KHOA CÔNG NGHỆ ĐIỆN TỬ VÀ TRUYỀN THÔNG
BỘ MÔN CÔNG NGHỆ KỸ THUẬT MÁY TÍNH

BÀI GIẢNG
MÔ PHỎNG VÀ MÔ HÌNH HÓA

THÁI NGUYÊN – 2011

MỤC LỤC

CHƯƠNG I
GIỚI THIỆU KIẾN THỨC MÔ PHỎNG

1.1. Một số định nghĩa cơ bản

- Đối tượng (object) là tất cả những sự vật, sự kiện mà hoạt động của con người có liên quan tới.
- Hệ thống (System) là tập hợp các đối tượng (con người, máy móc), sự kiện mà giữa chúng có những mối quan hệ nhất định.
- Trạng thái của hệ thống (State of system) là tập hợp các tham số, biến số dùng để mô tả hệ thống tại một thời điểm và trong điều kiện nhất định.
- Mô hình (Model) là một sơ đồ phản ánh đối tượng, con người dùng sơ đồ đó để nghiên cứu, thực nghiệm nhằm tìm ra quy luật hoạt động của đối tượng hay nói cách khác mô hình là đối tượng thay thế của đối tượng gốc để nghiên cứu về đối tượng gốc.
- Mô hình hóa (Modeling) là thay thế đối tượng gốc bằng một mô hình nhằm các thu nhận thông tin quan trọng về đối tượng bằng cách tiến hành các thực nghiệm trên mô hình. Lý thuyết xây dựng mô hình và nghiên cứu mô hình để hiểu biết về đối tượng gốc gọi lý thuyết mô hình hóa.

Nếu các quá trình xảy ra trong mô hình đồng nhất (theo các chỉ tiêu định trước) với các quá trình xảy ra trong đối tượng gốc thì người ta nói rằng mô hình đồng nhất với đối tượng. Lúc này người ta có thể tiến hành các thực nghiệm trên mô hình để thu nhận thông tin về đối tượng.

- Mô phỏng (Simulation, Imitation) là phương pháp mô hình hóa dựa trên việc xây dựng mô hình số (Numerical model) và dùng phương pháp số (Numerical method) để tìm các lời giải. Chính vì vậy máy tính số là công cụ hữu hiệu và duy nhất để thực hiện việc mô phỏng hệ thống.

Lý thuyết cũng như thực nghiệm đã chứng minh rằng, chỉ có thể xây dựng được mô hình gần đúng với đối tượng mà thôi, vì trong quá trình mô hình hóa bao

giờ cũng phải chấp nhận một số giả thiết nhằm giảm bớt độ phức tạp của mô hình, để mô hình có thể ứng dụng thuận tiện trong thực tế. Mặc dù vậy, mô hình hóa luôn luôn là một phương pháp hữu hiệu để con người nghiên cứu đối tượng, nhận biết các quá trình, các quy luật tự nhiên. Đặc biệt, ngày nay với sự trợ giúp đắc lực của khoa học kỹ thuật, nhất là khoa học máy tính và công nghệ thông tin, người ta đã phát triển các phương pháp mô hình hóa cho phép xây dựng các mô hình ngày càng gần với đối tượng nghiên cứu, đồng thời việc thu nhận, lựa chọn, xử lý các thông tin về mô hình rất thuận tiện, nhanh chóng và chính xác. Chính vì vậy, mô hình hóa là một phương pháp nghiên cứu khoa học mà tất cả những người làm khoa học, đặc biệt là các kỹ sư đều phải nghiên cứu và ứng dụng vào thực tiễn hoạt động của mình.

1.2. Mô hình hóa hệ thống

1.2.1. Vai trò của phương pháp mô hình hóa hệ thống

a) Khi nghiên cứu trên hệ thống thực gặp nhiều khó khăn do nhiều nguyên nhân gây ra như sau:

- Giá thành nghiên cứu trên hệ thống thực quá đắt.

Ví dụ: Nghiên cứu kết cấu tối ưu, độ bền, khả năng chống dao động của ô tô, tàu thủy, máy bay,… người ta phải tác động vào đối tượng nghiên cứu các lực đủ lớn đến mức có thể phá hủy đối tượng để từ đó đánh giá các chỉ tiêu kỹ thuật đã đề ra. Như vậy, giá thành nghiên cứu sẽ rất đắt. Bằng cách mô hình hóa trên máy tính ta dễ dàng xác định được kết cấu tối ưu của các thiết bị nói trên.

- Nghiên cứu trên hệ thống thực đòi hỏi thời gian quá dài.

Ví dụ: Nghiên cứu đánh giá độ tin cậy, đánh giá tuổi thọ trung bình của hệ thống kỹ thuật (thông thường tuổi thọ trung bình của hệ thống kỹ thuật khoảng 30 ÷ 40 năm), hoặc nghiên cứu quá trình phát triển dân số trong khoảng thời gian 20 ÷ 50 năm,… Nếu chờ đợi quãng thời gian dài như vậy mới có kết quả nghiên cứu thì không còn tính thời sự nữa. Bằng cách mô phỏng hệ thống và cho "hệ thống"

vận hành tương đương với khoảng thời gian nghiên cứu người ta có thể đánh giá được các chỉ tiêu kỹ thuật cần thiết của hệ thống.

- Nghiên cứu trên hệ thực ảnh hưởng đến sản xuất hoặc gây nguy hiểm cho người và thiết bị.

Ví dụ: Nghiên cứu quá trình cháy trong lò hơi của nhà máy nhiệt điện, trong lò luyện clanhke của nhà máy xi măng… người ta phải thay đổi chế độ cấp nhiên liệu (than, dầu), tăng giảm sản lượng gió cấp, thay đổi áp suất trong lò,… Việc làm các thí nghiệm như vậy sẽ cản trở việc sản xuất bình thường, trong nhiều trường hợp có thể xảy ra cháy, nổ gây nguy hiểm cho người và thiết bị. Bằng cách mô phỏng hệ thống, người ta có thể cho hệ thống "vận hành" với các bộ thông số, các chế độ vận hành khác nhau để tìm ra lời giải tối ưu.

- Trong một số trường hợp không cho phép làm thực nghiệm trên hệ thống thực.

Ví dụ: Nghiên cứu các hệ thống làm việc ở môi trường độc hại, nguy hiểm, dưới hầm sâu, dưới đáy biển, hoặc nghiên cứu trên cơ thể người,… Trong những trường hợp này dùng phương pháp mô phỏng là giải pháp duy nhất để nghiên cứu hệ thống.

b) Phương pháp mô hình hóa cho phép đánh giá độ nhạy của hệ thống khi thay đổi tham số hoặc cấu trúc của hệ thống cũng như đánh giá phản ứng của hệ thống khi thay đổi tín hiệu điều khiển. Những số liệu này dùng để thiết kế hệ thống hoặc lựa chọn thông số tối ưu để vận hành hệ thống.

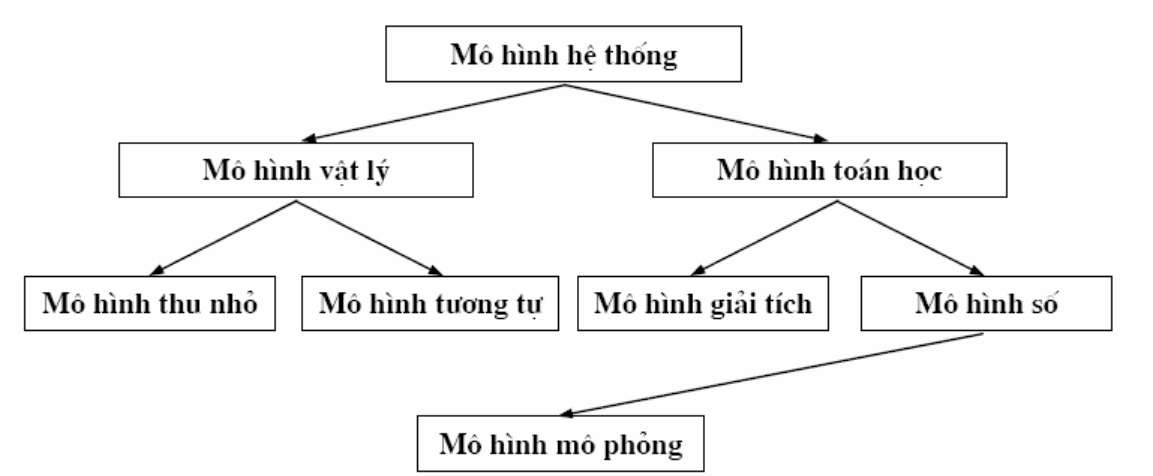
c) Phương pháp mô hình hóa cho phép nghiên cứu hệ thống ngay cả khi chưa có hệ thống thực

Trong trường hợp này, khi chưa có hệ thống thực thì việc nghiên cứu trên mô hình là giải pháp duy nhất để đánh giá các chỉ tiêu kỹ thuật của hệ thống, lựa chọn cấu trúc và thông số tối ưu của hệ thống… đồng thời mô hình cũng được dùng để đào tạo và huấn luyện.

Trong những trường hợp này dùng phương pháp mô phỏng mô hình hóa là giải pháp duy nhất để nghiên cứu hệ thống.

1.2.2. Phân loại mô hình hóa hệ thống

Có thể căn cứ vào nhiều dấu hiệu khác nhau để phân loại mô hình. Hình 1.1 biểu diễn một cách phân loại mô hình điển hình. Theo cách này mô hình chia thành hai nhóm: mô hình vật lý và mô hình toán học hay còn gọi là mô hình trừu tượng.



Hình 1.1- Sơ đồ phân loại mô hình

- Mô hình vật lý là mô hình được cấu tạo bởi các phần tử vật lý. Các thuộc tính của đối tượng phản ánh các định luật vật lý xảy ra trong mô hình. Nhóm mô hình vật lý được chia thành mô hình thu nhỏ và mô hình tương tự. Mô hình vật lý thu nhỏ có cấu tạo giống đối tượng thực nhưng có kích thước nhỏ hơn cho phù hợp với điều kiện của phòng thí nghiệm. Ví dụ, người ta chế tạo lò hơi của nhà máy nhiệt điện có kích thước nhỏ đặt trong phòng thí nghiệm để nghiên cứu các chế độ thủy văn của đập thủy điện. Ưu điểm của loại mô hình này là các quá trình vật lý xảy ra trong mô hình giống như trong đối tượng thực, có thể đo lường quan sát các đại lượng vật lý một cách trực quan với độ chính xác cao. Nhược điểm của mô hình vật lý thu nhỏ là giá thành đắt, vì vậy chỉ sử dụng khi thực sự cần thiết.

- Mô hình vật lý tương tự được cấu tạo bằng các phần tử vật lý không giống với đối tượng thực nhưng các quá trình xảy ra trong mô hình tương đương với quá trình xảy ra trong đối tượng thực. Ví dụ, có thể nghiên cứu quá trình dao động của con lắc đơn bằng mô hình tương tự là mạch dao động R-L-C vì quá trình dao động điều hòa trong mạch R-L-C hoàn toàn tương tự quá trình dao động điều hòa của con lắc đơn, hoặc người ta có thể nghiên cứu đường dây tải điện bằng mô hình tương tự là mạng bốn cực R-L-C. Ưu điểm của loại mô hình này là giá thành rẻ, cho phép chúng ta nghiên cứu một số đặc tính chủ yếu của đối tượng thực.

- Mô hình toán học thuộc loại mô hình trừu tượng. Các thuộc tính được phản ánh bằng các biểu thức, phương trình toán học. Mô hình toán học được chia thành mô hình giải tích và mô hình số. Mô hình giải tích được xây dựng bởi các biểu thức giải tích. Ưu điểm của loại mô hình là cho ta kết quả rõ ràng, tổng quát. Nhược điểm của mô hình giải tích là thường phải chấp nhận một số giả thiết đơn giản hóa để có thể biểu diễn đối tượng thực bằng các biểu thức giải tích, vì vậy loại mô hình này chủ yếu được dùng cho các hệ tiền định và tuyến tính.

- Mô hình số được xây dựng theo phương pháp số tức là bằng các chương trình chạy trên máy tính số. Ngày nay, nhờ sự phát triển của kỹ thuật máy tính và công nghệ thông tin, người ta đã xây dựng được các mô hình số có thể mô phỏng được quá trình hoạt động của đối tượng thực. Những mô hình loại này được gọi là mô hình mô phỏng. Ưu điểm của mô hình mô phỏng là có thể mô tả các yếu tố ngẫu nhiên và tính phi tuyến của đối tượng thực, do đó mô hình càng gần với đối tượng thực. Ngày này, mô hình mô phỏng được ứng dụng rất rộng rãi.

Có thể căn cứ vào các đặc tính khác nhau để phân loại mô hình như: mô hình tĩnh và mô hình động, mô hình tiền định và mô hình ngẫu nhiên, mô hình tuyến tính và mô hình phi tuyến, mô hình có thông số tập trung, mô hình có thông số dải, mô hình liên tục, mô hình gián đoạn, …

Mô hình phải đạt được hai tính chất cơ bản sau:

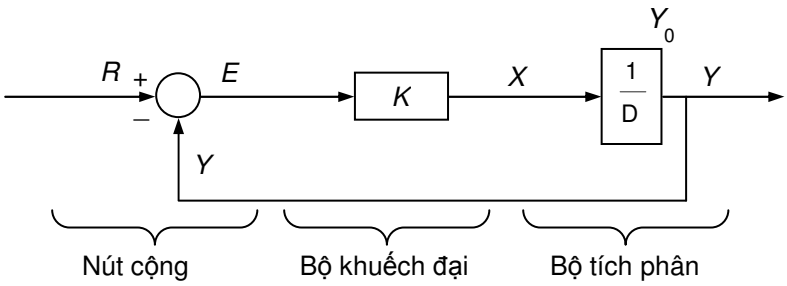
Tính đồng nhất: mô hình phải đồng nhất với đối tượng mà nó phản ánh theo những tiêu chuẩn định trước.

Tính thực dụng: Có khả năng sử dụng mô hình để nghiên cứu đối tượng. Rõ ràng, để tăng tính đồng nhất trong mô hình phải đưa vào nhiều yếu tố phản ánh đầy đủ các mặt của đối tượng. Nhưng như vậy nhiều khi mô hình trở nên quá phức tạp và cồng kềnh đến nỗi không thể dùng để tính toán được nghĩa là mất đi tính chất thực dụng của mô hình. Nếu quá chú trọng tính thực dụng, xây dựng mô hình quá đơn giản thì sai lệch giữa mô hình và đối tượng thực sẽ lớn, điều đó sẽ dẫn đến kết quả nghiên cứu không chính xác. Vì vậy, tùy thuộc vào mục đích nghiên cứu mà người ta lựa chọn tính đồng nhất và tính thực dụng của mô hình một cách thích hợp.

1.3. Phương pháp mô phỏng

1.3.1. Sơ đồ khối

Các mô hình sơ đồ khối gồm hai đối tượng, các đường dây tín hiệu và các khối. Chức năng của đường dây tín hiệu là truyền dẫn tín hiệu, hoặc giá trị, từ điểm gốc ban đầu của nó (thường là một khối) tới điểm kết thúc (thường là một khối khác). Hướng của dòng tín hiệu được xác định bởi mũi tên trên đường tín hiệu. Một hướng chỉ được xác định cho một đường tín hiệu, toàn bộ các tín hiệu truyền trên các nhánh khác phải theo hướng riêng. Mỗi khối là một thành phần xử lý để tác động tới tín hiệu và tham số đầu vào để tạo ra tín hiệu đầu ra. Bởi vì các khối chức năng có thể là phi tuyến cũng như tuyến tính nên tập hợp các khối chức năng riêng về thực tế là không giới hạn và hầu như không bao giờ có sự giống nhau giữa các nhà cung cấp về ngôn ngữ của khối chức năng. Tuy nhiên, một sơ đồ ba khối cơ bản phải được thiết lập để các ngôn ngữ sơ đồ khối có điểm chung. Các khối này là nút cộng, khối khuếch đại và bộ tích phân. Một hệ thống kết hợp chặt chẽ ba khối đó được mô tả như Hình 1.2.



Hình 1.2: Ví dụ về một hệ thống 3 khối

1.3.2. Bản chất của phương pháp mô phỏng

Phương pháp mô phỏng có thể định nghĩa như sau:

"Mô phỏng là quá trình xây dựng mô hình toán học của hệ thống thực và sau đó tiến hành tính toán thực nghiệm trên mô hình để mô tả, giải thích và dự đoán hành vi của hệ thống thực".

Theo định nghĩa này, có ba điểm cơ bản mà mô phỏng phải đạt được. Thứ nhất là phải có mô hình toán học tốt tức là mô hình có tính đồng nhất cao với hệ thực đòng thời mô hình được mô tả rõ ràng thuận tiện cho người sử dụng. Thứ hai là mô hình cần phải có khả năng làm thực nghiệm trên mô hình tức là có khả năng thực hiện các chương trình máy tính để xác định các thông tin về hệ thực. Cuối cùng là khả năng dự đoán hành vi của hệ thực tức là có thể mô tả sự phát triển của hệ thực theo thời gian.

Phương pháp mô phỏng được đề xuất vào những năm 80 của thế kỷ 20, từ đó đến nay phương pháp mô phỏng đã được nghiên cứu, hoàn thiện, và ứng dụng thành công vào nhiều lĩnh vực khác nhau như lĩnh vực khoa học kỹ thuật, khoa học xã hội, kinh tế, y tế,… Sau đây trình bày một số lĩnh vực mà phương pháp mô phỏng đã được ứng dụng và phát huy được ưu thế của mình.

- Phân tích và thiết kế hệ thống sản xuất, lập kế hoạch sản xuất.
- Đánh giá phần cứng, phần mềm của hệ thống máy tính.
- Quản lý và xác định chính sách sự trữ mua sắm vật tư của hệ thống kho vật tư, nguyên liệu.

- Phân tích và đánh giá hệ thống phòng thủ quân sự, xác định chiến lược phòng thủ, tấn công.
- Phân tích và thiết kế hệ thống thông tin liên lạc, đánh giá khả năng làm việc của mạng thông tin.
- Phân tích và thiết kế các hệ thống giao thông như đường sắt, đường bộ, hàng không, cảng biển.
- Đánh giá, phân tích và thiết kế các cơ sở dịch vụ như bệnh viện, bưu điện, nhà hàng, siêu thị.
- Phân tích hệ thống kinh tế, tài chính.

Phương pháp mô phỏng được ứng dụng vào các giai đoạn khác nhau của việc nghiên cứu, thiết kế và vận hành các hệ thống như sau:

+ Phương pháp mô phỏng được ứng dụng vào giai đoạn nghiên cứu, khảo sát hệ thống trước khi tiến hành thiết kế nhằm xác định độ nhạy của hệ thống đối với sự thay đổi cấu trúc và tham số của hệ thống.

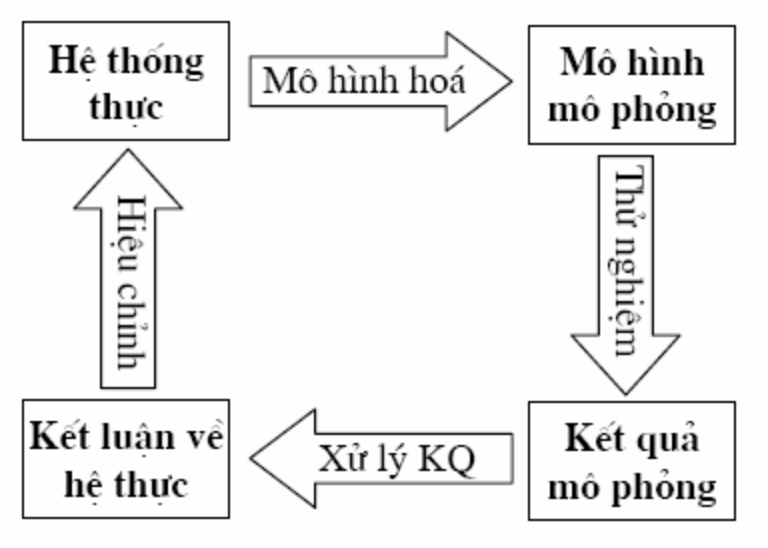
+ Phương pháp mô phỏng được ứng dụng vào giai đoạn thiết kế hệ thống để phân tích và tổng hợp các phương án thiết kế hệ thống, lựa chọn cấu trúc hệ thống thỏa mãn các chỉ tiêu cho trước.

+ Phương pháp mô phỏng được ứng dụng vào giai đoạn vận hành hệ thống để đánh giá khả năng hoạt động, giải bài toán vận hành tối ưu, chẩn đoán các trang thái đặc biệt của hệ thống.

Quá trình mô hình hóa được tiến hành như sau: Gọi hệ thống được mô phỏng là S. Bước thứ nhất người ta mô hình hóa hệ thống S với các mối quan hệ nội tại của nó. Để thuận tiện trong việc mô hình hóa, người ta thường chia hệ S thành nhiều hệ con theo các tiêu chí nào đó $S = S_1, S_2, S_3, \dots, S_n$. Tiếp đến người ta mô tả toán học các hệ con cùng các quan hệ giữa chúng. Thông thường giữa các hệ con có mối quan hệ trao đổi năng lượng và trao đổi thông tin. Bước thứ hai người ta mô hình hóa môi trường xung quanh E, nơi hệ thống S làm việc, với các

mối quan hệ tác động qua lại giữa S và E. Khi đã có mô hình của S và E, người ta tiến hành các thực nghiệm trên mô hình, tức là cho S và E làm việc ở một điều kiện xác định nào đó. Kết quả người ta thu đươc một bộ thông số của hệ thống, hay thường gọi là xác định được một điểm làm việc của hệ thống. Các thực nghiệm đó được lặp lại nhiều lần và kết quả mô phỏng được đánh giá theo xác suất thống kê. Kết quả mô phỏng càng chính xác nếu số lần thực nghiệm, còn gọi là bước mô phỏng càng lớn. Về lý thuyết bước mô phỏng là hữu hạn nhưng phải đủ lớn và phụ thuộc vào yêu cầu của độ chính xác.

Hình 1.3 trình bày quá trình nghiên cứu bằng phương pháp mô phỏng và quan hệ giữa hệ thống thực với kết quả mô phỏng.



Hình 1.3 Quá trinh nghiên cứu bằng phương pháp mô phỏng

Nhìn vào hình 1.3 ta thấy rằng để nghiên cứu hệ thống thực ta phải tiến hành mô hình hóa tức là xây dựng mô hình mô phỏng. Khi có mô hình mô phỏng sẽ tiến hành làm các thực nghiệm trên mô hình để thu được các kết quả mô phỏng. Thông thường kết quả mô phỏng có tính trừu tượng của toán học nên phải thông qua xử lý mới thu được các thông tin kết luận về hệ thống thực. Sau đó dùng các thông tin và kết luận trên để hiệu chỉnh hệ thực theo mục đích nghiên cứu đã đề ra.

1.3.3. Các bước nghiên cứu mô phỏng

Hình 1.4 Các bước nghiên cứu mô phỏng

Khi tiến hành nghiên cứu mô phỏng thông thường phải thực hiện qua 10 bước như được biểu diễn bởi lưu đồ như hình 1.4.

Bước 1: Xây dựng mục tiêu mô phỏng và kế hoạch nghiên cứu.

Điều quan trọng trước tiên là phải xác định rõ mục tiêu nghiên cứu mô phỏng. Mục tiêu đó được thể hiện bằng các chỉ tiêu đánh giá, bằng hệ thống các câu hỏi cần được trả lời.

Bước 2: Thu thập dữ liệu và xác định mô hình nguyên lý.

Tùy theo mục tiêu mô phỏng mà người ta thu thập các thông tin, các dữ liệu tương ứng của hệ thống S và môi trường E. Trên cơ sở đó xây dựng mô hình nguyên lý M_{nl}, mô hình nguyên lý phản ánh bản chất của hệ thống S.

Bước 3: Hợp thức hóa mô hinh nguyên lý M_{nl}

Hợp thức hóa mô hình nguyên lý là kiểm tra tính đúng đắn, hợp lý của mô hình. Mô hình nguyên lý phải phản ánh đúng bản chất của hệ thống S và môi trường E nhưng đồng thời cũng phải tiện dụng, không quá phức tạp cồng kềnh. Nếu mô hình nguyên lý M_{nl} không đạt phải thu thập thêm thông tin, dữ liệu để tiến hành xây dựng lại mô hình.

Bước 4: Xây dựng mô hình mô phỏng M_{mp} trên máy tính.

Mô hình mô phỏng M_{mp} là những chương trình chạy trên máy tính. Các chương trình này được viết bằng các ngôn ngữ thông dụng như FORTRAN, PASCAL, C^{++}, hoặc các ngôn ngữ chuyên dụng để mô phỏng như GPSS, SIMSCRIPT,…

Bước 5: Chạy thử

Sau khi cài đặt chương trình, người ta tiến hành chạy thử xem mô hình mô phỏng có phản ánh đúng các đặc tính của hệ thống S và môi trường E hay không. Ở giai đoạn này cũng tiến hành sửa chữa các lỗi về lập trình.

Bước 6: Kiểm chứng mô hình

Sauk hi chạy thử người ta có thể kiểm chứng và đánh giá mô hình mô phỏng có đạt yêu cầu hay không, nếu không phải quay lại từ bước 2.

Bước 7: Lập kế hoạch thử nghiệm

Ở bước này người ta phải xác định số lần thử nghiệm, thời gian mô phỏng của từng bộ phận hoặc toàn bộ mô hình. Căn cứ vào kết quả mô phỏng (ở bước 9), người ta tiến hành hiệu chỉnh kế hoạch thử nghiệm để đạt được kết quả với độ chính xác theo yêu cầu.

Bước 8: Thử nghiệm mô phỏng

Cho chương trình chạy thử nghiệm theo kế hoạch đã được lập ở bước 7. Đây là bước thực hiện việc mô phỏng, các kết quả lấy ra từ bước này.

Bước 9: Xử lý kết quả

Thử nghiệm mô phỏng thường cho nhiều dữ liệu có tính thống kê xác suất. Vì vậy, để có kết quả cuối cùng với độ chính xác theo yêu cầu, cần phải thực hiện việc xử lý các kết quả trung gian. Bước xử lý kết quả đóng vai trò quan trọng trong quá trình mô phỏng.

Bước 10: Sử dụng và lưu trữ kết quả.

Sử dụng kết quả mô phỏng vào mục đích đã định và lưu giữ dưới dạng các tài liệu để có thể sử dụng nhiều lần.

1.3.4 Một số môi trường mô phỏng thường gặp

- Matrix/ System Build

- Easy 5

- Matlab/ Simulink

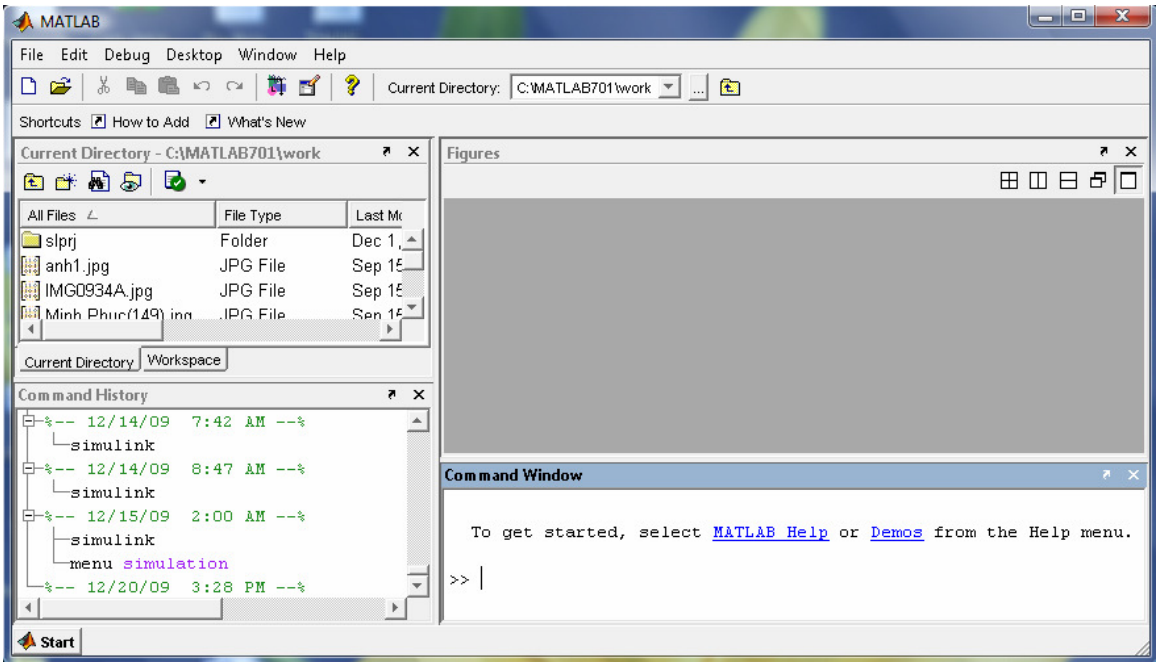
- LabView

- VisSim

…

CHƯƠNG II
MÔI TRƯỜNG MATLAB VÀ CÁCH LẬP TRÌNH

2.1 Giới thiệu môi trường làm việc Matlab



2.2 Các hàm toán

Chương trình Matlab có sẵn rất nhiều hàm toán tập hợp trong bảng sau đây. Để xem kỹ hơn, 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)*	Hàm arctg $\pm$ 180^0
tan(x)	Hàm tg	*sinc(x)*	Hàm sin (pi x)/(pi x)

2.3 Tính toán với vector và ma trận

2.3.1. Khai báo 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 (trong các trường hợp khác Matlab sẽ bỏ qua vị trí trống, các hàng ngăn cách bởi dấu (;) hoặc ngắt dòng.

Ví dụ:

```
>> my_vector = [2 3 4]
my_vector =
     2     3     4
>> my_matrix = [my_vector; 5 6 7]
my_matrix =
     2     3     4
     5     6     7
```

Vector có các phần tử tiếp diễn với một bước đi nhất định, có thể được nhập một cách đơn giản nhờ toán tử (:) như sau:

Start: increment: destination

Nếu chỉ nhập *start* và *destination* thì Matlab sẽ tự động đặt *increment* là +1.

Ví dụ:

```
>> my_vector = (1:3:8)

my_vector =

     1     4     7

>> my_vector1 = (1:8)

my_vector1 =

     1     2     3     4     5     6     7     8
```

Cũng có thể nhập các vector tuyến tính cũng như vector có phân hạng *logarithm* bằng cách dùng lệnh:

`linspace`*(start, destination, number)*

`logspace`*(start, destination, number)*

Đối với `logspace` thì *start* và *destination* được nhập bởi số mũ thập phân, ví dụ: $100 = 10^2$ ta chỉ cần nhập 2.

Ví dụ:

```
>> linspace(1,7,3)

ans =

     1     4     7

>> logspace(1,3,5)

ans =

  1.0e+003 *
```

```
    0.0100    0.0316    0.1000    0.3162    1.0000
```

Bằng các hàm `ones`*(line, column)* và `zeros`*(line, column)* ta tạo các ma trận có phần tử là 0 hoặc 1. 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.

Ví dụ:

```
>> ones(2,2)
ans =
     1     1
     1     1
>> zeros(2,2)
ans =
     0     0
     0     0
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     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 hang 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.

Ví dụ:

```
>> my_matrix(2,3)

ans =

     7
```

Matlab có một lệnh rất hữu ích, phục vụ tạo ma trận với chức năng tín hiệu thử, đó là `rand`*(line, column)*. Khi gọi, ta thu được ma trận với phần tử mang các giá trị ngẫu nhiên.

Ví dụ:

```
>> rand (2,3)

ans =

    0.9501    0.6068    0.8913

    0.2311    0.4860    0.7621
```

2.3.2. Tính toán với vector và ma trận

Nhiều phép tính có thể được áp dụng cho vector và ma trận.

- Các phép tính với từng phần tử: .* ./ .^

```
>> [2 3 4] .* [1 2 3]

ans =

     2     6    12

>> [2 3 4] .^ [1 2 3]

ans =

     2     9    64
```

- Chuyển vị ma trận matrix:

`transpose` *(maxtrix)* hoặc *matrix.'*

```
>> transpose (my_matrix)

ans =
```

```
     2     5
     3     6
     4     7
```

- Chuyển vị ma trận matrix có phần tử phức liên hợp:

  `ctranspose` *(maxtrix)* hoặc *matrix'*

  (đối với các giá trị thực, hai lệnh trên cho ra kết quả như nhau)

```
>> matrix = [1+i 1-i;2 3]
maxtrix =
   1.0000 + 1.0000i   1.0000 - 1.0000i
   2.0000             3.0000
>> ctranspose(matrix)
ans =
   1.0000 - 1.0000i   2.0000
   1.0000 + 1.0000i   3.0000
```

- Đảo ma trận:

  `inv`*(matrix)*

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

- Tính định thức của ma trận:

  `det`*(matrix)*

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

```
     1
```

- Tính các giá trị riêng của ma trận:

```
eig(matrix)
>> eig(matrix)
ans =
    0.1010
    9.8990
```

- Xác định hạng của ma trận:

```
rank(maxtrix)
>> rank (my_matrix)
ans =
     2
```

- Tính vector sai phân:

```
diff(vector [n])
>> vector= [1 2 3]
vector =
     1     2     3
>> diff(vector)
ans =
     1     1
```

- Chập vector (nhân đa thức): nếu hai vector cần chập có số 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.

```
conv(vector 1, vector 2)
```

```
>> vector1 = [2 3 4]
vector1 =
     2     3     4
>> vector2 = [1 2 3]
vector2 =


     1     2     3
>> conv(vector1,vector2)
ans =
     2     7    16    17    12
```

2.4 Các phép so sánh và phép toán Logic

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

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			
>=	*ge(a,b)*	lớn hơn hoặc bằng			

Chú ý:

- Các phép tính được thực hiện theo trình tự: trước hết là các biểu thức toán, tiếp theo là các biểu thức logic. Tuy nhiên, khi có cảm giác không chắc chắn, có thể dùng cách viết với dấu ngoặc đơn.

- Một lệnh 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: Nếu không ta thu được kết quả là số 0, nếu kết qủ là số khác 0, đó chính là số nói lên bản chất của variable, chẳng hạn 1 nói rằng variable là biến trong wordspace, 2 nói rằng variable là một Matlab File trong thư mục Matlab, … Có thể xem danh mục các lệnh nhờ help ops.

2.5 Biến, cấu trúc và trường

2.5.1. 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 đó có thể chứa tới 32 chữ cái, gạch ngang thấp (_) cũng hư chữ số. Chữ viết hoa và chữ thường 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.

Ví dụ:

```
>> 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 nên ta phải tránh sử dụng chúng. Đối với các phép tính bất định (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)

Ví dụ:

```
>> 1/0
Warning: Divide by zero.
ans =
   Inf
>> 0/0
Warning: Divide by zero.
ans =
   NaN
```

2.5.2. 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*) 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*, …)

```
>> matrix=[2 3 4;3 4 5]
matrix =
```

```
     2     3     4
     3     4     5
>> my_struct=struct('data',matrix,'size',[2,3]);
```

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

```
>> my_struct(2).data=matrix.^(-1);
>> my_struct(2).data(1,:)
ans =
    0.5000    0.3333    0.2500
```

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`:

```
>> 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, vì vậy có thể xử lý các phần tử của cấu trúc đó như 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
```

2.5.3. Trường

Tổng quát ở một mức cao hơn cấu trúc là *trường*. 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 thong 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_sell` như sau:

```
>> my_sell = cell(2,3)

my_sell =

     []     []     []

     []     []     []
```

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 2.5.2:

```
>> my_cell{1,1} = 'Xin chao cac ban! ';17

>> 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 đầy 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]    '23-Nov-2009'
>> my_cell{2,3}
ans =
23-Nov-2009
>> my_cell{2,1:2}
ans =
    name: 'Jahann Sebastian Bach'
    born: [1x1 struct]
ans =
    name: 'Wolfgang Amadeus Mozart'
    born: [1x1 struct]
>> my_cell{2,2}.born.Month
ans =
January
```

2.6 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 = []`.

```
>> a = [2 4 5;1 2 4]
a =
     2     4     5
     1     2     4
```

```
>> length(a)
ans =
     3
>> size(a)
ans =
     2     3
>> variable = []
variable =
     []
```

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 xoá có chủ đích một số biến nhất định, nếu chỉ gọi `clear` ta sẽ xoá toàn bộ biến trong *Workspace*.

```
>> whos
 Name            Size              Bytes  Class
 a               2x3                  48  double array
 ans             1x1                   8  double array
 b               3x2                  48  double array
 componist       1x2                1256  struct array
 datum           1x1                 398  struct array
 my_cell         2x3                1878  cell array
 variable        0x0                   0  double array

Grand total is 221 elements using 3636 bytes
```

2.7 Rẽ nhánh và vòng lặp

2.7.1 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`
- `swith` *term* `case` *term command* [...][`otherwise` *command*] `end`

```
>> 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 nhiều cấu trúc `if` và `switch` móc vòng, đan xen lẫn nhau.

2.7.2 Vòng lặp 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.

vidu.m

```
for k=1:0;
    k^2
```

```
end;
n = 1;
while 1
    n=n+1;
    m=n^2
    if m>10
        break;
    end;
end
>> vidu
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 một lần vì điều kiện ngừng chỉ được kiểm tra sau cùng. Ngoài ra, vòng `while` cần hai lệnh `end` để kết thúc.

2.7.3 Gián đoạn bằng `continue` và `break`

Hai lệnh 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 kế tiếp, mọi lệnh chưa thực hiện của vòng lặp 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: 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 tới (*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` 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.

Matlab đưa ra kết qủa trên màn hình như sau:

!! 3 is a prime number!

2 is not a prime number!

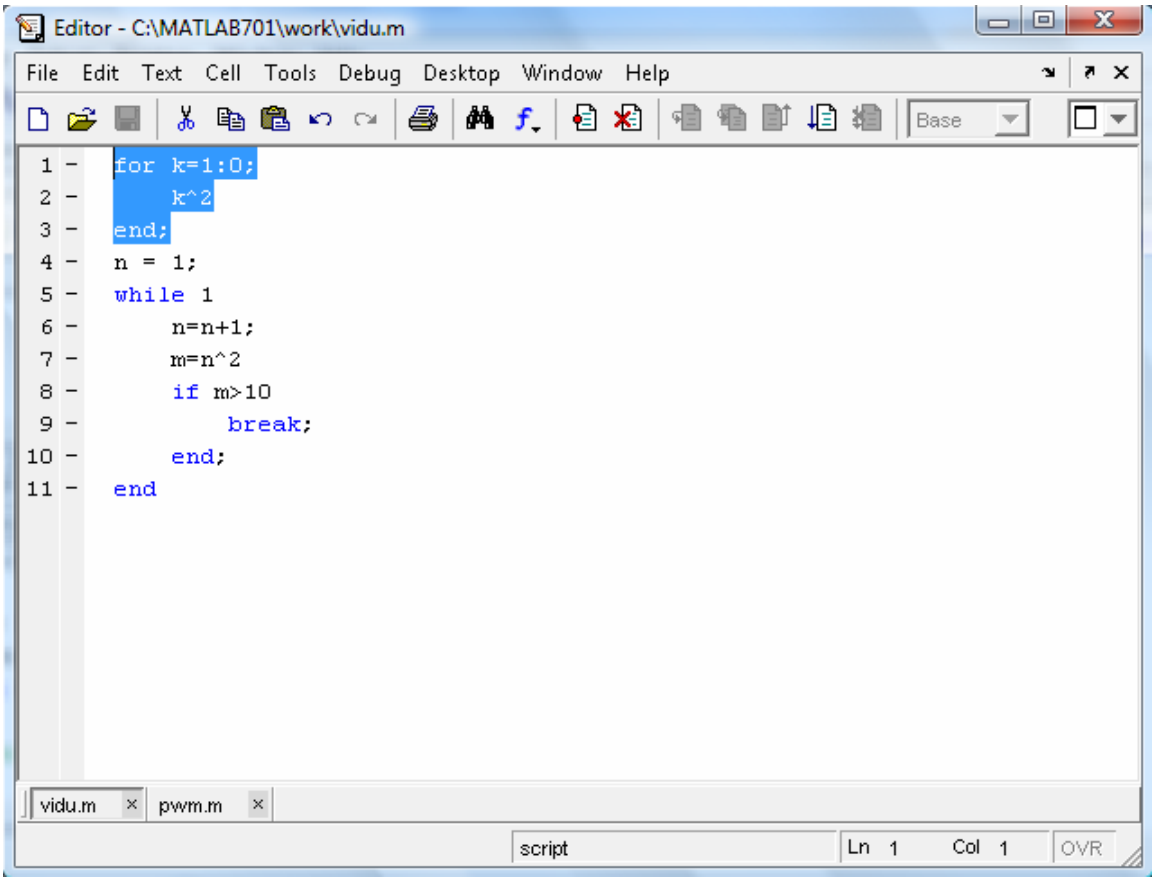
3 ….

Để xem tất cả các lệnh tạo khả năng điều khiển chương trình tính toán, ta gọi lệnh `help lang`.

2.8 Các scripts và các hàm của Matlab

2.8.1. Các scripts của Matlab

Bên cạnh khả năng nhập lệnh trực tiếp, ta có thể viết và cất nhiều chuỗi lệnh trong các *script* 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. Nếu chưa cài đặt trình soạn thảo đó, có thể sử dụng bất kỳ trình soạn thảo ASCII nào khác cũng được.



Hình 2.3 Trình soạn thảo của Matlab với ví dụ file PWM.M

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

2.8.2. Các hàm của Matlab

Một dạng đặc biệt của hàm *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.

Nếu một Matlab *script* hay một Matlab `function` lần đầu tiên được gọi, Matlab sẽ dịch ra mã ảo, là mã sẽ được kích hoạt để thực hiện nhiệm vụ đặt ra cho *script* hay `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 xoá cưỡng bức các hàm đã dịch, đồng thời giữ nguyên các *m-files.*

2.9 Nhập xuất dữ liệu

Thông thường, để Matlab tìm được các script hay dữ liệu, bắt buộc các file liên quan phải nằm tại thư mục hiện tại. Một số lệnh 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*	Xoá file
`!` *commando*	Gọi lệnh từ hệ điều hành

CHƯƠNG III
ĐỒ HỌA TRONG MATLAB

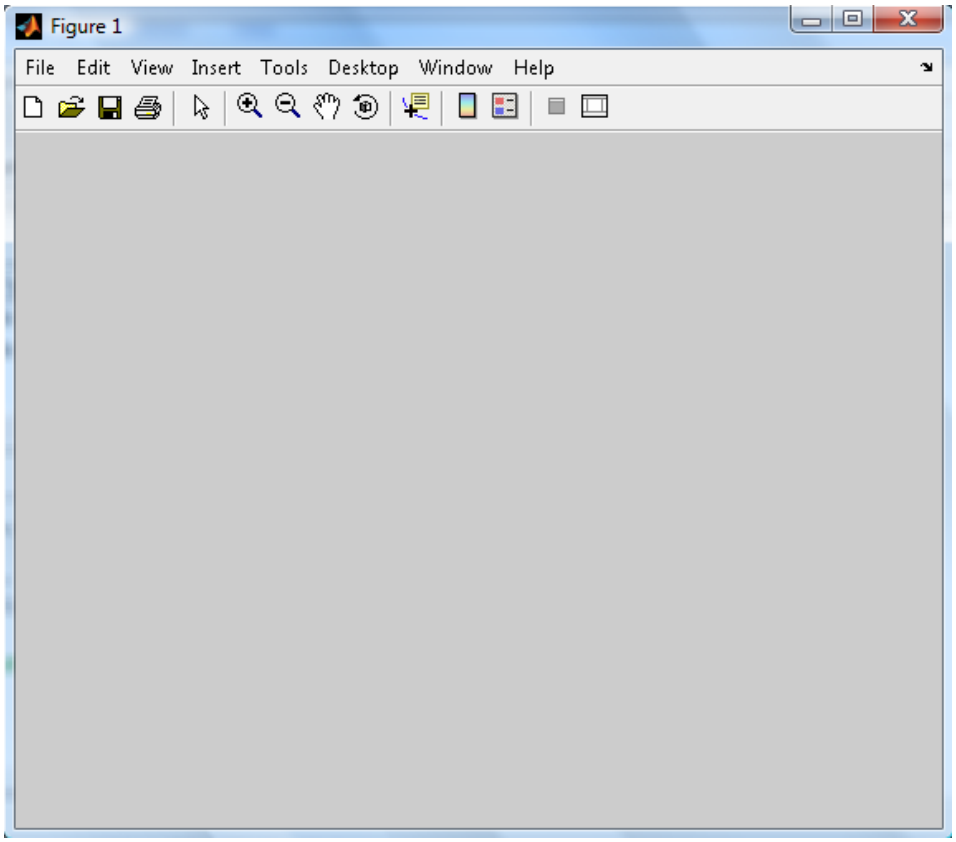
3.1 Cơ sở đồ hoạ Matlab

Khuôn khổ của mọi thao tác xuất đồ hoạ trên nền Matlab là *Figure*. 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
```

Có thể gọi một figure đã có số bằng lệnh `figure`(*number*). Số của *figure* sẽ hiển thị nếu ta gọi `gcf` (*Get hendle to Current figure*).

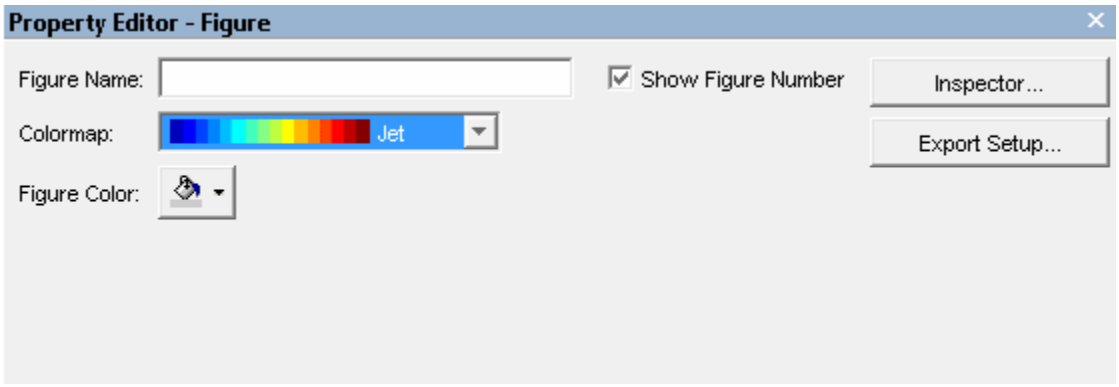
```
>> gcf
ans =
     1
```



Bằng lệnh `subplots` (*row, column, counter*), có thể chia đều một *figure* thành nhiều `subplots` (đồ hoạ 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.

Có thể xoá nội dung của một *figure* bằng lệnh `clf` (*clear current figure*), và lệnh `delete figure`(*number*) sẽ xoá 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. Có thể xem đặc điểm của một *figure* bằng lệnh `get`, với lệnh `set` ta lại có thể lập đặc điểm cho *figure*.

Việc lập trình đồ hoạ 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 hoạ (*property editor*) với cửa sổ giới thiệu ở hình sau:



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) `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 toạ độ ứng với cách chia trục đã xác định. Đối với đồ hoạ 3-D ta có thêm lệnh `box on` để tạo khung bao cho 3-D-Plot.

Để điền ký tự vào một đồ hoạ 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.

Bằng lệnh `legend`(*string_1,string_2*,…, [*position*]) ta có thể điền thêm một số lời ghi chú vào đồ hoạ. Vị trí của 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 hoạ. Lệnh `text`(*x_value,y_value,string*) cho phép ta điền một đoạn văn bản với nội dung *string* vào toạ độ bất kỳ *x_value, y_value* trong đồ hoạ.

Sử dụng lệnh `zoom on | off` để dung chuột cắt và co dãn mảng đó. Ngoài ra, cửa sổ *figure* còn có một vài nút cho phép dung chuột điền đoạn văn bản, vẽ thêm nét hoặc mũi tên, và mở *Property Editor*.

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

3.2 Đồ hoạ 2 chiều

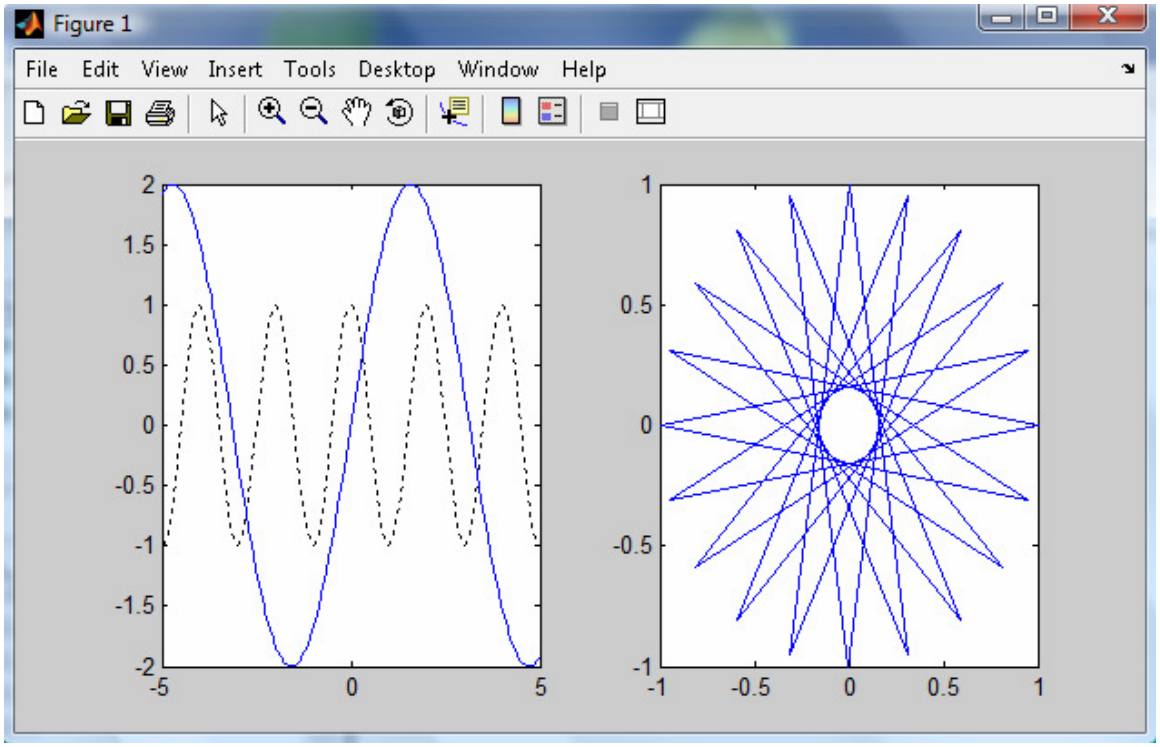
Lệnh `plot` (*x_value,y_value* …[,*plotstyle*]) vẽ đồ thị nối các điểm cho bởi cặp giá trị *x_value, y_value*. Thông thường các điểm đó được nối bởi một nét liền. Nếu ta nạp luân phiên nhiều vector x/y, ta sẽ thu được nhiều nét nối độc lập với nhau. Nếu thiếu *x_value*, khi ấy các giá trị của *y_value* 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 bậc thang.

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.

Mầu				Nét và điểm			
k	Đen	r	Đỏ	-	Nét liền	o	Chấm tròn
b	Xanh lam	m	Đỏ sẫm	--	Nét đứt	*	Chấm sao
c	Xám	y	Vàng	:	Nét gạch chấm	+	Dấu cộng
g	Xanh lá cây	w	Trắng	.	Nét chấm	×	Dấu nhân

Mỗi lần gọi mới lệnh `plot`, các đồ thị đã có trong *Figure* (hoặc trong *Subplot*) hiện tại sẽ bị xoá. Có thể ngăn chặn các điều đó bằng cách gọi lệnh `hold on` sau lệnh `plot` đầu tiên.

```
figure;

subplot (121);

plot([-5:0.1:5],cos((-5:0.1:5)*pi), 'k:');

hold on;

fplot ('2*sin(x)', [-5 5]);

subplot(122);

t = (0:20)*0.9*pi;

plot(cos(t),sin(t));
```

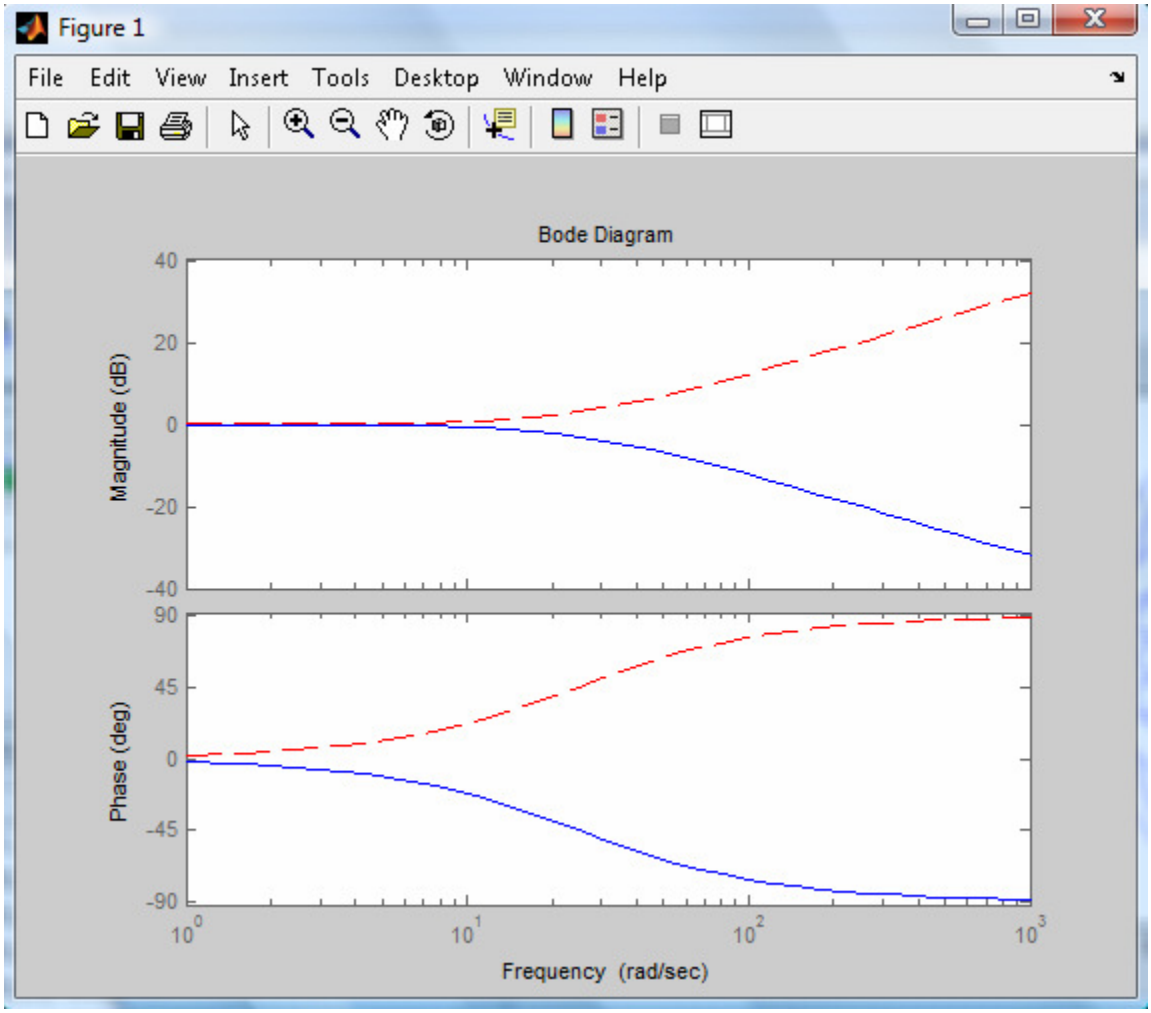


Lệnh `fplot`(*function, range*) trong ví dụ trên minh họa khả năng vẽ trực tiếp các hàm tường minh. Ngoài ra, Matlab còn tạo điều kiện vẽ các hàm không tường minh một cách dễ dàng nhờ lệnh `ezplot`(*function_1*, [*function_2*,] *range*).

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 thị BODE vẽ bằng lệnh bode:

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

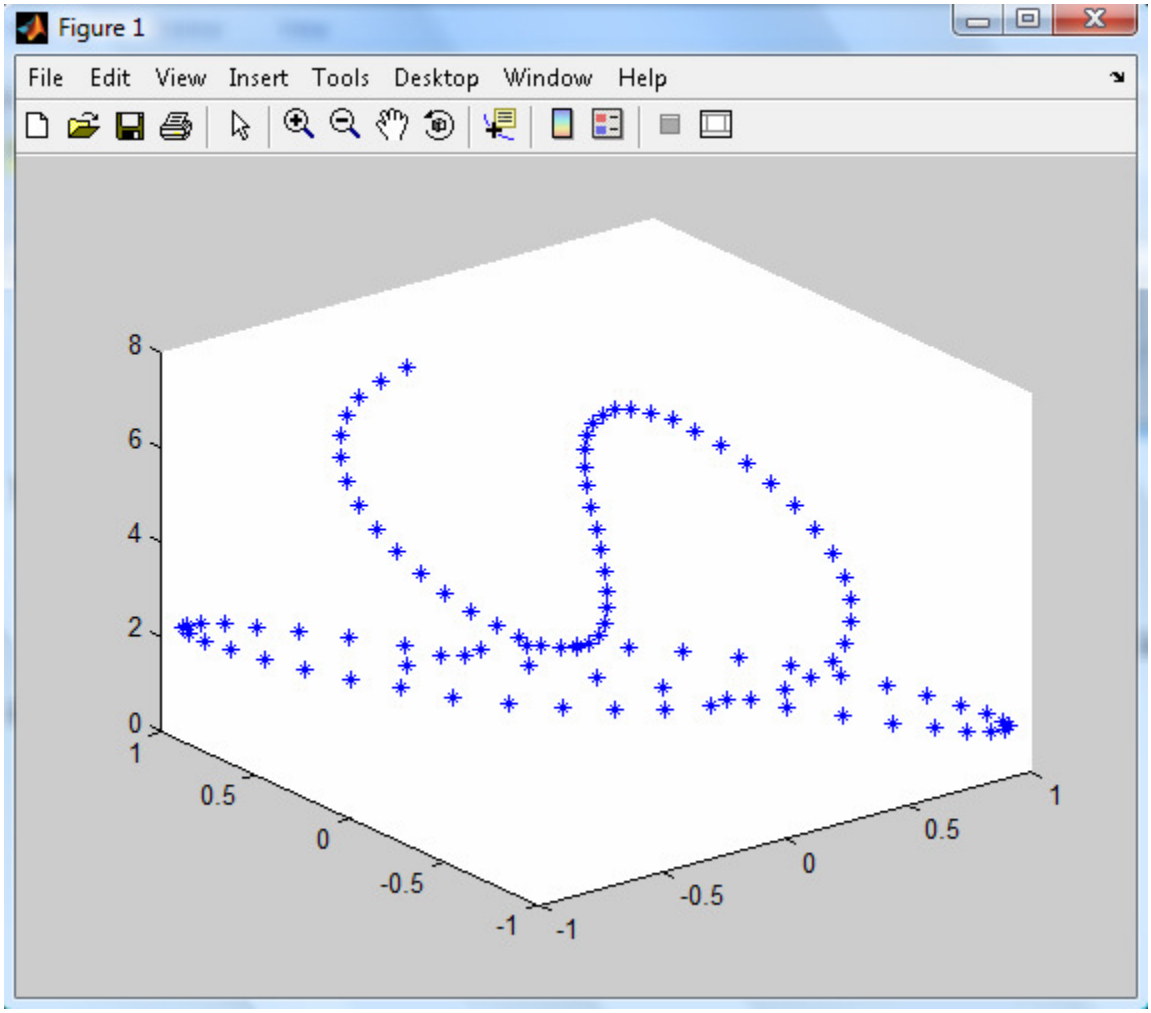


3.3 Đồ hoạ 3 chiều

3.3.1 Các lệnh Plots

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.

```
>> 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 hoạ 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 hang 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 hoạ 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 hoạ. 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*).

3.3.2 Phối cảnh trong đồ hoạ 3-D

Có thể dùng lệnh `view`(*horizontal,vertical*) để phối cảnh cho đồ hoạ 3 chiều bằng cách khai các góc theo phương nằm ngang và phương thẳng đứng tính bằng độ (^{o}, Degree). Góc chuẩn cho trước là (-37.5^o, 30^o). 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`.

3.3.3 Nhập, xuất và in đồ hoạ

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

```
>> anh = imread('anh1.jpg','jpeg');
>> image(anh)
```

Bằng lệnh `imread` ta gán *File* đồ hoạ với định dạng *fmt* cho biến *variable*. Nếu *variable* nhận hình ảnh chỉ bao gồm gam màu xám, *variable* sẽ là một biến 2 chiều. Nếu đó là hình ảnh mầu RGB, *variable* sẽ là một mảng 3 chiều. Định dạng của đồ hoạ được khai báo bởi *fmt*. Lệnh `image` (*variable*) sẽ xuất đồ hoạ mới gán cho *variable* ra màn hành *Figure* có chứa ảnh với định dạng *.jpg*

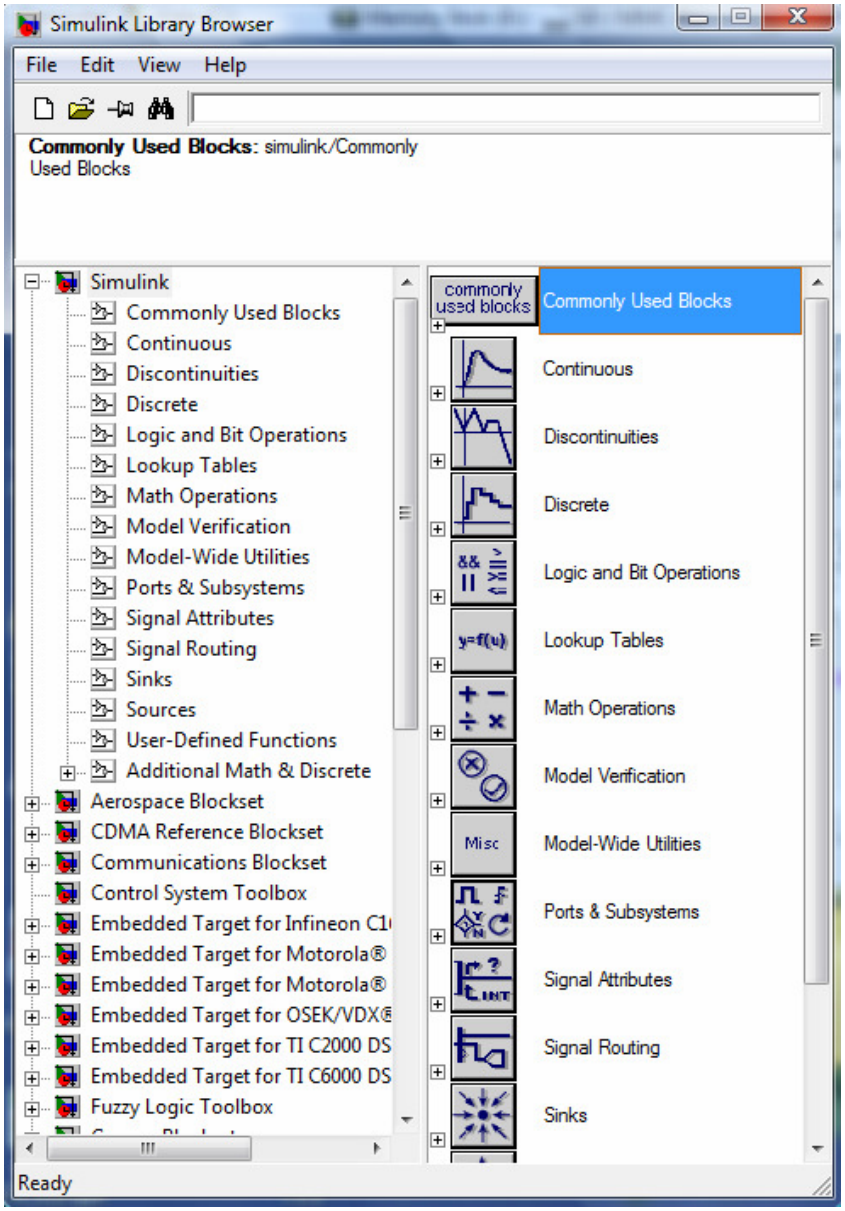
Đồ hoạ Figure của Matlab cũng có thể được xuất sang các định dạng khác. Lệnh `print -f`*number* sẽ in *Figure* mang số *number* ra máy in. Lệnh `print -f`*number* `-d`*fmt file* sẽ xuất *Figure* thành *file* với các định dạng đồ hoạ khác. Ví dụ: `bmp` (*Windows bitmap*), `emf` (*Enhanced meta*), `eps`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

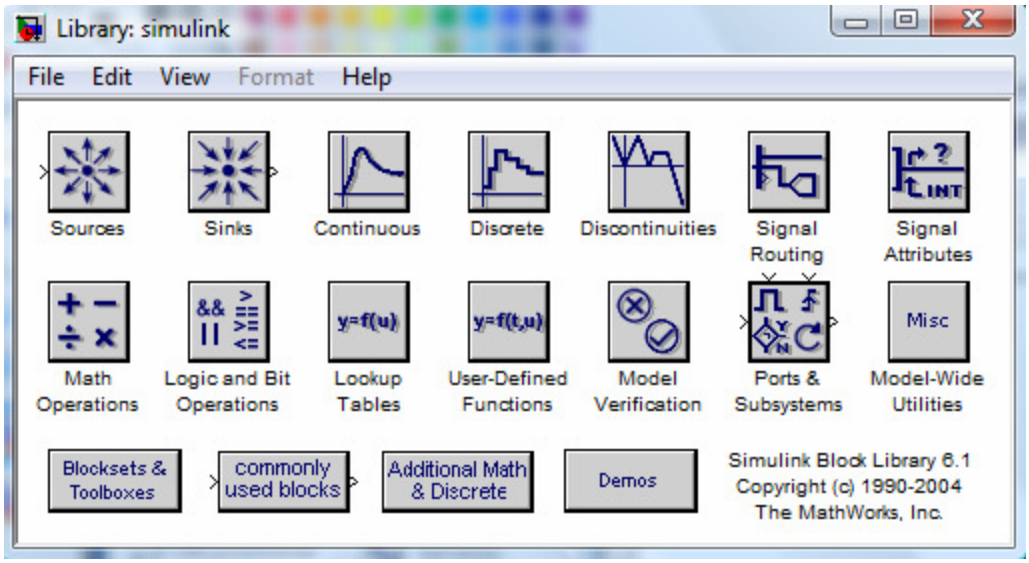
Nếu cần phải lưu lại để sau này xử lý, có thể cất các đồ hoạ đã thu được thành *File* với định dạng *fig* của Matlab. Để cất hoặc ta đi theo menu *File / Save as*, hoặc gọi lệnh `saveas`(*handle, 'file'* [,*format*]). Lệnh `saveas` cất *handle* (*Figure* hiện tại, có thể dùng `gcf` để hỏi) thành tệp có tên *file* với một trong các định dạng: `'fig'` (File nhị phân), `'m'` (gồm một *File fig* và một *File Script*).

CHƯƠNG IV
CƠ SỞ SIMULINK

4.1 Khởi động Simulink

Để có thể làm việc với Simulink, trước hết ta phải khởi động Matlab. Nếu chạy dưới hệ điều hành *Linux*, sau khi thực hiện lệnh `simulink3` ta sẽ thu được cửa sổ thư viện của *Simulink*. Nếu làm việc dưới Windows, sau khi gọi `simulink` ta có cửa sổ tra cứu thư viện như sau:

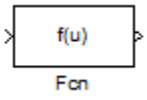




Các thư viện con *Source* (các khối nguồn tín hiệu), *Sinks* (các khối xuất tín hiệu), *Math* (các khối ghép nối toán học) và *Signals & Systems* (các khối tín hiệu và hệ con) sẽ được giới thiệu trong phạm vi chương này.

Tính chất của các khối chức năng

Tất cả các khối chức năng đều được xây dựng theo một mẫu giống nhau như sau:



Mỗi khối có một hay nhiều đầu vào/ra (trừ trường hợp ngoại lệ: các khối thuộc hai thư viện con *Source* và *Sinks*), có tên và ở trung tâm của hình khối chữ nhật có biểu tượng thể hiện đặc điểm riêng của khối. Người sử dụng có thể tuỳ 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 nhập xong, nháy chuột trái vào nút *OK* hay nút *Apply* 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 cho phép soạn thảo và lập định dạng khối.

Simulink phân biệt hai loại khối chức năng: Khối ảo (vitural) và khối thực (not vitural). 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 *Sum* hay khối *Product* của thư viện con *Math*. 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 hoạ của mô hình *Simulink*. Đó chính là các khối như *Mux*, *Demux*, hay *Enable* thuộc thư viện con *Signal & System*. Một số khối chức năng mang đặc tính ảo hay thực tuỳ thuộc theo vị trí hay cách thức sử dụng chúng trong mô hình *Simulink*.

Mô hình Simulink

Từ cửa sổ thư viện khối (L*ibrary*) hay từ cửa sổ truy cập thư viện (*Library Browser*) ta có thể tạo ra các cửa sổ mô phỏng mới bằng cách đi theo menu *File / New / Model*, hoặc mở các *File* có sẵn qua menu *File / Open*. Một *File Simulink* khi được cất giữ sẽ có đuôi *.mdl*.

4.2 Các thao tác cơ bản với Simulink

- Sao chép: Bằng cách gắp và thả *"Drag & Drop"* nhờ phím chuột phải là có thể sao 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: 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.

- Nối hai khối: Dùng phím chuột trái nháy vào đầu ra của một khối, sau đó di mũi tên của chuột tới đầu vào cần nối. Sau khu 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 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 tuỳ 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 và đường nối các tín hiệu theo định dạng vector, hoặc ma trận, hoặc mảng, ta có thể chọn menu *Format / Wide nonscalar lines* để tăng bề dầy của đường nối.

- Chỉ thị kích cỡ và dạng dữ liệu của tín hiệu: Lệnh chọn qua menu *Format / Signal dimensions* sẽ hiện thị kích cỡ của tín hiệu đi qua đường nối. Lệnh menu *Format / Port data types* chỉ thị thêm loại dữ liệu của tín hiệu qua đường nối.

- Định dạng 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 quang 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 sẽ mở ra. Tại đây có các lệnh cho phép cắt bỏ, chép hoặc xoá đường nối.

- Hộp đối thoại về đặc tính của khối: 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.

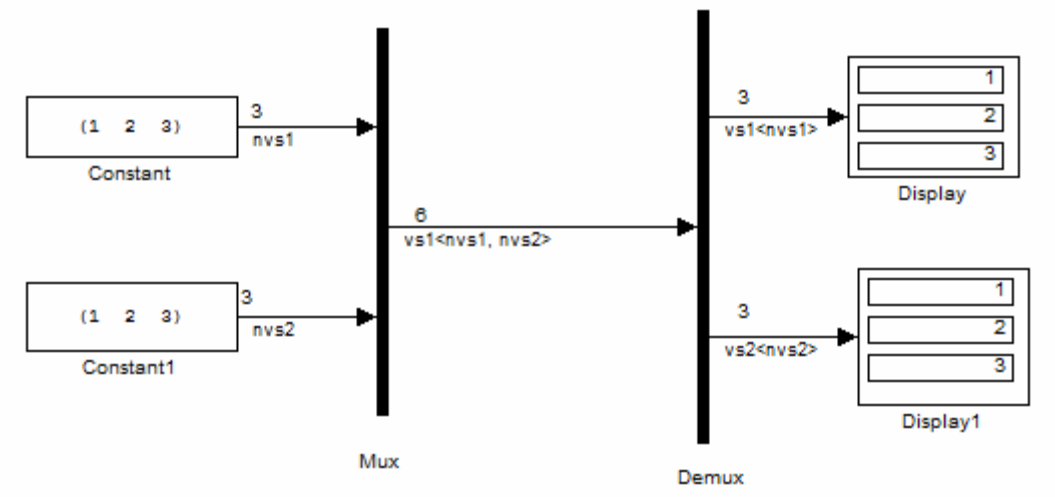
- Hộp đối thoại về đặc tính của tín hiệu: Có thể tới được hộp thoại *Signal Properties* của một đường nối hoặc bằng cách nháy chuột đánh dấu đường nối trên cửa sổ mô phỏng, sau đó đi theo menu *Edit / Signal Properties* từ cửa sổ định dạng đường. Trong hộp đối thoại ta có thể đặt tên cho đường nối một cách đơn giản hơn: Nháy kép phím chuột trái vào đường nối ta sẽ tự động tới được chế độ nhập văn bản.

4.3 Tín hiệu và các loại dữ liệu

4.3.1 Làm việc với tín hiệu

Trong *Simulink* ta phân biệt ba loại kích cỡ tín hiệu:

- Tín hiệu đơn.
- 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.



Khi tạo Simulink, các khối ảo sẽ tạo nên các đường tín hiệu ảo, 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 51hem51à 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 nào được ghép vào 51hem tín hiệu ảo.

Đối với các tín hiệu ảo ta có thể mở hộp thoại *Signal Properties* và khai chọn *Show propagated signals*. Sauk hi khai chọn tên của tín hiệu sẽ tự động được bổ sung 51hem phần trong ngoặc < >, cho biết các tín hiệu chứa trong đó.

Bus tín hiệu là tập hợp các tín hiệu ảo riêng rẽ. Khi tách Bus bởi bộ phận 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.

4.3.2 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*: chính xác cao, dấu phẩy động.

- *single*: chính xác vừa, dấu phẩy động.
- *int8, uint8, int16, uint16, int32, uint32*: số nguyên 8-, 16- hay 32- bit có / không có dấu.
- *52oolean*: biến logic 0 hoặc 1.

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 có đúng hay không, nhằm loại trừ các kết quả sai lầm có thể xảy ra.

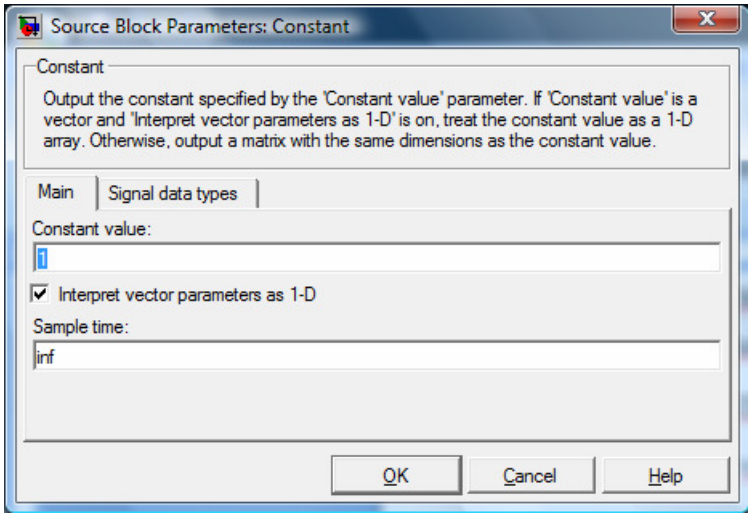
Khả năng khai báo, xác định loại số liệu của tín hiệu cũng như của tham số thuộc các khối chức năng trong Simulink là đặc biệt có ý nghĩa, nếu ta dự định tạo ra từ mô hình Simulink mã chạy cho các ứng dụng thời gian thực. Nhu cầu về bộ nhớ và tốc độ tính toán phụ thuộc vào loại số liệu được ta chọn.

4.4 Thư viện Sources và Sinks

4.4.1 Thư viện Sources

a) Constant

Khối *constant* tạo nên một hằng số thực hoặc phức, hằng số có thể là scalar, vector hay ma trận tuỳ 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 hang hay cột với kích cỡ [1×n] hay [n×1] dưới dạng ma trận. Nếu ô đó không được chọn, các vector hang cột đó chỉ được sử dụng như vector với chiều dài n, tức là tín hiệu 1-D.



b) Step và Ramp

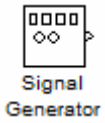
Nhờ hai khối *Stemp và Ramp* ta có thể tạo nên các tín hiệu dạng bậc thang hay dạng dốc tuyến tính, 53at để kích thích các mô hình Simulink. Trong đó hộp thoại *Block Parameters* của khối Step ta có thể khai báo giá trị đầu / giá trị cuối và cả thời điểm bắt đầu của tín hiệu bước nhẩy. Đối với Ramp ta có thể khai báo độ dốc, thời điểm và giá trị xuất phát của tín hiệu ở đầu ra.

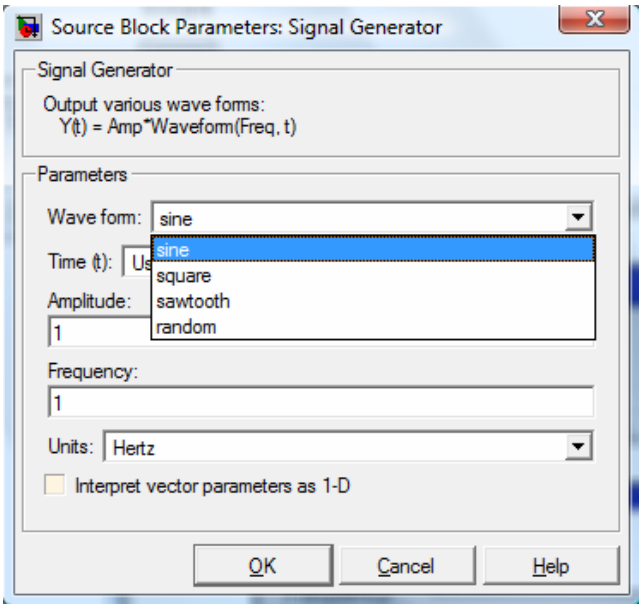
Đối với cả hai khối, ta có thể sử dụng tham số tuỳ chọn *Interpret vector parameters as 1-D* để quyết định các tín hiệu dạng bước nhẩy hay dạng dốc tuyến tính có giá trị scalar hay vector hay ma trận.

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.

c) Signal Generator và Pulse Generator

Bằng *Signal Generator* ta tạo ra các dạng tín hiệu kích thước khác nhau.





Cung cấp cho 4 dạng song khác nhau (giống như máy phát 54at):

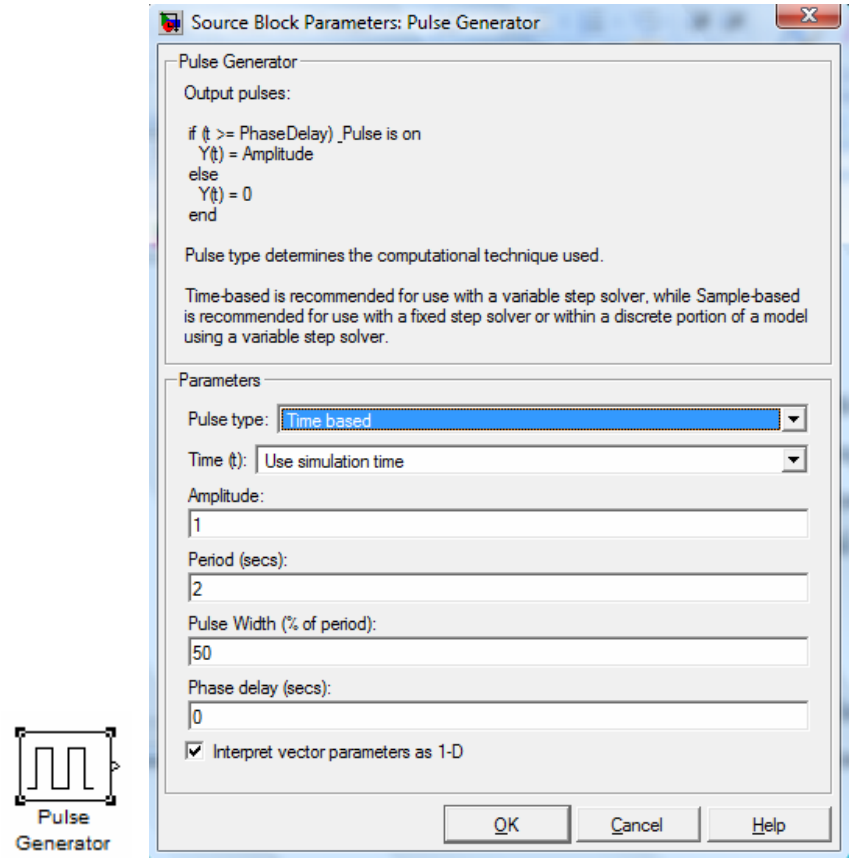
+ Sóng Sin

+ Sóng vuông (Square)

+ Sóng răng cưa (Sawtood)

+ Sóng ngẫu nhiên (Random)

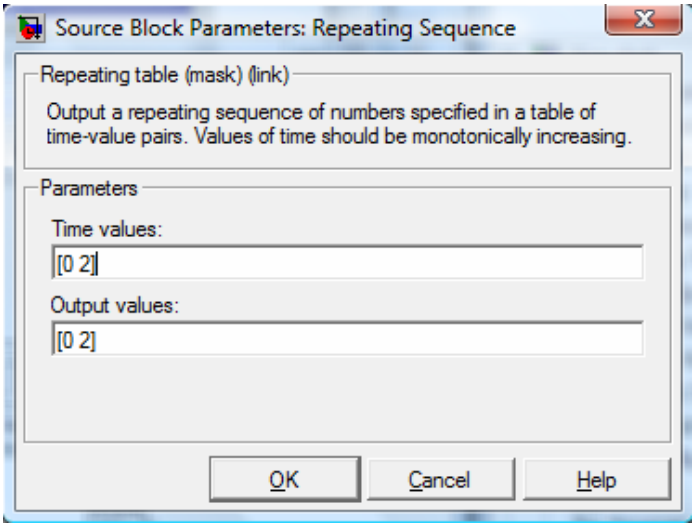
Với *Pulse Generator* tạo chuỗi xung hình chữ nhật. Biên độ và tần số có thể khai báo tuỳ ý. Đối với *Pulse Generator* ta còn có khả năng chọn tỷ lệ cho bề rộng xung (tính bằng phần trăm cho cả chu kỳ). Đối với cả hai khối ta có thể sử dụng tham số tuỳ chọn *Interpret vector parameters as 1-D* để quyết định các tín hiệu có giá trị scalar hay vector ma trận.



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.

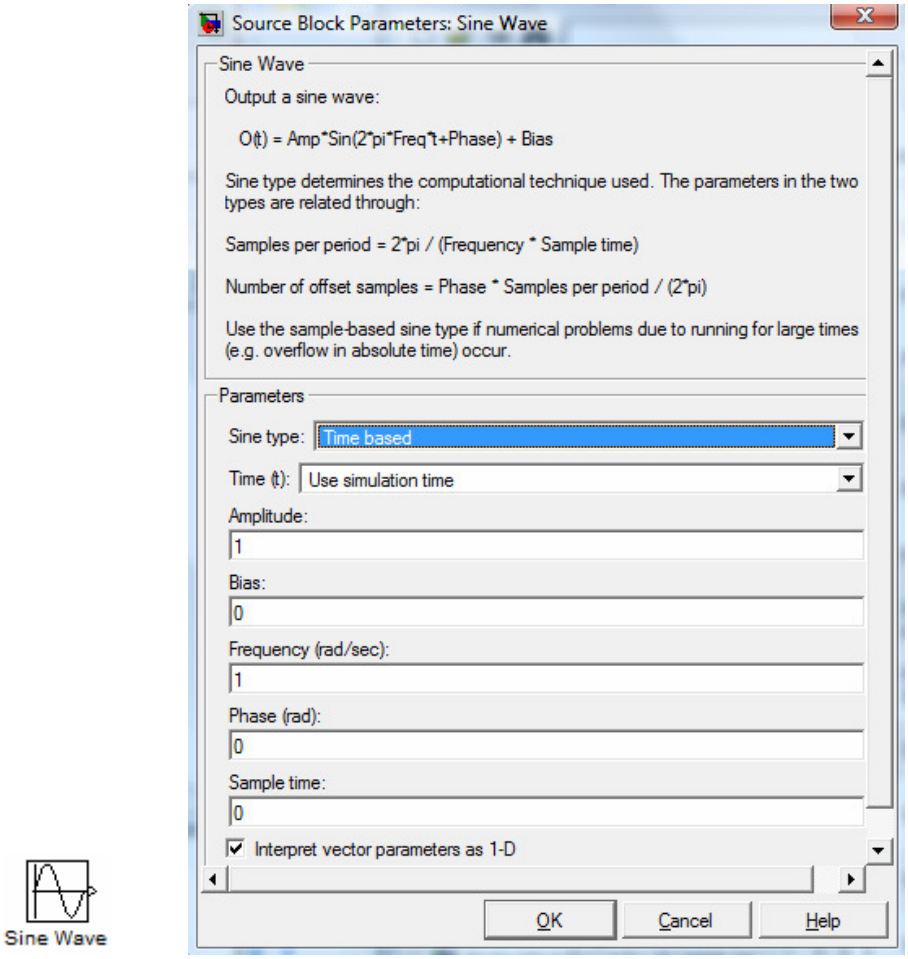
d) Repeating Sequence

Khối *Repeating Sequence* cho phép ta tạo nên một tín hiệu tuần hoàn tuỳ 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 phù hợp với chiều dài của tham số *Time values*. Giá trị lớn nhất của vector thời gian quyết định chu kỳ lặp lại của vector biến ra.



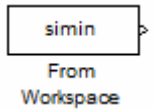
e) Sine Wave

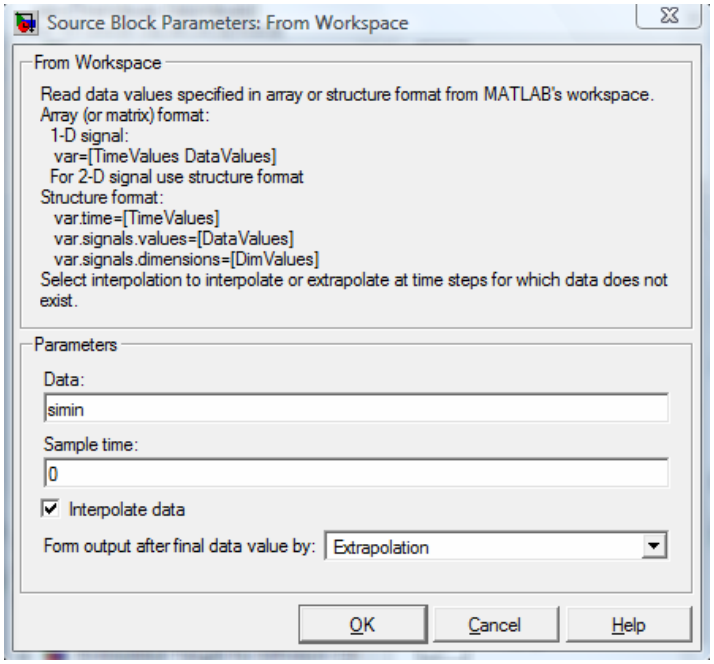
Khối *Sine Wave* được sử dụng để tạo tín hiệu hình sin cho cả hai loại mô hình: liên tục (tham số *Simple time = 0*) và gián đoạn (tham số *Simple time = 1*). Tín hiệu đầu ra y phụ thuộc vào 56at ham số chọn: *Amplitude, Frequency và Phase* trên cơ sở quan hệ *y = Amplitude*.`sin`(*Frequency*.`time` + *Phase*). Vì 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 tuỳ chọn *Interpret vector parameters as 1-D* để quyết định các tín hiệu có giá trị calar hay vector hay ma trận.



f) From Workspace

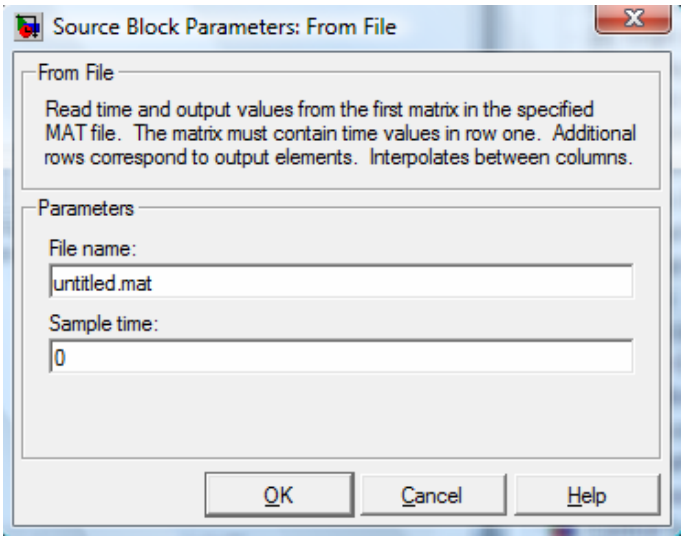
Khối *From Workspace* có nhiệm vụ lấy số liệu từ cửa sổ Matlab Workspace để cung cấp cho mô hình Simulink. Các số liệu lấy vào phải có dạng của biểu thức Matlab, khai báo tại dòng *Data*.





g) From File

Bằng khối From File ta có thể lấy số liệu từ một MAT-File 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.

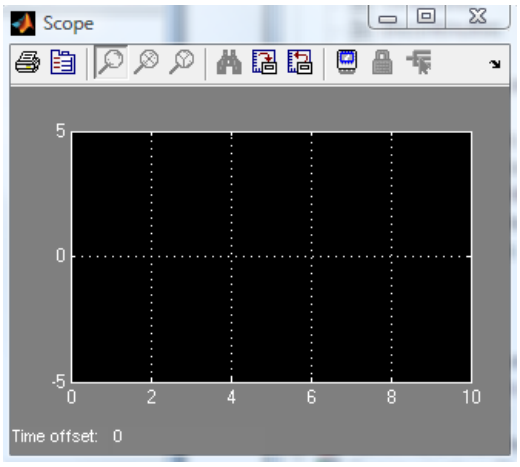


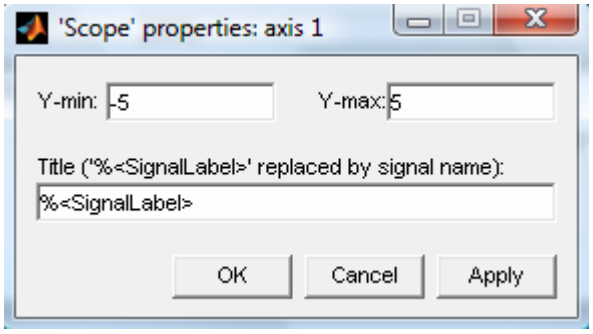
4.4.2 Thư viện Sinks

Thư viện này bao gồm các khối xuất chuẩn của Simulink. Ngoài khả năng hiển thị đơn giản bằng số, còn có các khối dao động kí để biểu diễn các tín hiệu phụ thuộc thời gian hay biểu diễn hai tín hiệu trên hệ toạ độ XY.

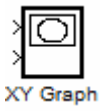
a) Scope

Nhờ khối Scope ta có thể hiển thị các tín hiệu của quá trình mô phỏng. Khi nhấn vào nút Properties, hộp thoại Scope Properties (đặc điểm của Scope) sẽ mở ra. 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 giống tương tự như lệnh Subplot của Matlab. Nếu điền một số cụ thể vào ô time range, đồ thị sẽ chỉ được biểu diễn tại thời điểm do giá trị của số xác định.





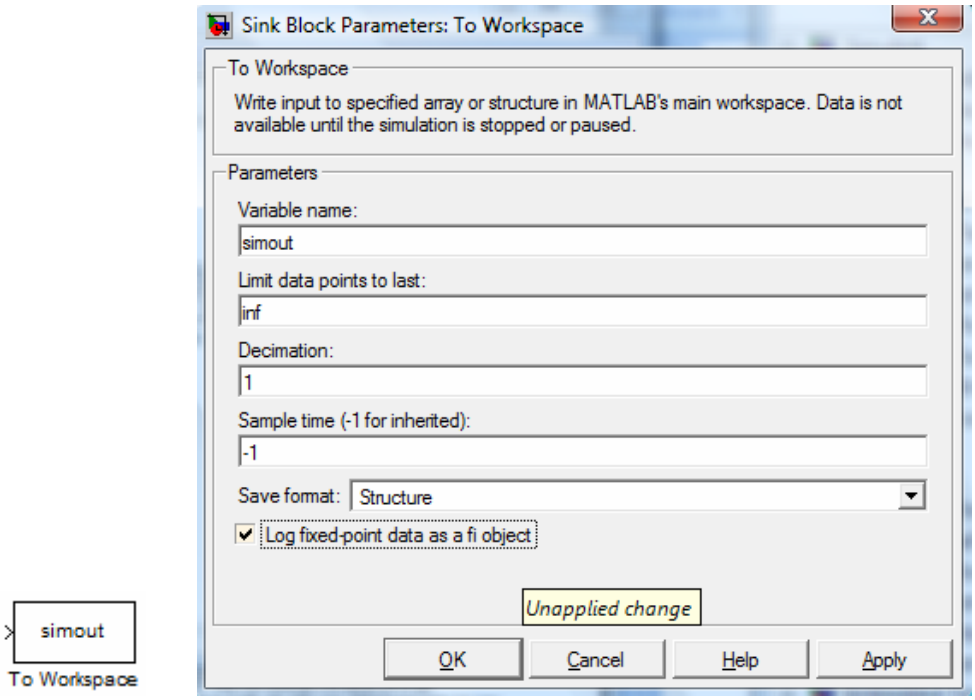
b) XY Graph



Khối này biểu diễn hai tín hiệu đầu vào trên hệ toạ độ XY dưới dạng đồ hoạ Matlab đầu vào thứ nhất (bên trên). Ứng với trục X đầu thứ hai ứng với trục Y.

c) To Workspace

Khối To Workspace gửi số liệu ở đầu vào của khối tới môi trường Matlab Workspace dưới dạng mảng (Array), Stracture hay Stracture with time và lấy chuỗi kí tự khai tại variable name để đặt tên cho tập số liệu được ghi.



d) To File

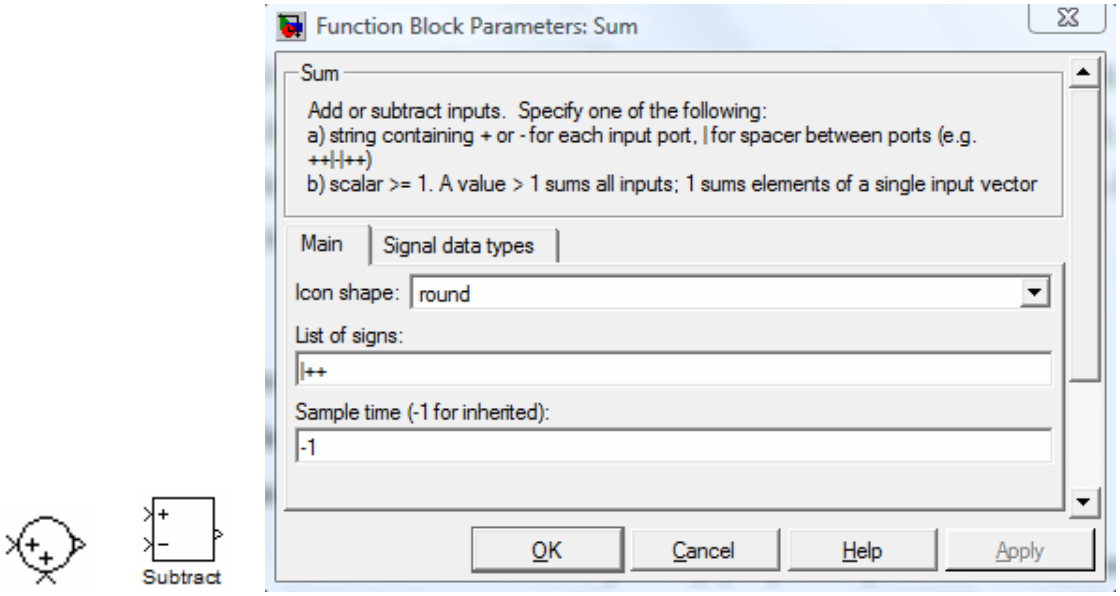
Khối này giúp ta cất tập số liệu (mảng hay ma trận) ở đầu vào của khối cùng với vector thời gian dưới dạng Mat-File. Array định dạng giống như định dạng mà khối From File cần, vì vậy số liệu do To File cất có thể được From File đọc trực tiếp mà không cần phải xử lý gì.

4.5 Thư viện Math

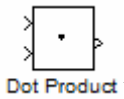
Thư viện này có một số khối có chức năng ghép toán học các tín hiệu khác nhau, có những khối đơn giản chỉ nhằm cộng hay nhân tín hiệu còn có các hàm phức tạp như lượng giác và logic … Sau đây ta xét chức năng của một số khối quan trọng trong thư viện này.

a) Sum

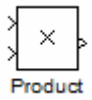
Tín hiệu ra của khối Sum là tổng các tín hiệu đầu vào (Ví dụ như tín hiệu đầu vào là các tín hiệu hình Sin thì tín hiệu đầu ra cũng là các tín hiệu hình Sin). Khối Sum cũng có thể tính tổng từng phần tử (ví dụ tín hiệu vào gồm hai tín hiệu: Sin(x) và [5 9 3] thì tín hiệu ra sẽ có dạng [Sin(x)+5 Sin(x)+9 Sin(x)+3]



b) Product và Dot Product



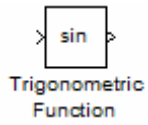
Khối Product thực hiện phép nhân từng phần tử hay ma trận cũng như phép chia giữa các tín hiệu vào (dạng 1- D hay 2 – D) của khối, ví dụ: nếu một khối Product có tham số Number of Inputs = */*, với ba tín hiệu vào là 5, sinx và [4 4 5 6] khi ấy tín hiệu đầu ra có dạng [20/Sinx 20/Sinx 25/Sinx 30/Sinx].



Khối Dot Product tính tích vô hướng của các Vector đầu vào. Giá trị đầu ra của khối tương đương với lệnh Matlab y = Sum(cọn(u_1)*$u_{2)}$.

c) Math Function và Trigonometric Function

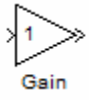
Cả hai khối này đều có thể xử lý tín hiệu 2-D. Khối Math Function có một lượng lớn các hàm toán đã được chuẩn bị sẵn cho phép ta lựa chọn theo nhu cầu sử dụng.

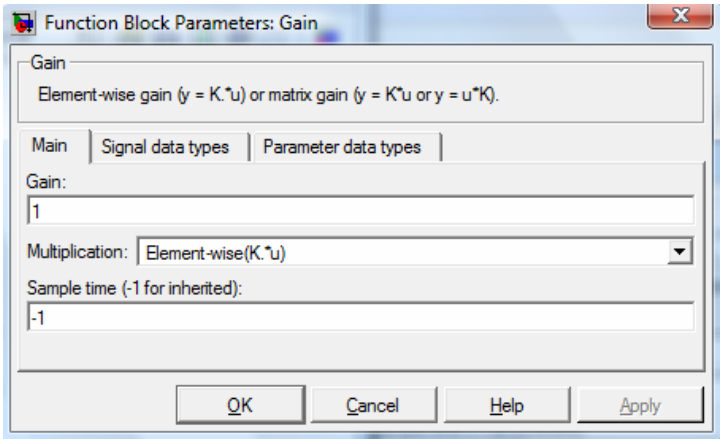


Còn khối Trigonometric Function có tất cả các hàm lượng giác quan trọng.

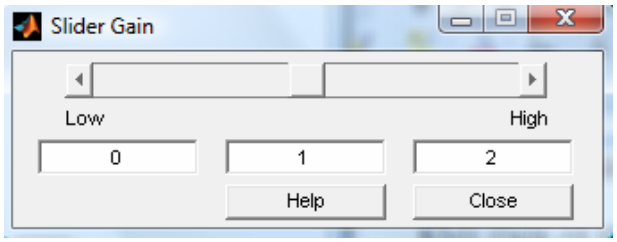
d) Gain và Slider Gain

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 chỉ có thể là một biến hay một số biến. Biến đó phải tồn tại trong môi trường Matlab Workspace thì khi ấy Simulink mới tính toán được biến.





Khối Slider Gain cho phép thay đổi hệ số khuếch đại vô hướng trong quá trình mô phỏng.



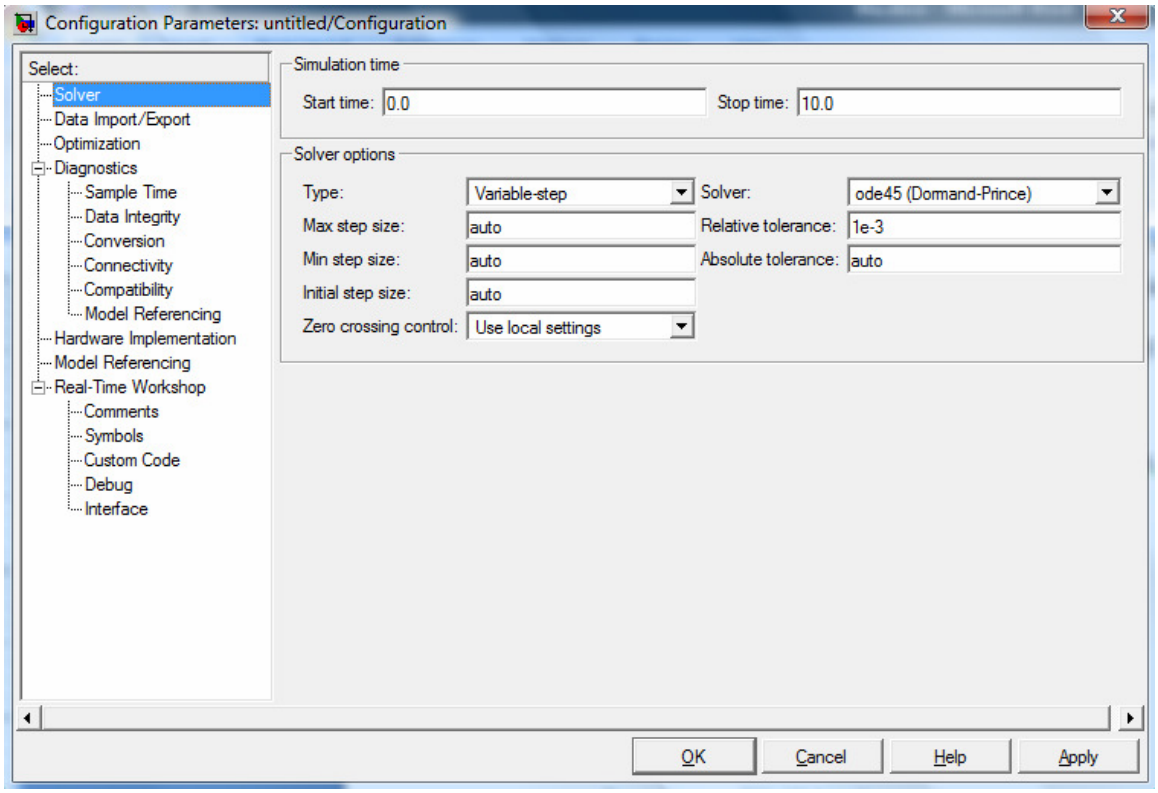
4.6 Khai báo tham số và phương pháp tích phân chuẩn bị cho mô phỏng.

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. Tại đó tất cả các tham số đều đã có một giá trị mặc định sẵn, nghĩa là: Có thể khởi động 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 các trang:

a) Solver (thuật toán)

Tại trang này 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.



Simulink cung cấp cho ta một số thuật toán khác nhau để giải bằng số phươngt rình vi phân. 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).

Thuật toán Variable-step làm việc với bước tích phân linh hoạt. Việc giải các phương trình vi phân 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 Intial step size. Trong quá trình mô phỏng, Simulink sẽ cố gắng giải phương trình vi phân bằng bước cho phép lớn nhất ghi tại Max step size. Kích cỡ Max step size có thể tính như sau:

$$Max\ step\ size = \frac{Stop\ time - Start\ time}{50}$$

Do có khả năng thích nghi bước tích phân, thuật toán Slover với Variable-step có thể giám sát biến thiên 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 như các đột biến dạng bước nhảy.

b) 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. Cứ mỗi bước tích phân, thuật toán Solver (dạng Variable-step) lại kiểm tra xem local error của mỗi biến trạng thái có thoả mãn điều kiện acceptable error (sai số có thể chấp nhận) được xác định bởi tham số Relative tolerance và Absolute tolerance ở hộp thoại Simulation Parameters (viết tắt là eltol và abstol). Điều kiện acceptable error được mô tả bằng công thức sau:

$$e_i \leq \underbrace{max\ (\ reltol.|x_i|, abstol)}_{\text{acceptable error}}$$

Nếu một trong số các biến trạng thái không thoả 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 có nguyên do như sau:

Giá trị khai báo tai 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 relation tolerance có thể trở nên quá bé, đồng nghĩa với việc: Biến trạng thái không được phép biến thiên gì nữa. Điều này 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

c) 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 sẽ 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 bắt đầu tính tiếp từ cận phải. Vì vậy, nếu chọn sai số quá thô sẽ có nguy cơ bỏ sót các điểm không. Nếu có nghi vấn bỏ sót điểm không, cần phải giảm sai số đã khai báo để đảm bảo là Solver với Variable-step sẽ chọn bước tính đủ nhỏ.

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 cài đặt mô hình (sau khi mô phỏng thành công) trên một cấu hình Hardware nào đó.

d) 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 rat a có thể khai báo giá trị ban đầu cho các biến trạng thái tại đây.

Input: tên của các tập số liệu cần đọc từ Workspace, các tập số liệu có thể định dạng Array, Structure và Structure with time.

Initial State: 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 quan trọng 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.

Biến ra của mô hình Simulink được cất bằng cách điền tên biến ra vào ô Output, sau khi đã kích hoạt Output. Tương tự ô State để cất biến trạng thái vào Workspace

e) Advance (khai báo nâng cao)

Sauk hi nhấn nút Configure của ô Inline parameters, ta thu được cửa sổ mới để khai báo cấu trúc tham số của mô hình. 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. Vì những tham số không thay đổi được sẽ bị coi là hằng số, thời gian sẽ giảm đi đáng kể.

4.6.1 Khởi động và ngừng mô phỏng

Quá trình mô phỏng của mô hình Simulink được khởi động qua menu Simulation/Start. Trong khi mô phỏng, có thể chọn Simulation/Pause để tạm ngừng, hay Simulink/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 của Matlab. Điều này đặc biệt có ý nghĩa khi ta muốn tự

động hoá toàn bộ các chu trình mô phỏng, không muốn khởi động, ngừng hay xử lý, … bằng tay. Đó là các lệnh set_param và sim.

- Lệnh set_param được gọi như sau:

set_param('sys','SimulinkCommand','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, tar a lệnh tiếp tục mô phỏng bằng continue. Nếu chọn cmd = update, mô hình sẽ được cập nhật mới

- Lệnh sim được gọi như sau:

[t,x,y] = sim('model')

Nếu muốn chuyển giao cả tham số mô phỏng, ta 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. 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 option 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,… Việc tạo cấu trúc tham số options được thực hiện bằng lệnh :

options = simset (property, value, …)

Với lệnh trên, các tham số đặt trong hộp thoại Simulation Parameters sẽ không bị thay đổi, mà chỉ bị vô hiệu hoá 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 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 = simset('model')

ta sẽ thu được trọn vẹn bộ tham số options đã được khai báo nhờ lệnh simser hay nhờ hộp thoại Simulations Parameters.

Ví dụ lệnh :

[t,x,y] = sim('model', [],simset(simget('model'), … 'slover', 'ode23', 'MaxStep', 0.01) ;

sẽ đưa thuật toán tích phân của sơ đồ Simulink có tên model về ode23 với bước lớn nhất là 0,01 giây. Tại vị trí của timespan ta viết [], nghĩa là : các giá trị của Start time và Stop time ở hộp thoại Simulation Parameters được giữ nguyện.

4.6.2 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.

Trong phần phía trên của hộp thoại báo lỗi ta thấy có danh sách các khối gây nên lỗi. Khi chuyển vạch chọn tới khối nào, 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 tat hay đổi, sửa lại các tham số khai báo tại đó. Đôi khi nguồn gây lỗi trên sơ đồ còn được tôn nổi bật 68hem bằng mầu, giúp ta nhanh chóng xác định được vị trí của khối gây lỗi.

4.6.3 Tập hợp các tham số trong Script cửa 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 rang, 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 để kích hoạt một script chứa các tham số mô hình, là việc sử dụng các thủ tục Callback. Khả năng này cho phép ta tiết kiệm, không cần mất công gọi script đó bằng dòng lệnh trong cửa sổ lệnh. 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 trước khi đọc Block Parameters.

Ví dụ:

Set_param(‘model’, ‘InitFcn’, ‘model_init’)

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 huỷ 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 biệt 69hem về lệnh set_param và Callback Routines ta gọi lệnh help set_param.

4.6.4 In mô hình Simulink

Cũng giống như đồ hoạ Matlab, ta có thể xuất mô hình Simulink dưới các dạng khác nhau. Bằng lệnh print –smodel ta sẽ xuất mô hình có tên model ra máy

in. Tuy nhiên, nếu in qua menu File/Print ta sẽ có nhiều khả năng khai báo tham số in hơn. Ví dụ: Chỉ in một tầng mô hình 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 hoạ Matlab:

Print –smodel;

Print –smodel –dmeta model;

Print –smodel –deps model;

4.7 Hệ thống con (Sub system)

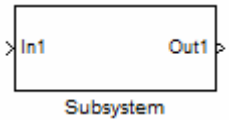
4.7.1 Tạo hệ thống con

Có hai cách để tạo hệ thống con:

- Cách 1: Dùng chuột đánh dấu tất cả các khối 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 menu 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 của khối và lần lượt gắp các khối cần thiết để tạo thành hệ thống con. Việc ghép nối với hệ thống mẹ 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).

4.7.2 Thư viện signals và Subsystem

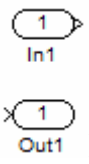
Subsystem



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 ghép trên được thực hiện nhờ khối Inport và Outport. 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 menu 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 với ý nghĩa của chúng.

Inport và Outport

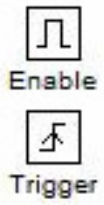


Inport và Outport là các khối đầu vào, đầu ra của một mô hình mô phỏng. Tại hộp thoại Block Parameters ta có thể điền vào ô Port number số thứ tự của khối. Simulink tự động đánh số các khối Inport và Outport một cách độc lập với nhau, bắt đầu từ 1. Khi ta bổ sung thêm khối Inport hay Outport, khối mới sẽ nhận số thứ tự kế tiếp. Khi xoá một khối nào đó, các khối còn lại sẽ 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 tới một vài tham số quan trọng khác của khối Outport. Ví dụ, Outport when disabled cho hệ thống cần xử lý tín hiệu ra như thế nào khi hệ thống mô phỏng đang ngừng không chạy (xoá về không hay giữ nguyên giá trị cuối cùng). Initial Output cho biết giá trị cần 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 tên của các biến cần lấy số liệu vào, hay tên của các biến mà ta sẽ gửi số liệu tới

Enable và 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 thống 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 hay Trigger được đưa tới.

Các hệ con có khối Enable được gọi là hệ 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 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 do ta tự viết.

Các khối Enable và Trigger là khối ảo có điều kiện.

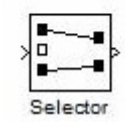
Mux và Demux

Khối Mux có tác dụng giống như một bộ chập kênh, có tác dụng chập các tín hiệu 1-D riêng rẽ thành một vector tín hiệu mới. Nếu như một trong số các tín hiệu riêng rẽ là 2-D, khi ấy ta chỉ có thể tập hợp các tín hiệu riêng rẽ thành Bus tín hiệu. 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 hai có bề rộng là 3, còn đầu vào thứ ba chưa xác định vì giá trị khai là -1.

Khối Demux có tác dụng ngược lại với Mux: Tách các 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 vecter 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.

Bus Selector và Selector

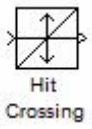


Các tín hiệu do khối Mux chập lại, có thể được tách ra không chỉ bằng khối Demux. Ta có thể sử dụng khối Bus Selector để tái tạo lại các tín hiệu từ một Bus tín hiệu, đồng thời gom chúng lại thành các tín hiệu riêng rẽ ban đầu.

Tại hộp thoại Block Parameters của khối Bus Selector 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.

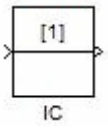
Khối Selector cho ta khả năng lựa chọn linh hoạt hơn Bus Selector: Khả năng tách ra khỏi Bus tín hiệu 1-D hay 2-D các phần tử riêng lẻ rồi sau đó gom chúng lại thành một tín hiệu 1-D hay 2-D mới.

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.

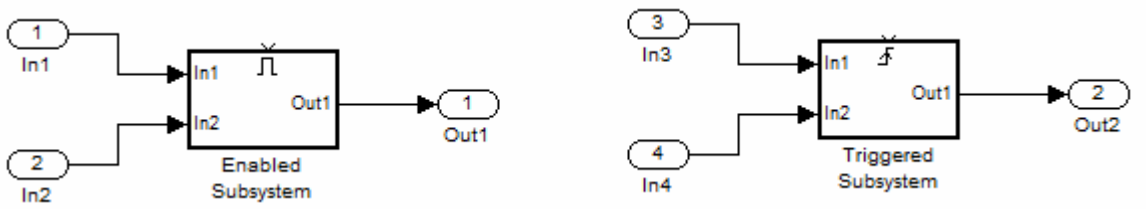
IC



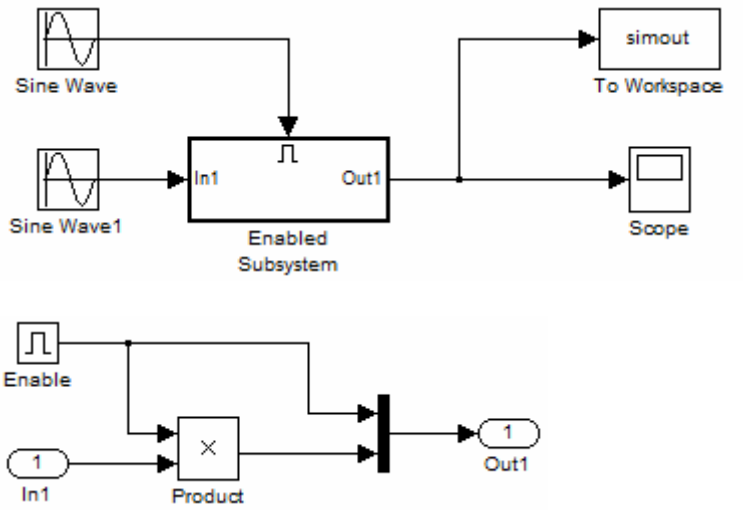
Khối IC có tác dụng gán cho tín hiệu ra của khối một điều kiện ban đầu nhất định.

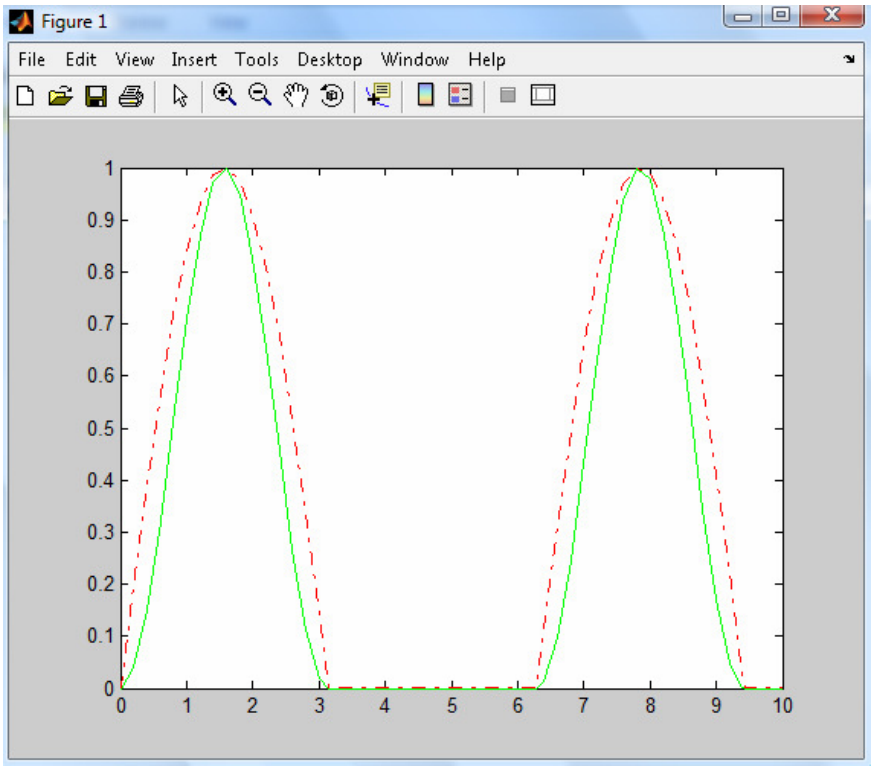
4.7.3 Kích hoạt có điều kiện các hệ thống con

Các hệ thống con có chứa khối Enable hay Trigger gọi là các hệ 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 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.



Ví dụ tiếp theo 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 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.

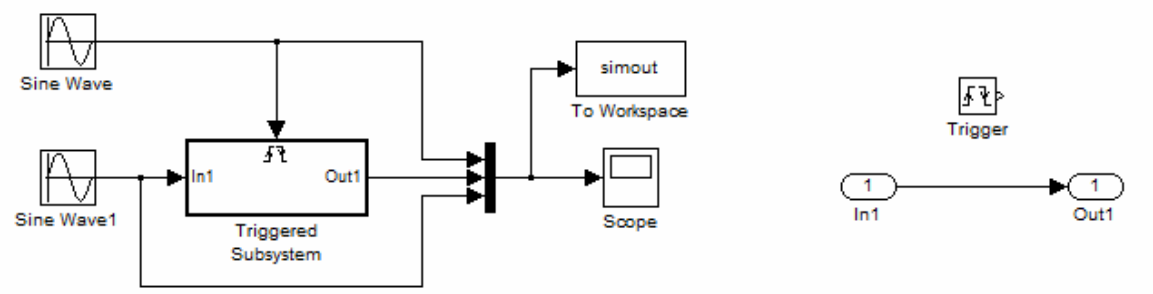


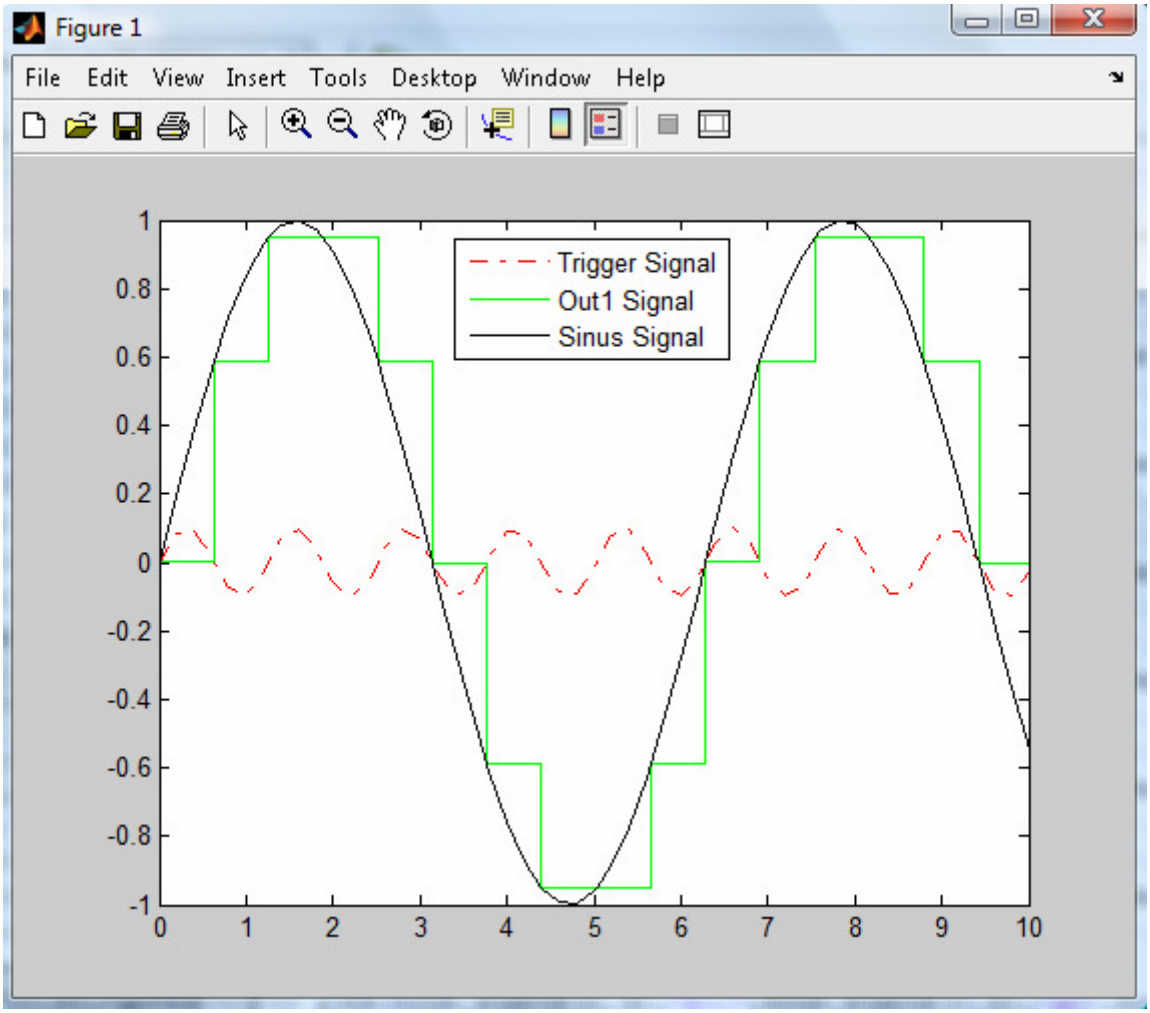


Sau khi chạy enable1.mdl, để thu được đồ thị trên ta cần thực hiện các dòng lệnh sau:

```
>> plot(tout,simout(:,1),'r-.',tout,simout(:,2),'g-');
>> legen('Signal Enable','Signal Enable x Sinus');
```

Ví dụ tiếp theo sẽ minh hoạ 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 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. Tượng tự ví dụ trên, 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.





Sau khi chạy trigger1.mdl, tức là khi: Kết quả mô phỏng đã được cất vào Workspace, ta thực hiện chum lệnh dưới đây để thu được đồ thị như hình trên.

```
>> plot(tout,simout(:,1),'r-.',tout,simout(:,2),'g-');
>> legend('Trigger Signal','Out1 Signal','Sinus Signal');
```

TÀI LIỆU THAM KHẢO

[1] Angermann, A.; Beuschel, M.; Rau, M.; Wohlfarth, U.: Simulation mit SIMULINK/MATLAB: Skriptum mit Ubungsaufgaben. Stand: 29. November 2001, TU munchen:

[2] Bishop, R. H.: Modern control systems analysis and disign using MATLAB. Addison - Wesley, 1993.

[3] Nguyễn Phùng Quang MATLAB & SIMULINK dành cho kỹ sư điều khiển tự động - Nhà xuất bản khoa học và kỹ thuật.