,-0.5,-1.0,-0.5};
  double move = 0.0,FluxAngle,theta;

/* Computing of the current flux angle (position of the
   flux vector) */
  if (sign(Psira < 0.0))
    move = TWO_PI * 0.5;
  if (Psira == 0.0)
    FluxAngle = TWO_PI * 0.25 * sign(Psirb);
  else
    FluxAngle = atan(Psirb / Psira) + move;
  if (FluxAngle < 0.0) FluxAngle = FluxAngle + TWO_PI;

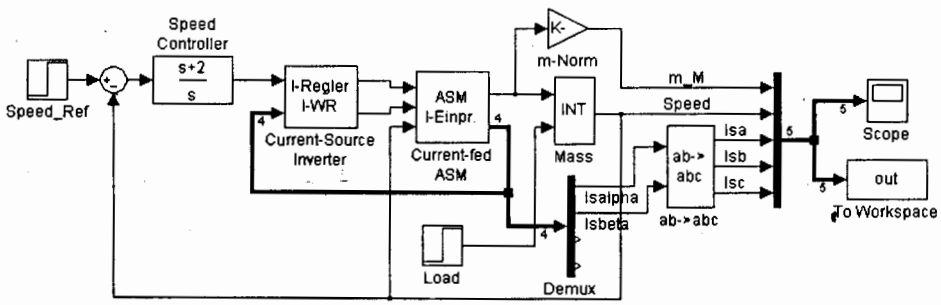
/* The absolute angle of the current vector to be switch-on
   results from the load angle (atan(Tr*w_r) and the flux
   angle */
  theta = atan(Tr * w_r) + FluxAngle;

/* Limiting the angle on 0 <= theta <= 2PI */
  if (theta >= TWO_PI)
    theta = theta - TWO_PI;
  if (theta < 0.0)
    theta = theta + TWO_PI;

  for (i=0;i<6;i++)
    if ((theta >= T[i]) && (theta < T[i+1]))
      point = i;
      isa = I_s * S1[point];
      isb = I_s * S2[point];
}
/* mdlDerivatives - compute the derivatives */
static void mdlDerivatives(dx, x, u, S, tid)
double *dx, *x, *u;
SimStruct *S;
int tid;
{
}
```

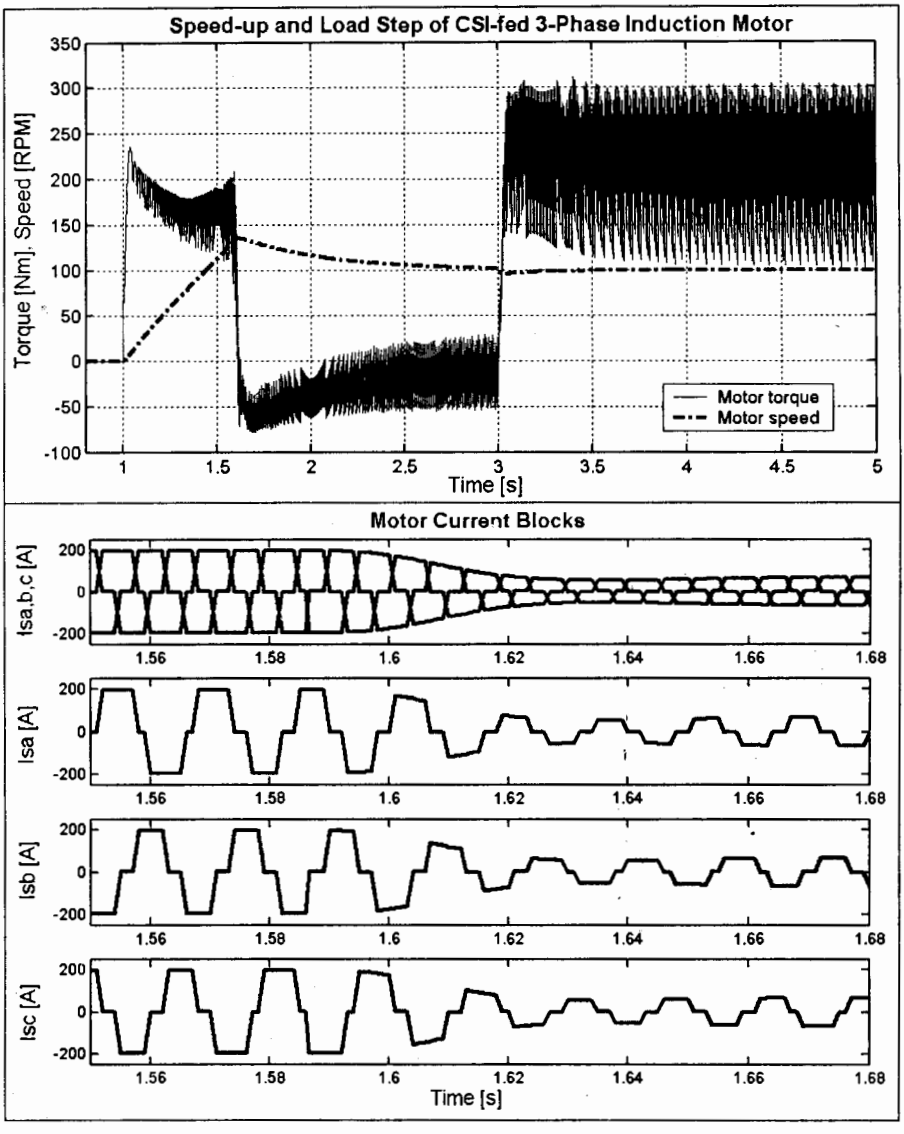
```
/* mdlTerminate - called when the simulation is
   terminated */
static void mdlTerminate(S)
SimStruct *S;
{ }
#undef xo
#undef p
#undef TWO_PI
#undef Psira
#undef Psirb
#undef isa
#undef isb
#undef mm
#undef w_r
#undef I_s
#ifdef MATLAB_MEX_FILE    /* Is this file being compiled
                             as a C-MEX-file? */
#include "simulink.c"     /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"      /* Code generation registration
                             function */
#endif
```

Đến đây chắc chắn những bạn đọc đã xem kỹ đoạn mã trên sẽ có nhận xét: Nếu theo cấu trúc của NLND (mô tả ở hình 11.12), đoạn mã đó mới thực hiện hai khối *tạo hàm* (tính góc) và *logic điều khiển* (đưa dòng tới đầu ra theo góc đã tính). Đây là nhận xét chuẩn xác, để hoàn thiện ta sẽ còn phải bổ sung thêm một khâu ĐC với đặc tính PI và một khâu PT_2 thay cho mô hình đối tượng ĐC dòng. Có thể bổ sung bằng cách lấy các khối có sẵn từ thư viện của SIMULINK. Để thử nghiệm ta xây dựng hệ thống truyền động sử dụng MĐDB nuôi bởi NLND như hình 11.13.



Hình 11.13 *Mô hình hệ truyền động(đòi hỏi chất lượng thấp) có ổn định tốc độ quay, sử dụng MĐDB nuôi bởi nghịch lưu nguồn dòng*

Hình tiếp theo minh họa các kết quả mô phỏng thu được từ sơ đồ 11.13.

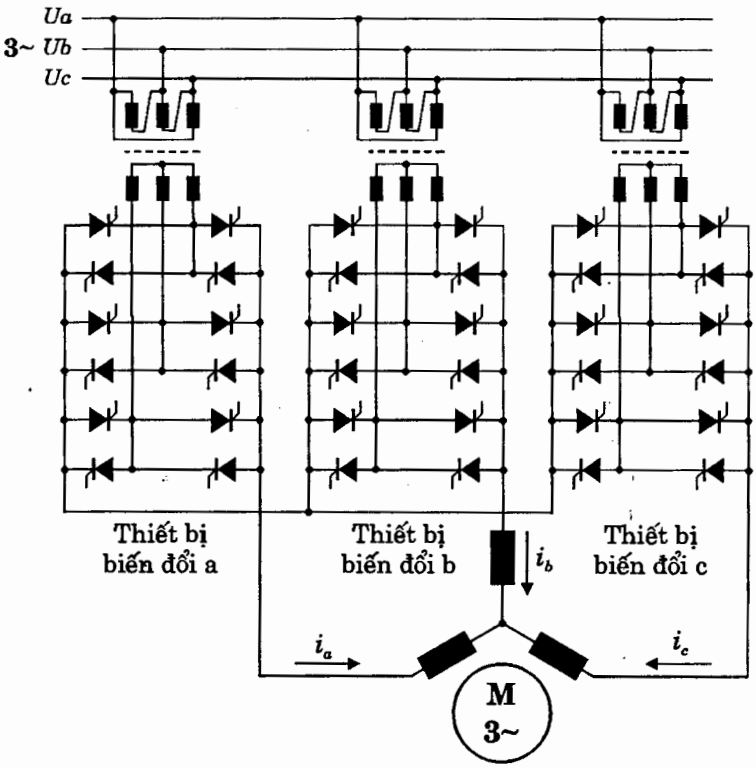


Hình 11.14 *Nửa trên: Đặc tính mômen và tốc độ khi khởi động và đột biến tải; Nửa dưới: Dạng của ba dòng pha*

Qua hình 11.14 ta thấy rõ chất lượng tương đối thấp của mômen quay, thể hiện qua biên độ khá lớn của các thành phần mômen hài. Dạng Block của ba dòng pha được (so với hình 11.11) minh chứng ở nửa hình dưới, đây chính là nguyên nhân sinh ra hài trong mômen quay.

11.4 Mô hình thiết bị biến đổi trực tiếp nuôi MĐDB và MĐĐB

Thiết bị biến đổi trực tiếp[1] (BĐTT)là loại biến tần với khả năng chuyển trực tiếp tần số lưới sang tần số công tác của máy điện mà không cần qua mạch một chiều trung gian. Vì không qua mạch DC trung gian, tần số tạo được ở đầu ra bao giờ cũng bé hơn tần số lưới. Thiết bị BĐTT được sử dụng để điều khiển máy điện XCBP có công suất rất lớn thuộc phạm vi MW. Những hệ truyền động này thường hay thấy trong công nghiệp nặng, ví dụ: Trong các dây chuyền cán thép. Hình 11.15 mô tả sơ đồ nguyên lý của thiết bị BĐTT.

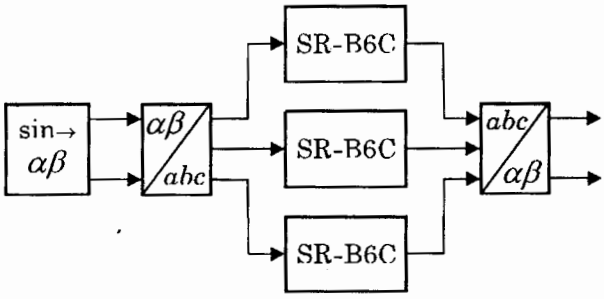


Hình 11.15 *Sơ đồ nguyên lý thiết bị biến đổi trực tiếp, sử dụng trong các hệ thống truyền động công suất rất lớn*

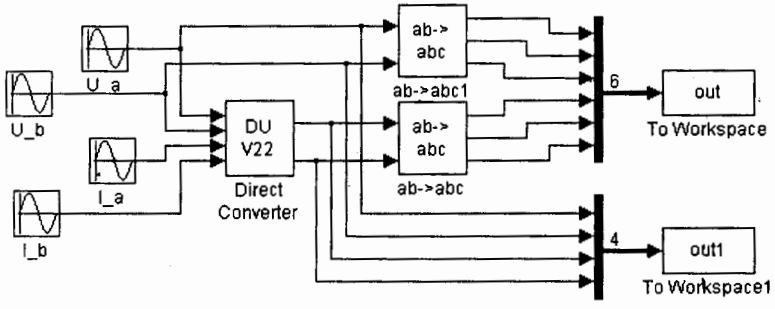
Thiết bị bao gồm ba thành phần dành cho ba pha, và mỗi thành phần được cấu tạo từ hai cầu CL 6 xung mắc song song ngược với nhau. Một cấu trúc mạch như vậy chỉ cho phép ta hạ tần số, và (theo một số công trình nghiên cứu) vì lý do hạn chế méo dạng dòng lưới, tần số đầu ra chỉ được phép tối đa bằng ½ tần số lưới.

[1] Thiết bị biến đổi trực tiếp: Direct Converter

Từ sơ đồ nguyên lý ở hình 11.15 ta có thể xây dựng mô hình cho thiết bị bằng cách sử dụng các module cầu CL 6 xung đã có sẵn như hình 11.16, hoặc bằng cách soạn thảo *C-mex-File*. Hình 11.17 giới thiệu sơ đồ SIMULINK sử dụng thiết bị BĐTT tạo theo phương thức *C-mex-File*.



Hình 11.16 *Thực hiện mô hình bằng cách sử dụng các module cầu CL 6 xung có sẵn*

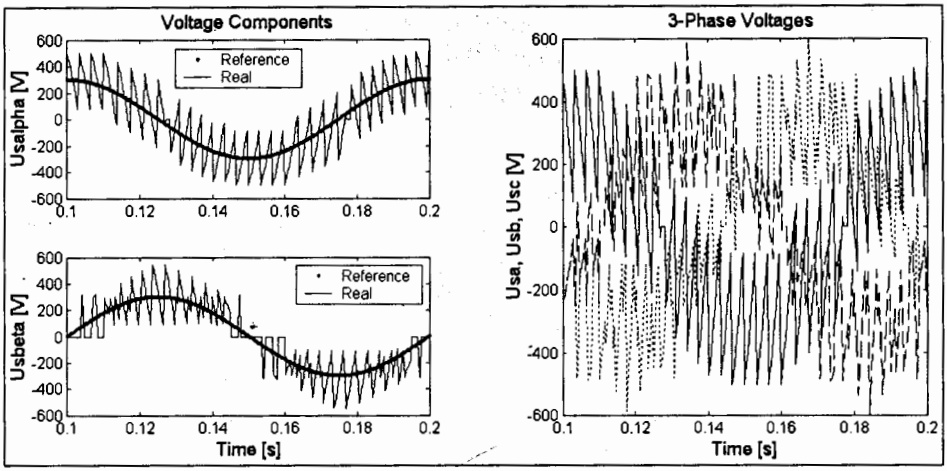


Hình 11.17 *Mô hình SIMULINK mô phỏng thiết bị BĐTT*

Trong mô hình thiết bị BĐTT ở hình 11.17 (khối có tên *Direct Converter*), nguyên lý ở hình 11.16 đã được mở rộng thêm khả năng *ngăn ngừa dòng quẩn*, là dòng xuất hiện tại thời điểm chuyển dẫn dòng giữa hai cầu CL mắc song song ngược với nhau. Giải pháp ngăn ngừa là tạo khoảng thời gian nghỉ khi dòng đi qua điểm 0. Để tạo được ta cài đặt thêm 2 tín hiệu dòng của máy điện ở đầu vào của mô hình *Direct Converter*.

Mô hình thiết bị BĐTT mới chỉ tìm cách tái tạo lại chế độ vận hành động cơ điện trực tiếp với lưới cứng. Trong mô hình chưa tích hợp khâu ĐC dòng. Tuy nhiên, sẽ hợp lý hơn nếu ta tìm cách cài đặt trong mô hình khâu ĐC dòng, được thiết kế tựa theo hệ tọa độ *dq* (tọa độ từ thông).

Hình 11.18 giới thiệu vài đặc tính điện áp, thu được sau mô phỏng bằng mô hình 11.17. Ta có thể nhận thấy rõ chất lượng tái tạo điện áp rất tốt.



Hình 11.18 *Trái: Hai thành phần điện áp α, β trước và sau thiết bị BĐTT; Phải: Ba điện áp pha ở đầu ra của thiết bị*

11.5 Tóm tắt nội dung chương 11

Sau khi đã nghiên cứu kỹ chương 11, người đọc cần nắm vững các nội dung sau đây:

1. Mô hình hóa trên nền MATLAB và SIMULINK các loại thiết bị biến đổi sử dụng van điện tử công suất như: Cầu chỉnh lưu 6 xung có điều khiển cắt pha, nghịch lưu băm xung nguồn áp, nghịch lưu nguồn dòng và thiết bị biến tần trực tiếp.
2. Kỹ năng tạo hàm S (tạo *S-Function*) bằng *m-File* hay *C-mex-File* để tạo ra những khối SIMULINK mới, mô phỏng các thiết bị biến đổi kể trên.

12 Thư viện mô hình phần cơ và ví dụ ứng dụng của Motion Control Blockset

12.1 Thư viện mô hình các phần tử truyền động cơ học

Sau khi ta đã làm quen với phương pháp mô hình hóa các phần tử của hệ thống truyền động điện, mục này tập trung chủ yếu vào các mô hình của hệ thống cơ. Khi nghiên cứu, khảo sát tổng thể các hệ thống điện – cơ (hình 10.1) ta cần thiết phải mô hình hóa được các phần tử cơ bằng thứ "*ngôn ngữ*" hay theo "*dạng*" giống như các phần tử điện. Chính vì vậy, ta lựa chọn "*lưu đồ tín hiệu*" với các mô tả toán học trên miền thời gian làm phương thức biểu diễn. Các phương thức biểu diễn khác của cơ khí như xác định "*tần số riêng*" của hệ / của phần tử vv..., không tạo điều kiện ghép nối với các mô hình phần điện. Các khảo sát về tần số của hệ thống cơ bằng mô phỏng sẽ phải được quy về miền thời gian. Tương tự, ta cũng không thể tính trực tiếp các phương trình đạo hàm riêng với thông số rải. Phải tìm được cách xấp xỉ gần đúng hoặc chuyển sang phương trình vi phân thường. Tùy theo mức độ lượng tử hóa, các phương pháp sử dụng phần tử hữu hạn có thể đưa tới kết quả rất chính xác, nhưng ta lại không có khả năng tích hợp các kết quả đó vào môi trường MATLAB. Ta sẽ phải tạo được các giao diện cho phép ghép nối các hệ mô phỏng khác nhau, ví dụ: Ghép ADAMS (mô phỏng cơ khí) với MATLAB. Nhưng những mô phỏng đó đòi hỏi máy tính rất mạnh, vì vậy giải pháp dung hòa chấp nhận được sẽ là *phương pháp mô hình hóa các hệ thống nhiều vật*.

Hệ thống cơ của máy thường được thiết kế với trình tự: Thiết bị truyền động điện (như động cơ điện) được ghép với thiết bị phụ tải (như dụng cụ, đồ gá hay máy công tác) qua các thành phần truyền như trục cứng, bánh răng, côn hay trục vít vô tận. Trong nhiều trường hợp ta có thể phân tích các thành phần đó thành hệ cơ sở (có thể gọi là hệ tế bào) bao gồm: Phần tử lò so – phần tử làm nhụt – phần tử khối[1]. Khi phân tích như vậy ta đã *phân rã hệ thống cơ nhiều vật*[2] thành nhiều *hệ một vật* nối với nhau. Tiếp theo, ta tìm cách mô hình hóa hệ tế bào, để từ đó xây dựng mô hình tổng thể của cả hệ lớn. Trong cách mô hình hóa như vậy, không nhất thiết phải bảo đảm sự đồng nhất về biên giới giữa các mô hình con với biên giới thực tế giữa các phần tử cơ. Ví dụ: Một trục cơ với kích thước dài rất có thể cần được chia thành vài ba đoạn ngắn, và đoạn

[1] Viết gọn lại: Lò so – Nhụt – Khối

[2] Hệ nhiều vật: Multi-Body System; hệ một vật: Single-Body System

ngắn ở đầu trục có thể nhận thêm vào mô hình của nó quán tính trễ của côn nối. Khi đã hoàn thành việc phân rã, ta có thể xây dựng phương trình chuyển động của hệ tế bào với điều kiện luôn thỏa mãn quan hệ sau:

$$J\ddot{\varphi}(t)+D\dot{\varphi}(t)+C\varphi(t)=\sum m_{\varphi}(t) \tag{12.1}$$

φ	Góc xoắn	C	Hệ số độ cứng của lò so
$\dot{\varphi}$	Đạo hàm bậc nhất theo thời gian của góc xoắn: Vận tốc góc	D	Hệ số tắt dần
		J	Hệ số (mômen) quán tính
$\ddot{\varphi}$	Đạo hàm bậc hai theo thời gian của góc xoắn: Gia tốc góc	m_{φ}	Mômen ngoài

Trong thực tiễn, bên cạnh các biến đầu vào và biến trạng thái, rất có thể một vài tham số là hàm của thời gian hay của biến trạng thái. Ví dụ: Mômen quán tính của một cánh tay đòn sẽ biến thiên phụ thuộc vào góc của tay đòn đó. Trong nhiều trường hợp ta có thể xấp xỉ gần đúng bằng cách coi các tham số đó là hằng, và vì vậy đã loại trừ mất nguyên nhân kích dao động do tham số biến thiên theo thời gian gây nên. Để khắc phục sai lạc đó ta có thể bổ sung thêm các hàm tham số vào mô hình.

Việc xác định giá trị của các tham số bao giờ cũng xuất phát từ tài liệu thiết kế. Ví dụ: Mômen quán tính được tính từ khối lượng và kích thước hình học của phần tử. Xét một ống rỗng với khối lượng m, bán kính trong r_i, bán kính ngoài r_a ta có mômen quán tính J:

$$J=\frac{1}{2}m\left(r_a^2+r_i^2\right) . \tag{12.2}$$

Trong tài liệu chuyên ta tìm thấy công thức tính hệ số độ cứng lò so với S là hệ số trượt (*Sliding*), I_p mômen quán tính bề mặt, l là độ dài của vật.

$$C=\frac{S\,I_p}{l} \tag{12.3}$$

Với vật là ống như trên, các tham số trong (12.3) được tính như sau:

$$I_{p,round_tube}=\frac{\pi}{2}\left(r_a^4+r_i^4\right)\rightarrow C=\frac{S\pi\left(r_a^4+r_i^4\right)}{2l} \tag{12.4}$$

Việc xác định hệ số tắt dần rất phức tạp do vật chịu ảnh hưởng nhiều của các tác động từ bên ngoài, và việc xác định hầu hết được tiến hành bằng thực nghiệm. Theo tài liệu nước ngoài, có thể tính gần đúng cho các hệ tắt dần yếu trong chế tạo máy theo:

$$D=\sqrt{JC\left(1-\eta_{res}^2\right)} \tag{12.5}$$

$\eta_{res} \approx 0{,}97...0{,}98$ (Tỷ lệ tần số)

Với các tham số từ số liệu thiết kế ta có thể tham số hóa các mô hình cơ. Mô hình tế bào "*Spring–Damping–Mass*" ("*Lò so–Nhụt–Khối lượng*") được chia thành hai mô hình con.

- Mô hình "*Mass*" mô tả một vật thể quay, chịu tác động của một mômen gia tốc và một mômen hãm. Xuất phát từ phương trình (12.1), vế bên

phải bao gồm các biến đầu vào, để tính đạo hàm bậc nhất của góc (tính tần số). Các tham số là mômen quán tính và giá trị ban đầu của tần số. Ta thu được:

$$\omega = \frac{1}{J}\int \left(m_{I1} - m_{I2}\right) dt + \omega_0 \tag{12.6}$$

m_{I1} Mômen dương tại đầu vào 1 [Nm]
m_{I2} Mômen âm tại đầu vào 2 [Nm]
ω Tần số quay của khối [s^{-1}]
ω_0 Giá trị ban đầu của tần số quay của khối [s^{-1}]

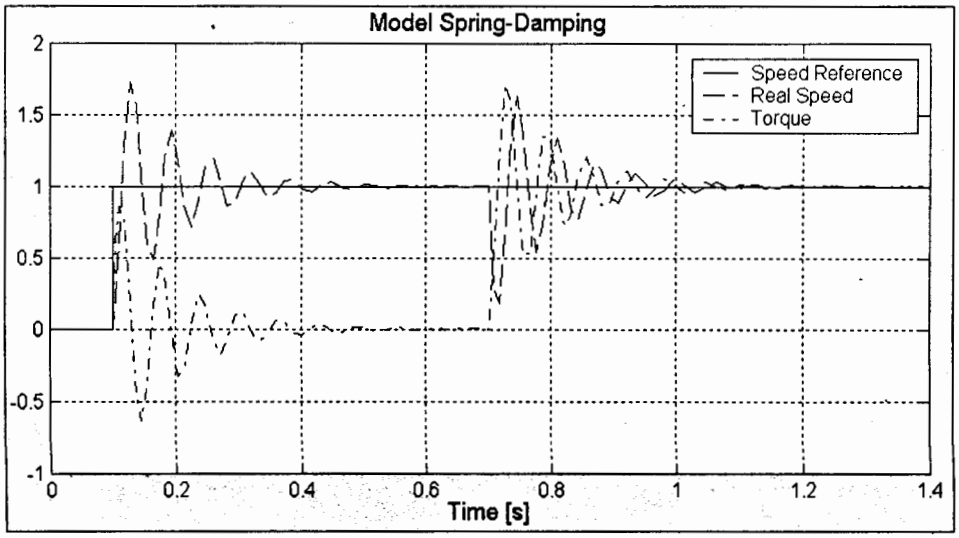
- Mô hình "*Spring*" mô phỏng mối liên kết đàn hồi giữa các "*Mass*", trong đó chỉ duy nhất một mômen được truyền đạt. φ trong (12.1) mô phỏng góc xoắn, là chênh lệch về góc xuất hiện giữa hai "*Mass*" ở hai đầu. $\dot{\varphi}$ là vận tốc góc, đạo hàm bậc nhất của φ. Mô hình "*Spring*" tính biểu thức thứ hai của (12.1).

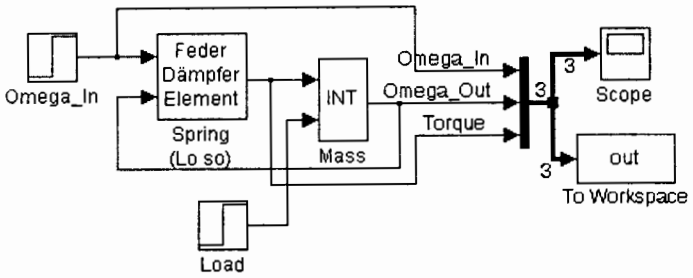
$$m_\varphi = C\left[\int \left(\omega_{I1} - \omega_{I2}\right) dt + \varphi_0\right] + D\left(\omega_{I1} - \omega_{I2}\right) \tag{12.7}$$

ω_{I1} Tần số quay của khối lượng đầu lò so
ω_{I2} Tần số quay của khối lượng cuối lò so
m_φ Mômen truyền qua phần tử nối
φ_0 Góc truyền qua phần tử nối

Với hai khối "*Mass*" và "*Spring*" ta có thể mô hình hóa các hệ tế bào (hệ hai vật), tiến tới mô hình hóa và mô phỏng các hệ nhiều vật. Hình 12.1a) giới thiệu kết quả mô phỏng một hệ hai vật (hệ tế bào, hình 12.1b).

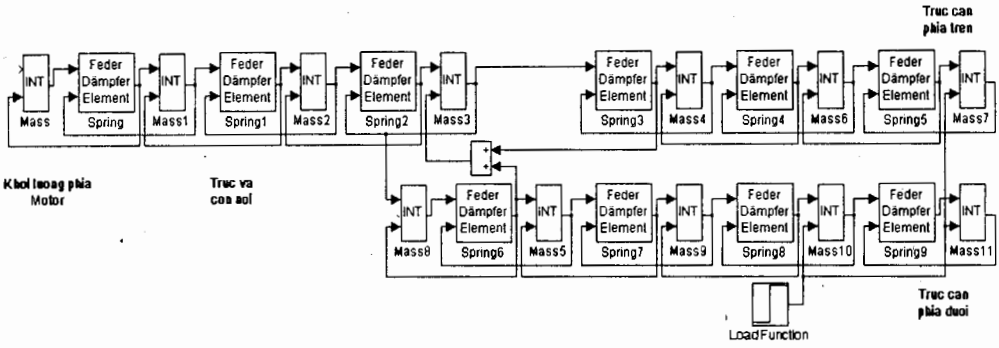
a)





b)

Hình 12.1 *a) Đáp ứng tốc độ (nét --) tại đầu cuối lò so khi có bước nhảy (nét liền) tốc độ ở đầu vào lò so, và đặc tính mômen (nét -.-) tại đầu cuối. b) Mô hình SIMULINK của hệ tế bào (hệ hai vật)*



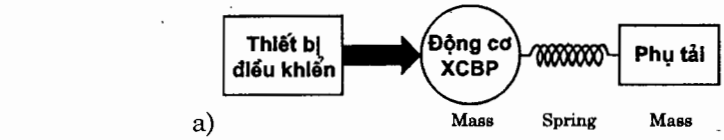
Hình 12.2 *Cấu trúc mô phỏng một hệ tuyến tính 11 vật*

Hình 12.2 giới thiệu một mô hình mô phỏng phần cơ của thiết bị truyền động chính cán thép. Trong ví dụ đó, hệ thống cơ được phân rã thành 11 khối "*Mass*" và các khâu ghép nối.

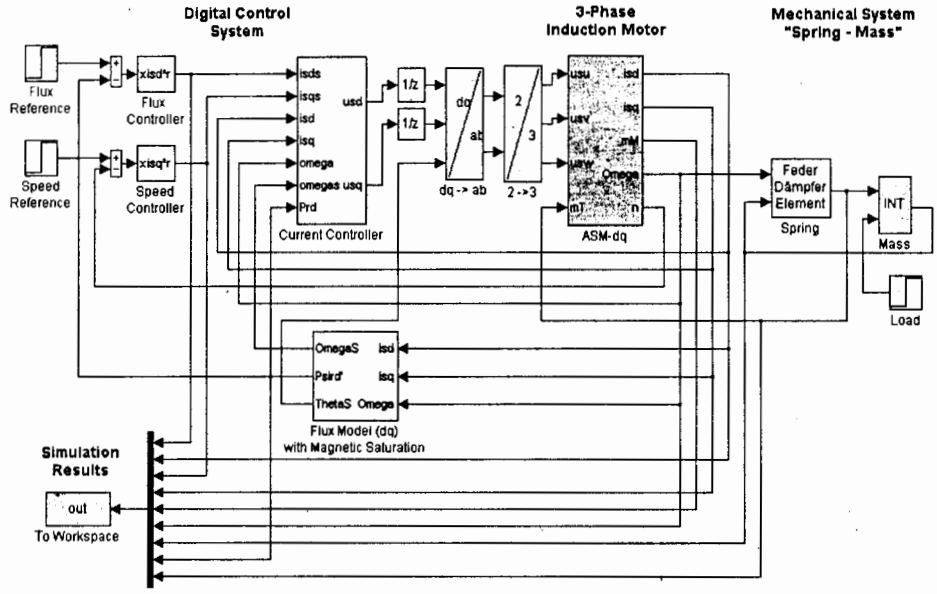
12.2 Vài ví dụ ứng dụng

12.2.1 Mô phỏng một trục chuyển động của tay máy sử dụng MĐDB

Một trục chuyển động của tay máy (hay cũng có thể của các ứng dụng khác) được mô tả một cách đơn giản ở hình 12.3a. Đặc điểm của trục truyền động đơn là: Động cơ và phụ tải được ghép với nhau qua một phần tử cơ với đặc tính có thể bị xoắn, gây dao động với một hệ số tắt dần nào đó. Có thể diễn đạt đơn giản: Động cơ và phụ tải được ghép với nhau thành một hệ hai vật, đã được mô phỏng minh họa ở hình 12.1.



a)



b)

Hình 12.3 *Điều khiển một trục chuyển động của tay máy: a) Sơ đồ khối truyền động của hệ hai vật; b) Cấu trúc mô phỏng tổng thể hệ thống điện - cơ*

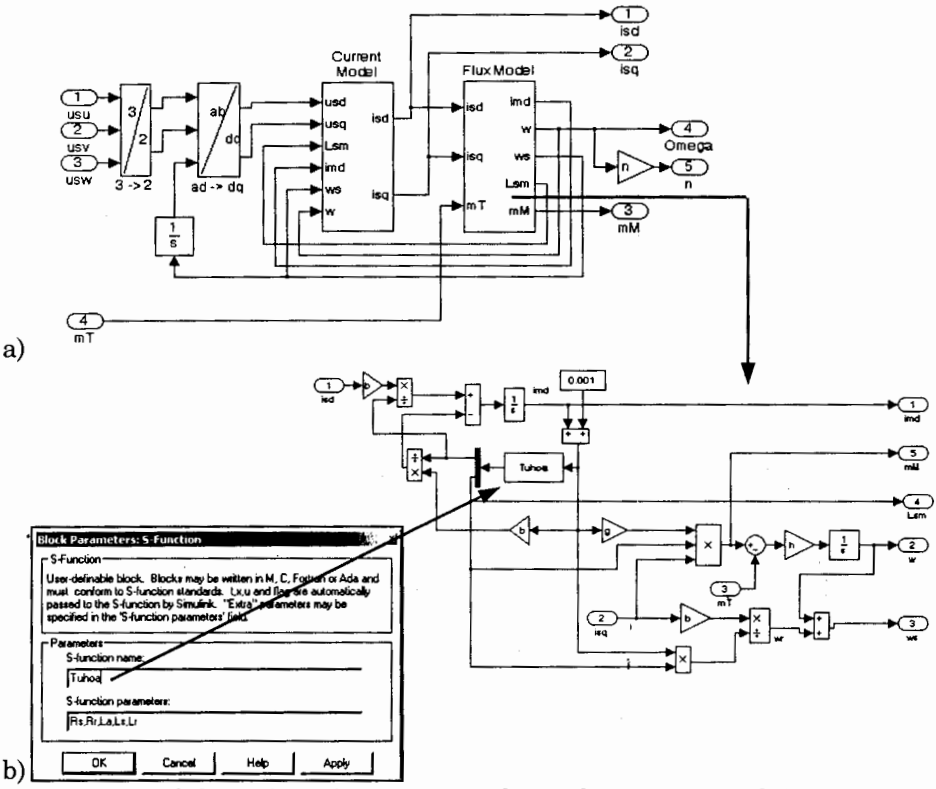
Động cơ XCBP sử dụng trong hệ trên là một MĐDB có Rotor lồng sóc với mô hình trên hệ tọa độ từ thông *dq* theo công thức (10.13), minh họa ở hình 10.14, thậm chí đã được mô hình hóa trên SIMULINK như hình 10.15. Tuy nhiên, mô hình 10.15 chưa thể hiện được bản chất phi tuyến tiềm ẩn trong phương trình (10.13) và hình 10.14, đó là: Đặc điểm bão hòa của điện cảm, được mô tả bởi đường đặc tính từ hóa $L_m = L_m(i_m)$.

Để cài đặt đường đặc tính từ hóa $L_m = L_m(i_m)$, mô hình MĐDB ở cấu trúc 12.3 đã được tách thành 2 phần: Mô hình dòng và mô hình từ thông (hình 12.4a). Khi tách ra như vậy ta sẽ dễ dàng soạn thảo một *S-Function* (hàm S: có tên là `Tu hoa`, hình 12.4b) mô tả đường đặc tính ta đã biết. Bạn đọc cũng có thể sử dụng khối *Look-Up Table* để cài đặt đặc tính.

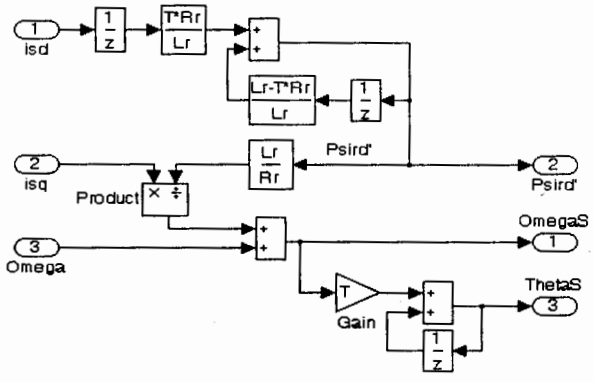
Khi ĐK một MĐDB ta biết: Từ thông Rotor ψ_r là một đại lượng ta không muốn đo, muốn biết giá trị của ψ_r ta có thể sử dụng một mô hình đơn giản, hoặc một khâu QS để tính (hình 12.5). Với ta tính được các đại lượng quan trọng khác như tần số Rotor ω_r, Stator ω_s và vị trí của hệ tọa độ ϑ_s.

$$\omega_r = \frac{i_{sq}}{T_r\left(\psi_{rd}/L_m\right)} \rightarrow \omega_s = \omega + \omega_r \rightarrow \vartheta_s(k) = \vartheta_s(k-1) + \omega_s(k)T \quad (12.8)$$

k = 1, 2, ... Thời điểm trích mẫu T Chu kỳ trích mẫu



Hình 12.4 *a) Mô hình MĐDB được tách thành hai mô hình con: Mô hình dòng và mô hình từ thông; b) Mô hình từ thông và đường đặc tính "Từ hóa", thực hiện dưới dạng hàm S*

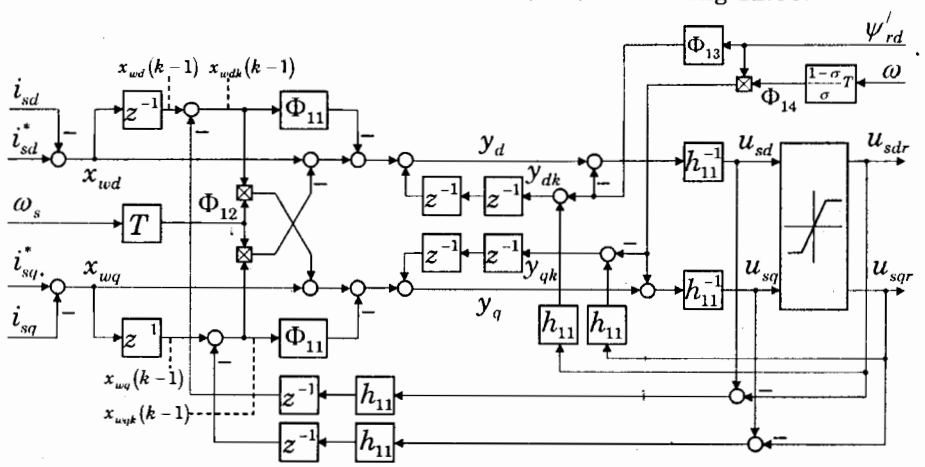


Hình 12.5 *Mô hình từ thông*

Hệ thống ĐK ở hình 12.3 được cài đặt trên hệ tọa độ *dq*, tức là hệ tọa độ tựa theo từ thông Rotor ψ_r. Lợi thế khi mô hình hóa MĐDB trên hệ tọa độ đó thể hiện qua hai quan hệ sau:

$$\psi_{rd}(s) = \frac{L_m}{1+sT_r} i_{sd}\ ; \quad m_M = \frac{3}{2}\frac{L_m}{L_r} z_p \psi_{rd} i_{sq} \tag{12.9}$$

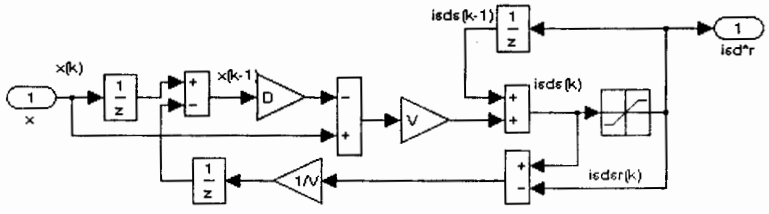
Các ký hiệu sử dụng trong (12.9) đã được giải thích ở mục 10.3. Có thể diễn đạt lợi thế đó như sau: Nếu thành công trong việc áp đặt không chậm trễ hai thành phần dòng i_{sd}, i_{sq}, ta có thể sử dụng i_{sd} làm đại lượng ĐK ổn định từ thông ψ_r. Khi ψ_r đã được ĐK ổn định (có thể coi là hằng), ta dùng i_{sq} để ĐK mômen quay m_M. Điều kiện tiên quyết để tận dụng lợi thế đó là: Ta phải thành công trong việc *áp đặt không trễ hai thành phần dòng*. Để thực hiện ta sử dụng một khâu ĐC hai chiều như hình 12.6a với mô hình SIMULINK trong 12.6b.



Hình 12.6 *Khâu ĐC hai chiều có khả năng ĐK cách ly hai thành phần dòng i_{sd} và i_{sq} với động học cao*

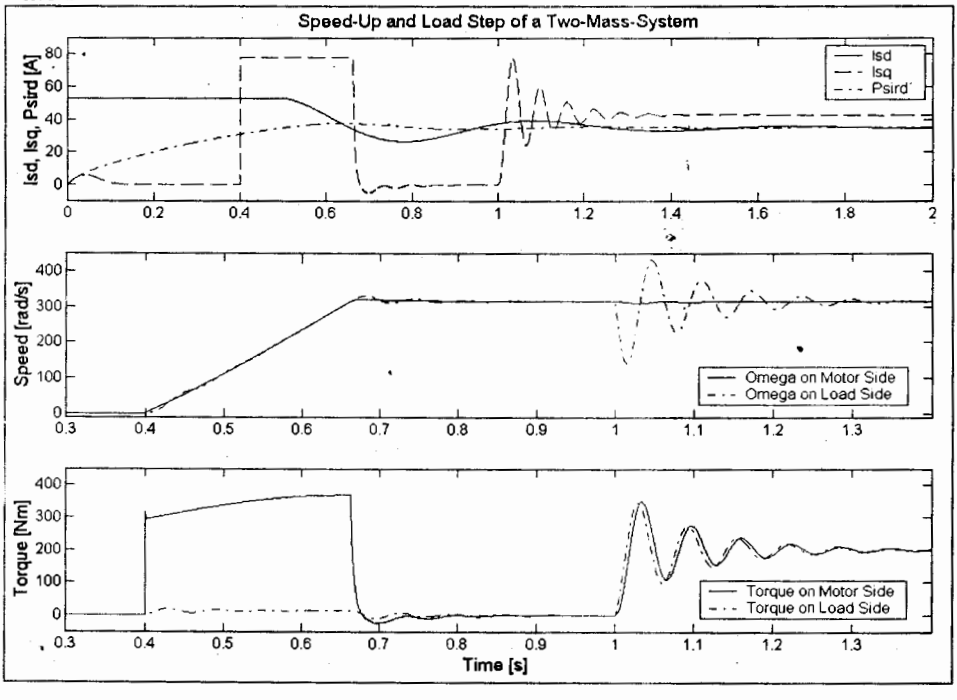
Tại đây chỉ lưu ý bạn đọc rằng, so với hình 12.6a, mô hình 12.6b chưa cài đặt khối hạn chế đầu ra, là khối cho phép ta xét đến điều kiện biên quan trọng của hệ thống: Do điện áp mạch một chiều trung gian U_{DC} có giới hạn, điện áp đầu ra của khâu ĐC cần phải được hạn chế tương ứng.

Khối cuối cùng cần được đề cập đến trong cấu trúc 12.3 là hai khâu ĐC số với đặc tính PI, dùng để ĐC từ thông (*Flux Controller*) và tốc độ quay (*Speed Controller*). Hai khâu đó có mô hình SIMULINK như hình 12.7.



Hình 12.7 *Khâu điều chỉnh PI số (Digital PI-Controller)*

Có thể dễ dàng tới được sơ đồ như 12.7 bằng cách gián đoạn hóa một khâu PI tương tự có sẵn, và vấn đề tham số hóa sẽ tự động được giải quyết trong quá trình số hóa. Ngoài ra, cũng có thể thiết kế bộ tham số theo tiêu chuẩn tối ưu module số.



Hình 12.8 *Kết quả mô phỏng cấu trúc ở hình 12.3: Quá trình khởi động động cơ và đóng phụ tải tại đầu cuối của phần tử lò so.*

Hình 12.8 giới thiệu một số *kết quả mô phỏng tổng thể hệ thống điện – cơ một trục chuyển động* của tay máy theo sơ đồ 12.3b. Hình trên cùng cho ta thấy diễn biến của các thành phần dòng tạo từ thông i_{sd}, tạo mômen i_{sq} và từ thông ψ_{rd} khi khởi động động cơ và khi đóng tải ở đầu phía bên kia của phần tử nối (có đặc điểm lò so).

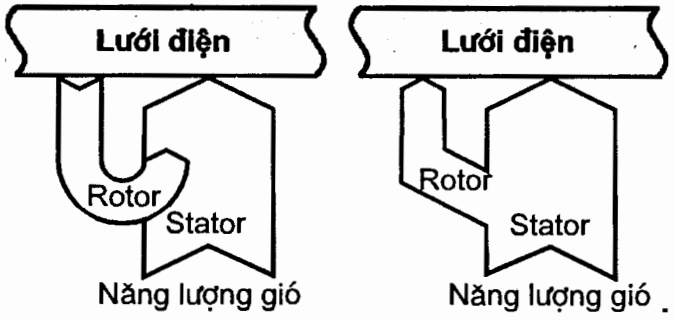
Hình giữa cho ta thấy khả năng dao động của tốc độ quay bên phía tải khi có đột biến phụ tải. Điều này minh chứng rất rõ: Việc ĐK tốc độ quay khối đầu cuối qua ghép nối lò so bằng khâu PI đơn giản sẽ khó đưa lại kết quả tốt. Trong thực tế có thể ta sẽ phải sử dụng các phương pháp ĐK khác, ví dụ: ĐK trên không gian trạng thái, hay ĐK có sử dụng khâu QS tốc độ quay phía phụ tải.

12.2.2 Mô phỏng hệ thống phát điện chạy sức gió sử dụng MĐDB-RDQ

Hình 10.28 mô tả sơ đồ khối của một hệ thống phát điện chay sức gió sử dụng máy điện dị bộ nguồn kép (còn gọi là máy điện dị bộ Rotor dây quấn: MĐDB-RDQ, để tiện trong mục này ta chỉ viết MĐ). Lợi thế chính của giải pháp là khả năng ĐK máy phát từ Rotor, cho phép giảm đáng kể công suất và giá thành của thiết bị ĐK.

Khi đã hòa đồng bộ với lưới điện, dòng năng lượng qua máy phát có thể được mô tả dễ hiểu nhờ hình 12.9. Có thể xảy ra hai trường hợp sau:

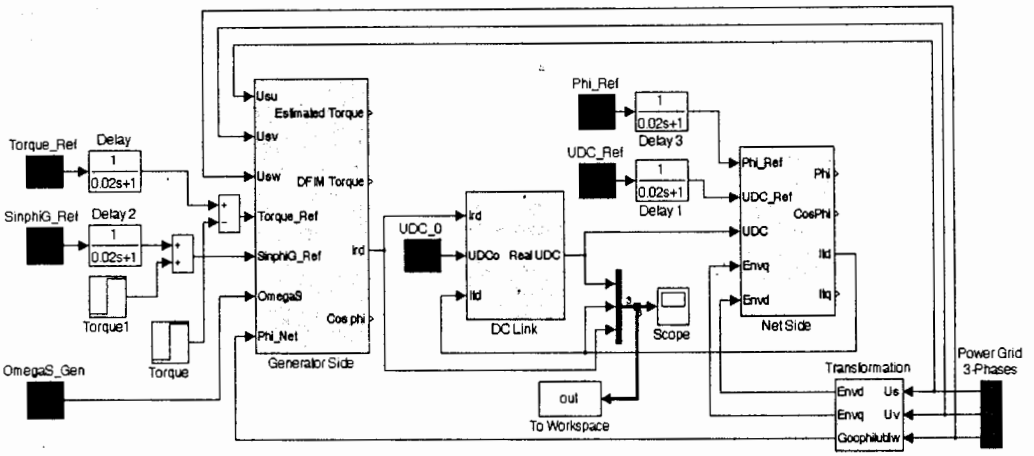
- Gió thổi cánh quạt quay ứng với tần số thấp hơn tần số của lưới điện: Đó là *chế độ vận hành dưới đồng bộ*, và MĐ lấy năng lượng từ lưới qua Rotor.
- Gió thổi cánh quạt quay ứng với tần số cao hơn tần số của lưới điện: Đó là *chế độ vận hành trên đồng bộ*, và MĐ hoàn năng lượng trở về lưới qua Rotor.



Hình 12.9 Chiều của dòng năng lượng ở chế độ dưới (trái) và trên (phải) đồng bộ

Dễ dàng thấy rằng: Để đảm bảo vận hành MĐ ở hai chế độ trên, thiết bị biến đổi ở cả hai phía lưới và máy phát (hình 10.28: khối NI và GI) đều phải là NL có khả năng ĐK dòng năng lượng chảy theo hai chiều. Nối giữa hai NL là mạch

điện áp một chiều sử dụng tụ lọc. Mô hình SIMULINK của một hệ thống phức hợp như vậy được minh họa ở hình 12.10.



Hình 12.10 *Mô hình mô phỏng cấu trúc ĐK hệ thống phát điện chạy sức gió sử dụng MĐDB-RDQ (Generator Side) ghép với NL phía lưới (Net Side) qua mạch một chiều trung gian (DC Link)*

Mô hình 12.10 là một hệ thống rất phức hợp được gom gọn lại thành ba khối chính: Khối "*Generator Side*" với mô hình máy phát và hệ thống ĐK, khối "*Net Side*" với mô hình phía lưới và hệ thống ĐK, hai khối đó trao đổi điện năng với nhau qua khối "*DC Link*". Sau đây ta sẽ lần lượt xét từng khối.

a) Khối "Generator Side" với mô hình máy phát và hệ thống điều khiển

Tại mục 10.6 ta đã xây dựng hai mô hình của máy phát, đó là các hệ phương trình (10.26) và (10.30). Ta sẽ sử dụng (10.30) làm mô hình để mô phỏng máy phát (hình 10.29), còn (10.26) làm xuất phát điểm để tìm các thuật toán ĐK máy phát.

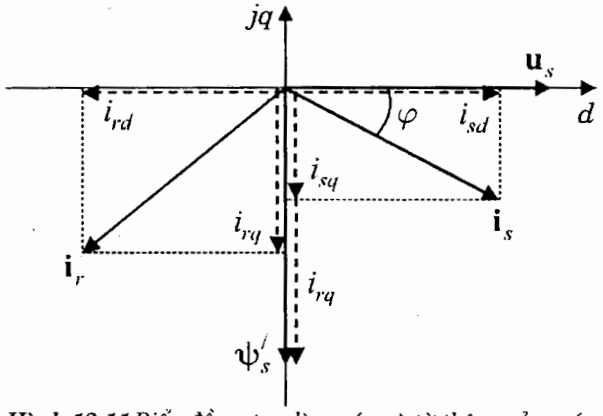
Để hiểu phương pháp ĐK ta viết lại phương trình từ thông Stator trong (10.25) dưới dạng thành phần như sau:

$$\begin{cases} \psi'_{sd} = \dfrac{L_s}{L_m} i_{sd} + i_{rd} \approx 0 \\ \psi'_{sq} = \dfrac{L_s}{L_m} i_{sq} + i_{rq} \approx \left|\psi'_s\right| \end{cases} \tag{12.10}$$

Vì công suất của máy phát thường là rất lớn, ta có thể viết $\frac{L_s}{L_m} \approx 1$ và do đó:

$$\begin{cases} i_{sd} + i_{rd} \approx 0 \\ i_{sq} + i_{rq} \approx \left|\psi'_s\right| = \psi'_{sq} \end{cases} \tag{12.11}$$

Các quan hệ dòng, áp và từ thông trong (10.24) và (12.11) được minh họa một cách dễ hiểu bằng biểu đồ vector như sau:



Hình 12.11 Biểu đồ vector dòng, áp và từ thông của máy phát

Để tiện theo dõi, ta viết lại một lần nữa công thức (10.27):

$$m_M = -\frac{3}{2} z_p \frac{L_m}{L_s} \psi_{sq} i_{rd} = -\frac{3}{2} z_p (1-\sigma) L_r \psi'_{sq} i_{rd} \qquad (12.12)$$

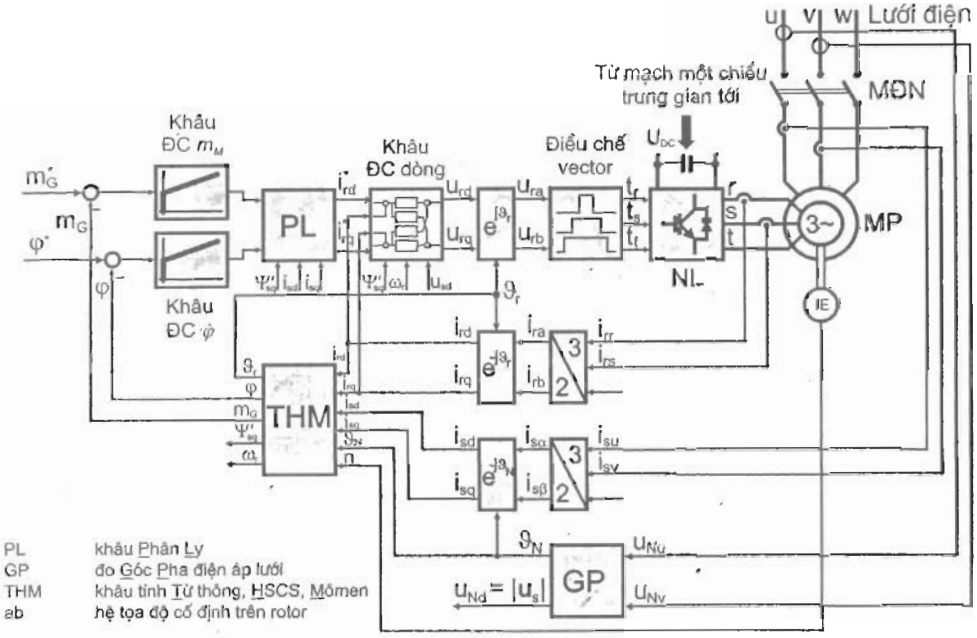
Kết hợp giữa (10.24), (12.11), (12.12) và hình 12.11 ta rút ra các nhận xét quan trọng sau đây:

- Theo (10.24), vector từ thông ψ_s luôn đứng vuông góc với vector điện áp $\mathbf{u}_s$ và có biên độ cũng do $\mathbf{u}_s$ quyết định (do lưới điện quyết định).
- Do $|\psi_s| = \psi_{sq}$ có module là hằng, theo (12.12) mômen m_M (tức là công suất hữu công) chỉ phụ thuộc vào i_{rd}. Vì vậy i_{rd} *chính là đại lượng ĐK mômen* m_M (*ĐK công suất hữu công*). Khi i_{rd} thay đổi, thành phần dòng Stator i_{sd} cũng biến thiên theo.
- Khi công suất vô công nhỏ, tức là $\cos\varphi \to 1$ ta có $\sin\varphi \to 0; \sin\varphi \approx \varphi$. Theo hình 12.11 ta có:

$$\sin\varphi = \frac{i_{sq}}{|\mathbf{i}_s|} \approx \frac{\psi'_{sq} - i_{rq}}{|\mathbf{i}_s|} \qquad (12.13)$$

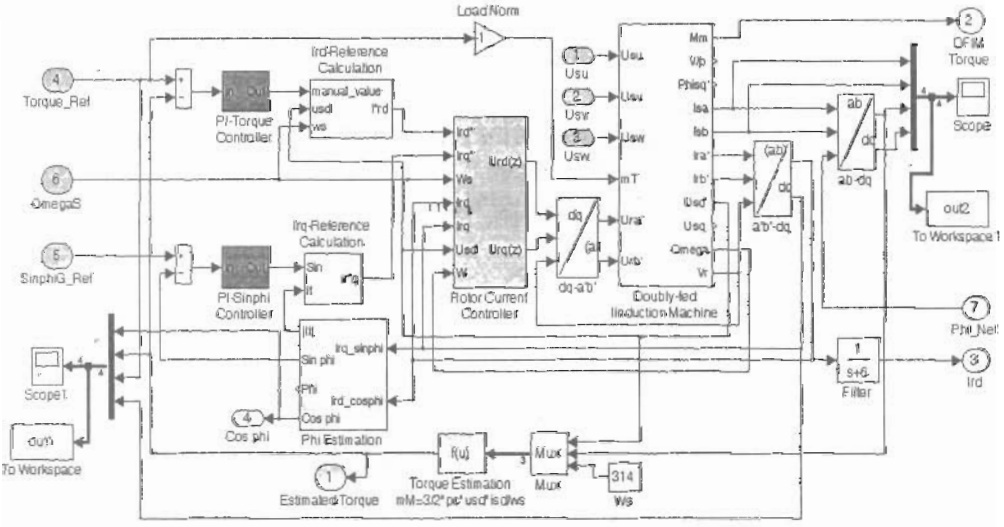
Vậy là theo (12.13) ta có thể sử dụng thành phần dòng i_{rq} *làm đại lượng ĐK góc pha* φ (*ĐK công suất vô công*).

Theo các nhận xét trên, để có thể sử dụng hữu hiệu hai thành phần dòng i_{rd}, i_{rq} làm đại lượng ĐK công suất hữu công, vô công, ta sẽ phải thực hiện thành công vòng ĐC trong cùng với một khâu ĐC hai chiều, bảo đảm cách ly và áp đặt không trễ hai thành phần dòng đó. Giá trị chủ đạo (giá trị đặt, *Set Points*) của i_{rd}, i_{rq} sẽ là các biến đầu ra của hai khâu ĐC mômen m_M và $\sin\varphi$ ở vòng ngoài. Cấu trúc của hệ thống ĐK được mô tả ở hình 12.12.



Hình 12.12 Cấu trúc hệ thống điều khiển phía máy phát

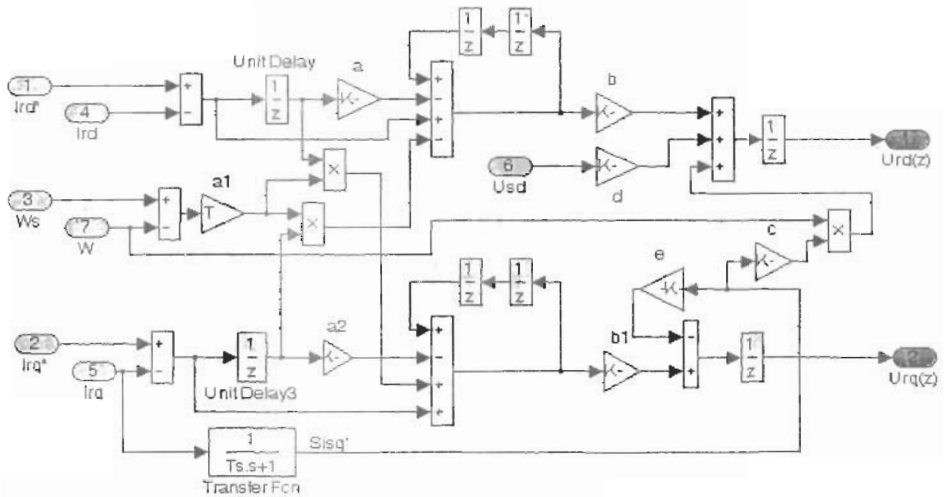
Từ sơ đồ cấu trúc ở hình 12.12 ta có thể xây dựng sơ đồ SIMULINK mô phỏng hệ thông điều khiển phía máy phát ở hình 12.13.



Hình 12.13 Sơ đồ SIMULINK mô phỏng cấu trúc điều khiển ở hình 12.12 với mô hình máy phát (khối Doubly-fed Induction Machine) theo hình 10.29

Khâu ĐC dòng trong 10.13 (khối *Rotor Current Controller*) là một khâu với đặc tính *Dead-Beat* có động học cao, thực hiện theo sơ đồ SIMULINK ở hình 12.14. Thuật toán ĐC như sau:

$$\begin{cases} u_{rd}(k+1) = h_{11r}^{-1}[x_{wd}(k) - \Phi_{11}x_{wd}(k-1) - \Phi_{12}x_{wq}(k-1) \\ \qquad + y_d(k-2) - \Phi_{14}\psi'_{sq} - h_{11s}u_{sd}] \\ u_{rq}(k+1) = h_{11r}^{-1}[x_{wq}(k) + \Phi_{12}x_{wd}(k-1) - \Phi_{11}x_{wq}(k-1) \\ \qquad + y_q(k-2) - \Phi_{13}\psi'_{sq}] \end{cases} \tag{12.14}$$



Hình 12.14 Sơ đồ SIMULINK của khâu ĐC dòng hai chiều

b) Khối "Net Side" với mô hình phía lưới và hệ thống điều khiển

Nhiệm vụ quan trọng của phần ĐK phía lưới là *lấy năng lượng từ lưới* để cung cấp cho mạch một chiều *ở chế độ dưới đồng bộ* hoặc *hoàn năng lượng* từ mạch một chiều lên lưới *ở chế độ trên đồng bộ*. Trong cả hai quá trình đó, điện áp một chiều trung gian U_{DC} phải được giữ ổn định không đổi.

Yêu cầu kể trên có thể được thực hiện một cách đơn giản. Quan sát dòng i_N ở đầu ra phía lưới của CL trên hệ tọa độ *dq* (*hệ tọa độ tựa hướng điện áp lưới*[1]), ta có:

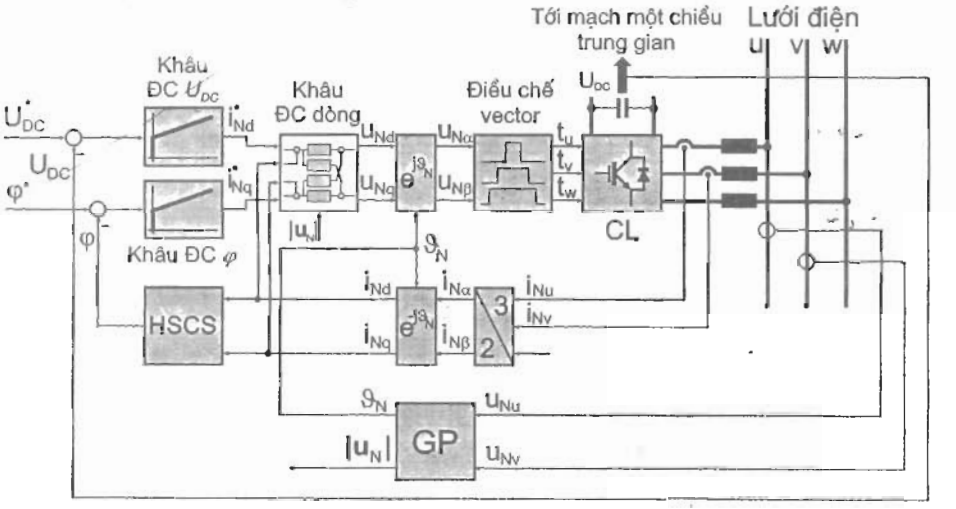
$$\underline{i}_N = i_{Nd} + j i_{Nq} \tag{12.15}$$

Trong dòng kể trên, thành phần i_{Nd} có tác dụng sản sinh công suất hữu công (trùng pha với vector điện áp lưới $\mathbf{u}_N$), i_{Nq} sản sinh công suất vô công (vuông góc với vector $\mathbf{u}_N$). Vì vậy, *dòng i_{Nd} có thể được sử dụng làm dòng nạp hoặc xả mạch một chiều trung gian*. Có nghĩa là, nếu thành công trong việc áp đặt nhanh và

[1] Tựa hướng điện áp lưới: THĐAL, Grid-Voltage Orientation

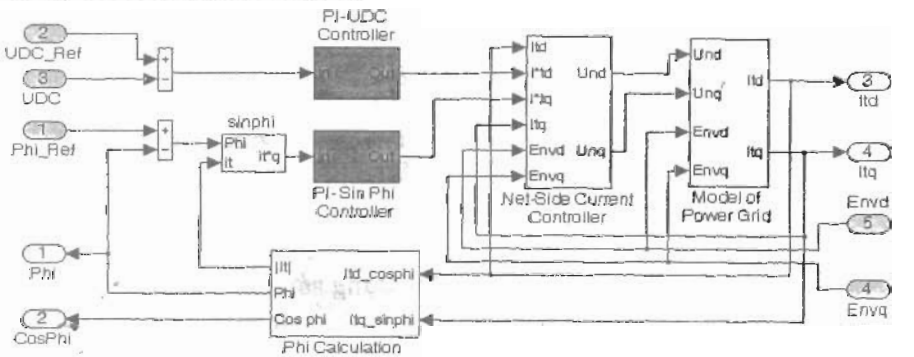
chính xác dòng i_{Nd} nhờ một khâu ĐC dòng, đầu ra của khâu điều chỉnh U_{DC} có thể được sử dụng trực tiếp làm giá trị đặt của dòng i_{Nd}.

Theo diễn giải trên, thành phần i_{Nq} dường như không có vai trò quan trọng và vì vậy có thể đặt bằng không. Tuy vậy, vai trò sản sinh công suất vô công có thể được tận dụng để ĐK hệ số công suất (ví dụ: HSCS của toàn hệ) và đầu ra của khâu ĐC φ có thể được sử dụng làm giá trị đặt của i_{Nq}. Lúc này, khâu ĐC φ thứ hai giữ vai trò của tụ bù quen biết. Giống như hệ thống ĐK phía máy phát, HSCS được điều chỉnh gián tiếp qua điều chỉnh sinφ. Cả hai khâu ĐC φ và U_{DC} đều là khâu điều chỉnh PI thông thường. Đến đây ta đã có thể xây dựng cấu trúc ĐK cho phía lưới như trong hình 12.15.

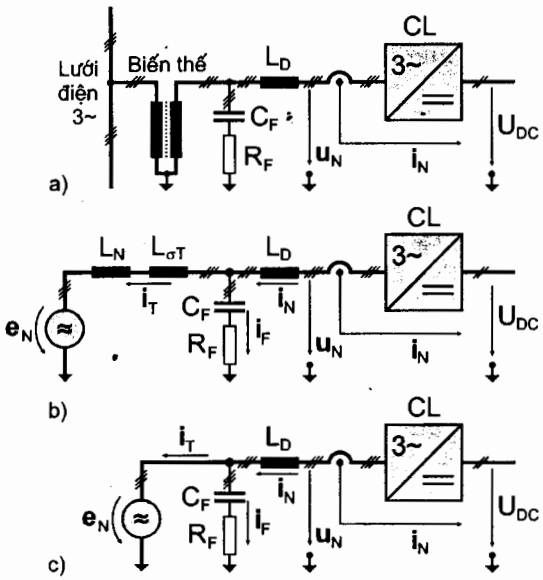


Hình 12.15 Cấu trúc hệ thống điều khiển phía lưới

Từ sơ đồ cấu trúc ở hình 12.15 ta có thể xây dựng sơ đồ SIMULINK mô phỏng hệ thống điều khiển phía máy phát ở hình 12.16.



Hình 12.16 Sơ đồ SIMULINK mô phỏng cấu trúc điều khiển ở hình 12.15 với mô hình phía lưới (khối Model of Power Grid)



Hình 12.17 Sơ đồ mạch điện phía lưới

Để hiểu được chính xác sơ đồ 12.16 ta hãy xét cụ thể mạch điện phía lưới. Đầu ra của CL thường được ghép với lưới qua cảm kháng L_D (có điện trở cuộn dây là R_D), qua khâu lọc RC và biến áp (hình 12.17a). Điện cảm của lưới cho trước là L_N, biến áp được thay thế tương đương (gần đúng) bởi điện cảm tiêu tán $L_{\sigma T}$ và điện áp lưới được thay bởi nguồn áp $\mathbf{e}_N$, từ đó ta có sơ đồ thay thế cho mạch điện phía lưới như hình 12.17b. Vì tổng điện áp rơi trên biến thế và điện cảm lưới rất nhỏ so với điện áp rơi trên khâu lọc RC, ta có thể bỏ qua chúng và thu được sơ đồ tối giản phục vụ thiết kế ở hình 12.17c.

Ở trạng thái xác lập, ta thu được phương trình lọc RC từ hình 12.17c:

$$\mathbf{e}_N = \frac{1}{j\omega_s C_F}\mathbf{i}_F + R_F\mathbf{i}_F \tag{12.16}$$

Trên hệ tọa độ THĐAL với $\underline{e}_N = e_{Nd} + je_{Nq}$ và $e_{Nq} = 0$ ta có thể viết:

$$\begin{cases} e_{Nd} = R_F i_{Fd} - \dfrac{1}{\omega_s C_F} i_{Fq} \\ 0 = R_F i_{Fq} + \dfrac{1}{\omega_s C_F} i_{Fd} \end{cases} \tag{12.17}$$

Trong đó: $\mathbf{i}_F = i_{Fd} + ji_{Fq}$. Nhờ (12.17) ta có thể tính chính xác giá trị của các thành phần dòng i_{Fd}, i_{Fq} ở chế độ xác lập. Ở chế độ xác lập, các thành phần dòng kể trên cùng với điện áp lưới $\mathbf{e}_N$ là không đổi, vì vậy, đối với mạch vòng ĐC dòng phía lưới chúng chỉ giữ vai trò của các đại lượng nhiễu cố định. Tương tự khâu ĐC dòng Rotor, các đại lượng nhiễu phía lưới cũng có thể bị triệt tiêu ảnh hưởng nhờ khâu bù nhiễu hoặc nhờ thành phần tích phân tiềm ẩn trong thuật toán ĐC. Từ hình 12.17c ta có thể xây dựng các phương trình dòng áp phía lưới.

$$\begin{cases} \mathbf{u}_N = R_D\mathbf{i}_N + L_D\dfrac{d\mathbf{i}_N}{dt} + \mathbf{e}_N \\ \mathbf{i}_N = \mathbf{i}_T + \mathbf{i}_F \end{cases} \tag{12.18}$$

Sau khi chuyển (12.18) sang hệ tọa độ THĐAL, đồng thời thế dòng $\mathbf{i}_N$ trong phương trình trên bởi $\mathbf{i}_N$ từ phương trình dưới, ta thu được phương trình điện áp mới:

$$\mathbf{u}_N = R_D\mathbf{i}_T + L_D\frac{d\mathbf{i}_T}{dt} + j\omega_s L_D\mathbf{i}_T + \mathbf{e}_{Nv} \tag{12.19}$$

với:

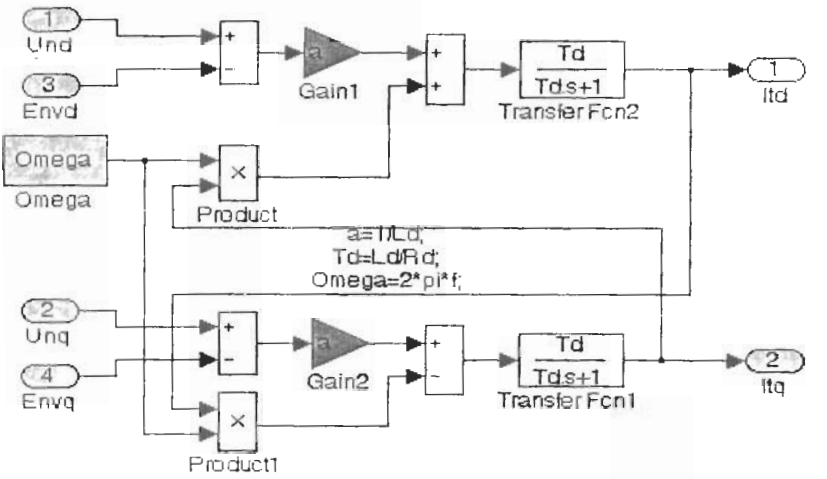
$$\begin{aligned}\mathbf{e}_{Nv} &= \mathbf{e}_N + j\omega_s L_D\mathbf{i}_F + R_D\mathbf{i}_F = e_{Nvd} + je_{Nvq}\\ L_D\frac{d\mathbf{i}_F}{dt} &= \mathbf{0}\end{aligned} \tag{12.20}$$

Vậy (12.18) có thể được viết lại dưới dạng mô hình trạng thái như sau:

$$\begin{cases}\dfrac{di_{Td}}{dt} = -\dfrac{1}{T_D}i_{Td} + \omega_s i_{Tq} + \dfrac{1}{L_D}\left(u_{Nd} - e_{Nvd}\right)\\ \dfrac{di_{Tq}}{dt} = -\omega_s i_{Td} - \dfrac{1}{T_D}i_{Tq} + \dfrac{1}{L_D}\left(u_{Nq} - e_{Nvq}\right)\end{cases} \tag{12.21}$$

Mô hình (12.21) được ta sử dụng để mô phỏng mạch điện phía lưới và cũng là xuất phát điểm để thiết kế khâu ĐC dòng phía lưới. Giả thiết rằng ta chỉ trực tiếp đo dòng $\mathbf{i}_N$ ở đầu ra của CL, vậy ta phải sử dụng phương trình (12.17) để tính dòng $\mathbf{i}_F$ để rồi sau đó nhờ phương trình thứ 2 của (12.18) tính giá trị thực dòng lưới (dòng biến thế) $\mathbf{i}_T$.

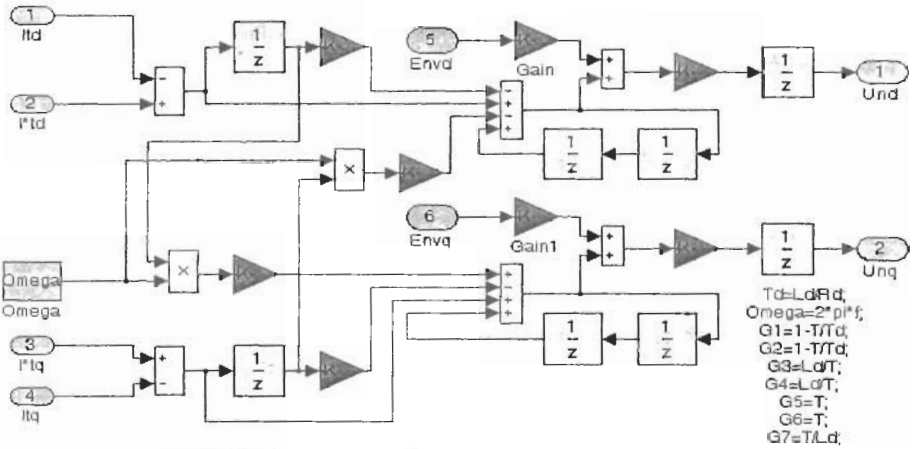
Mô hình SIMULINK của mạch điện phía lưới được giới thiệu ở hình 12.18.



Hình 12.18 Sơ đồ SIMULINK mô phỏng mạch điện phía lưới

Tương tự phía máy phát, khâu ĐC dòng trong 12.16 (khối *Net-Side Current Controller*) cũng là một khâu với đặc tính *Dead-Beat* có động học cao, thực hiện theo sơ đồ SIMULINK ở hình 12.19 với thuật toán ĐC như sau:

$$\begin{cases} u_{Nd}(k+1) = \dfrac{L_D}{T}\left[x_{Td}(k) - \left(1 - \dfrac{T}{T_D}\right)x_{Td}(k-1) - \omega_s T x_{Tq}(k-1) \right. \\ \qquad\qquad \left. + y_{Nd}(k-2)\right] \\ u_{Nq}(k+1) = \dfrac{L_D}{T}\left[x_{Tq}(k) + \omega_s T x_{Td}(k-1) - \left(1 - \dfrac{T}{T_D}\right)x_{Tq}(k-1) \right. \\ \qquad\qquad \left. + y_{Nq}(k-2)\right] \end{cases} \tag{12.22}$$

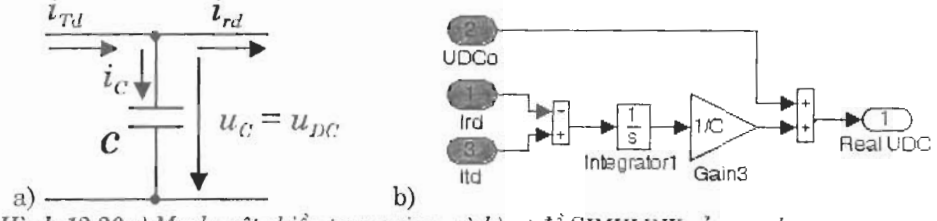


Hình 12.19 Sơ đồ SIMULINK của khâu ĐC dòng phía lưới

c) Khối "DC Link" với mô hình mạch điện một chiều trung gian

Trên cơ sở cấu trúc thiết bị ở hình 10.28 ta biết: Hai NL phía lưới và phía máy phát trao đổi điện năng với nhau qua mạch điện một chiều trung gian với điện áp U_{DC}. Trong sơ đồ SIMULINK ở hình 12.20, mạch được mô phỏng bởi khối *"DC Link"*.

Để đơn giản bớt việc mô hình hóa, ta tạm giả thiết: *Công suất vô công ở cả hai phía máy phát và lưới đã được ĐC ổn định*, và (tùy theo chế độ vận hành trên hay dưới đồng bộ) chỉ xảy ra trao đổi năng lượng do biến động công suất hữu công đưa từ *Turbine* gió lên lưới.



Hình 12.20 a) Mạch một chiều trung gian, và b) sơ đồ SIMULINK của mạch

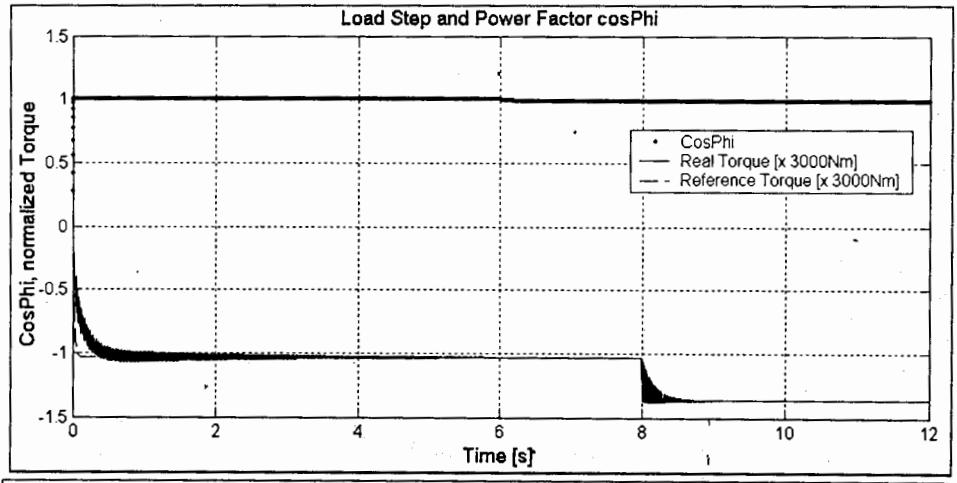
Đại lượng ĐK công suất hữu công phía lưới là i_{Td}, phía máy phát là i_{rq}, và dòng chảy qua tụ lọc C là i_C (hình 12.20a). Phương trình điện mô tả mạch là:

$$\left.\begin{aligned} i_C &= C\frac{du_{DC}}{dt} \\ i_C &= i_{Td} - i_{rq} \end{aligned}\right\} \Rightarrow u_{DC} = U_{DC0} + \frac{1}{C}\int \left(i_{Td} - i_{rq}\right) dt \qquad (12.23)$$

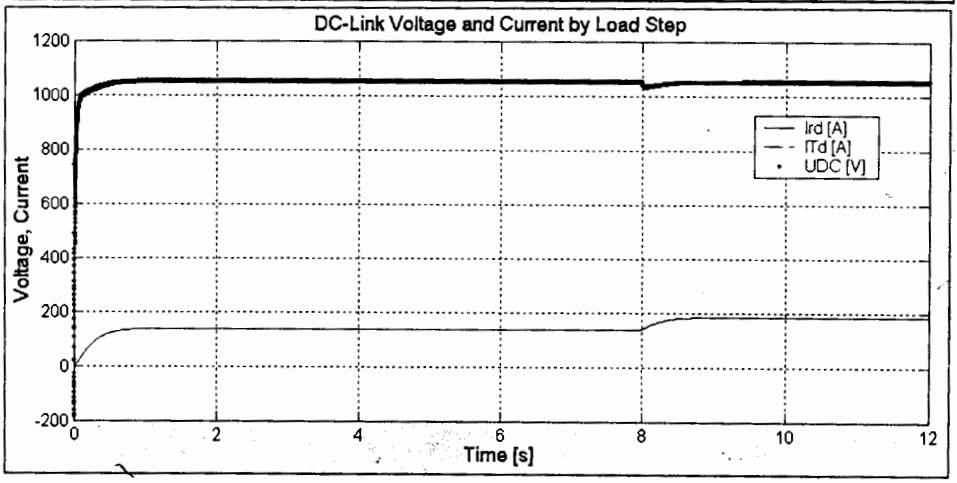
U_{DC0} Giá trị ban đầu của U_{DC}

Với (12.23) ta xây dựng mô hình SIMULINK của mạch một chiều trung gian như hình 12.20b. Hình 12.21 giới thiệu một số kết quả mô phỏng với hệ thống ta vừa xây dựng.

a)



b)



***Hình 12.21** a) Khả năng cách ly rất cao của hệ thống ĐK phía máy phát: Khi tăng công suấthữu công thêm 33%, cosφ vẫn giữ ổn định giá trị 1; b) Tại thời điểm tăng công suất, hệ thống ĐK phía lưới bảo đảm ổn định U_{DC} = 1050V*

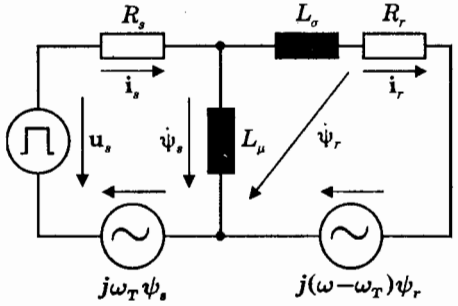
Hình 12.21a minh họa khả năng cách ly rất tốt công suất hữu công và vô công của hệ thống ĐK phía máy phát: Khi tăng công suất phát thêm ≈33% (m_G tăng từ 3000 lên 4000Nm), hệ số công suất vẫn giữ ổn định $\cos\varphi = 1$. Khi tăng như vậy, thông thường U_{DC} sẽ giảm do tác dụng của dòng i_{rd}, nhờ có hệ thống ĐK phía lưới đã kịp thời tăng i_{Td} bù lại nên đã giữ ổn định được U_{DC} = 1050V (hình 12.21b).

Ví dụ mô phỏng trên là một hệ thống vô cùng phức hợp và ta có thể thông qua mô phỏng khảo sát diễn biến của rất nhiều đại lượng khi xảy ra quá trình quá độ: Quá trình hòa đồng bộ, quá trình đóng ngắt thêm phụ tải hay biến động của nguồn năng lượng gió ở đầu vào của Turbine vv... Vì dung lượng sách bị hạn chế nên người viết không đưa thêm các kết quả mô phỏng khác nữa.

12.2.3 Mô phỏng hệ truyền động dị bộ theo phương pháp "Direct Torque Control" sử dụng Logic mờ[1]

"Direct Torque Control[2]*"* là phương pháp khá phổ biến được sử dụng để ĐK các máy điện XCBP công suất lớn, có tiền thân là phương pháp *"Direct Self-Control*[3]*"*. Bản chất của phương pháp là: Dựa trên ảnh hưởng trực tiếp của điện áp đầu vào để ĐK vector từ thông Stator ψ_s, qua đó trực tiếp ĐK mômen quay m_M mà không cần tới vòng ĐC dòng trong cùng như các phương pháp khác.

Phương pháp DSC kinh điển là một phương pháp dùng kỹ thuật analog với các khâu ĐC kiểu 2 hoặc 3 điểm, có nhiệm vụ giữ ổn định biên độ của từ thông stator và áp đặt nhanh mômen quay trong điều kiện: van bán dẫn là GTO (*Gate-turn-off-Thyristor*) có tốc độ đóng ngắt chậm. Do bản chất vật lý của phương pháp trong ví dụ này là DSC, khâu *Fuzzy* chủ yếu tham gia vào phần quyết định chế độ đóng ngắt van, ta còn có thể gọi khâu ĐC mờ trong trường hợp này là *khâu điều chế phi tuyến bề rộng xung*[4]. Để hiểu được trình tự các bước xây dựng mô hình SIMULINK của hệ thống ta sẽ phải điểm qua vài nét của kiến thức cơ sở.



Hình 12.22 Sơ đồ thay thế chữ Γ của MĐDB (chỉ số "T" viết phía dưới bên phải ω : hệ tọa độ được chọn để tựa hướng)

[1] Fuzzy Logic

[2] DTC: Điều khiển mômen trực tiếp

[3] DSC: Tự chỉnh trực tiếp

[4] Nonlinear Puls Width Modulator: NPWM

a) Mô hình của MĐDB sử dụng trong phương pháp DTC

Xuất phát điểm để thiết kế các thuật toán ĐK là sơ đồ thay thế (SĐTT) dạng chữ Γ ở hình 12.22. Từ sơ đồ ở hình 12.22 có thể viết ngay được hệ phương trình điện áp mô tả MĐDB trên một hệ tọa độ tựa quay với vận tốc góc bất kỳ ω_T.

$$\begin{aligned}\frac{d\psi_s}{dt} &= \mathbf{u}_s - R_s\mathbf{i}_s - j\omega_T\psi_s \\ \frac{d\psi_r}{dt} &= R_r\mathbf{i}_r + j(\omega-\omega_T)\psi_r\end{aligned} \tag{12.24}$$

Khi chọn hệ tọa độ Stator $\alpha\beta$ cố định làm hệ tọa độ tựa hướng ta sẽ có $\omega_T = 0$. Quan hệ giữa từ tản ψ_σ, dòng rotor $\mathbf{i}_r$, dòng stator $\mathbf{i}_s$ như sau:

$$\psi_\sigma = \psi_s - \psi_r \tag{12.25}$$

$$\mathbf{i}_r = \frac{1}{L_\sigma}(\psi_s - \psi_r) = \frac{\psi_\sigma}{L_\sigma} \tag{12.26}$$

$$\mathbf{i}_s = \frac{L_\mu + L_\sigma}{L_\mu L_\sigma}\psi_s - \frac{1}{L_\sigma}\psi_r \tag{12.27}$$

Mômen nội tại của MĐDB chịu ảnh hưởng của giá trị module của hai từ thông ψ_s, ψ_r và góc δ xen giữa chúng.

$$m = \frac{3}{2}\frac{1}{L_\sigma}|\psi_s||\psi_r|\sin\delta \tag{12.28}$$

với:

$$\text{tg}\,\delta = |\psi_\sigma|/|\psi_r| \tag{12.29}$$

Mômen đưa tới trục cơ của MĐDB sẽ là:

$$m_M = z_p m \tag{12.30}$$

Tốc độ góc ω_r (tần số trượt của mạch rotor) có thể tính được nếu ta xét phương trình thứ hai của hệ (12.24) khi chọn $\omega_T = \omega_s$ (hệ tọa độ tựa quay với vận tốc góc Stator), chỉ nhằm mục đích tính module của từ thông.

$$|\psi_r| = \frac{R_r|\mathbf{i}_r|}{\omega_r} \tag{12.31}$$

Kết hợp (12.26), (12.29) và (12.31) ta tính được ω_r cho chế độ xác lập.

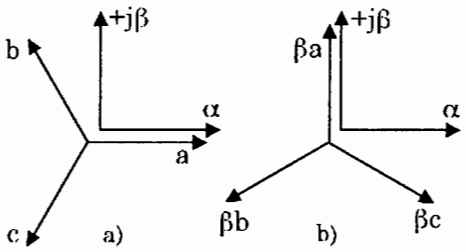
$$\omega_r = \frac{R_r\,\text{tg}\delta}{L_\sigma} = \frac{\text{tg}\delta}{T_\sigma} \tag{12.32}$$

Module của từ thông stator ψ_s được tính từ hai thành phần.

$$|\psi_s| = \sqrt{\psi_{s\alpha}^2 + \psi_{s\beta}^2} \tag{12.33}$$

Giá trị danh định (gọi là *giá trị chuẩn*) tính nhờ (12.33) là cơ sở để ta *dẫn dắt* ψ_s *đi theo một quỹ đạo tròn*, môdul của ψ_s sẽ là hằng và do các thành phần từ thông ba pha có dạng sin tạo nên.

Trong các phần sau ta sẽ thấy: *ở dải tốc độ quay lớn hơn cỡ 25% danh định*, để tận dụng điện áp ψ_s cần được dẫn dắt đi theo quỹ đạo hình lục giác và vì vậy module tính theo giá trị chuẩn (12.33) sẽ biến động liên tục, bất lợi cho mômen quay. Ta có thể chọn một hệ tọa độ Stator (hình 12.23b) khác với hệ tọa độ Stator quen biết (hình 12.23a) làm cơ sở tính giá trị chuẩn: cho phép ta *dẫn dắt* ψ_s *theo quỹ đạo lục giác nhưng có giá trị chuẩn là hằng*.



Hình 12.23 *a) Hệ tọa độ Stator quen biết; b) Hệ tọa độ Stator mới*

Khác với thông lệ, trên hệ tọa độ Stator mới: Trục trùng với trục của pha "*a*" là trục "*β*" chứ không phải trục "*α*" (hình 12.23b).

$$\begin{bmatrix}\psi_a\\ \psi_b\\ \psi_c\end{bmatrix}=\begin{bmatrix}1 & 0\\ -\frac{1}{2} & \frac{\sqrt{3}}{2}\\ -\frac{1}{2} & -\frac{\sqrt{3}}{2}\end{bmatrix}\begin{bmatrix}\psi_\alpha\\ \psi_\beta\end{bmatrix};\quad\begin{bmatrix}\psi_{\beta a}\\ \psi_{\beta b}\\ \psi_{\beta c}\end{bmatrix}=\begin{bmatrix}0 & 1\\ -\frac{\sqrt{3}}{2} & -\frac{1}{2}\\ \frac{\sqrt{3}}{2} & -\frac{1}{2}\end{bmatrix}\begin{bmatrix}\psi_\alpha\\ \psi_\beta\end{bmatrix} \qquad (12.34)$$

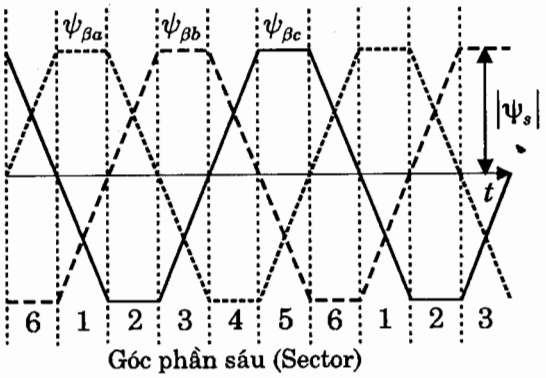
Công thức chuyển đổi của hệ cũ là (12.34) bên trái, của hệ mới là (12.34) bên phải. Để phân biệt với hệ cũ, ta *gọi các thành phần từ thông ba pha mới là từ thông β*. Khi chọn quỹ đạo lục giác, *các thành phần từ thông β đều có dạng hình thang* (hình 12.24) với tổng giá trị tức thời

$$|\psi_s|=\frac{1}{2}\left(|\psi_{\beta a}|+|\psi_{\beta b}|+|\psi_{\beta c}|\right) \qquad (12.35)$$

luôn lớn gấp đôi giá trị lớn nhất trong ba thành phần.

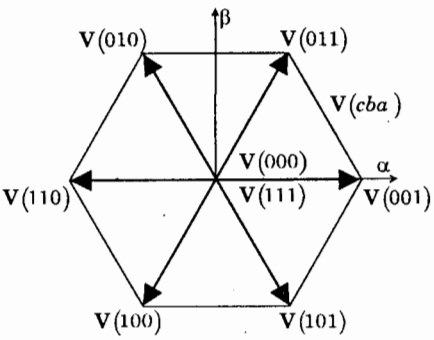
b) Nghịch lưu trong phương pháp DTC

Nghịch lưu sử dụng trong DTC có sơ đồ mạch như ở hình 11.6a. Ba nhánh van của nghịch lưu cho phép nối ba cực *a*, *b* và *c* của động cơ với thế năng "+" (lôgich 1) hoặc thế năng "-" (lôgich 0) của mạch một chiều trung gian. Chúng tạo thành 8 trạng thái lôgich ứng với 8 vector điện áp chuẩn minh họa ở hình 12.25.



Hình 12.24 Các thành phần từ thông β khi ψ_s được dẫn dắt theo quỹ đạo lục giác

Hai vector V(000) và V(111) là hai vector có module bằng 0. Nếu van bán dẫn là loại có tốc độ đóng ngắt cao (ví dụ loại IGBT: *Insulated Gate Bipolartransistor*), ta có thể sử dụng 6 vector có module khác 0, kết hợp với 2 vector module 0 để tạo điện áp Stator với vị trí bất kỳ. Trong trường hợp công suất lớn, van sử dụng thường là GTO (*Gate-turn-off-Thyristor*) là loại tốc độ chậm nên chỉ sử dụng 6 vector có môdul khác 0. Trường hợp này là đối tượng chính của phương pháp DTC.



Hình 12.25 Các vector điện áp Stator chuẩn

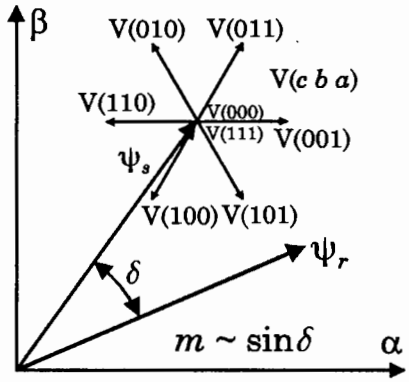
c) Phương pháp DTC

Khác với *phương pháp tựa theo từ thông rotor*[1] hoạt động dựa trên vòng ĐC dòng Stator (mục 12.2.1), áp đặt 2 thành phần dòng tạo từ thông i_{sd} và dòng tạo mômen quay i_{sq}. Xung kích thích van được tạo từ các giá trị điện áp do khâu ĐC dòng đưa tới *khâu điều chế vector*. Tư tưởng chính của phương pháp DTC là: *Tạo xung kích thích van trực tiếp trên cơ sở sai lệch từ thông stator và sai lệch mômen quay* (hình 12.26). Vì vậy, phương pháp DTC không cần khâu chuyển hệ tọa độ $dq \rightarrow \alpha\beta$, không phải tính góc chuyển hệ là khâu gây không ít khó khăn khi thực hiện.

Cơ sở vật lý của phương pháp DTC được mô tả bởi hai phương trình (12.24), (12.28). Khi bỏ qua điện áp sụt trên điện trở R_s trong (12.24), ta thu được quan hệ tích phân giữa điện áp Stator $\mathbf{u}_s$ và từ thông Stator $\boldsymbol{\psi}_s$. Điều ấy có nghĩa là: *Vector $\boldsymbol{\psi}_s$ sẽ chuyển động theo hướng của vector điện áp được chọn*. Với 7 vector điện áp chuẩn (hình 12.25) ta sẽ có 7 khả năng tác động tới $\boldsymbol{\psi}_s$ (hình 12.26).

Trong phương pháp DTC (bản chất là SFO[2]) vector $\boldsymbol{\psi}_s$ đáp lại trực tiếp mỗi thay đổi điện áp $\mathbf{u}_s$. Trong phương pháp RFO, điện áp $\mathbf{u}_s$ chỉ tác động trực tiếp tới dòng $\mathbf{i}_s$, để rồi dòng tác động tới từ thông rotor $\boldsymbol{\psi}_r$ với quán tính thể hiện bởi hằng số thời gian rotor T_r. Có thể nói: Giữa $\mathbf{u}_s$ và $\boldsymbol{\psi}_r$ có quan hệ với quán tính T_r.

[1] Rotor Flux Orientation: RFO
[2] Stator Flux Orientation: SFO

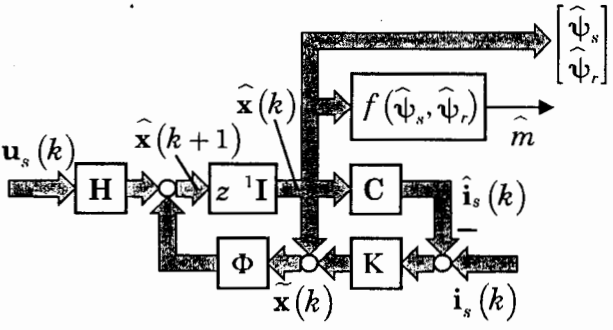


Hình 12.26 Cơ sở vật lý của phương pháp DTC

Vậy là: *bằng cách chọn vector điện áp thích hợp ta có thể đồng thời tác động tới module của* ψ_s *và góc* δ *xen giữa* ψ_s, ψ_r (hình 12.26). Theo (12.28), mômen quay m tỷ lệ với tích của $|\psi_s|$ và $\sin\delta$, nhờ đó ta có thể điều khiển được mômen quay với động học cao nhất.

Khi ta đặt lên động cơ một vector điện áp chuẩn phù hợp, vector ψ_s có thể chuyển động trên quỹ đạo với gia tốc tối đa. Nếu dấu của gia tốc trùng với dấu của chiều chuyển động, góc δ sẽ tăng lên, bởi vì ψ_r chỉ có thể bám theo với quán T_r. Nếu vector điện áp được chọn có module 0, gia tốc của ψ_s sẽ gần như bằng 0, và nhờ đó góc δ bé đi do ψ_r vẫn tiếp tục quay theo hướng cũ.

Các giá trị thực của từ thông ψ_s và mômen m sẽ được tính nhờ một *khâu quan sát từ thông* (hình 12.27). Sau khi so sánh với các giá trị chủ đạo, trên cơ sở sai lệch ta có thể quyết định được chùm xung kích thích van phù hợp. Trong giải pháp DTC kinh điển, các quyết định được đưa ra bởi các khâu ĐC kiểu 2 hoặc 3 điểm.



Hình 12.27 Khâu quan sát tức thời (khâu Luenberger)

Trong khâu quan sát Luenberger thông thường, các sai lệch đầu ra ở thời điểm thứ (k) chỉ được sử dụng để hiệu chỉnh vector trạng thái ở thời điểm thứ (k+1). Có nghĩa là: chỉ sau khi 1 chu kỳ trích mẫu đã trôi qua lãng phí. Thực tế, ta có thể hiệu chỉnh ngay ở thời điểm thứ (k), và một khâu quan sát như vậy còn có tên là *khâu quan sát tức thời* (*Current Estimator*, hình 12.27) với phương trình tính toán sau đây:

$$\psi_s^N(k+1) = \left(1-\rho\frac{T}{T_\sigma}\right)\psi_s^N(k) + \rho(1-\sigma)\frac{T}{T_\sigma}\psi_r^N(k) + \mathbf{u}_s^N(k)n_0 \qquad (12.36)$$

$$\psi_r^N(k+1) = e^{jT\omega(k)}\left[\frac{T}{T_\sigma}\psi_s^N(k) + \left(1-\frac{T}{T_\sigma}\right)\psi_r^N(k)\right]$$

Trong công thức (12.36), hai đại lượng từ thông và điện áp đã được chuẩn hóa (nhận biết nhờ chỉ số "*N*" viết bên phải, trên cao) bởi các giá trị:

$$\Psi_N = \sqrt{2}\frac{\mathbf{U}_{\text{Rated}}}{\omega_0};\ \mathbf{U}_N = \omega_0 \Psi_N \tag{12.37}$$

và do đó không có thứ nguyên, thuận lợi cho việc lập trình vi xử lý. Công thức (12.36) còn chứa lượng biến thiên góc của Rotor trong phạm vi 1 chu kỳ trích mẫu T:

$$T\omega(k) = \Delta\vartheta = \vartheta(k+1) - \vartheta(k) \tag{12.38}$$

Từ đó ta có thể tìm được các công thức tính chi tiết của khâu QS.

$$\begin{bmatrix} \widehat{\psi}^N_{s\alpha}(k+1) \\ \widehat{\psi}^N_{s\beta}(k+1) \\ \widehat{\psi}^N_{r\alpha}(k+1) \\ \widehat{\psi}^N_{r\beta}(k+1) \end{bmatrix} = \begin{bmatrix} a_1 & 0 & a_2 & 0 \\ 0 & a_1 & 0 & a_2 \\ a_3\cos\Delta\vartheta & -a_3\sin\Delta\vartheta & a_4\cos\Delta\vartheta & -a_4\sin\Delta\vartheta \\ a_3\sin\Delta\vartheta & a_3\cos\Delta\vartheta & a_4\sin\Delta\vartheta & a_4\cos\Delta\vartheta \end{bmatrix} \begin{bmatrix} \widetilde{\psi}^N_{s\alpha}(k) \\ \widetilde{\psi}^N_{s\beta}(k) \\ \widetilde{\psi}^N_{r\alpha}(k) \\ \widetilde{\psi}^N_{r\beta}(k) \end{bmatrix} + \begin{bmatrix} a_5 & 0 \\ 0 & a_5 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u^N_{s\alpha}(k) \\ u^N_{s\beta}(k) \end{bmatrix} \tag{12.39}$$

$$\begin{bmatrix} \widehat{i}^N_{s\alpha}(k) \\ \widehat{i}^N_{s\beta}(k) \end{bmatrix} = \begin{bmatrix} a_6 & 0 & -1 & 0 \\ 0 & a_6 & 0 & -1 \end{bmatrix} \begin{bmatrix} \widehat{\psi}^N_{s\alpha}(k) \\ \widehat{\psi}^N_{s\beta}(k) \\ \widehat{\psi}^N_{r\alpha}(k) \\ \widehat{\psi}^N_{r\beta}(k) \end{bmatrix} \tag{12.40}$$

$$\begin{bmatrix} \widetilde{\psi}^N_{s\alpha}(k) \\ \widetilde{\psi}^N_{s\beta}(k) \\ \widetilde{\psi}^N_{r\alpha}(k) \\ \widetilde{\psi}^N_{r\beta}(k) \end{bmatrix} = \begin{bmatrix} \widehat{\psi}^N_{s\alpha}(k) \\ \widehat{\psi}^N_{s\beta}(k) \\ \widehat{\psi}^N_{r\alpha}(k) \\ \widehat{\psi}^N_{r\beta}(k) \end{bmatrix} + \begin{bmatrix} k_1 & -k_2 \\ k_2 & k_1 \\ k_3 & -k_4 \\ k_4 & k_3 \end{bmatrix} \begin{bmatrix} i^N_{s\alpha}(k) - \widehat{i}^N_{s\alpha}(k) \\ i^N_{s\beta}(k) - \widehat{i}^N_{s\beta}(k) \end{bmatrix} \tag{12.41}$$

$$\widehat{m}^N(k) = 2\left[\widehat{\psi}^N_{s\beta}(k)\widehat{\psi}^N_{r\alpha}(k) - \widehat{\psi}^N_{s\alpha}(k)\widehat{\psi}^N_{r\beta}(k)\right] \tag{12.42}$$

Đại lượng chuẩn hóa của dòng và mômen trong 4 công thức trên là:

$$\mathbf{I}_N = \frac{\boldsymbol{\Psi}_N}{L_\sigma};\ m_N = \frac{3}{4}\boldsymbol{\Psi}_N \mathbf{I}_N = \text{Mômen lật} \tag{12.43}$$

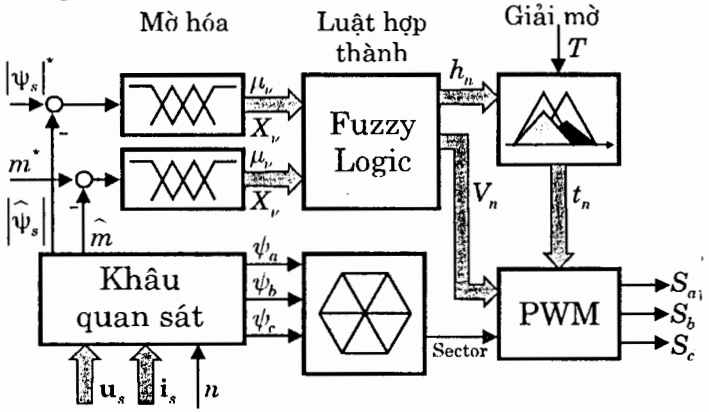
Các tham số của khâu quan sát có công thức:

$$\begin{aligned} a_1 &= 1-\frac{T}{T_\sigma}\rho;\ a_2 = \frac{T}{T_\sigma}\rho(1-\sigma);\ a_3 = \frac{T}{T_\sigma} \\ a_4 &= 1-\frac{T}{T_\sigma};\ a_5 = \frac{T}{T_\sigma}n_0;\ a_6 = \frac{1}{1-\sigma} \end{aligned} \tag{12.44}$$

Cho đến đây, tất cả các tham số của khâu quan sát đều đã biết trước, trừ các tham số $k_1 \ldots k_4$ của ma trận trọng lượng **K**.

d) Phương pháp DTC sử dụng Fuzzy Logic

Sau khi đã có mô hình của MĐDB, của khâu QS, ta có thể bắt đầu xây dựng hệ thống DTC sử dụng logic mờ với sơ đồ nguyên lý như hình 12.28.



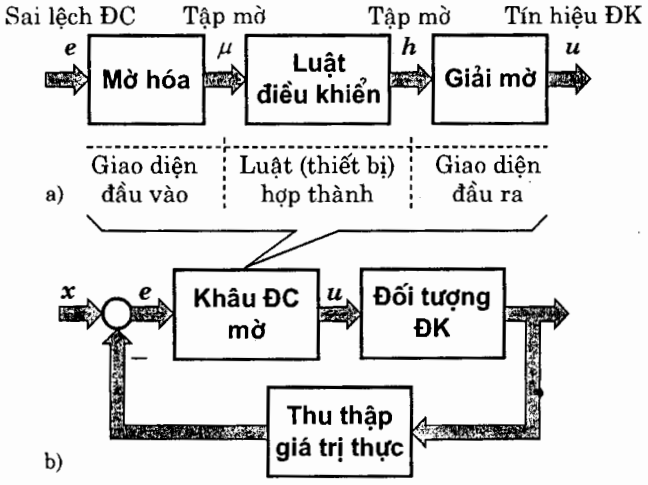
Hình 12.28 *Cấu trúc của khâu ĐC mờ cho MĐDB trên cơ sở phương pháp DTC*

Ba thành phần cơ bản (hình 12.29a) của một khâu ĐC chỉnh mờ là: *khâu mờ hóa* (*Fuzzyfication*), *khâu hợp thành* (*Fuzzy Rule Inference*) và *khâu giải mờ* (*Defuzzyfication*).

Các biến vào của khâu ĐC mờ là sai lệch ĐC của từ thông Stator $\boldsymbol{\psi}_s$ và mômen quay m với giá trị thực do khâu quan sát cung cấp (tính từ $\mathbf{u}_s$, $\mathbf{i}_s$ và n). Khâu mờ hóa biến các đại lượng rõ đó thành các giá trị ngôn ngữ, xác định bởi tập mờ X_ν và mức thuộc μ_ν. Sau đó chúng được khâu hợp thành xử lý, đưa ra các kết luận dưới dạng biến ngôn ngữ (vector điện áp V_n với mức thỏa mãn h_n). Trên cơ sở đó, khâu giải mờ tính các thời gian đóng ngắt t_n của ba nhánh van nghịch lưu cho từng chu kỳ xung.

Toàn bộ mặt phẳng vector thuộc hệ tọa độ $\alpha\beta$ được chia thành 6 Sector (mỗi Sector 60^0). Khâu *nhận dạng Sector* có nhiệm vụ xác định Sector hiện tại, phục vụ tính trạng thái đóng ngắt thực sự S_a, S_b và S_c từ biến ngôn ngữ V_n.

Khâu hợp thành được thiết kế sao cho vector từ thông Stator được dẫn dắt hoặc theo quỹ đạo hình tròn (phía thấp của dải tốc độ danh định) hoặc theo quỹ đạo hình lục giác (phía cao của dải tốc độ danh định).



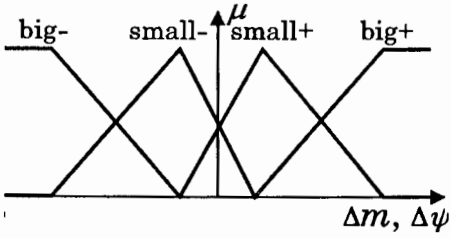
Hình 12.29 Cấu trúc của khâu ĐC mờ

Mờ hóa module từ thông và mômen quay

Sai lệch ĐC của module từ thông và mômen quay được mô tả bởi các biến ngôn ngữ như hình 12.30 với kích cỡ đặc trưng bởi 4 tập mờ "*big*" và "*smal*", có kèm thêm dấu "+" hoặc "-". Để cấu trúc của khâu ĐC không trở nên quá phức tạp và khối lượng tính toán không quá lớn, ta giới hạn ở số lượng 4 tập cho mỗi biến và cấu thành hàm thuộc μ bởi hàm tuyến tích theo công thức.

$$\mu_n(x) = \left[1 - a\left|x - \Delta x_n\right|\right] \tag{12.45}$$

a	Độ dốc	x	Biến vào
Δx	Sai lệch ĐC của biến vào		
n	Số thứ tự của tập mờ		



***Hình 12.30** Các biến ngôn ngữ đầu vào*

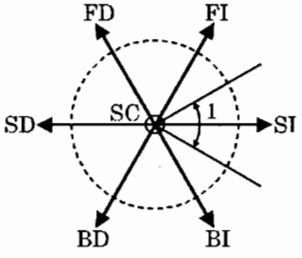
Hình 12.30 đã chỉ rõ: tại mỗi thời điểm chỉ có 2 tập mờ ở trạng thái tích cực, số lượng luật điều khiển cần xử lý ít, nhờ đó khối lượng tính toán sẽ thấp ở mức chấp nhận được.

Luật điều khiển

Luật ĐK mô tả phản ứng cần có của vector từ thông Stator ψ_s ứng với các cặp giá trị cụ thể của hai biến vào. Ví dụ:

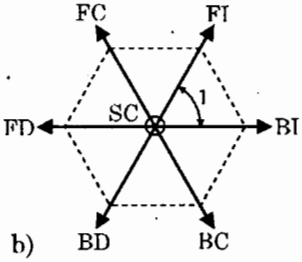
IF (Δm là "big+") *and* ($\Delta\psi$ là "smal-") **THEN** (module từ thông được tăng theo chiều tiến)

Trong ví dụ trên, "*chiều tiến*" được hiểu là chiều quay hiện tại của vectơ từ thông. Tất cả các luật ĐK có kết luận giống nhau sẽ được kết hợp với nhau nhờ toán tử MAX. Các kết luận đều luôn dựa trên 1 trong 7 vector điện áp chuẩn của nghịch lưu (hình 12.26) với ảnh hưởng của nó trực tiếp tới vector từ thông trong phạm vi của Sector đang xét. Kết luận minh họa tác dụng cần có của việc lựa chọn vector điện áp đặt lên MĐDB, ví dụ như "*tăng môdul từ thông theo chiều tiến (flux forwards increase)*". Giả sử, hiện tại vector ψ_s đang nằm trong Sector 1, kết luận ngôn ngữ về ảnh hưởng của các vector điện áp tới ψ_s được minh họa trong hình 12.31.



a)

FI		increase
	Flux forwards	
FD		decrease
SI		increase
SD	*Flux standing still*	decrease
SC		constant
BI		increase
	Flux backwards	
BD		decrease



b)

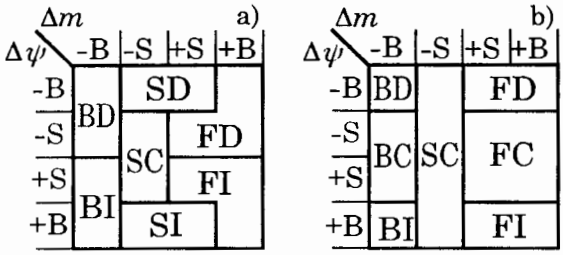
FI		increase
FC	*Flux forwards*	constant
FD		decrease
SC	*Flux standing still*	constant
BI		increase
BC	*Flux backwards*	constant
BD		decrease

***Hình 12.31** Ảnh hưởng của các vector điện áp tới ψ_s khi ψ_s ở Sector 1 cho trường hợp: **a)** quỹ đạo tròn, **b)** quỹ đạo lục giác*

Dễ dàng thấy rằng: do toàn bộ mặt phẳng vector được chia thành 6 Sector 60^0, kết luận về tác dụng của từng vector điện áp chuẩn sẽ chỉ được thay đổi tương ứng với vị trí Sector của từ thông. Điều này đã làm nổi bật ý nghĩa của khâu *nhận dạng Sector* đã nhắc đến ở trên.

Nhằm mục đích khử sai lệch ĐC của từ thông và mômen (ứng với dạng quỹ đạo đã chọn) một cách nhanh chóng, ta có thể xây dựng luật ĐK, tập hợp dưới dạng ma trận ở hình 12.32. Khi áp dụng các kết luận ở hình 12.31 và ma trận (luật) ĐK ở hình 12.32 cần ghi nhớ:

- Khi chuyển trạng thái đóng ngắt phải chọn trình tự có lợi cho việc giảm thiểu tổn hao đóng ngắt van.
- Quỹ đạo tròn chỉ thực hiện được nếu kết hợp 2 vector biên và 1 vector không. Ngược lại, để thực hiện quỹ đạo lục giác ta chỉ cần 1 vector biên và 1 vector không.



B: big, S: small

Hình 12.32 *Luật điều khiển của: **a)** quỹ đạo tròn và **b)** quỹ đạo lục giác*

Giải mờ và tính thời gian đóng ngắt

Có hai phương pháp thực hiện giải mờ và tính thời gian đóng ngắt van bán dẫn của NL như sau:

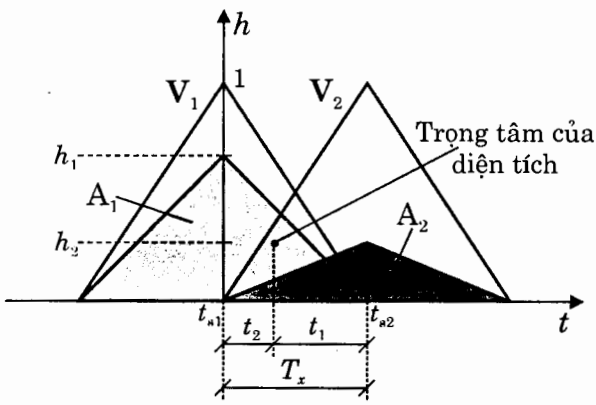
Phương pháp điều chế với 2 vector

Đơn giản hơn cả là: trong 1 chu kỳ xung chỉ sử dụng 2 vector có mức thỏa mãn cao nhất (ví dụ: vector $\mathbf{V}_1$, $\mathbf{V}_2$ như hình 12.33). Sau khi nhân mức thỏa mãn h với tập mờ của từng vector, ta đi tìm hoành độ trọng tâm của phần diện tích mới xuất hiện (hình 12.33).

Vị trí của hoành độ trọng tâm chính là điểm chia chu kỳ xung T thành hai khoảng thời gian thực hiện hai vector điện áp được tính như sau:

$$t_1 = \frac{\int f(t)t\,dt}{\int f(t)dt} \approx \frac{A_1 t_{s1} + A_2 t_{s2}}{A_1 + A_2} = T\frac{h_1}{h_1 + h_2};\ t_2 = T - t_1 \qquad (12.46)$$

Cơ sở của công thức tính gần đúng trọng tâm (12.46) là việc sử dụng các tập mờ có dạng tam giác cân với hệ tọa độ lựa chọn như hình 12.33.



Hình 12.33 *Giải mờ theo phương pháp 2 vector*

Phương pháp điều chế với 3 vector

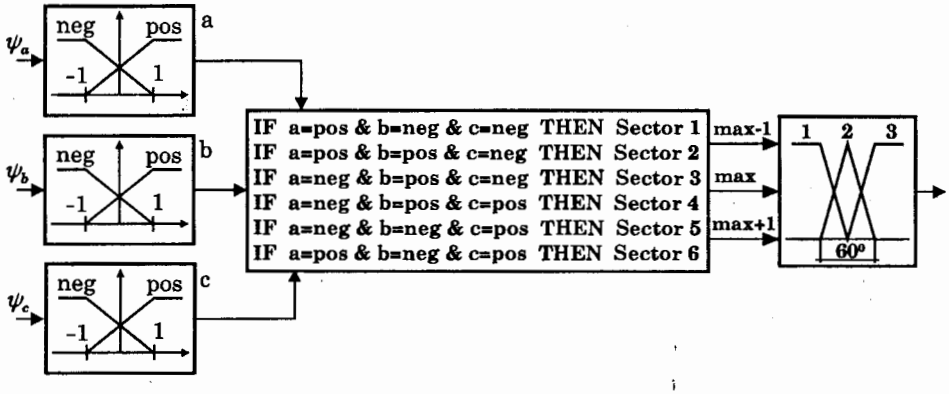
Việc điều chế được mở rộng với sự tham gia của 3 vector điện áp chuẩn. Tương tự với công thức (12.46), ta sử dụng dưới đây để tính thời gian đóng ngắt t_n cho 3 vector có mức thỏa mãn cao nhất.

$$t_n = T\frac{h_n}{h_1 + h_2 + h_3} \qquad n = 1,2,3 \tag{12.47}$$

Tại đây cần lưu ý bạn đọc về hai phương pháp giải mờ trên:

- Phương pháp 3 vector rất thích hợp với quỹ đạo tròn, vì quỹ đạo được duy trì (ở trạng thái tĩnh) chủ yếu bởi hai vector "*forwards increase*", "*forwards decrease*" và một vector không "*standing still constant*".
- Phương pháp 2 vector có lợi thế hơn đối với quỹ đạo lục giác. Vì ở chế độ tĩnh, vector từ thông chỉ được dẫn dắt bởi một vector biên "*forwards constant*" và một vector không "*standing still constant*".

e) Nhận dạng mờ góc phần sáu (Sector)



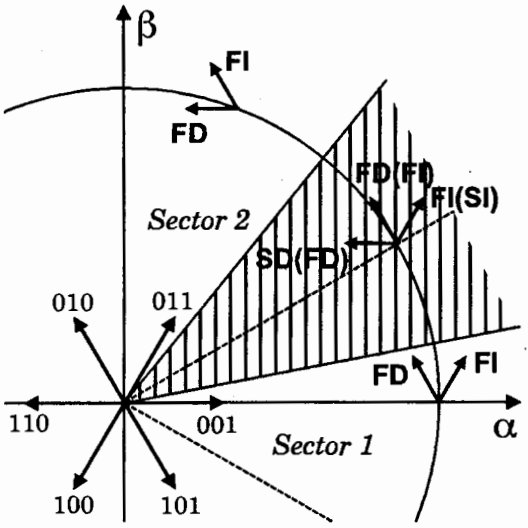
***Hình 12.34** Nhận dạng Sector theo phương thức mờ*

Để có thể áp dụng chính xác luật ĐK mô tả ở hình 12.32, ta phải biết vị trí hiện tại của ψ_s. Hình 12.31 đã minh họa cho ví dụ Sector 1. Khi vector ψ_s chuyển Sector, nghĩa là, khi vị trí của nó đang nằm ở biên giới giữa hai Sector, khi đó ảnh hưởng của các vector điện áp chuẩn tới ψ_s cũng thay đổi. Để loại trừ các nhược điểm của cách nhận dạng kinh điển, ta sử dụng phương pháp nhận dạng mờ góc phần sáu mô tả ở hình 12.34.

Ở đầu vào, ba thành phần pha của ψ_s được chuyển thành các biến ngôn ngữ a, b và c. Khâu hợp thành sẽ đưa ra kết luận về mức thuộc của ψ_s vào các Sector. Theo luật hợp thành ở hình 12.34, *Sector hiện tại* có mức thuộc là (max), *Sector vừa qua* là (max-1) và *Sector sắp tới* là (max+1). Từ đó ta thu được một biến ngôn ngữ mới, mô tả vị trí của ψ_s trong phạm vi góc ta đang quan tâm. Để hiểu rõ hơn tác dụng của phương pháp, ta hãy xem xét kỹ từng dạng quỹ đạo.

Quỹ đạo tròn

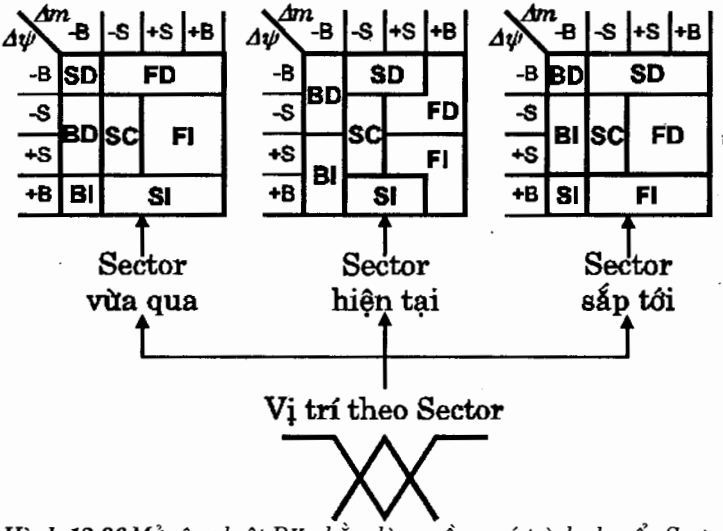
Các ảnh hưởng của điện áp (tại biên giữa hai Sector) theo hướng tiếp tuyến và hướng ly tâm được minh họa ở hình 12.35.



Hình 12.35 Quá trình chuyển đổi Sector trên quỹ đạo tròn

Bộ luật ĐK đã được xây dựng ở hình 12.31, 12.32 chỉ được áp dụng cho vị trí chính giữa Sector. Vì vậy, đối với miền biên giới (ngoài vị trí điểm giữa) ta phải biến đổi luật ĐK, sao cho quá trình chuyển Sector trở nên "*mềm*" hơn.

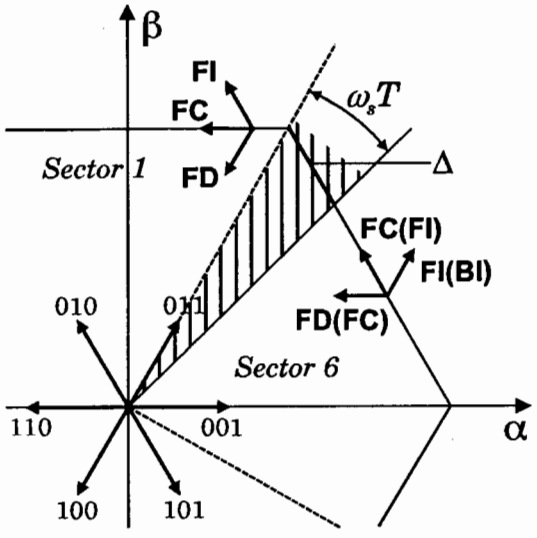
Vậy là, bên cạnh hai biến ngôn ngữ dành cho sai lệch ĐC của mômen quay m và module từ thông Stator ψ_s, hệ thống ĐC mờ có thêm một biến ngôn ngữ thứ ba "*Vị trí theo Sector*" (là biến ra của khâu nhận dạng ở hình 12.34) mô tả vị trí của ψ_s. Trên cơ sở biến thứ ba ta chọn luật ĐK đã được biến đổi phù hợp (hình 12.36), riêng khâu hợp thành và giải mờ vẫn được giữ nguyên như ban đầu.



Hình 12.36 Mở rộng luật ĐK nhằm làm mềm quá trình chuyển Sector trên quỹ đạo tròn

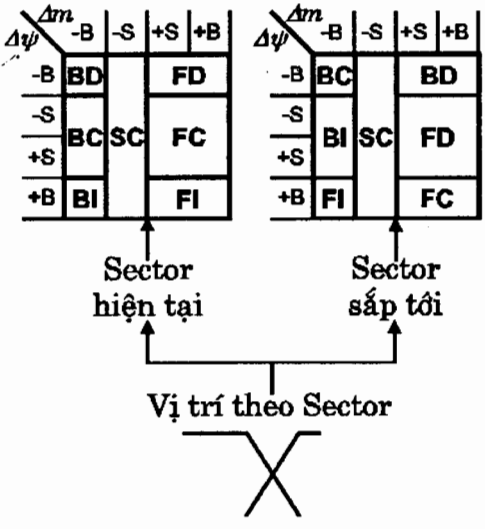
Quỹ đạo lục giác

Đối với dạng quỹ đạo lục giác, biên giới giữa các Sector đồng thời là vị trí của các vector điện áp chuẩn, vì vậy các nhận xét của quỹ đạo tròn ở trên không còn ý nghĩa. Tuy vậy, ta có thể *sử dụng nhận dạng Sector mờ để bù lại thời gian trễ khi chuyển Sector.*



Hình 12.37 Quá trình chuyển đổi Sector trên quỹ đạo lục giác

Nếu như ở giải pháp tương tự (Analog), thời điểm chuyển được ghi nhận một cách tự nhiên liên tục. Thì ngược lại, ở giải pháp số (Digital), thời điểm chuyển Sector hoàn toàn có thể xảy ra trong phạm vi chu kỳ trích mẫu, và không trùng với thời điểm trích mẫu là thời điểm khởi tính thuật toán nhận dạng Sector (ví dụ: nhờ dấu của từ thông pha). Việc chậm trễ đó có thể gây nên các quá trình cân bằng dòng/áp bất lợi. Có thể sử dụng *giải pháp quá độ Sector mờ* để khắc phục hiện tượng cân bằng này, trong đó *miền biên giới được coi là hàm của tần số Stator* (hình 12.37).



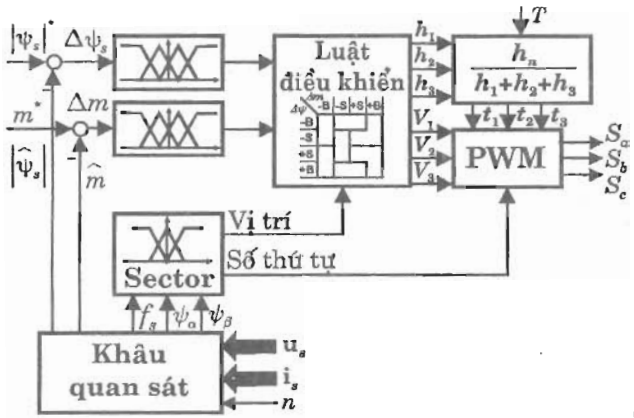
Hình 12.38 Mở rộng luật ĐK nhằm bù thời gian trễ khi nhận biết quá trình chuyển Sector trên quỹ đạo lục giác

Phần diện tích đánh dấu miền biên giới chính là miền có cạnh là đoạn quỹ đạo Δ mà vector ψ_s sẽ phải vượt qua trong phạm vi một chu kỳ trích mẫu. Sử dụng công thức (12.32) ta tính được tần số mạch Rotor và từ đó tần số Stator. Với tần số Stator, có thể tính trước xem thời điểm chuyển Sector sẽ xảy ra lúc nào trong phạm vi chu kỳ trích mẫu.

Đối với mục tiêu đặt ra, dễ dàng thấy rằng biến ngôn ngữ "*Vị trí theo Sector*" chỉ cần hai mức thuộc cho *Sector hiện tại* và *Sector sắp tới*, để từ đó biến đổi luật ĐK (hình 12.38), sao cho quá trình chuyển sang cạnh kế tiếp của lục giác đủ chính xác và mềm mại.

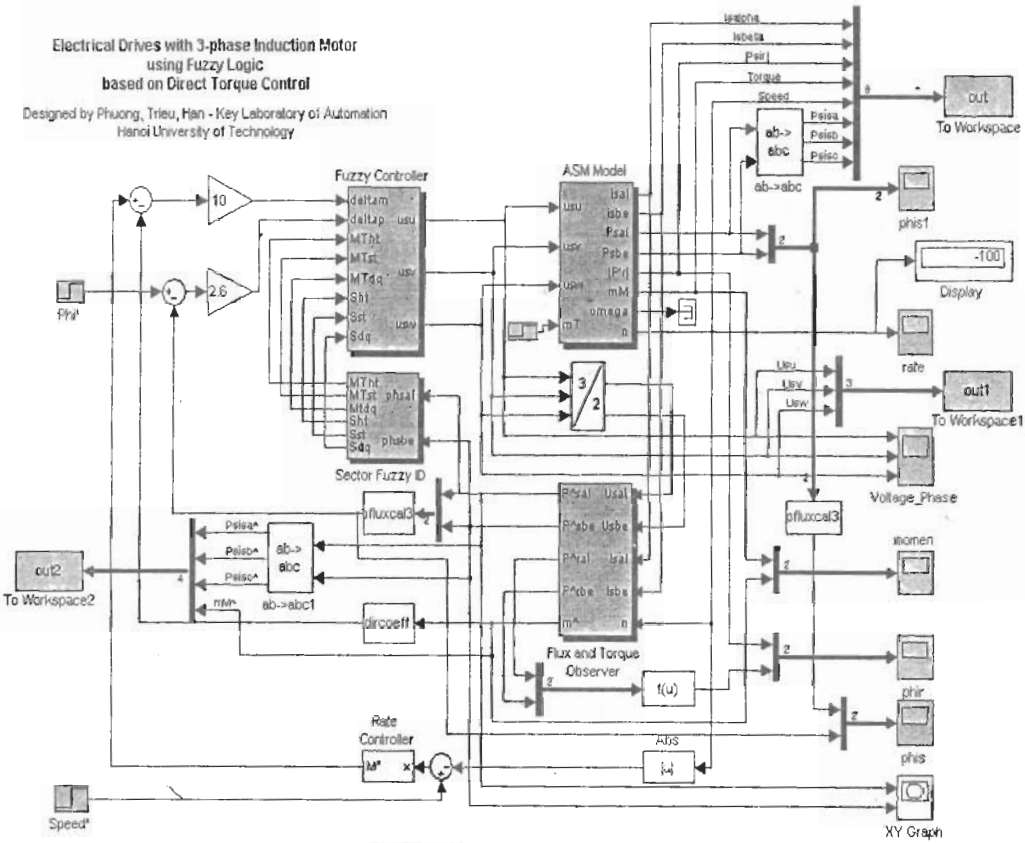
Trong luật ĐK của vùng biên giới (hình 12.38), ta phải chú ý bảo đảm chuyển từ chế độ điều chế với 2 vector (chỉ giành cho quỹ đạo lục giác) sang chế độ điều chế với 3 vector, sử dụng 2 vector biên ("*flux forwards constant*" của Sector hiện tại và sắp tới) và 1 vector không.

Từ kết quả của các phần trình bầy ở trên ta có thể chi tiết hóa cấu trúc ở hình 12.28 và thu được sơ đồ cấu trúc của hệ thống ĐC mờ cho MĐDB như hình 12.39 dưới đây.



Hình 12.39 Cấu trúc hệ thống ĐC mờ cho MĐDB

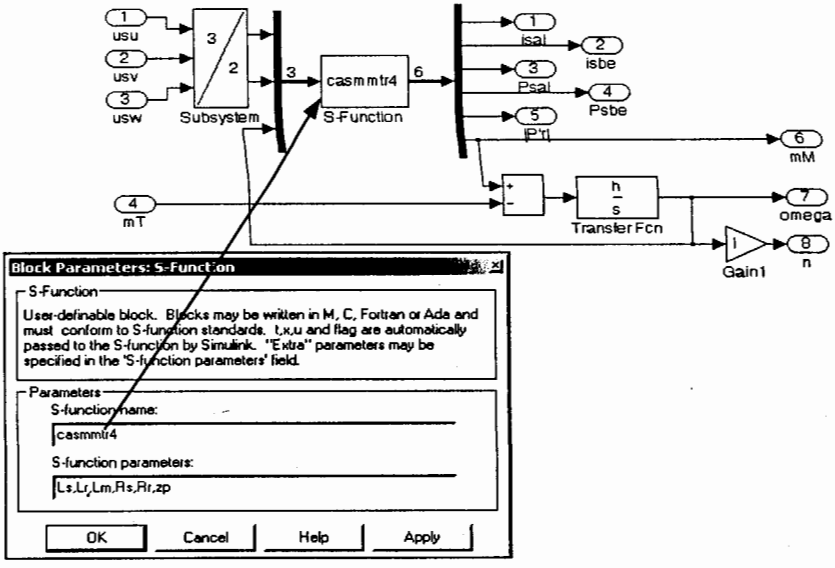
Cấu trúc ở hình 12.39 là xuất phát điểm để ta xây dựng sơ đồ SIMULINK *mô phỏng hệ thống truyền động điện theo nguyên lý DTC sử dụng logic mờ* ở hình 12.40. Sau đây, ta sẽ lần lượt làm quen với các khối của sơ đồ.



Hình 12.40 Sơ đồ SIMULINK mô phỏng hệ thống truyền động điện không đồng bộ xoay chiều ba pha theo nguyên lý DTC và sử dụng Fuzzy Logic

f) Mô hình SIMULINK của MĐDB

Đây là vấn đề đã được giải quyết ở các mục trước: Mô hình SIMULINK của MĐDB (hình 12.40: khối *ASM Model*) được xây dựng trên hệ tọa độ $\alpha\beta$, là hệ tọa độ cố định so với Stator. Mô hình có phương trình toán (10.11) đã cho ở mục 10.3. Để tăng tốc độ mô phỏng ta sử dụng *S-Function* viết dưới dạng *C-mex-File*.



Hình 12.41 *Khối ASM-Model thực hiện bằng C-mex-File (hàm* `casmmtr4`*)*

g) Mô hình SIMULINK của khâu quan sát từ thông và mômen quay

Mô hình SIMULINK của khâu QS từ thông được thực hiện theo các phương trình (12.38) - (12.43) với các tham số đã được định nghĩa tại đó. Việc mô hình hóa được thực hiện theo hai bước chính như sau:

- Sử dụng các công cụ thuộc chương 3 để xác định bộ tham số k_1 ... k_4 của ma trận trọng lượng thuộc công thức (12.41). Phương pháp sử dụng để xác định bộ tham số đó là phương pháp gán cực.
- Soạn thảo một *C-File* tính toán khâu QS theo theo các phương trình (12.38) - (12.43). Sau khi gọi `mex observer_name.c` (giả thiết *C-File* có tên là `observer_name.c`) tại cửa sổ lệnh của MATLAB ta sẽ thu được hàm S (*C-mex-File*) có tên `observer_name.dll`.

Khi sử dụng các công cụ của MATLAB để tìm bộ tham số k_1 ... k_4 cần phải lưu ý rằng:

- Xét phương trình đặc tính của khâu QS ta thấy: Hệ thống có hai cặp điểm cực phức liên hợp với vị trí phụ thuộc tần số (còn gọi là đặc điểm nhảy tần số, phụ thuộc điểm làm việc).

- Vị trí các điểm cực của khâu QS gián đoạn (đặc điểm của hệ thống số) trên miền ảnh z còn phụ thuộc vào chu kỳ trích mẫu T.

Chính vì vậy, cần xây dựng (bằng công cụ MATLAB) lớp các quỹ đạo điểm cực (với hai tham số là ω và T) của đối tượng MĐDB trên miền ảnh z để phân tích hiểu rõ đối tượng. Việc tìm bộ tham số $k_1 \ldots k_4$ cần bảo đảm:

- *Các điểm cực mới phải nằm trong đường tròn đơn vị*: Nhằm bảo đảm tính ổn định của khâu QS.
- *Các điểm cực mới nằm càng gần gốc tọa độ càng tốt*: Nhằm nâng cao tốc độ hội tụ (đặc tính động học) của kết quả QS.

Sau khi đã tìm được bộ tham số ta có thể thực hiện khâu QS với đoạn mã C như dưới đây.

```
/* OBSERVER.C: Torque, rotor and stator flux observer  */
#define  S_FUNCTION_NAME  observer
#define  S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define  U(element) (*uPtrs[element])

static void mdlInitializeSizes(SimStruct *S)
{ ssSetNumSFcnParams(S,12);
/*Ls(0),Lr(1),Lm(2),Rs(3),Rr(4),w0(5),k1(6),k2(7),
  k3(8),k4(9),T(10),zp(11) */
if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))
  return;
  ssSetNumContStates(S,0);
  ssSetNumDiscStates(S,4);
                          //Psalpha,Psbeta,Pralpha,Prbeta
if (!ssSetNumInputPorts(S,1)) return;
  ssSetInputPortWidth(S,0,5);
                          //Usalpha,Usbeta,Isalpha,Isbeta,w
  ssSetInputPortDirectFeedThrough(S,0,0);
                          //See mdlOutputs
if (!ssSetNumOutputPorts(S,1)) return;
  ssSetOutputPortWidth(S,0,5);
                          //Psalpha,Psbeta,Pralpha,Prbeta,M
  ssSetNumSampleTimes(S,1);
                          //Discrete Sample Time T
  ssSetNumRWork(S,0);
  ssSetNumIWork(S,0);
  ssSetNumPWork(S,0);
  ssSetNumModes(S,0);
  ssSetNumNonsampledZCs(S,0);
  ssSetOptions(S,SS_OPTION_EXCEPTION_FREE_CODE);
}
```

```
static void mdlInitializeSampleTimes(SimStruct *S)
{ real_T T = mxGetPr(ssGetSFcnParam(S,10))[0];
  ssSetSampleTime(S,0,T);
  ssSetOffsetTime(S,0,0.0);
}

#define MDL_INITIALIZE_CONDITIONS
#if defined(MDL_INITIALIZE_CONDITIONS)
static void mdlInitializeConditions(SimStruct *S)
{ int_T i;
  real_T *x0 = ssGetRealDiscStates(S);
  for(i=0;i<4;i++)
  { *x0++=0.0;
  }
}
#endif

static void mdlOutputs(SimStruct *S, int_T tid)
{ real_T *y = ssGetOutputPortRealSignal(S,0);
  real_T *x = ssGetRealDiscStates(S);
  real_T zp = mxGetPr(ssGetSFcnParam(S,11))[0];
  real_T Ls = mxGetPr(ssGetSFcnParam(S,0))[0];
  real_T Lr = mxGetPr(ssGetSFcnParam(S,1))[0];
  real_T Lm = mxGetPr(ssGetSFcnParam(S,2))[0];
/* Xuất các thành phần từ thông và mômen tới đầu ra */
  y[0] = x[0];
  y[1] = x[1];
  y[2] = x[2];
  y[3] = x[3];
  y[4] = ((1.5*zp)/(Ls+Lr-2*Lm))*(x[1]*x[2]-x[0]*x[3]);
}

#define MDL_UPDATE
#if defined(MDL_UPDATE)
static void mdlUpdate(SimStruct *S, int_T tid)
{ real_T *x              = ssGetRealDiscStates(S);
  real_T tempX[4]        = {0.0, 0.0, 0.0, 0.0};
  InputRealPtrsType uPtrs =
                ssGetInputPortRealSignalPtrs(S,0);
  real_T Ls = mxGetPr(ssGetSFcnParam(S,0))[0];
  real_T Lr = mxGetPr(ssGetSFcnParam(S,1))[0];
  real_T Lm = mxGetPr(ssGetSFcnParam(S,2))[0];
  real_T Rs = mxGetPr(ssGetSFcnParam(S,3))[0];
  real_T Rr = mxGetPr(ssGetSFcnParam(S,4))[0];
  real_T w0 = mxGetPr(ssGetSFcnParam(S,5))[0];
  real_T k1 = mxGetPr(ssGetSFcnParam(S,6))[0];
  real_T k2 = mxGetPr(ssGetSFcnParam(S,7))[0];
```

```
  real_T k3 = mxGetPr(ssGetSFcnParam(S,8))[0];
  real_T k4 = mxGetPr(ssGetSFcnParam(S,9))[0];
  real_T T  = mxGetPr(ssGetSFcnParam(S,10))[0];
  real_T zp = mxGetPr(ssGetSFcnParam(S,11))[0];
/* Tính các tham số và khai báo biến phụ */
  real_T Lsi     = Ls+Lr-2*Lm;
  real_T Tsi     = Lsi/Rr;
  real_T Tr      = (Ls+Lsi)/Rr;
  real_T Ts      = Ls/Rs;
  real_T zo      = Tr/Ts;
  real_T sigma   = Lsi/(Ls+Lsi);
  real_T U_nmlz = 1/w0;
  real_T I_nmlz = Lsi;
  real_T hso     = T*4*3.1416/60;
  real_T a3      = T/Tsi;
  real_T a1      = 1-a3*zo;
  real_T a2      = a3*zo*(1-sigma);
  real_T a4      = 1-a3;
  real_T a5      = a3*w0*Tsi;
  real_T a6      = 1/(1-sigma);
  real_T co      = cos(hso*U(4));
  real_T si      = sin(hso*U(4));
/* Tính các thành phần từ thông thuộc khâu quan sát */
  tempX[0] =    (a1-k1*a1*a6-a2*k3*a6)*x[0]
             + (a1*k2*a6+a2*k4*a6)*x[1]
             + (a1*k1+a2*k3+a2*x[2]
             - (a1*k2+a2*k4)*x[3] + U_nmlz*a5*U(0)
             + I_nmlz*((a1*k1+a2*k3)*U(2)
             - (a1*k2+a2*k4)*U(3));
  tempX[1] = - (k2*a1*a6+k4*a2*a6)*x[0]
             - (k1*a1*a6+k3*a2*a6-a1)*x[1]
             + (a1*k2+a2*k4)*x[2]
             + (a1*k1+a2*k3+a2)*x[3] + U_nmlz*a5*U(1)
             + I_nmlz*((a1*k2+a2*k4)*U(2)
             + (k1*a1+k3*a2)*U(3));
  tempX[2] = (- (k1*a3*a6+k3*a4*a6-a3)*co
             + (k2*a3*a6+k4*a4*a6)*si)*x[0]
             + ((k2*a3*a6+k4*a4*a6)*co
             + (k1*a3*a6+k3*a4*a6-a3)*si)*x[1]
             + ((k1*a3+k3*a4+a4)*co
             - (k2*a3+k4*a4)*si)*x[2] + (-(k2*a3+k4*a4)*co
             - (k1*a3+k3*a4+a4)*si)*x[3]
             + I_nmlz*((k3*a4+k1*a3)*co
             - (k2*a3+k4*a4)*si)*U(2)
             - I_nmlz*((k2*a3+k4*a4)*co
             + (k1*a3+k3*a4)*si)*U(3);
  tempX[3] = (-(k1*a3*a6+k3*a4*a6-a3)*si
```

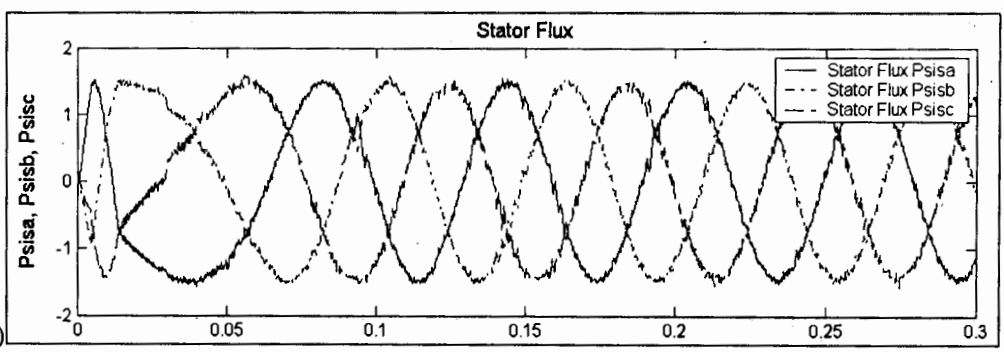
```
                - (k2*a3*a6+k4*a4*a6)*co)*x[0]
                + ((k2*a3*a6+k4*a4*a6)*si
                - (k1*a3*a6+k3*a4*a6-a3)*co)*x[1]
                + ((k1*a3+k3*a4+a4)*si
                + (k2*a3+k4*a4)*co)*x[2]
                + (-(k2*a3+k4*a4)*si+(k1*a3+k3*a4+a4)*co)*x[3]
                + I_nmlz*((k1*a3+k3*a4)*si
                + (k2*a3+k4*a4)*co)*U(2)
                + I_nmlz*(-(k2*a3+k4*a4)*si
                + (k1*a3+k3*a4)*co)*U(3);
  /* Gán các biến trạng thái của khâu quan sát */
    x[0] = tempX[0];
    x[1] = tempX[1];
    x[2] = tempX[2];
    x[3] = tempX[3];
  }
  #endif

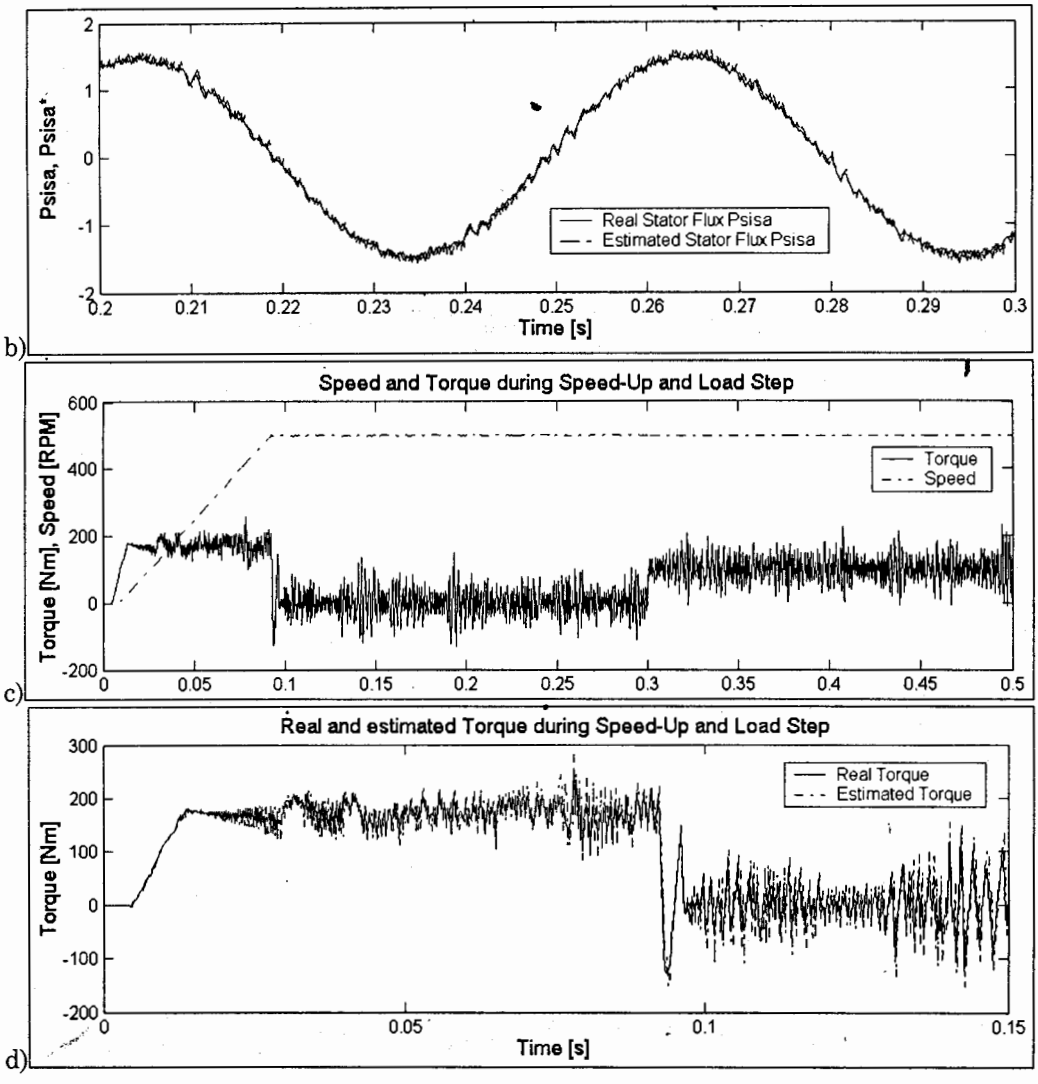
  static void mdlTerminate(SimStruct *S)
  {
  }

  #ifdef MATLAB_MEX_FILE
  #include "simulink.c"
  #else
  #include "cg_sfun.h"
  #endif
```

Các đặc tính ở hình 12.42 là kết quả mô phỏng của khâu QS trong trường hợp từ thông Stator được dẫn dắt theo quỹ đạo tròn.

a)



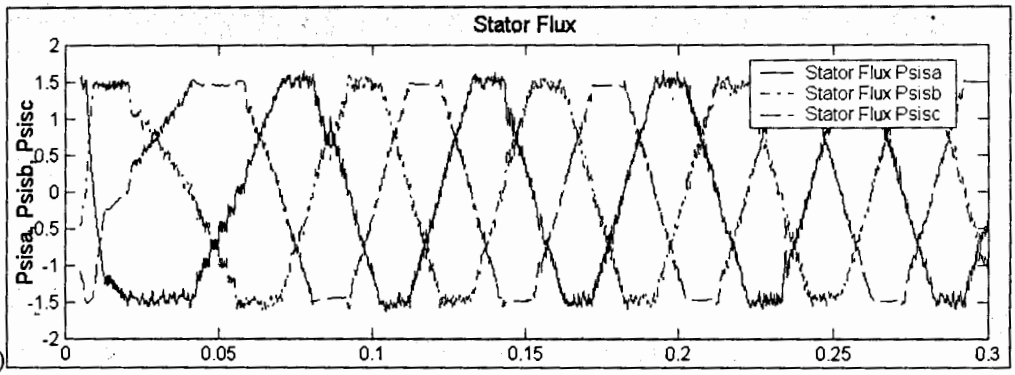


Hình 12.42 *Kết quả mô phỏng với khâu QS khi từ thông được dẫn dắt theo quỹ đạo tròn: a) Ba từ thông pha phía Stator; b) Từ thông pha a (kéo dãn trục thời gian) của động cơ và giá trị QS được của nó; c) Tốc độ và mômen quay của quá trình khởi động và đột biến phụ tải sau 0,3s; d) Mômen thực và mômen QS được (trích từ hình c)*

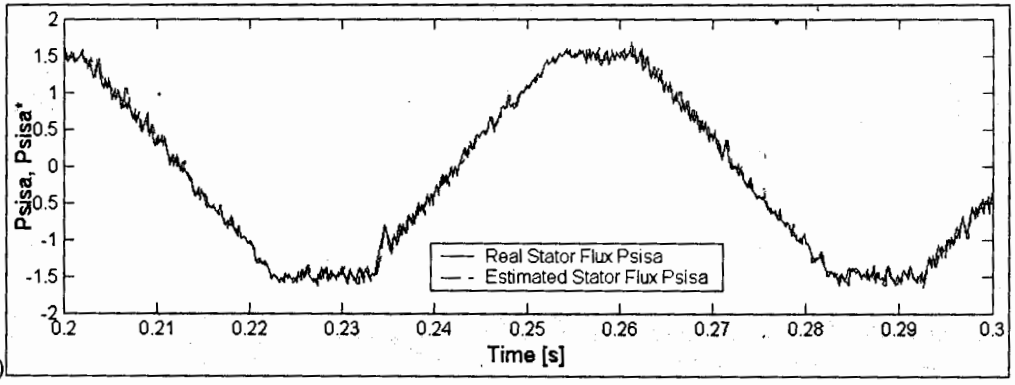
Theo dõi hình 12.42b ta có thể nhận thấy sự trùng khớp chính xác giữa từ thông Stator liên tục (pha a) của động cơ với giá trị gián đoạn (*digital*) do khâu QS tính được. Hình 12.42c minh họa diễn biến tốc độ, mômen quay của quá trình khởi động và đột ngột đóng tải sau 0,3s. Nếu kéo dãn trục thời gian ta có thể thấy chất lượng QS mômen quay rất tốt tại hình 12.42d.

Hình 12.43 giới thiệu các kết quả mô phỏng khi dẫn dắt từ thông Stator đi theo quỹ đạo hình lục giác. Có thể thấy rất rõ dạng hình thang của ba từ thông pha ở hình 12.43a, đã được mô tả lý thuyết ở hình 12.24. Hình 12.43b,d minh chứng một lần nữa chất lượng của khâu QS từ thông và mômen quay. Thành phần hài chứa trong đặc tính mômen ở hình 12.43c cho thấy rõ nhược điểm khi ĐK theo quỹ đạo lục giác.

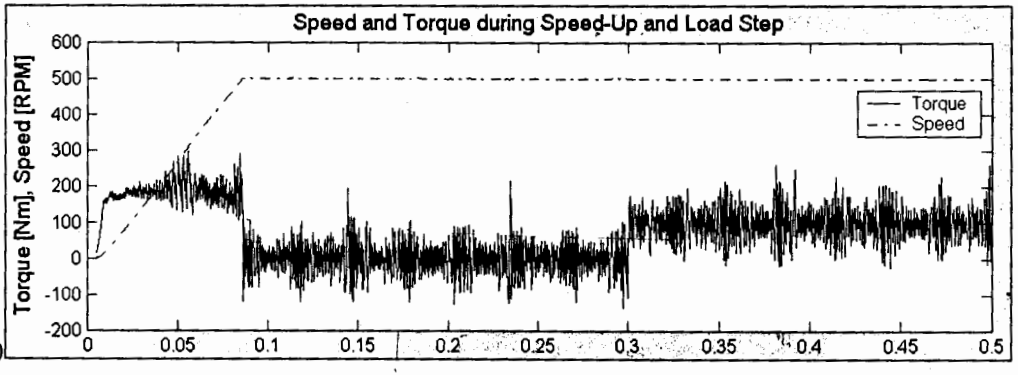
a)

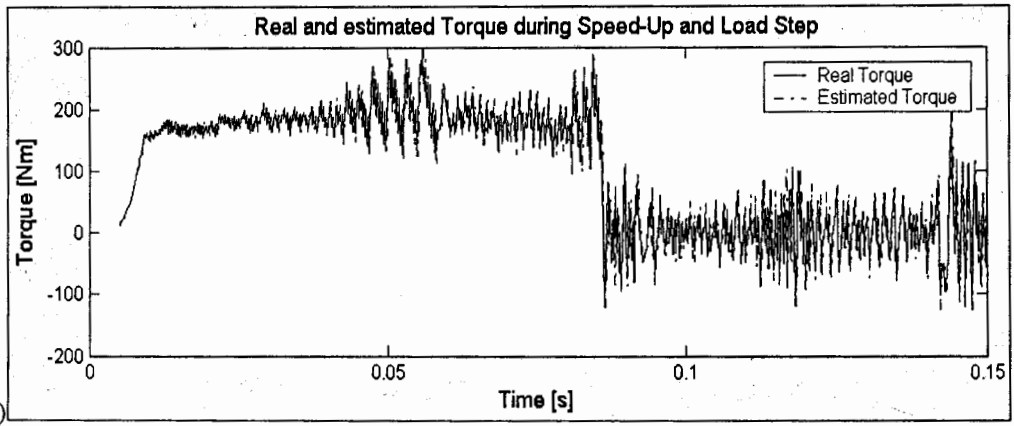


b)



c)





d)

Hình 12.43 Kết quả mô phỏng với khâu QS khi từ thông được dẫn dắt theo quỹ đạo lục giác: a) Ba từ thông pha phía Stator; b) Từ thông pha a của động cơ và giá trị QS được của nó; c) Tốc độ và mômen quay của quá trình khởi động và đột biến phụ tải sau 0,3s; d) Mômen thực và mômen QS được

h) Mô hình SIMULINK của khâu nhận dạng mờ Sector

Mô hình SIMULINK của khâu nhận dạng mờ góc phần sáu được thực hiện theo nguyên lý đã mô tả cho quỹ đạo tròn (hình 12.35 và 36), quỹ đạo lục giác (hình 12.37 và 38). Thông tin chính xác về Sector hiện tại của vector từ thông Stator mang ý nghĩa quyết định khi áp dụng luật ĐK (hình 12.31 và 32), chọn đúng vector điện áp cần thiết để đưa tới Stator.

Tuy nhiên, việc nhận dạng chính xác Sector (ví dụ: Bằng cách xét dấu các thành phần từ thông pha) sẽ khiến cho quá trình chuyển động từ Sector cũ sang Sector kế tiếp trở nên không sạch sẽ (hậu quả của việc xét dấu khi từ thông đi qua điểm 0). Khâu nhận dạng mờ Sector đã *"mềm hóa"* quá trình chuyển tiếp đó, nâng cao chất lượng truyền động lên rất nhiều, điều này có thể nhận thấy dễ dàng thông qua mô phỏng.

Mô hình SIMULINK của khâu nhận dạng Sector cũng được thực hiện bằng hàm S theo dạng *C-mex-File* với đoạn mã nguồn C như dưới đây.

```
/* File: sector_id.c
   Abstract: This is a C-code part of the C-MEX S-function
   file (dll) for sector indentifier.
Input in order:
   Psu: Projection of stator flux vector on u axis.
   Psv: Projection of stator flux vector on v axis.
   Psw: Projection of stator flux vector on w axis.
Output in order:
   y[0] Membership value of current sector.
   y[1] Membership value of forthcoming sector.
```

```
    y[2] Membership value of just bypast sector.
    y[3] Sector Index of current sector.
    y[4] Sector Index of forthcoming sector.
    y[5] Sector Index of just bypast sector.
/* Chú ý: Membership value:   Mức thuộc
          Current sector:     Sector hiện tại
          Fortcoming sector:  Sector sắp tới
          Just bypast sector: Sector vừa qua */
Parameter:
   neg_threshold:  negative threshold
   pos_threshold:  positive threshold
   sample time
   direction of rotor's rotation. */

#define S_FUNCTION_NAME sector_id
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) (*uPtrs[element])

/* Function: mdlInitializeSizes */
static void mdlInitializeSizes(SimStruct *S)
{ ssSetNumSFcnParams(S,4); /* pos_threshold, neg_threshold,
                              T and direction of rotor */
  if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))
     return;
  ssSetNumContStates(S,0);
  ssSetNumDiscStates(S,0);

  if (!ssSetNumInputPorts(S,1)) return;
  ssSetInputPortWidth(S,0,3);          //Psu and Psv and Psw
  ssSetInputPortDirectFeedThrough(S,0,1);

  if(!ssSetNumOutputPorts(S,1)) return;
  ssSetOutputPortWidth(S,0,6);
  ssSetNumSampleTimes(S,1);
  ssSetNumRWork(S,0);
  ssSetNumIWork(S,0);
  ssSetNumPWork(S,0);
  ssSetNumModes(S,0);
  ssSetNumNonsampledZCs(S,0);
  ssSetOptions(S,SS_OPTION_EXCEPTION_FREE_CODE);
} /* end mdlInitializeSizes */

/* Function: mdlInitializeSampleTimes */
static void mdlInitializeSampleTimes(SimStruct *S)
{ int_T T=mxGetPr(ssGetSFcnParam(S,2))[0];
  ssSetSampleTime(S,0,T);
```

```
  ssSetOffsetTime(S,0,0.0);
} /* end mdlInitializeSampeTimes */

/* Function: mdlOutputs */
static void mdlOutputs(SimStruct *S, int_T tid)
{ InputRealPtrsType uPtrs =
                          ssGetInputPortRealSignalPtrs(S,0);
  real_T *y             = ssGetOutputPortRealSignal(S,0);
  real_T pos_threshold = mxGetPr(ssGetSFcnParam(S,0))[0];
  real_T neg_threshold = mxGetPr(ssGetSFcnParam(S,1))[0];
  real_T chquay        = mxGetPr(ssGetSFcnParam(S,3))[0];
  real_T neg[3],pos[3];    //nega,negb,negc,posa,posb,posc
  int_T  i,j,Sector_Idx[7];                  //Sector Index
  real_T Sector[7];                          //0,1,2,3,4,5,6

  for(i=0;i<3;i++)
  { if ((neg_threshold <= U(i)) && (U(i) <= pos_threshold))
    { pos[i] = (U(i) - neg_threshold) / (pos_threshold
                                        - neg_threshold);
      neg[i] = (U(i) - pos_threshold) / (neg_threshold
                                        - pos_threshold);
    }
    else
    { if (U(i)<neg_threshold)
      { pos[i] = 0;
        neg[i] = 1;
      }
      else
      { pos[i] = 1;
        neg[i] = 0;
      }
    }
  }
  Sector[1] = pos[0]*neg[1]*neg[2];
  Sector[2] = pos[0]*pos[1]*neg[2];
  Sector[3] = neg[0]*pos[1]*neg[2];
  Sector[4] = neg[0]*pos[1]*pos[2];
  Sector[5] = neg[0]*neg[1]*pos[2];
  Sector[6] = pos[0]*neg[1]*pos[2];

  for(i=0;i<7;i++) Sector_Idx[i] = i;

  for(i=1;i<6;i++)
    for(j=i+1;j<7;j++)
    { if(Sector[i]<Sector[j])
      { Sector[0] = Sector[i];
        Sector[i] = Sector[j];
```

```
            Sector[j]=Sector[0];
            Sector_Idx[0]=Sector_Idx[i];
            Sector_Idx[i]=Sector_Idx[j];
            Sector_Idx[j]=Sector_Idx[0];
/* Sector[0] and Sector_Idx[0] is temporary variable */
          }
      }

/* chquay==1: Quay thuận chiều kim đồng hồ */
  if(chquay==1)
  { if((Sector_Idx[1]==1)||(Sector_Idx[1]==6))
    { if(Sector_Idx[2]<Sector_Idx[3])
      { y[1] = Sector[3];
        y[2] = Sector[2];
        y[4] = Sector_Idx[3];
        y[5] = Sector_Idx[2];
      }
      else
      { y[1] = Sector[2];
        y[2] = Sector[3];
        y[4] = Sector_Idx[2];
        y[5] = Sector_Idx[3];
      }
    }
    else
    { if(Sector_Idx[2]>Sector_Idx[3])
      { y[1] = Sector[3];
        y[2] = Sector[2];
        y[4] = Sector_Idx[3];
        y[5] = Sector_Idx[2];
      }
      else
      { y[1] = Sector[2];
        y[2] = Sector[3];
        y[4] = Sector_Idx[2];
        y[5] = Sector_Idx[3];
      }
    }
  }
/* chquay==2: Quay ngược chiều kim đồng hồ */
  if(chquay==2)
  { if((Sector_Idx[1]==1)||(Sector_Idx[1]==6))
    { if(Sector_Idx[2]>Sector_Idx[3])
      { y[1] = Sector[3];
        y[2] = Sector[2];
        y[4] = Sector_Idx[3];
        y[5] = Sector_Idx[2]; }
```

```
        else
        { y[1] = Sector[2];
          y[2] = Sector[3];
          y[4] = Sector_Idx[2];
          y[5] = Sector_Idx[3];
        }
      }
      else
      { if(Sector_Idx[2]<Sector_Idx[3])
        { y[1] = Sector[3];
          y[2] = Sector[2];
          y[4] = Sector_Idx[3];
          y[5] = Sector_Idx[2];
        }
        else
        { y[1] = Sector[2];
          y[2] = Sector[3];
          y[4] = Sector_Idx[2];
          y[5] = Sector_Idx[3];
        }
      }
    }
    y[0] = Sector[1];
    y[3] = Sector_Idx[1];
  } /* end mdlOutputs */

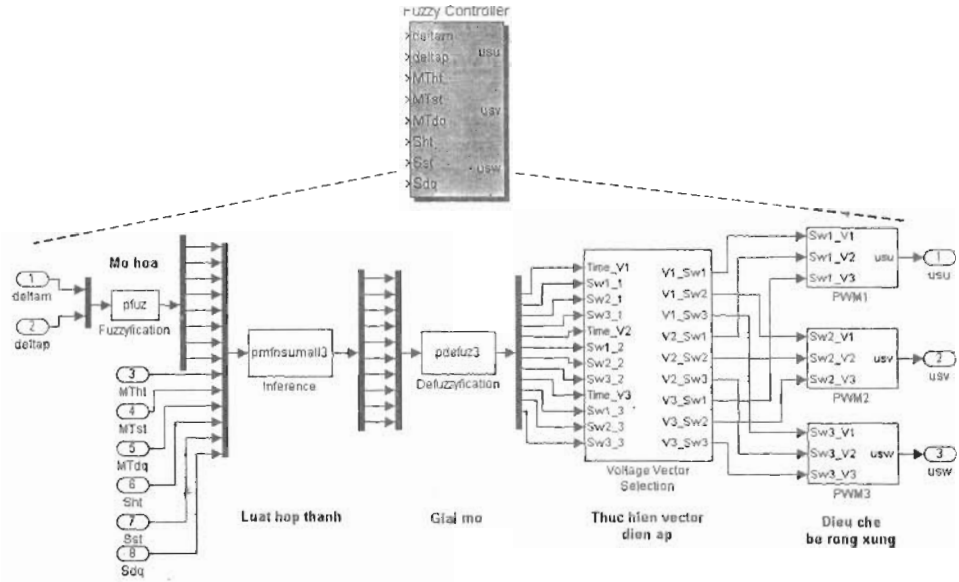
  /* Function: mdlTerminate */
  static void mdlTerminate(SimStruct *S)
  {
  } /* end mdlTerminate */
  #ifdef MATLAB_MEX_FILE
  #include "simulink.c"
  #else
  #include "cg_sfun.h"
  #endif
```

i) Mô hình SIMULINK của khâu ĐC mờ (khối Fuzzy Controller)

Khối Fuzzy Controller có tất cả 8 đầu vào với ý nghĩa như sau:

- `deltam`, `deltap`: sai lệch mômen quay, từ thông Stator (hiệu số giữa giá trị đặt và giá trị thực)
- `MTht`, `MTst`, `MTdq`: mức thuộc của Sector hiện tại, sắp tới và đã qua
- `Sht`, `Sst`, `Sdq`: Chỉ số của Sector hiện tại, sắp tới và đã qua

Với 3 điện áp ra `usu`, `usv` và `usw`. Ba điện áp ra có dạng xung vuông điều chế với biên độ phụ thuộc điện áp một chiều trung gian U_{DC}.

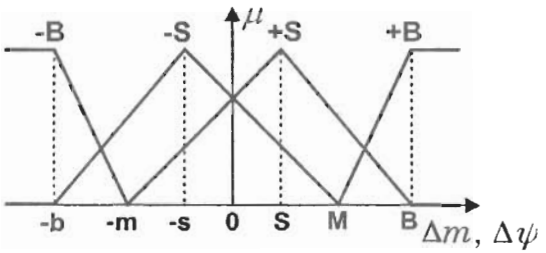


***Hình 12.44** Khối Fuzzy Controller và sơ đồ SIMULINK chi tiết gồm các khâu Fuzzyfication (mờ hóa), Inference (luật hợp thành), Defuzzyfication (giải mờ), Voltage Vector Selection (thực hiện vector điện áp) và 3 khâu PWM (điều chế bề rộng xung)*

Sau đây ta sẽ lần lượt tìm hiểu cách thực hiện từng khối con thuộc sơ đồ SIMULINK chi tiết của khối *Fuzzy Controller*.

Khối Fuzzyfication (mờ hóa)

Khối được thực hiện dưới dạng hàm S. Trong hình 12.44 hàm S có tên `pfuz` và được soạn thảo bằng *m-File* hay *C-mex-File*. Hàm liên thuộc tổng quát ở hình 12.30 được cụ thể hóa lại như dưới đây.



***Hình 12.45** Dạng hàm liên thuộc sử dụng trong S-Function `pfuz`*

Khối *Fuzzyfication* có hai đầu vào là các tín hiệu sai lệch mômen quay `deltam` và sai lệch từ thông Stator `deltap`.

Tám đầu ra của khối là:

- 4 đầu ra ứng với 4 mức thuộc của `deltam` vào 4 hàm liên thuộc –B, –S, +S và +B.
- 4 đầu ra ứng với 4 mức thuộc của `deltap` vào 4 hàm liên thuộc –B, –S, +S và +B.

Các tham số b, m, s, S, M và B có thể được khai báo trước và giữ vai trò quan trọng đối với chất lượng truyền động. Có thể thực hiện khâu mờ hóa với các hàm liên thuộc ở hình 12.45 bằng *script* (*m-File*) sau đây.

```
function [sys,x0,str,ts] = pfuz(t,x,u,flag,T)
switch flag,
case 0
  [sys,x0,str,ts]=mdlInitializeSizes(T);
case 3
  sys = mdlOutputs(t,x,u);
case {1,2,4,9}
  sys = [];
otherwise
  error(['Unhandled flag=',num2str(flag)]);
end
%-----------------------------------------------
function [sys,x0,str,ts] = mdlInitializeSizes(T)
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs    = 8;
% sys(1),sys(2),sys(3),sys(4)là 4 mức thuộc của delta m
% vào (-B,-S,+S,+B)
% sys(5),sys(6),sys(7),sys(8)là 4 mức thuộc của delta Psi
% vào (-B,-S,+S,+B)

sizes.NumInputs      = 2;              % delta m, delta Psi
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [-1 0];
%-----------------------------------------------
function sys = mdlOutputs(t,x,u)
% Mờ hóa delta m %
b=-12.00000; m=-5.00000; s=-0.20000;
S=1.00000; M=3.00000; B=5.00000;
if (u(1)<=b)
   sys(1)=1;                          % B-
   sys(2)=0;                          % S-
   sys(3)=0;                          % S+
   sys(4)=0;                          % B+
end
if ((u(1)>b)&(u(1)<=m))
   sys(1)=u(1)/(b-m)+m/(m-b);
   sys(2)=u(1)/(s-b)+b/(b-s);
```

```
    sys(3)=0;
    sys(4)=0;
end
if (u(1)>m)&(u(1)<=s)
    sys(1)=0;
    sys(2)=u(1)/(s-b)+b/(b-s);
    sys(3)=u(1)/(S-m)+m/(m-S);
    sys(4)=0;
end
if (u(1)>s)&(u(1)<=S)
    sys(1)=0;
    sys(4)=0;
    sys(2)=u(1)/(s-M)+M/(M-s);
    sys(3)=u(1)/(S-m)+m/(m-S);
end
if (u(1)>S)&(u(1)<=M)
    sys(4)=0;
    sys(1)=0;
    sys(2)=u(1)/(s-M)+M/(M-s);
    sys(3)=u(1)/(S-B)+B/(B-S);
end
if (u(1)>M)&(u(1)<B)
    sys(1)=0;
    sys(2)=0;
    sys(3)=u(1)/(S-B)+B/(B-S);
    sys(4)=u(1)/(B-M)+M/(M-B);
end
if(u(1)>=B)
    sys(1)=0;
    sys(2)=0;
    sys(3)=0;
    sys(4)=1;
end

% Mờ hóa delta Psi %
b1=-.03000; m1=-0.02000; s1=-0.02000;
S1=0.02000; M1=0.02000; B1=0.03000;%
if (u(2)<=b1)
    sys(5)=1;                                % B-
    sys(6)=0;                                % S-
    sys(7)=0;                                % S+
    sys(8)=0;                                % B+
end
if ((u(2)>b1)&(u(2)<=m1))
    sys(5)=u(2)/(b1-m1)+m1/(m1-b1);
    sys(6)=u(2)/(s1-b1)+b1/(b1-s1);
    sys(7)=0;
```

```
    sys(8)=0;
end
if (u(2)>m1)&(u(2)<=s1)
    sys(5)=0;
    sys(6)=u(2)/(s1-b1)+b1/(b1-s1);
    sys(7)=u(2)/(S1-m1)+m1/(m1-S1);
    sys(8)=0;
end
if (u(2)>s1)&(u(2)<=S1)
    sys(5)=0;
    sys(8)=0;
    sys(6)=u(2)/(s1-M1)+M1/(M1-s1);
    sys(7)=u(2)/(S1-m1)+m1/(m1-S1);
end
if (u(2)>S1)&(u(2)<=M1)
    sys(8)=0;
    sys(5)=0;
    sys(6)=u(2)/(s1-M1)+M1/(M1-s1);
    sys(7)=u(2)/(S1-B1)+B1/(B1-S1);
end
if (u(2)>M1)&(u(2)<=B1)
    sys(5)=0;
    sys(6)=0;
    sys(7)=u(2)/(S1-B1)+B1/(B1-S1);
    sys(8)=u(2)/(B1-M1)+M1/(M1-B1);
end
if(u(2)>B1)
    sys(5)=0;
    sys(6)=0;
    sys(7)=0;
    sys(8)=1;
end
```

Để tăng tốc độ mô phỏng ta cũng có thể thực hiện khối *Fuzzyfication* bằng *C-mex-File* với đoạn mã dưới đây.

```
#define  S_FUNCTION_NAME pfuz.c
#define  S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define  u(element) (*uPtrs[element])
#define  sys(element) y[element]

/* Function: mdlInitializeSizes */
static void mdlInitializeSizes(SimStruct *S)
{ ssSetNumSFcnParams(S,12);//b,m,s,S,M,B  b1,m1,s1,S1,M1,B1
  if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))
      return;
```

```
    ssSetNumContStates(S,0);
    ssSetNumDiscStates(S,0);

    if (!ssSetNumInputPorts(S,1)) return;
    ssSetInputPortWidth(S,0,2);
    ssSetInputPortDirectFeedThrough(S,0,1);

    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S,0,8);
    ssSetNumSampleTimes(S,1);
    ssSetNumRWork(S,0);
    ssSetNumIWork(S,0);
    ssSetNumPWork(S,0);
    ssSetNumModes(S,0);
    ssSetNumNonsampledZCs(S,0);

    ssSetOptions(S,SS_OPTION_EXCEPTION_FREE_CODE);
} /* end mdlInitializeSizes */

/* Function: mdlInitilizeSampleTimes */
static void mdlInitializeSampleTimes(SimStruct *S)
{ ssSetSampleTime(S,0,INHERITED_SAMPLE_TIME);
  ssSetOffsetTime(S,0,0.0);
} /* end mdlInitializeSampleTimes */

/* Function: mdlOuputs */
static void mdlOutputs(SimStruct *S,int_T tid)
{ InputRealPtrsType uPtrs
              = ssGetInputPortRealSignalPtrs(S,0);
  real_T  *y  = ssGetOutputPortRealSignal(S,0);
  real_T  b   = mxGetPr(ssGetSFcnParam(S,0))[0];
  real_T  m   = mxGetPr(ssGetSFcnParam(S,1))[0];
  real_T  s   = mxGetPr(ssGetSFcnParam(S,2))[0];
  real_T  Ss  = mxGetPr(ssGetSFcnParam(S,3))[0];
  real_T  M   = mxGetPr(ssGetSFcnParam(S,4))[0];
  real_T  B   = mxGetPr(ssGetSFcnParam(S,5))[0];
  real_T  b1  = mxGetPr(ssGetSFcnParam(S,6))[0];
  real_T  m1  = mxGetPr(ssGetSFcnParam(S,7))[0];
  real_T  s1  = mxGetPr(ssGetSFcnParam(S,8))[0];
  real_T  S1  = mxGetPr(ssGetSFcnParam(S,9))[0];
  real_T  M1  = mxGetPr(ssGetSFcnParam(S,10))[0];
  real_T  B1  = mxGetPr(ssGetSFcnParam(S,11))[0];

// Delta m //
  if (u(0)<=b)
  { sys(0)=1.00000;
    sys(1)=0.00000;
```

```
    sys(2)=0.00000;
    sys(3)=0.00000;
  }
  if ((u(0)>b)&&(u(0)<=m))
  { sys(0)=u(0)/(b-m)+m/(m-b);
    sys(1)=u(0)/(s-b)+b/(b-s);
    sys(2)=0.00000;
    sys(3)=0.00000;
  }
  if ((u(0)>m)&&(u(0)<=s))
  { sys(0)=0.00000;
    sys(1)=u(0)/(s-b)+b/(b-s);
    sys(2)=u(0)/(Ss-m)+m/(m-Ss);
    sys(3)=0.00000;
  }
  if ((u(0)>s)&&(u(0)<=Ss))
  { sys(0)=0.00000;
    sys(3)=0.00000;
    sys(1)=u(0)/(s-M)+M/(M-s);
    sys(2)=u(0)/(Ss-m)+m/(m-Ss);
  }
  if ((u(0)>Ss)&&(u(0)<=M))
  { sys(3)=0.00000;
    sys(0)=0.00000;
    sys(1)=u(0)/(s-M)+M/(M-s);
    sys(2)=u(0)/(Ss-B)+B/(B-Ss);
  }
  if ((u(0)>M)&&(u(0)<B))
  { sys(0)=0.00000;
    sys(1)=0.00000;
    sys(2)=u(0)/(Ss-B)+B/(B-Ss);
    sys(3)=u(0)/(B-M)+M/(M-B);
  }
  if(u(0)>=B)
  { sys(0)=0.00000;
    sys(1)=0.00000;
    sys(2)=0.00000;
    sys(3)=1.00000;
  }

// Delta Psi //
  if (u(1)<=b1)
  { sys(4)=1.00000;
    sys(5)=0.00000;
    sys(6)=0.00000;
    sys(7)=0.00000;//B+
  }
```

```
    if ((u(1)>b1)&&(u(1)<=m1))
    { sys(4)=u(1)/(b1-m1)+m1/(m1-b1);
      sys(5)=u(1)/(s1-b1)+b1/(b1-s1);
      sys(6)=0.00000;
      sys(7)=0.00000;
    }
    if ((u(1)>m1)&&(u(1)<=s1))
    { sys(4)=0.00000;
      sys(5)=u(1)/(s1-b1)+b1/(b1-s1);
      sys(6)=u(1)/(S1-m1)+m1/(m1-S1);
      sys(7)=0.00000;
    }
    if ((u(1)>s1)&&(u(1)<=S1))
    { sys(4)=0.00000;
      sys(7)=0.00000;
      sys(5)=u(1)/(s1-M1)+M1/(M1-s1);
      sys(6)=u(1)/(S1-m1)+m1/(m1-S1);
    }
    if ((u(1)>S1)&&(u(1)<=M1))
    { sys(7)=0.00000;
      sys(4)=0.00000;
      sys(5)=u(1)/(s1-M1)+M1/(M1-s1);
      sys(6)=u(1)/(S1-B1)+B1/(B1-S1);
    }
    if ((u(1)>M1)&&(u(1)<=B1))
    { sys(4)=0.00000;
      sys(5)=0.00000;
      sys(6)=u(1)/(S1-B1)+B1/(B1-S1);
      sys(7)=u(1)/(B1-M1)+M1/(M1-B1);
    }
    if (u(1)>B1)
    { sys(4)=0.00000;
      sys(5)=0.00000;
      sys(6)=0.00000;
      sys(7)=1.00000;
    }
} /* end mdlOutputs */

/* Function: mdlTerminate */
static void mdlTerminate(SimStruct *S)
{
} /* end mdlTerminate */
#ifdef   MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

Khối Inference (luật hợp thành)

Khối có tất cả là 14 đầu vào và 10 đầu ra. Trong đó:

- 8 đầu vào từ khối *Fuzzyfication* tới (mức thuộc của `deltam` và `deltap` vào các tập mờ, biểu diễn ở hình 12.45).
- 6 đầu vào còn lại `MTht`, `MTst`, `MTdq`, `Sht`, `Sst`, `Sdq` (mức thuộc và chỉ số của Sector hiện tại, sắp tới và đã qua). Cần chú ý rằng: Nói đến Sector là nói đến vị trí của vector từ thông Stator. 6 đầu vào này chính là các đầu ra của khâu nhận dạng mờ Sector.

10 đầu ra của khối có các ý nghĩa như sau:

- *Mức thỏa mãn cao nhất*
- Vector (ngôn ngữ) có mức thỏa mãn cao nhất
- Sector của vector có mức thỏa mãn cao nhất
- *Mức thỏa mãn cao thứ 2*
- Vector (ngôn ngữ) có mức thỏa mãn cao thứ 2
- Sector của vector có mức thỏa mãn cao thứ 2
- *Mức thỏa mãn cao thứ 3*
- Vector (ngôn ngữ) có mức thỏa mãn cao thứ 3
- Sector của vector có mức thỏa mãn cao thứ 3
- Giá trị của biến `choice` (1: quỹ đạo tròn; 2: quỹ đạo lục giác)

Các luật hợp thành của khối được xây dựng theo hình 12.36 (quỹ đạo tròn) và 12.38 (quỹ đạo lục giác). Theo lý thuyết, có thể thực hiện điều kiện AND bằng phép MIN hoặc PROD, điều kiện OR bằng SUM hoặc MAX. Tuy nhiên, mô phỏng đã cho kết quả tốt hơn khi ta sử dụng phép PROD và SUM.

Sau đây là nội dung mã nguồn C của khối *Inference*.

```
/* File: pffnsumall.c
   Abstract: This is a C-code part of the C-MEX S-function
   (.dll-file) for a fuzzy law block (inference rules)
   Input in order (14):
       U(0->3)    Linguistics variable of moment
                  error (B-,S-,S+,B+)
       U(4->7)    Linguistics varialbe of flux's magnitude
                  error (B-,S-,S+,B+)
       U(8)       Membership value of current vector
       U(9)       Membership value of forthcoming vector
       U(10)      Membership value of just bypast sector
       U(11)      Sector index of current vector
       U(12)      Sector index of forthcoming vector
       U(13)      Sector index of just bypast sector
```

```
    Output in order:
        y[0]  Max of membership values (membership value of
              primary vector).
        y[1]  Vector index of primary vector.
        y[2]  Sector index correspond to primary vector.
        y[3]  Membership value of secondary vector.
        y[4]  Vector index of secondary vector.
        y[5]  Sector index correspond to secondary vector.
        y[6]  Membership value of tertiary vector.
        y[7]  Vector index of tertiary vector.
        y[8]  Sector index correspond to tertiary vector.
        y[9]  Choice between circle orbit or hexagonal orbit
              (1: cirlce, 2: hexagon).
    Parameter: No */

#define  S_FUNCTION_NAME  fuzzylawtable
#define  S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define  U(element)       (*uPtrs[element])

/* Function: mdlInitializeSizes */
static void mdlInitializeSizes(SimStruct *S)
{ ssSetNumSFcnParams(S,2);           // choice (circle=1;
                                     // hexagonal=2), T
  if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))
      return;
  ssSetNumContStates(S,0);
  ssSetNumDiscStates(S,0);

  if (!ssSetNumInputPorts(S,1)) return;
  ssSetInputPortWidth(S,0,14);
  ssSetInputPortDirectFeedThrough(S,0,1);

  if( !ssSetNumOutputPorts(S,1)) return;
  ssSetOutputPortWidth(S,0,10);
  ssSetNumSampleTimes(S,1);
  ssSetNumRWork(S,0);
  ssSetNumIWork(S,0);
  ssSetNumPWork(S,0);
  ssSetNumModes(S,0);
  ssSetNumNonsampledZCs(S,0);
  ssSetOptions(S,SS_OPTION_EXCEPTION_FREE_CODE);
} /* end mdlInitializeSizes*/

/*Function: mdlInitializeSampleTimes */
static void mdlInitializeSampleTimes(SimStruct *S)
{ real_T T=mxGetPr(ssGetSFcnParam(S,1))[0];
```

```
  ssSetSampleTime(S,0,T);
  ssSetOffsetTime(S,0,0.0);
} /* end mdlInitializeSampleTimes */

/* Function: mdlOutputs */
static void mdlOutputs(SimStruct *S, int_T tid)
{ InputRealPtrsType uPtrs =
                          ssGetInputPortRealSignalPtrs(S,0);
  real_T  *y = ssGetOutputPortRealSignal(S,0);
  real_T  M[4],P[4],a[4][4];
  int_T   i,j;
  int_T   V_Idxbypast[8],V_Idxcurr[8];
  int_T   V_Idxfcom[8],V_Idx[22];
  real_T  V[22],S_Idx[22];
  real_T  SIbypast,FIbypast,FDbypast,SDbypast;
  real_T  BDbypast,BIbypast,SCbypast;
  real_T  SIcurr,FIcurr,FDcurr,SDcurr;
  real_T  BDcurr,BIcurr,SCcurr;
  real_T  SIfcom,FIfcom,FDfcom,SDfcom;
  real_T  BDfcom,BIfcom,SCfcom;
  real_T  hexBIbypast,hexFIbypast,hexFCbypast,hexFDbypast;
  real_T  hexBDbypast,hexBCbypast, hexSCbypast;
  real_T  hexBIcurr,hexFIcurr,hexFCcurr,hexFDcurr;
  real_T  hexBDcurr,hexBCcurr,hexSCcurr;
  real_T  hexBIfcom,hexFIfcom,hexFCfcom,hexFDfcom;
  real_T  hexBDfcom, hexBCfcom,hexSCfcom;
  real_T  choice=mxGetPr(ssGetSFcnParam(S,0))[0];

  for (i=0;i<4;i++) M[i]=U(i);
  for (i=0;i<4;i++) P[i]=U(i+4);
  for (i=0;i<4;i++)
    for (j=0;j<4;j++)
    { a[i][j]=P[i]*M[j]; }

  if(choice==1)                              // Quỹ đạo tròn
  { /* Sector đã qua */
    SIbypast = U(10)*(a[3][1]+a[3][2]+a[3][3]);
    FIbypast = U(10)*(a[1][2]+a[1][3]+a[2][2]+a[2][3]);
    FDbypast = U(10)*(a[0][1]+a[0][2]+a[0][3]);
    SDbypast = U(10)*a[0][0];
    BDbypast = U(10)*(a[1][0]+a[2][0]);
    BIbypast = U(10)*a[3][0];
    SCbypast = U(10)*(a[1][1]+a[2][1]);
    /* Sector hiện tại */
    SIcurr = U(8)*(a[3][1]+a[3][2]);
    FIcurr = U(8)*(a[3][3]+a[2][2]+a[2][3]);
    FDcurr = U(8)*(a[1][2]+a[1][3]+a[0][3]);
```

```
    SDcurr = U(8)*(a[0][1]+a[0][2]);
    BDcurr = U(8)*(a[0][0]+a[1][0]);
    BIcurr = U(8)*(a[2][0]+a[3][0]);
    SCcurr = U(8)*(a[1][1]+a[2][1]);
    /* Sector sắp tới */
    SIfcom = U(9)*a[3][0];
    FIfcom = U(9)*(a[3][1]+a[3][2]+a[3][3]);
    FDfcom = U(9)*(a[2][2]+a[2][3]+a[1][2],a[1][3]);
    SDfcom = U(9)*(a[0][1]+a[0][2]+a[0][3]);
    BDfcom = U(9)*a[0][0];
    BIfcom = U(9)*(a[1][0]+a[2][0]);
    SCfcom = U(9)*(a[1][1]+a[2][1]);

    V[1]  = SIcurr;
    V[2]  = FIcurr;
    V[3]  = FDcurr;
    V[4]  = SDcurr;
    V[5]  = BDcurr;
    V[6]  = BIcurr;
    V[7]  = SCcurr;
//=======================
    V[8]  = SIfcom;
    V[9]  = FIfcom;
    V[10] = FDfcom;
    V[11] = SDfcom;
    V[12] = BDfcom;
    V[13] = BIfcom;
    V[14] = SCfcom;
//=======================
    V[15] = SIbypast;
    V[16] = FIbypast;
    V[17] = FDbypast;
    V[18] = SDbypast;
    V[19] = BDbypast;
    V[20] = BIbypast;
    V[21] = SCbypast;
  }
  else                              // Quỹ đạo lục giác
  { /* Sector đã qua */
    hexBIbypast = 0;
    hexFIbypast = 0;
    hexFCbypast = 0;
    hexFDbypast = 0;
    hexBDbypast = 0;
    hexBCbypast = 0;
    hexSCbypast = 0;
    /* Sector hiện tại */
```

```
    hexBIcurr = U(8)*a[3][0];
    hexFIcurr = U(8)*(a[3][2]+a[3][3]);
    hexFCcurr = U(8)*(a[2][2]+a[2][3]+a[1][2]+a[1][3]);
    hexFDcurr = U(8)*(a[0][2]+a[0][3]);
    hexBDcurr = U(8)*a[0][0];
    hexBCcurr = U(8)*(a[1][0]+a[2][0]);
    hexSCcurr = U(8)*(a[0][1]+a[1][1]+a[2][1]+a[3][1]);
    /* Sector sắp tới */
    hexBIfcom = U(9)*(a[1][0]+a[2][0]);
    hexFIfcom = U(9)*a[3][0];
    hexFCfcom = U(9)*(a[3][2]+a[3][3]);
    hexFDfcom = U(9)*(a[2][2]+a[2][3]+a[1][2]+a[1][3]);
    hexBDfcom = U(9)*(a[0][2]+a[0][3]);
    hexBCfcom = U(9)*a[0][0];
    hexSCfcom = U(9)*(a[0][1]+a[1][1]+a[2][1]+a[3][1]);

    V[1]  = hexBIcurr;
    V[2]  = hexFIcurr;
    V[3]  = hexFCcurr;
    V[4]  = hexFDcurr;
    V[5]  = hexBDcurr;
    V[6]  = hexBCcurr;
    V[7]  = hexSCcurr;
//=======================
    V[8]  = hexBIfcom;
    V[9]  = hexFIfcom;
    V[10] = hexFCfcom;
    V[11] = hexFDfcom;
    V[12] = hexBDfcom;
    V[13] = hexBCfcom;
    V[14] = hexSCfcom;
//=======================
    V[15] = hexBIbypast;
    V[16] = hexFIbypast;
    V[17] = hexFCbypast;
    V[18] = hexFDbypast;
    V[19] = hexBDbypast;
    V[20] = hexBCbypast;
    V[21] = hexSCbypast;
  }
//=======================

  for(i=1;i<8;i++)
  { V_Idxbypast[i]=i;
    V_Idxcurr[i]=i;
    V_Idxfcom[i]=i;
  }
```

```
    for(i=1;i<8;i++)
    { V_Idx[i]=V_Idxcurr[i];
      S_Idx[i]=U(11);
    }
    for(i=8;i<15;i++)
    { V_Idx[i]=V_Idxfcom[i-7];
      S_Idx[i]=U(12);
    }
    for(i=15;i<22;i++)
    { V_Idx[i]=V_Idxbypast[i-14];
      S_Idx[i]=U(13);
    }
    for(i=1;i<4;i++)
      for(j=i+1;j<22;j++)
      { if(V[i]<V[j])
        { V[0]=V[i]; V[i]=V[j]; V[j]=V[0];
          V_Idx[0]=V_Idx[i];
          V_Idx[i]=V_Idx[j];
          V_Idx[j]=V_Idx[0];
          S_Idx[0]=S_Idx[i];
          S_Idx[i]=S_Idx[j];
          S_Idx[j]=S_Idx[0];
        }
      }
    y[0]=V[1];              // Mức thỏa mãn cao nhất
    y[1]=V_Idx[1];          // Vector có mức thỏa mãn cao nhất
    y[2]=S_Idx[1];          // Sector của vector có mức thỏa mãn
                            // cao nhất
    y[3]=V[2];              // Mức thỏa mãn cao thứ 2
    y[4]=V_Idx[2];          // Vector có mức thỏa mãn cao thứ 2
    y[5]=S_Idx[2];          // Sector của vector có mức thỏa mãn
                            // cao thứ 2
    y[6]=V[3];              // Mức thỏa mãn cao thứ 3
    y[7]=V_Idx[3];          // Vector có mức thỏa mãn cao thứ 3
    y[8]=S_Idx[3];          // Sector của vector có mức thỏa mãn
                            // cao thứ 3
    y[9]=choice;            // Quỹ đạo được chọn
} / *end mdlOutputs */

/* Function: mdlTerminate */
static void mdlTerminate(SimStruct *S)
{ } /* end mdlTerminate */
#ifdef   MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

Khối Defuzzyfication (giải mờ)

Khối có 10 đầu vào và 12 đầu ra. Trong đó đầu vào chính là đầu ra của khối *Inference* (luật hợp thành) vừa đề cập đến ở trên. 12 đầu ra bao gồm (theo số thứ tự ở đầu ra của khối):

1	Thời gian thực hiện vector có mức thỏa mãn lớn nhất
2, 3, 4	Trạng thái đóng ngắt S_a, S_b, S_c của vector có mức thỏa mãn lớn nhất
5	Thời gian thực hiện vector có mức thỏa mãn lớn thứ 2
6, 7, 8	Trạng thái đóng ngắt S_a, S_b, S_c của vector có mức thỏa mãn lớn thứ 2
9	Thời gian thực hiện vector có mức thỏa mãn lớn thứ 3 (=0 khi chọn quỹ đạo lục giác)
10, 11, 12	Trạng thái đóng ngắt S_a, S_b, S_c của vector có mức thỏa mãn lớn thứ 3

Tùy theo từng Sector mà các vector điện áp chuẩn (ứng với các trạng thái đóng ngắt chuẩn) có tác dụng khác nhau. Do đó, cũng tùy theo Sector mà các vector điện áp ngôn ngữ sẽ sử dụng các vector điện áp chuẩn (trạng thái đóng ngắt) khác nhau. Trạng thái đóng ngắt của các vector điện áp ngôn ngữ đối với hai quỹ đạo được tập hợp trong hai bảng dưới đây:

- *Quỹ đạo tròn*

Vector ngôn ngữ \ Sector	S_1	S_2	S_3	S_4	S_5	S_6
SI	100	110	010	011	001	101
FI	110	010	011	001	101	100
FD	010	011	001	101	100	110
SD	011	001	101	100	110	010
BD	001	101	100	110	010	011
BI	101	100	110	010	011	001
SC	111	111	111	111	111	111

- *Quỹ đạo lục giác*

Vector ngôn ngữ \ Sector	S_6	S_1	S_2	S_3	S_4	S_5
BI	100	110	010	011	001	101
FI	110	010	011	001	101	100
FC	010	011	001	101	100	110
FD	011	001	101	100	110	010
BD	001	101	100	110	010	011
BC	101	100	110	010	011	001
SC	111	111	111	111	111	111

Nếu xem kỹ hai bảng trên ta dễ dàng rút ra nhận xét: Cả hai bảng có nội dung với các tổ hợp giá trị 0 hay 1 (ngắt / đóng) giống hệt nhau nhưng lại ứng với các Sector khác nhau. Ví dụ: Cột đầu tiên của quỹ đạo tròn thuộc về Sector S_1, nhưng của quỹ đạo lục giác lại ứng với Sector S_6. Có thể hiểu được điều này nếu ta biết rằng: Các Sector của quỹ đạo tròn được định nghĩa trên hệ tọa độ Stator quen biết (hình 12.23a), còn của quỹ đạo lục giác lại dựa trên hệ tọa độ mới định nghĩa (hình 12.23b).

Nhiệm vụ của khối *Defuzzyfication* là:

- Dựa trên sự lựa chọn quỹ đạo (tròn hay lục giác), trên các vector điện áp ngôn ngữ và thông tin về Sector để *tra trạng thái đóng ngắt 3 nhánh van NL* từ một trong hai bảng trên.
- *Tính thời gian đóng ngắt* (thời gian duy trì trạng thái đóng ngắt tra từ bảng) theo các công thức (12.46) hoặc (12.47).

Khối *Defuzzyfication* được thực hiện bằng hàm S có tên là `pdefuz3` (hình 12.44) với mã *m-File* dưới đây.

```
function [sys,x0,str,ts] = ...
                              pdefuz3(t,x,u,flag,Tx,chquay)
switch flag
case 0
  [sys,x0,str,ts]=mdlInitializeSizes;
case 3
  sys = mdlOutputs(t,x,u,Tx,chquay);
case {1,2,4,9}
  sys=[];
otherwise
  error(['Unhandled flag=',num2str(flag)]);
end

function [sys,x0,str,ts] = mdlInitializeSizes
  sizes=simsizes;
  sizes.NumContStates=0;
  sizes.NumDiscStates=0;
  sizes.NumOutputs=12;
              %T1,Su1,Sv1,Sw1,T2,Su2,Sv2,Sw2,T3,Su3,Sv3,Sw3
  sizes.NumInputs=10;
% u(1),u(2),u(3): mức thuộc max, vector max (1-7),Sht,
% u(4),u(5),u(6): mức thuộc thứ 2, vector 2 (1-7),Sst,
% u(7),u(8),u(9): mức thuộc thứ 3, vector 3 (1-7),Sdq
% u(10):          biến choice
% Ý nghĩa giá trị của các đầu vào u(2), u(5), u(8):
% Chỉ số 1 tương ứng với SI(tròn), BI(lục giác)
% Chỉ số 2 tương ứng với FI(tròn), FI(lục giác)
% Chỉ số 3 tương ứng với FD(tròn), FC(lục giác)
% Chỉ số 4 tương ứng với SD(tròn), FD(lục giác)
```

```
% Chỉ số 5 tương ứng với BD(tròn), BD(lục giác)
% Chỉ số 6 tương ứng với BI(tròn), BC(lục giác)
% Chỉ số 7 tương ứng với SC(tròn), SC(lục giác)
  sizes.DirFeedthrough=0;
  sizes.NumSampleTimes=1;
  sys=simsizes(sizes);
  x0=[];
  str=[];
  ts=[-1 0];

function sys = mdlOutputs(t,x,u,Tx,chquay)
% Bảng xác định trạng thái đóng ngắt van ứng với sector %
ar=[1 0 0 1 1 0 0 1 0 0 1 1 0 0 1 1 0 1   %SI(tròn);BI(lục giác)
    1 1 0 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0   %FI(tròn);FI(lục giác)
    0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0   %FD(tròn);FC(lục giác)
    0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 0 1 0   %SD(tròn);FD(lục giác)
    0 0 1 1 0 1 1 0 0 1 1 0 0 1 0 0 1 1   %BD(tròn);BD(lục giác)
    1 0 1 1 0 0 1 1 0 0 1 0 0 1 1 0 0 1   %BI(tròn);BC(lục giác)
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]; %SC(tròn);SC(lục giác)
ar1=ar;                    % ar: tròn; ar1: lục giác
%%%%%%%%%%%%%%%%%%%%%%
%    Quỹ đạo tròn    %
%%%%%%%%%%%%%%%%%%%%%%
if u(10)==1                % choice
   tthm=u(1)+u(4)+u(7);
   sys(1)=Tx*u(1)/tthm;        % Thời gian thực hiện u(2)
   sys(5)=Tx*u(4)/tthm;        % Thời gian thực hiện u(5)
   sys(9)=Tx*u(7)/tthm;        % Thời gian thực hiện u(8)
%-------------------------------------------------------%
% Sector hiện tại chứa vector có mức thuộc max: u(2) %
  if u(3)==1               % Sector hiện tại là S1
     sys(2)=ar(u(2),1);        % Pha U: hàng u(2), cột 1
     sys(3)=ar(u(2),2);        % Pha V: hàng u(2), cột 2
     sys(4)=ar(u(2),3);        % Pha W: hàng u(2), cột 3
  elseif u(3)==2           % Sector hiện tại là S2
     sys(2)=ar(u(2),4);        % Pha U: hàng u(2), cột 4
     sys(3)=ar(u(2),5);        % Pha V: hàng u(2), cột 5
     sys(4)=ar(u(2),6);        % Pha W: hàng u(2), cột 6
  elseif u(3)==3           % Sector hiện tại là S3
     sys(2)=ar(u(2),7);        % Pha U: hàng u(2), cột 7
     sys(3)=ar(u(2),8);        % Pha V: hàng u(2), cột 8
     sys(4)=ar(u(2),9);        % Pha W: hàng u(2), cột 9
  elseif u(3)==4           % Sector hiện tại là S4
     sys(2)=ar(u(2),10);       % Pha U: hàng u(2), cột 10
     sys(3)=ar(u(2),11);       % Pha V: hàng u(2), cột 11
     sys(4)=ar(u(2),12);       % Pha W: hàng u(2), cột 12
```

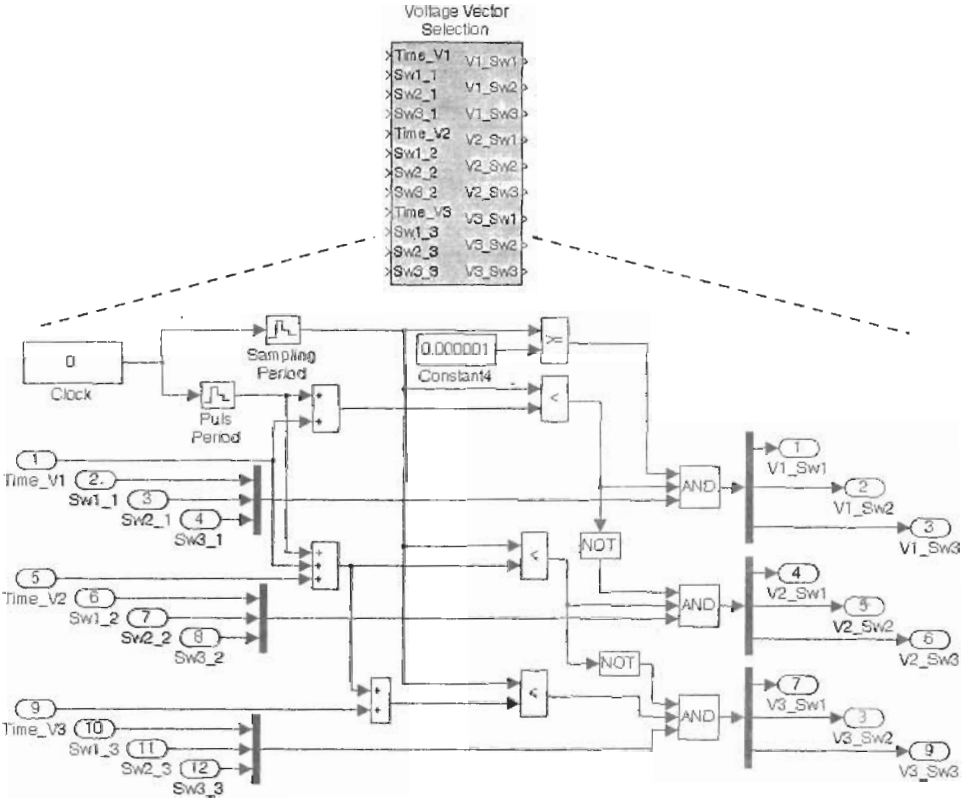
```
  elseif u(3)==5          % Sector hiện tại là S5
     sys(2)=ar(u(2),13);      % Pha U: hàng u(2), cột 13
     sys(3)=ar(u(2),14);      % Pha V: hàng u(2), cột 14
     sys(4)=ar(u(2),15);      % Pha W: hàng u(2), cột 15
  elseif u(3)==6          % Sector hiện tại là S6
     sys(2)=ar(u(2),16);      % Pha U: hàng u(2), cột 16
     sys(3)=ar(u(2),17);      % Pha V: hàng u(2), cột 17
     sys(4)=ar(u(2),18);      % Pha W: hàng u(2), cột 18
  else
     sys(2)=0;
     sys(3)=0;
     sys(4)=0;
  end
%-------------------------------------------------------%
% Sector sắp tới chứa vector có mức thuộc thứ 2: u(5) %
  if u(6)==1              % Sector sắp tới là S1
     sys(6)=ar(u(5),1);       % Pha U: hàng u(5), cột 1
     sys(7)=ar(u(5),2);       % Pha V: hàng u(5), cột 2
     sys(8)=ar(u(5),3);       % Pha W: hàng u(5), cột 3
  elseif u(6)==2          % Sector sắp tới là S2
     sys(6)=ar(u(5),4);       % Pha U: hàng u(5), cột 4
     sys(7)=ar(u(5),5);       % Pha V: hàng u(5), cột 5
     sys(8)=ar(u(5),6);       % Pha W: hàng u(5), cột 6
  elseif u(6)==3          % Sector sắp tới là S3
     sys(6)=ar(u(5),7);       % Pha U: hàng u(5), cột 7
     sys(7)=ar(u(5),8);       % Pha V: hàng u(5), cột 8
     sys(8)=ar(u(5),9);       % Pha W: hàng u(5), cột 9
  elseif u(6)==4          % Sector sắp tới là S4
     sys(6)=ar(u(5),10);      % Pha U: hàng u(5), cột 10
     sys(7)=ar(u(5),11);      % Pha V: hàng u(5), cột 11
     sys(8)=ar(u(5),12);      % Pha W: hàng u(5), cột 12
  elseif u(6)==5          % Sector sắp tới là S5
     sys(6)=ar(u(5),13);      % Pha U: hàng u(5), cột 13
     sys(7)=ar(u(5),14);      % Pha V: hàng u(5), cột 14
     sys(8)=ar(u(5),15);      % Pha W: hàng u(5), cột 15
  elseif u(6)==6          % Sector sắp tới là S6
     sys(6)=ar(u(5),16);      % Pha U: hàng u(5), cột 16
     sys(7)=ar(u(5),17);      % Pha V: hàng u(5), cột 17
     sys(8)=ar(u(5),18);      % Pha W: hàng u(5), cột 18
  else
     sys(6)=0;
     sys(7)=0;
     sys(8)=0;
  end
%-------------------------------------------------------%
% Sector đã qua chứa vector có mức thuộc thứ 3: u(8) %
```

```
   if u(9)==1              % Sector đã qua là S1
      sys(10)=ar(u(8),1);      % Pha U: hàng u(8), cột 1
      sys(11)=ar(u(8),2);      % Pha V: hàng u(8), cột 2
      sys(12)=ar(u(8),3);      % Pha W: hàng u(8), cột 3
   elseif u(9)==2          % Sector đã qua là S2
      sys(10)=ar(u(8),4);      % Pha U: hàng u(8), cột 4
      sys(11)=ar(u(8),5);      % Pha V: hàng u(8), cột 5
      sys(12)=ar(u(8),6);      % Pha W: hàng u(8), cột 6
   elseif u(9)==3          % Sector đã qua là S3
      sys(10)=ar(u(8),7);      % Pha U: hàng u(8), cột 7
      sys(11)=ar(u(8),8);      % Pha V: hàng u(8), cột 8
      sys(12)=ar(u(8),9);      % Pha W: hàng u(8), cột 9
   elseif u(9)==4          % Sector đã qua là S4
      sys(10)=ar(u(8),10);     % Pha U: hàng u(8), cột 10
      sys(11)=ar(u(8),11);     % Pha V: hàng u(8), cột 11
      sys(12)=ar(u(8),12);     % Pha W: hàng u(8), cột 12
   elseif u(9)==5          % Sector đã qua là S5
      sys(10)=ar(u(8),13);     % Pha U: hàng u(8), cột 13
      sys(11)=ar(u(8),14);     % Pha V: hàng u(8), cột 14
      sys(12)=ar(u(8),15);     % Pha W: hàng u(8), cột 15
   elseif u(9)==6          % Sector đã qua là S6
      sys(10)=ar(u(8),16);     % Pha U: hàng u(8), cột 16
      sys(11)=ar(u(8),17);     % Pha V: hàng u(8), cột 17
      sys(12)=ar(u(8),18);     % Pha W: hàng u(8), cột 18
   else
      sys(10)=0;
      sys(11)=0;
      sys(12)=0;
   end
%%%%%%%%%%%%%%%%%%%%%%
%  Quỹ đạo lục giác  %
%%%%%%%%%%%%%%%%%%%%%%
elseif u(10)==2            %choice
% Quỹ đạo lục giác: Không sử dụng u(8) %
   sys(9)=0;                % Thời gian thực hiện u(8)
   sys(10)=0;               % Pha U: logic 0
   sys(11)=0;               % Pha V: logic 0
   sys(12)=0;               % Pha W: logic 0
%-------------------------------------------------------%
   tthm=u(1)+u(4);
   sys(1)=Tx*u(1)/tthm;     % Thời gian thực hiện u(2)
   sys(5)=Tx-sys(1);        % Thời gian thực hiện u(5)
%-------------------------------------------------------%
% Sector hiện tại chứa vector có mức thuộc max: u(2) %
  if u(3)==1               % Sector hiện tại là S1
     sys(2)=ar1(u(2),4);      % Pha U: hàng u(2), cột 4
```

```
      sys(3)=arl(u(2),5);       % Pha V: hàng u(2), cột 5
      sys(4)=arl(u(2),6);       % Pha W: hàng u(2), cột 6
   elseif u(3)==2           % Sector hiện tại là S2
      sys(2)=arl(u(2),7);       % Pha U: hàng u(2), cột 7
      sys(3)=arl(u(2),8);       % Pha V: hàng u(2), cột 8
      sys(4)=arl(u(2),9);       % Pha W: hàng u(2), cột 9
   elseif u(3)==3           % Sector hiện tại là S3
      sys(2)=arl(u(2),10);      % Pha U: hàng u(2), cột 10
      sys(3)=arl(u(2),11);      % Pha V: hàng u(2), cột 11
      sys(4)=arl(u(2),12);      % Pha W: hàng u(2), cột 12
   elseif u(3)==4           % Sector hiện tại là S4
      sys(2)=arl(u(2),13);      % Pha U: hàng u(2), cột 13
      sys(3)=arl(u(2),14);      % Pha V: hàng u(2), cột 14
      sys(4)=arl(u(2),15);      % Pha W: hàng u(2), cột 15
   elseif u(3)==5           % Sector hiện tại là S5
      sys(2)=arl(u(2),16);      % Pha U: hàng u(2), cột 16
      sys(3)=arl(u(2),17);      % Pha V: hàng u(2), cột 17
      sys(4)=arl(u(2),18);      % Pha W: hàng u(2), cột 18
   elseif u(3)==6           % Sector hiện tại là S6 .
      sys(2)=arl(u(2),1);       % Pha U: hàng u(2), cột 1
      sys(3)=arl(u(2),2);       % Pha V: hàng u(2), cột 2
      sys(4)=arl(u(2),3);       % Pha W: hàng u(2), cột 3
   else
      sys(2)=0;
      sys(3)=0;
      sys(4)=0;
   end
%-------------------------------------------------------%
% Sector sắp tới chứa vector có mức thuộc thứ 2: u(5) %
   if u(6)==1               % Sector sắp tới là S1
      sys(6)=arl(u(5),4);       % Pha U: hàng u(5), cột 4
      sys(7)=arl(u(5),5);       % Pha V: hàng u(5), cột 5
      sys(8)=arl(u(5),6);       % Pha W: hàng u(5), cột 6
   elseif u(6)==2           % Sector sắp tới là S2
      sys(6)=arl(u(5),7);       % Pha U: hàng u(5), cột 7
      sys(7)=arl(u(5),8);       % Pha V: hàng u(5), cột 8
      sys(8)=arl(u(5),9);       % Pha W: hàng u(5), cột 9
   elseif u(6)==3           % Sector sắp tới là S3
      sys(6)=arl(u(5),10);      % Pha U: hàng u(5), cột 10
      sys(7)=arl(u(5),11);      % Pha V: hàng u(5), cột 11
      sys(8)=arl(u(5),12);      % Pha W: hàng u(5), cột 12
   elseif u(6)==4           % Sector sắp tới là S4
      sys(6)=arl(u(5),13);      % Pha U: hàng u(5), cột 13
      sys(7)=arl(u(5),14);      % Pha V: hàng u(5), cột 14
      sys(8)=arl(u(5),15);      % Pha W: hàng u(5), cột 15
   elseif u(6)==5           % Sector sắp tới là S5
```

```
        sys(6)=arl(u(5),16);      % Pha U: hàng u(5), cột 16
        sys(7)=arl(u(5),17);      % Pha V: hàng u(5), cột 17
        sys(8)=arl(u(5),18);      % Pha W: hàng u(5), cột 18
    elseif u(6)==6          % Sector sắp tới là S6
        sys(6)=arl(u(5),1);       % Pha U: hàng u(5), cột 1
        sys(7)=arl(u(5),2);       % Pha V: hàng u(5), cột 2
        sys(8)=arl(u(5),3);       % Pha W: hàng u(5), cột 3
    else
        sys(6)=0;
        sys(7)=0;
        sys(8)=0;
    end
end
```

Khối Voltage Vector Selection (thực hiện vector điện áp)



Hình 12.46 Khối Voltage Vector Selection và sơ đồ SIMULINK chi tiết

Để dễ hiểu chức năng của khối *Voltage* ***Vector*** *Selection*, ta tóm tắt lại hoạt động **của** khối ***Defuzzyfication***, là khối tạo **ra 12 tín** hiệu đầu vào **của** *Voltage*

Vector Selection. Khối *Defuzzyfication* đã tùy theo quỹ đạo được chọn là hình tròn hay hình lục giác, tức là `u(10) = 1` hay 2:

- *Đối với quỹ đạo tròn (điều chế 3 vector)*: Tính 3 thời gian thực hiện vector có mức thuộc cao nhất là `u(2)`, vector có mức thuộc cao thứ 2 là `u(5)` và vector có mức thuộc cao thứ 3 là `u(8)`. 3 giá trị thời gian đó được đưa tới các đầu ra số 1, 5 và 9. *Đối với quỹ đạo lục giác (điều chế 2 vector)*: Không thực hiện `u(8)`, thời gian đưa tới đầu ra số 9 sẽ là 0.
- *Để thực hiện vector có mức thuộc cao nhất là* `u(2)`, ta kiểm tra giá trị của `u(3)` để biết `u(2)` nằm ở Sector nào (S1, hay S2, ... hay S6). Từ đó tra hàng thứ `u(2)` và cột tương ứng của bảng `ar` để tìm trạng thái logic (1 hoặc 0; đóng lên nguồn + hay xuống nguồn -) của pha U, V và W. Ba giá trị logic của pha U, V và W được đưa tới đầu ra 2, 3 và 4.
- *Để thực hiện vector có mức thuộc cao thứ 2 là* `u(5)`, ta kiểm tra giá trị của `u(6)` để biết `u(5)` nằm ở Sector nào (S1, hay S2, ... hay S6). Từ đó tra hàng thứ `u(5)` và cột tương ứng của bảng `ar` để tìm trạng thái logic (1 hoặc 0; đóng lên nguồn + hay xuống nguồn -) của pha U, V và W. Ba giá trị logic của pha U, V và W được đưa tới đầu ra 6, 7 và 8.
- *Để thực hiện vector có mức thuộc cao thứ 3 là* `u(8)`, ta kiểm tra giá trị của `u(9)` để biết `u(8)` nằm ở Sector nào (S1, hay S2, ... hay S6). Từ đó tra hàng thứ `u(8)` và cột tương ứng của bảng `ar` để tìm trạng thái logic (1 hoặc 0; đóng lên nguồn + hay xuống nguồn -) của pha U, V và W. Ba giá trị logic của pha U, V và W được đưa tới đầu ra 10, 11 và 12. Khi quỹ đạo được chọn là lục giác, ba giá trị logic đưa tới đầu ra đều là 0 (cả 3 pha U, V và W đều được nối với nguồn -).

Như vậy, 12 đầu ra của khối *Defuzzyfication* được chia thành 3 nhóm ứng với 3 vector `u(2)`, `u(5)` và `u(8)` (của quỹ đạo tròn). Khi quỹ đạo là lục giác ta chỉ cần 2 nhóm đầu ứng với 2 vector `u(2)` và `u(5)`, nhóm thứ 3 của `u(8)` không được sử dụng. Mỗi nhóm có 4 đầu ra với chức năng:

- *Đầu ra thứ nhất* (số thứ tự 1, 5 và 9): Giá trị thời gian thực hiện tổ hợp logic U, V và W.
- *Đầu ra thứ 2* (số thứ tự 2, 6 và 10): Giá trị logic của pha U.
- *Đầu ra thứ 3* (số thứ tự 3, 7 và 11): Giá trị logic của pha V.
- *Đầu ra thứ 4* (số thứ tự 4, 8 và 12): Giá trị logic của pha W.

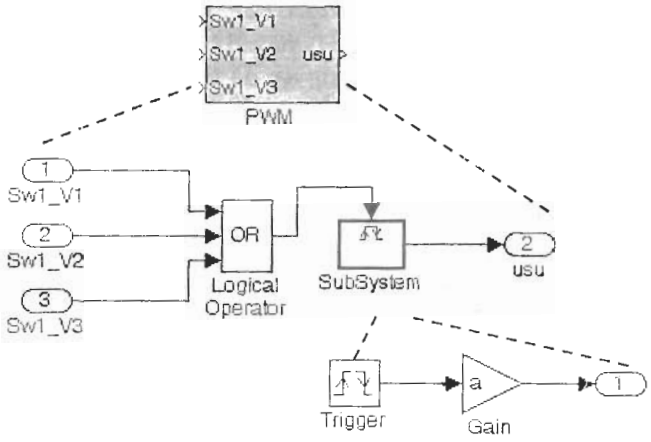
Vậy nhiệm vụ của khối *Voltage Vector Selection* (với đầu vào chính là đầu ra của khối *Defuzzyfication*) là kết hợp thế nào đó, để lần lượt đưa các tổ hợp trạng thái logic ở đầu vào tới 3 đầu ra duy nhất ứng với ba pha U, V và W:

- Tổ hợp đầu vào số 2/3/4, duy trì trong khoảng thời gian ở đầu vào 1.
- Tổ hợp đầu vào số 6/7/8, duy trì trong khoảng thời gian ở đầu vào 5.
- Tổ hợp đầu vào số 10/11/12, duy trì trong khoảng thời gian ở đầu vào 9.

Mặc dù có thể thực hiện khối bằng C-mex-File, ví dụ ở hình 12.46 đã sử dụng một cách rất thuận lợi các khối logic có sẵn trong thư viện của SIMULINK.

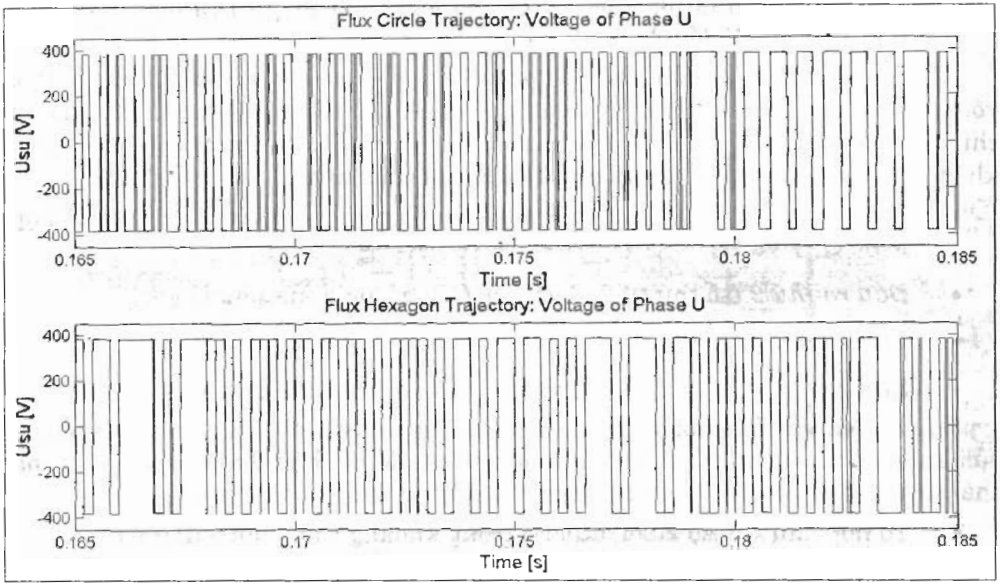
Khối PWM (điều chế bề rộng xung)

Mỗi pha U, V và W đều có một khối PWM của riêng mình. Khối PWM không có chức năng điều chế bề rộng xung như ta thường quen. Tại đây khối PWM chỉ duy nhất có nhiệm vụ kết hợp 3 giá trị logic (do 3 vector với ba mức thuộc cung cấp) để ĐK đầu ra của pha:



- Nhận giá trị $+(2/3)U_{DC}$ khi giá trị logic là 1 và
- nhận giá trị $-(2/3)U_{DC}$ khi giá trị logic là 0.

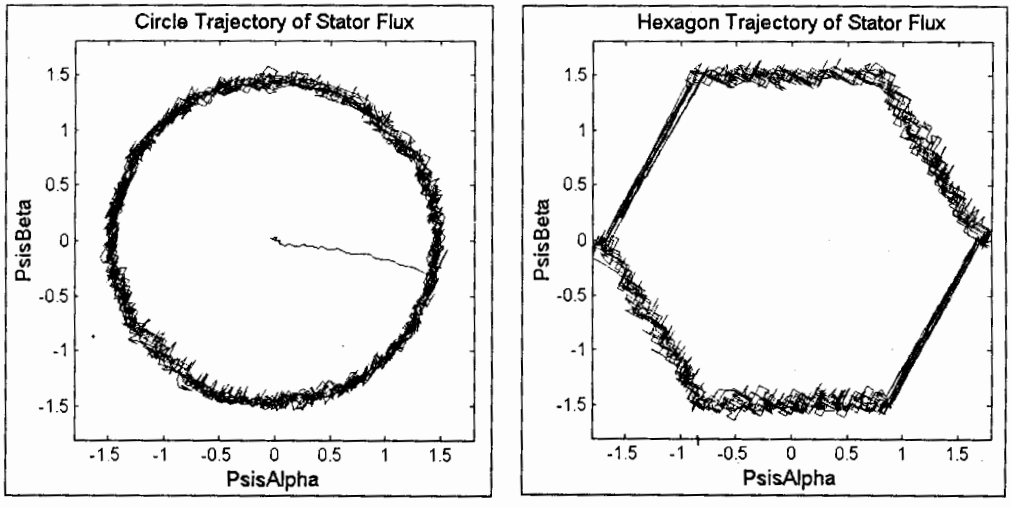
***Hình 12.47** Khối PWM và sơ đồ SIMULINK chi tiết*



***Hình 12.48** Điện áp pha U ở đầu ra khối PWM: quỹ đạo tròn (trên), lục giác (dưới)*

Hình 12.48 là kết quả mô phỏng dạng điện áp đưa tới pha U của MĐDB trong hai trường hợp: Quỹ đạo hình tròn và hình lục giác. Có thể nhận xét một cách rất chính xác rằng đó là kết quả đáng tin cậy. Trong cả hai hình thuộc 12.48, bạn đọc bắt gặp lại dạng điện áp (chuỗi xung vuông có bề rộng thay đổi) với sóng cơ bản hình sin, thường gặp khi sử dụng PWM. Tuy nhiên, như ở phần mở đầu mục 12.2.3 đã nói: Đây là khâu NPWM (Nonlinear Puls Width Modulator), tức là khâu điều chế phi tuyến bề rộng xung chứ không phải khâu điều chế vector thông thường.

Hình 12.49 dưới đây khẳng định lại một lần nữa tính đúng đắn của mô hình SIMULINK (hình 12.40), mô phỏng hệ thống truyền động điện không đồng bộ xoay chiều ba pha theo nguyên lý DTC và sử dụng *Fuzzy Logic* để số hóa.



Hình 12.49 *Quỹ đạo hình tròn (trái) và hình lục giác (phải) của từ thông Stator* ψ_s

12.3 Tóm tắt nội dung chương 12

Sau khi đã nghiên cứu kỹ chương 12, người đọc cần nắm vững các nội dung sau đây:

1. Mô hình hóa trên nền MATLAB và SIMULINK các phần tử truyền động cơ học trên cơ sở phân rã hệ thống cơ nhiều vật thành nhiều hệ một vật.
2. Mô hình hóa và mô phỏng một trục chuyển động (ví dụ: Một trục của tay máy) bao gồm cả ba loại phần tử: Thiết bị điều khiển (thuật toán và điện tử công suất), máy điện và phụ tải (hệ thống cơ nhiều vật).
3. Mô hình hóa và mô phỏng thiết bị phát điện chạy sức gió sử dụng máy điện dị bộ nguồn kép, bao gồm các hệ thống con: Máy phát và hệ thống điều khiển máy phát, lưới / nghịch lưu phía lưới và hệ thống điều khiển

nghịch lưu phía lưới, quá trình trao đổi điện năng giữa hai phía qua mạch điện một chiều trung gian.

4. Mô hình hóa và mô phỏng hệ truyền động dị bộ theo phương pháp *Direct Torque Control* sử dụng *Fuzzy Logic*. Trong đó có các nội dung mới: Khâu nhận dạng mờ góc phần sáu (*Sector*), khâu điều chỉnh mờ (*Fuzzy Controller*, thực hiện các nhiệm vụ: Mờ hóa, luật hợp thành, giải mờ, tạo vector điện áp, điều chế bề rộng xung).
5. Kỹ năng tạo hàm S (tạo *S-Function*) bằng *m-File* hay *C-mex-File* để tạo ra những khối SIMULINK mới.

13 Tài liệu tham khảo

13.1 Tài liệu tham khảo chung

[1] Angermann, A.; Beuschel, M.; Rau, M.; Wohlfarth, U.: *Simulation mit SIMULINK/MATLAB: Skriptum mit Übungsaufgaben*. Stand: 29. November 2001, TU München[1]

[2] Biran, A.; Breiner, M.: *MATLAB für Ingenieure: Systematische und praktische Einführung*. Addison-Wesley, 1995

[3] Bishop, R. H.: *Modern control systems analysis and design using MATLAB*. Addison-Wesley, 1993

[4] Bode, H.: *MATLAB in der Regelungstechnik: Analyse linearer Systeme*. B.G. Teubner Stuttgart-Leipzig, 1998

[5] Gander, W.; Hřebíček, J.: *Solving problems in scientific computing using Maple and MATLAB*. Springer, 1993

[6] Hoffmann, J.: *MATLAB und SIMULINK: Beispielorientierte Einführung in die Simulation dynamischer Systeme*. Addison-Wesley, 1998

[7] Kamen, E. W.; Heck B. S.: *Fundamentals of signals and systems using MATLAB*. Prentice Hall Inc., 1997

[8] Krause, M.: *Fuzzy-Regelung eines Traktionsantriebes im ständerfesten Koordinatensystem*. Dissertation 1995, TU Dresden

[9] Meyer, J.: *Simulation dynamischer Systeme unter Echtzeitbedingungen*. Abschlußbericht, Dezember 1997, TU Dresden[2]

[10] Nguyễn Phùng Quang: *Máy điện dị bộ nguồn kép dùng làm máy phát trong hệ thống phát điện chạy sức gió: Các thuật toán điều chỉnh bảo đảm phân ly giữa mômen và hệ số công suất*. The 3rd Vietnam Conference on Automation (3rd VICA), Hà Nội, 4/1998, tr. 413-437

[11] Nguyễn Phùng Quang: *Lôgich mờ trong điều khiển động cơ dị bộ rotor lồng sóc nuôi bởi nghịch lưu nguồn áp. Phần 1: Mô hình động cơ và phương pháp tự chỉnh (Direct Self-Control)*. Tạp chí Tự động hóa ngày nay, số 6(11)/2001, tr. 17 – 22.

[12] Nguyễn Phùng Quang: *Lôgich mờ trong điều khiển động cơ dị bộ rotor lồng sóc nuôi bởi nghịch lưu nguồn áp. Phần 2: Khâu điều chỉnh mờ trên*

[1] Bài giảng của TU München: Tài liệu tham khảo chính

[2] Báo cáo đề tài của TU Dresden: Tài liệu tham khảo chính

cơ sở phương pháp DSC. Tạp chí Tự động hóa ngày nay, số 7(12)/2001, tr. 35 – 41

[13] Nguyễn Phùng Quang: *Khái quát về nguyên lý và cấu trúc điều khiển động cơ từ kháng loại đóng ngắt (Switched Reluctance Motors)*. Tạp chí Tự động hóa ngày nay, số 1(30)/2003.

[14] Nguyễn Phùng Quang: *Động cơ từ kháng và triển vọng ứng dụng trong các hệ thống Mechatronics*. Proceedings of the 1st National Conference on Mechatronics, Technopark Hoa Lac, Hanoi 09/2002, pp. 354 – 365

[15] Nguyễn Phùng Quang, Chu Đức Việt, Lưu Hồng Việt, Trần thị Thanh Yên, Chu Đình Đức, Ngô Quốc Tùng: *Toolbox Motion Control: Tool for Motion Control Research based on MATLAB & Simulink*. RESCCE'2000, Japan–USA–Vietnam Workshop on Research and Education in Systems, Computation and Control Engineering, Ho Chi Minh City, 7-9 June 2000

[16] Nguyễn Phùng Quang; Dittrich, A.: *Truyền động điện thông minh*. Nhà xuất bản Khoa học và Kỹ thuật, 2002

[17] Ong, Chee-M.: *Dynamic simulation of electric machinery using MATLAB / SIMULINK*. Prentice Hall PTR, 1998

[18] Redfern, D.; Campbell, C.: *The MATLAB 5 Handbook*. Springer, 1997

[19] Shahian, B.; Hassul, M.: *Control system design using MATLAB*. Prentice Hall, 1993

13.2 Các tài liệu dưới dạng .pdf-Files

[20] The MATHWORK Inc.: ...\help\pdf_doc\control*usingcontrol.pdf*

[21] The MATHWORK Inc.: ...\help\pdf_doc\matlab*graphg.pdf*

[22] The MATHWORK Inc.: ...\help\pdf_doc\matlab\gui*buildgui.pdf*

[23] The MATHWORK Inc.: ...\help\pdf_doc\matlab*using_ml.pdf*

[24] The MATHWORK Inc.: ...\help\pdf_doc\signal*signal_tb.pdf*

[25] The MATHWORK Inc.: ...\help\pdf_doc\simulink*sfunctions.pdf*

[26] The MATHWORK Inc.: ...\help\pdf_doc\simulink*sl_using.pdf*

[27] The MATHWORK Inc.: ...\help\pdf_doc\stateflow*sf_ug.pdf*

13.3 Các luận văn, đồ án tốt nghiệp tại ĐHBK Hà Nội đã góp phần xây dựng Motion Control Blockset

[28] Cao Bảo Tuấn, Trần Ngọc Sơn: *Đề xuất phương án điều chỉnh hệ thống phát điện chạy sức gió và sử dụng máy điện đồng bộ kích thích vĩnh cửu*. Đồ án tốt nghiệp K41, ĐHBK Hà Nội, 5/2001

[29] Đoàn Quảng Trị, Nguyễn Văn Quyết: *Bổ xung Toolbox Motion Control: thực hiện khâu quan sát từ thông Rotor của động cơ dị bộ ba pha, khi có*

và không có đo tốc độ quay Rotor. Đồ án tốt nghiệp K41, ĐHBK Hà Nội, 5/2001

[30] Đỗ Mạnh Cường: *Mô hình hóa và thiết kế hệ thống điều khiển động cơ từ kháng loại đóng ngắt*. Luận văn Cao học, ĐHBK Hà Nội, 12/2002

[31] Nguyễn Đắc Trung, Mai Văn Hải: *Xây dựng mô hình trên nền MATLAB + Simulink, mô phỏng hệ thống phát điện chạy sức gió và sử dụng máy điện dị bộ nguồn kép*. Đồ án tốt nghiệp K41, ĐHBK Hà Nội, 5/2001

[32] Nguyễn Thanh Tùng: *Điều khiển hòa đồng bộ cho máy phát điện chạy sức gió sử dụng máy điện dị bộ nguồn kép*. Đồ án tốt nghiệp K42, ĐHBK Hà Nội, 5/2002

[33] Nguyễn thị Hồng Hạnh, Ngô Đức Tuân: *Xây dựng mô hình và đề xuất cấu trúc điều khiển động cơ tuyến tính kiểu đồng bộ kích thích vĩnh cửu*. Đồ án tốt nghiệp K42, ĐHBK Hà Nội, 5/2002

[34] Nguyễn Thị Mai Phương, Nguyễn Ngọc Hân, Đinh Phương Triệu: *Xây dựng hệ thống điều khiển động cơ không đồng bộ ba pha theo nguyên lý Direct Self-Control trên cơ sở lôgic mờ, kiểm chứng bằng công cụ MATLAB & Simulink*. Đồ án tốt nghiệp K43, ĐHBK Hà Nội, 5/2003

[35] Phùng Ngọc Lân: *Tổng hợp hệ thống điều khiển thiết bị phát điện chạy sức gió dùng máy điện dị bộ nguồn kép, kiểm chứng nguyên lý qua mô phỏng trên nền MATLAB & Simulink*. Luận văn Cao học, ĐHBK Hà Nội, 12/2001

[36] Trần thị Thanh Yên, Chu Đình Đức, Ngô Thanh Tùng: *Góp phần xây dựng Toolbox Motion Control, sử dụng để mô phỏng điều khiển các quá trình chuyển động trên nền MATLAB + Simulink*. Đồ án tốt nghiệp K40, ĐHBK Hà Nội, 5/2000

Danh mục từ tra cứu

A

acker 134, 137, 140
allmargin 114, 118
ans 2
append 90, 92
Array
  Cell - 12
Averaging
  phương pháp - 196
Auto Correlation 103
axis 32

B

BackgroundColor 55
barlett 198, 200
Basic Fitting-Tool 193, 194
besself 211
biến 4
  - ngôn ngữ 425
  - options 155
bình phương
  - sai phân bé nhất 177
blackman 198, 200
bmp 42
bode 37, 118, 262, 297
BODE 110, 114, 299
box 41
boxcar 198, 200
break 16, 17
butter 209, 210, 211

C

c2d 92, 95
Callback 46, 55, 243
canon 122
care 140
Cascade Control 300
caxis 39, 41
cấu trúc 10, 31
cd 30
cell 13, 96
Cell Array 12, 67, 79, 83, 112
char 96
cheby1 209, 210, 211
cheby2 209, 210, 211
Check Box 49
Chu kỳ
  - trích mẫu 94, 113
class 96, 98
clc 24
clf 32
clear 13, 14, 20
close 31, 32
colormap 39, 41
Command
  - History 2
  - Windows 1, 19, 23, 29, 36, 44, 49 243
cond 148, 149
condeig 148, 149
condest 148, 149
continue 17
Continuous
  thư viện - 213, 257
contour 39, 41
Control System Toolbox 65
conv 9, 10
copyfile 30
Correlation Functions 200
covar 103, 104
csd 205

ctranspose 8, 10
ctrb 125
ctrbf 125
Cubic 191
Current Directory Browser 3
curve fitting 153, 177, 182

D

d2c 92, 95
d2d 95
damp 99, 104
dare 140
Data Statistics 193, 194
dcgain 98, 104
Dead-Beat 412
Defuzzyfication 425, 458
det 9, 10
delay2z 78
DelayMargin 115
delete 30, 32
den 66, 82, 206
denominator 66
diary 24
diff 9, 10
Digital Signal Processing 75
dir 30
Dirac
  xung - 106
Direct Torque Control 418
DirFeedthrough 277
Discrete
  thư viện - 213, 283, 286
disp 26
Display 156
dlinmod 290, 296, 298
dlqr 141, 146
DMFrequency 115
doc 4
double 96
drmodell 90, 92
drss 90, 92
dsort 100, 104

Đ

đảo loại
  - mô hình 85
điều chế
  - bề rộng xung 464
  - vector điện áp 383, 428, 429
điều chỉnh 300
  - dòng 301, 391, 416
  - hai chiều 406, 412
  - mờ 426, 432
  - tốc độ quay 303, 391, 407
  - từ thông 407
điều khiển
  - chuyển động 319
  luật - 426
  - thuật toán 155
  - trạng thái 313
đường đẳng mức
Đồ họa
  - 2 chiều 34
  - 3 chiều 38
động cơ
  - dị bộ ba pha 178
  - một chiều 291

E

echo 23, 24
edit 45
eig 9, 10, 104
EigenValue 99
EigName 79
EigValue 79
ellip 209, 210, 211
emeas 145
emf 42
eobs 145
eps 5
eps 42
esort 100, 104
estim 135, 140
Euclid
  chuẩn toàn phương - 177, 179
eval 29
evalfr 109, 118
eye 8
exist 15

exitflag 163
Extrapolation 191
ezplot 35

F
fclose 28
feedback 90, 92
fft 195, 198, 200
fft2 200
fgoalattain 176
figure 30, 32, 59
FigureHandle 31
filt 75
filter 207, 209, 210
filtfilt 207, 209, 210
Filter
  Analog - 211
  Digital - 205
findobj 59, 60
firl 206, 209
First Order Hold 92
flag 273
fminbnd 166
fmincon 171
fminimax 176
fminsearch 168
fminunc 168
foh 93
fopen 28
for 16, 17, 112
format 23, 216
Fourier
  - Transformation 194
  Discrete - Transformation 194
fplot 35
fprintf 17, 28
Frames 45
frd 73, 74, 76, 80, 85
frddata 83
freqresp 109, 118
freqs 211
Frequency 82
freqz 206, 209, 210
fsolve 157, 161, 165
function 19, 20
Functions & Tables
  thư viện - 213
function
  windows - 196
Fuzzy Controller 444
Fuzzy Logic 425, 432
Fuzzy Rule Inference 425
Fuzzyfication 425, 445
fval 163, 167
fzero 157, 165

G
GainMargin 115
gán cực
  phương pháp - 134
Gate-turn-off-Thyristor 417
gca 36
gcbo 53
gcf 30, 32
gensig 104, 108, 109
get 24, 31, 32, 60
giao diện
  - đồ họa 43
global 4, 19
GMFrequency 115
Graphical User Interface 43
grid 33
griddata 192, 194
GUI 43, 53, 56, 59
guide 43

H
hàm 19
  - cửa sổ 196, 198
  - điều kiện 173
  - lượng giác 6
  - mục tiêu 173
  - toán 6
  - trọng lượng 106
  - tự tương quan 103
  - tương quan 200
hamming 198, 200
Hamming

cửa sổ - 198
handle 31, 55, 60
handle 31, 58
hann 198, 200
hasdelay 97, 98
hệ
- lai 285, 288
- LTI 65, 76, 109
- MIMO 67, 69, 71, 74, 77, 105, 134
- một vật 401
- nhiều vật 401
- SIMO 107
- SISO 66, 69, 105, 109, 127, 134
- tuyến tính – dừng 65, 76
Help 3
help
-[command] 3
- datafun 5
- elfun 5
- elmat 14
- general 30
- graph2d 33
- graph3d 33, 39
- iofun 28
- log 4
- ops 15
- specgraph 33
helpwin 4
hold off 36
hold on 34, 36, 112

I

if 16
ifft 200
ifft2 200
image 41
impulse 104, 106, 109
imread 41
inf 5, 10
Inference 452
initial 104, 109
inline 154
Inline Function 154
inline objects 153
input 25
InputDelay 78, 82
InputGroup 82
InputName 82
interp1 191, 194
interp2 191
interp3 191
Interpolation 191
inv 87, 161
IODelay 77, 82
isa 96, 98
isct 96, 98
isdt 96, 98
isempty 96, 98
isproper 97, 98
issiso 96, 98
iteration steps 157, 162

J

jpg 42

K

kalman 143, 146
Kalman
lọc - 126, 141, 146
kế thừa
Tính - 84
khảo sát 98
- động học 294
khối
- Algebraic Constraint 271
- ảo 215
- Backlash 266
- Bus Selector 249
- Constant 220
- Coulomb & Viscous Friction 267
- Dead Zone 267
- Demux 248
- Derivative 258
- Discrete Filter (scalar) 286
- Discrete State Space 287
- Discrete Transfer Function (scalar) 286
- Discrete Zero-Pole (scalar) 287

- Discrete-Time Integrator 286
- Dot Product 230
- Enable 248
- From File 223
- From Workspace 222
- Function 269
- Gain 231
- Hit Crossing 251
- IC 251
- Inport 247
- Integrator 257
- Logical Operator 231
- Look-Up Table 268
- Look-Up Table (2-D) 268
- Manual Switch 268
- Math Function 230
- Mathlab Function 270
- Matrix Gain 231
- Memory 259
- Multiport Switch 268
- Mux 248
- Outport 247
- Product 230
- Pulse Generator 221
- Quantizer 265
- Ramp 221
- Rate Limiter 265
- Relational Operator 231
- Relay 268
- Repeating Sequence 222
- S-Function 270
- Saturation 265
- Scope 224
- Selector 249
- Signal Generator 221
- Sine Wave 222
- Slider Gain 231
- State-Space 258
- Step 221
- Subsystem 247
- Sum 230
- Switch 268
- thực 215
- To File 228
- To Workspace 226
- Transfer Function 258
- Transport Delay 259
- Trigger 248
- Trigonometric Function 230
- Unit Delay 286
- Variable Transport Delay 259
- XY Graph 226
- Zero-Order Hold 287
- Zero-Pole 258

L

Layout 43, 53
Leakage Effect 196
least squares 153, 177
  nonnegative - 179
`legend` 33
`length` 13, 14, 105, 108, 111
`linespace` 6, 8
`linmod` 261, 262, 296, 298, 308
`linmod2` 262, 296, 298
List Box 49
`load` 27
lọc số 205
lọc tương tự 211
logarithm 35
`log10` 149
`loglog` 35, 111
logic mờ 416
`logspace` 6, 8, 110
`lookfor` 4
lỗi
  - dữ liệu, đầu vào 147
  - phương pháp 147
  - tròn số 147
`lqgreg` 146
`lqr` 141, 146
`lqrd` 141, 146
`lqry` 141, 146
`ls` 30
`lsim` 104, 108, 136, 144
`lsqcurvefit` 182, 184, 185
`lsqlin` 180, 185
`lsqnonlin` 184, 185
`lsqnonneg` 179, 185
LTI-Viewer 298

ltimodels 66
ltiprops 79
Luenberger
khâu quan sát - 133

M

m-file 18, 273
ma trận 6, 71, 89, 141
- Jacobi 163, 164
mag 111
margin 114, 118
matched 95
Math
thư viện - 213, 230
máy điện
- không đồng bộ 326
- một chiều 321
- từ kháng 365
mdlDerivatives 274
mdlGetTimeOfNextVarHit 274
mdlInitializeSize 274
mdlOutputs 274
mdlTerminate 274
mdlUpdate 274
mesh 39, 41
meshgrid 39, 41
mex 273, 278
mex-file 273
minreal 119, 122
mkdir 30
mod 17
modred 120, 122
mô hình
- cầu chỉnh lưu 375
Chuẩn của - 102
điều hòa - 148
- dữ liệu đặc tính tần số 72
- điểm không - điểm cực 68
- giảm bậc 119
- gián đoạn 74, 76
- LTI 79, 83, 87, 96, 102, 295
- mạch điện một chiều 415
- máy phát 410
- mô phỏng MĐDB 330, 334, 340, 417
- mô phỏng MĐDB nguồn kép 361
- mô phỏng MĐĐB 343, 346, 349 351, 359
- mô phỏng MĐMC 324
- mô phỏng MĐTK 370
- nghịch lưu băm xung 379
- nghịch lưu nguồn dòng 389
- phía lưới 413
- thiết bị biến đổi trực tiếp 397
- truyền đạt 66
- trạng thái 71
- tuyến tính hóa 295
- từ thông 406
more 23, 24

N

NaN 5
nargin 19, 20
nargout 19, 20
nhận dạng
- đường đặc tính từ hóa 178
- mờ 429, 440
- Sector 425
Nhát cắt vàng
phương pháp - 168
nội suy 191
Nonlinear
thư viện - 213, 265
norm 102, 104, 180
num2str 26, 47, 60
num 66, 82, 206
NumContStates 277
NumDiscStates 277
numerator 66
NumInputs 277
NumOutputs 277
NumSampleTimes 277
NYQUIST 116, 299
nyquist 111, 117, 118

O

obsv 123
obsvf 124

ode1, ode15, ode2, ode23, ode45 234
Offset 283, 287
ones 7, 8
optimget 155, 157
optimization
- *constrained* 153
- *constrained nonlinear* 170
multiobjective - 176
- *unconstrained* 166, 168
Optimization Toolbox 153
optimset 155, 157, 172
options 155, 157, 160
ord2 92
OutputDelay 78, 82
OutputGroup 82
OutputName 82

P

pade 78
parallel 90, 92
Parent 55
pathtool 29
pause 24
pcx 42
phase 111
PhaseMargin 115
phương sai 103
PID 186
place 134, 137, 140
plot 34, 36
plot3 38, 41
Plot
3-D- 32
PMFrequency 115
pole 99, 104
polo 135, 309
Popup Menu 49, 50
Position 55
prewarp 95
print 42, 246
Property Editor 32
Property Inspector 54, 56
psd 205
pwd 30
pzmap 101, 104, 297

Q

quan sát
- Luenberger 308, 423
- mômen quay 433
- nhiễu 310
- trạng thái 306, 313
- từ thông 433
- tức thời 423
quán tính bậc hai
Khâu - 93, 107, 143
quán tính bậc nhất
Khâu - 36, 73, 94, 107, 275
quỹ đạo
- lục giác 420, 431, 458, 467
- tròn 420, 430, 458, 467

R

Radio Button 49, 50
rand 7, 8, 192, 200
randn 196, 200
rank 9, 10, 124, 126
Rate Limiter 186, 265
reg 137, 140, 146
ResponseData 82
Riccati
phương trình - 140
rlocfind 129, 131
rlocus 127, 131
rmodell 90, 92
roots 99, 104, 160
rotate3d 41
rss 90, 92
Runge-Kutta 234, 285

S

S-Function 272
sai số 235
- tuyệt đối 147
- tương đối 147
Sampling time 74
Saturation 186, 265

save 27
saveas 42
Scale
Large - 168, 171, 174
Medium - 168, 171
script 18, 96, 154
semilogx 35, 111, 211
semilogy 35
series 90, 92
set 31, 32, 36, 60, 77, 80
sgrid 101, 104, 129
sign 99
Signal & Systems
thư viện - 213
Signal Processing Toolbox 192
sim 242
simget 242
Simplex
phương pháp - 168
simset 187, 242
simsizes 277
Simulation
- Diagnostics 242
- Parameters 233
simulink3 213
Sinks
thư viện - 213, 220, 224
SISO Design Tool 131
sisotool 131
size 13, 14, 97
slider 47
sminreal 119, 122
Solver 233, 284, 287
Sources
thư viện - 213, 220
spectrum 200
Splines 191
sprintf 26, 28
sqrt 119, 149
ss 71, 74, 76, 80, 85, 262, 296, 298
ss2tf 85, 262
ss2zp 85
ssbal 150
sscanf 60
ssdata 83
Stable 115
stairs 36, 283
StateName 82
step 85, 104, 107
Step
Fixed - 234, 284, 287
- *Size* 234, 284
Variable - 234, 284, 287
Style 55
str2num 47
string 10, 25, 55, 60
struct 10, 13, 29, 131
structure.name 13
subplot 31, 32, 34
Subsystem 246
Mask - 254
surf 39, 41
switch 16
sysss 84
systf 84

T

Tag. 55
text 33, 45
tf 37, 66, 74, 76, 80, 85
tf2ss 85
tf2zp 85
tfdata 83
thanh trượt 47
thư viện
- Continuous 213, 257
- Discrete 213, 283, 286
- Functions & Tables 213
- Math 213, 230
- mô hình các phần tử truyền 401
- mô hình máy điện 321
- mô hình thiết bị biến đổi 375
- Nonlinear 213, 265
- Signal & Systems 213
- Sinks 213, 220, 224
- Sources 213, 220
Thyristor 389, 417
tif 42

tín hiệu
- ảo 218
- đơn 218
ma trận - 218
vector - 218
title 33
tolerance
Absolute - 235
Relative - 235
TolFun 170
TolX 160, 170
Toolbox
Control System - 65
Optimization - 153
Signal Processing - 192
totaldelay 79
Transfer function 66
Transformation
Fourier - 194
Discrete Fourier - 194
transpose 8, 10
trễ
Thời gian - 76
trích mẫu
chu kỳ - 283, 287
trim 263
truy cập
- dữ liệu 83
trường 10, 12
Ts 82
turn off 53
turn on 53
tustin 93
tỷ lệ – vi phân
Khâu - 36

U

uicontrol 45, 60
uimenu 60
Units 55, 82

V

Value 55
Variable 82
variable 15, 60
global 20
local 20
vector 6, 67, 71, 161
view 39, 41
virtual 215
not - 215

W

waterfall 39, 41
while 16, 17
who 14
whos 14, 25
workspace 14
Workspace 13, 28, 187
- Broser 2, 14
- I/O 237
Show - 14

X

xấp xỉ 78
đa thức - 177
- hình chữ nhật 92
- Tustin 92
xcorr 201, 205
xlabel 33

Y

ylabel 33

Z

zero 99, 104
zero crossing 234, 236
Zero Order Hold 92, 284, 288
zero/pole/gain 69, 102
zeros 7, 8
zgrid 101, 104
zlabel 33, 41
zoh 93
zoom 33
zp2ss 85
zp2tf 85
zpk 69, 74, 76, 80, 85
zpkdata 83

Nguyễn Phùng Quang

MATLAB & Simulink dành cho kỹ sư điều khiển tự động

Chịu trách nhiệm xuất bản: PGS. TS. Tô Đăng Hải
Biên tập: Nguyễn Đăng
Trình bầy và chế bản: Nguyễn Phùng Quang
Vẽ bìa: Trần Thắng

Nhà xuất bản Khoa học và Kỹ thuật
70 Trần Hưng Đạo, Hà Nội

In 1000 cuốn khuôn khổ 16×24 cm, tại Xưởng in NXB Văn hóa Dân tộc
Giấy phép xuất bản số: 1189-34-15/9/2003
In xong và nộp lưu chiểu quý 4/2003