Chương 2: Sử dụng Symbolic Math Toolbox trong Matlab

Viện Toán ứng dụng và Tin học, ĐHBK Hà Nội

Hà Nội, tháng 8 năm 2015

Nội dung

Phần mềm "Symbolic Math Toolbox" kết hợp tính toán "symbolic" vào môi trường số của phần mềm Matlab. Các công cụ này bổ sung cho khả năng tính toán số học và đồ họa của Matlab thêm một số dạng của tính toán toán học, được tóm tắt dưới bảng sau:

Tiện ích	**Nội dung**
Giải tích (Calculus)	Các phép tính đạo hàm, tích phân, giới hạn, tổng và khai triển chuỗi Taylor
Đại số tuyến tính (Linear Algebra)	Nghịch đảo, định thức, giá trị riêng, SVD và dạng chính tắc của các ma trận symbolic
Rút gọn (Simplification)	Các phương pháp rút gọn biểu thức đại số
Nghiệm của phương trình (Solutions of Equations)	Nghiệm symbolic và nghiệm số của phương trình đại số và phương trình vi phân
Các hàm toán học đặc biệt (Specials Mathematical Functions)	Các hàm đặc biệt trong toán học ứng dụng cổ điển
Các phép biến đổi (Transforms)	Fourier, Laplace, z và các dạng biến đổi ngược tương ứng.

Ví dụ sau minh họa sự khác nhau giữa một dữ liệu chuẩn của Matlab, ví dụ `double` và đối tượng symbolic tương ứng.

Ví dụ 1

Câu lệnh Matlab:

```
>> sqrt(2)
```

cho kết quả là một số

```
ans =
    1.4142
```

Mặt khác, câu lệnh:

```
>> a=sqrt(sym(2))
```

cho kết quả

```
a=
2^(1/2)
```

Chú ý 1.1

- Matlab cho kết quả `2^(1/2)` nghĩa là $2^{1/2}$, bằng cách sử dụng ký hiệu symbolic cho phép toán căn bậc hai, mà không tính toán giá trị số cụ thể.
- Matlab lưu biểu thức symbolic này dưới dạng `string` thay thế cho $2^{1/2}$. Ta có thể nhận được giá trị số của đối tượng symbolic bằng cách dùng lệnh `double`:

```
>> double(a)
ans =
     1.4142
```

Chú ý 1.1 (tiếp)

- Khi ta tạo một phân số dạng `symbolic`, Matlab sẽ lưu tử số và mẫu số. Ví dụ

```
>> sym(2)/sym(5)
 ans =
 2/5
```

- Matlab thực hiện các phép tính trên các đối tượng `symbolic` khác với trên các kiểu dữ liệu chuẩn. Ví dụ:

```
>> 2/5+1/3
ans =
    0.7333
```

```
>> sym(2)/sym(5)+sym(1)/sym(3)
 ans =
 11/15
```

sym và syms

Cho phép ta xây dựng, biến đổi các số, biến và đối tượng thành symbolic.

Ví dụ 2

Lệnh

```
>> x=sym('x')
>> a=sym('alpha')
```

tạo ra các biến symbolic x hiển thị bởi x và a hiển thị bởi alpha.

Ví dụ 3

Giả sử ta muốn dùng symbolic để biểu diễn "tỷ lệ vàng" $\rho = \frac{1+\sqrt{5}}{2}$ bằng lệnh:

```
>> rho=sym('(1+sqrt(5))/2')
```

Bây giờ ta có thể thực hiện các phép toán khác nhau với rho. Ví dụ

```
>> f=rho^2-rho-1
  f =
  (5^(1/2)/2 + 1/2)^2 - 5^(1/2)/2 - 3/2
```

Sau đó rút gọn biểu thức f sẽ thu được

```
>> simplify(f)
  ans =
  0
```

Ví dụ 4

Giả sử muốn giải phương trình bậc hai $f = ax^2 + bx + c$. Một cách tiếp cận là dùng lệnh

```
f=sym('a*x^2+b*x+c')
```

sẽ gắn biểu thức symbolic $ax^2 + bx + c$ cho biến f. Tuy nhiên, trong trường hợp này Symbolic Math Toolbox không tạo ra các biến tương ứng với các số hạng a, b, c, x của biểu thức. Để thực hiện các phép toán symbolic (ví dụ tích phân, đạo hàm, thay thế, etc) trên f, ta phải tạo các biến một cách rõ ràng. Cách tốt hơn đó là dùng các lệnh:

```
>> a=sym('a');  b=sym('b'); c=sym('c') ; x=sym('x');
```

hoặc đơn giản hơn `syms a b c x;`

Để xác định các biến `symbolic` nào được có mặt trong biểu thức, sử dụng lệnh `findsym`.

Ví dụ 5

Cho các biểu thức `symbolic f` và `g` được xác định bởi

```
>> syms a b n t x z
f = x^n; g = sin(a*t + b);
```

Khi đó, ta có thể tìm các biến `symbolic` có mặt trong `f` bởi lệnh

```
>> findsym(f)
ans =
n,x
```

Thay thế các biến symbolic

Lệnh subs

Ta có thể thay giá trị số cho một biến symbolic bằng cách sử dụng lệnh subs.

Ví dụ 6

Để thay giá trị $x = 2$ trong biểu thức symbolic

```
f = 2*x^2 - 3*x + 1
```

nhập vào lệnh

```
>> subs(f,2)
```

sẽ trả về giá trị của $f(2)$:

```
ans=
3
```

Thay thế các biến symbolic

Lệnh subs

Chú ý 1.2

- Để thay thế một ma trận vào trong một biểu thức symbolic f, sử dụng lệnh `polyvalm(sym2poly(f), A)`, sẽ thay thế x bởi A, và thay thế các hằng số trong f bởi một hằng số nhân với ma trận đơn vị.
- Khi một biểu thức có nhiều hơn một biến symbolic, ta có thể xác định biến cần thay thế. Ví dụ, để thay giá trị $x = 3$ trong biểu thức symbolic

```
>> syms x y
>> f = x^2*y + 5*x*sqrt(y)
```

sử dụng câu lệnh

```
>> subs(f, x, 3)
```

sẽ thu được

```
ans =
9*y+15*y^(1/2)
```

- Nếu ta không xác định một biến để thay thế, Matlab sẽ chọn một biến mặc định theo qui tắc sau. Đối với biến một chữ cái, Matlab chọn biến **gần với** `x` **nhất** trong bảng chữ cái. Nếu có hai biến gần `x` như nhau, Matlab sẽ chọn biến đứng sau trong bảng chữ cái.
  Trong ví dụ trên, hai lệnh `subs(f,3)` `subs(f,x,3)` cho kết quả giống nhau.
- Ta có thể sử dụng lệnh `findsym` để xác định biến mặc định. Ví dụ

```
>> syms s t
>> g = s + t;
>> findsym(g,1)
```

sẽ trả về biến mặc định:

```
ans =
t
```

Xét giá trị ban đầu trong Matlab

```
>> t = 0.1
```

Hàm sym có bốn tùy chọn cho việc trả về các biểu diễn symbolic của giá trị số lưu trữ trong t.

Tùy chọn 'f'

```
>> sym(t,'f')
```

trả về một biểu diễn symbolic dưới dạng dấu chấm động

```
ans =
  3602879701896397/36028797018963968
```

Tùy chọn 'r'

```
>> sym(t,'r')
```

trả về dạng hữu tỷ

```
1/10
```

Đây là tùy chọn mặc định của hàm sym. Nghĩa là, nếu gọi sym mà không có thành phần thứ hai cũng giống như sử dụng sym với cùy chọn 'r':

```
sym(t)
ans =
1/10
```

Tùy chọn 'e'

Tùy chọn 'e' trả về dạng hữu tỷ của t cộng với sự sai khác giữa giá trị thực của dạng hữu tỷ của t và giá trị thực (máy) dưới dạng dấu chấm động trong dạng eps (độ chính xác tương đối dạng dấu chấm động)

```
sym(t,'e')
ans =
1/10+eps/40
```

Tùy chọn 'd'

Tùy chọn thứ tư 'd' trả về dạng thập phân mở rộng đến số các chữ số có nghĩa, xác định bởi hàm digits:

```
sym(t,'d')
ans =
.100000000000000055511512312578
```

Giá trị mặc định của digits là 32. Nếu muốn dạng ngắn hơn, ta có thể dùng lệnh như sau:

```
digits(7)
sym(t,'d')
ans =
.1000000
```

Một tính năng riêng của lệnh sym đó là chuyển một ma trận dạng số về dạng symbolic. Ví dụ, lệnh

```
>> A = hilb(3)
```

sẽ tạo ra ma trận Hilbert cấp 3:

```
A =

    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000
```

Áp dụng lệnh sym lên A:

```
>> A = sym(A)
```

ta sẽ nhận được dạng symbolic của ma trận Hilbert cấp 3:

```
A =

[   1, 1/2, 1/3]
[ 1/2, 1/3, 1/4]
[ 1/3, 1/4, 1/5]
```

Biến đổi giữa symbolic và số

Tạo các biến thực và phức

Lệnh sym cho phép ta định rõ tính chất của biến symbolic bằng cách sử dụng tùy chọn 'real'. Các câu lệnh

```
>> x = sym('x','real'); y = sym('y','real');
```

hoặc đơn giản hơn

```
>> syms x y real
>> z = x + i*y
```

tạo các biến thực x và y. Do đó, z là một biến phức và ta có thể thực hiện các lệnh

```
conj(x), conj(z), expand(z*conj(z))
```

sẽ trả về

```
x, x-i*y, x^2+y^2
```

Biến đổi giữa `symbolic` và số

Xóa các biến trong không gian làm việc của nhân Maple

Khi ta tổ chức biến thực x với lệnh

```
>> syms x real
```

x trở thành một đối tượng `symbolic` trong không gian làm việc của Matlab và là một biến thực dương trong nhân làm việc của Maple. Nếu muốn bỏ thuộc tính thực của x, nhập vào

```
>> syms x unreal
```

Nếu bạn muốn xóa toàn bộ các định nghĩa biến trong không gian làm việc của nhân Maple, nhập vào

```
>> maple restart
```

Chú ý rằng lệnh

```
>> clear x
```

chỉ xóa biến x trong không gian làm việc cùa Matlab. Nếu sau đó ta nhập `syms x` mà không xóa x trong môi trường làm việc của nhân Maple thì Matlab sẽ xem x như là một số thực dương.

Nếu muốn tạo một hàm trừu tượng $f(x)$, nhập vào

```
>> f = sym('f(x)')
```
syms f(x)

Khi đó, f hoạt động như là $f(x)$ và có thể xử lý bằng các lệnh Matlab. Ví dụ, để xây dựng tỷ sai phân cấp 1, viết

```
>> df = (subs(f,'x','x+h') - f)/'h'
```

hoặc

```
>> syms x h
>> df = (subs(f,x,x+h)-f)/h
```

sẽ trả về

```
df =
(f(x+h)-f(x))/h
```

Ứng dụng này rất hữu ích trong các phép biến đổi Fourier, Laplace và $z-$.

Biến đổi giữa symbolic và số

Dùng sym để truy cập các hàm của Maple

Ta có thể truy cập hàm tính giai thừa k bằng cách sử dụng lệnh sym

```
>> kfac = sym('k!')
```

Khi đó, để tính $5!$ hoặc $n!$, viết

```
>> syms k n
>> subs(kfac,k,5), subs(kfac,k,n)
ans =
120
ans =
factorial(n)
```

Biến đổi giữa symbolic và số

Tạo một ma trận symbolic

Ta có thể tạo một ma trận vòng A từ các phần tử a, b, c bằng cách nhập

```
>> syms a b c
>> A=[a b c; b c a; c a b]

A =

[ a, b, c]
[ b, c, a]
[ c, a, b]
```

Để thay A(2,3) bằng beta và b bằng alpha, dùng các lệnh

```
>> syms alpha beta;
>> A(2,3) = beta;
>> A = subs(A,b,alpha)
```

sẽ thu được

```
A =

[ a, alpha, c]
[ alpha, c, beta]
[ c, a, alpha]
```

Sử dụng biểu thức symbolic

Các lệnh

```
>> syms x y z
>> r = sqrt(x^2 + y^2 + z^2)
>> t = atan(y/x)
>> f = sin(x*y)/(x*y)
```

tạo ra các biểu thức symbolic `r`, `t`, và `f`. Ta có thể dùng `diff`, `int`, `subs`, và các hàm khác trong Symbolic Math Toolbox để xử lý các biểu thức trên.

Tạo các M-file

M-file cho phép ta dùng các hàm tổng quát hơn. Giả sử, muốn tạo hàm

$$\text{sinc}(x) = \begin{cases} \dfrac{\sin(x)}{x} & x \neq 0 \\ 1 & x = 0 \end{cases}$$

ta viết file sinc.m có nội dung sau

```
function z = sinc(x)
if isequal(x,sym(0))
z = 1;
else
z = sin(x)/x;
end
```

Nội dung

Để minh họa việc tính đạo hàm sử dụng `Symbolic Math Toolbox`, trước hết tạo biểu thức `symbolic`

```
>> syms x
>> f = sin(5*x)
```

Khi đó, lệnh

```
>> diff(f)
```

sẽ tính đạo hàm của f theo đối x:

```
>> ans =
5*cos(5*x)
```

Để tính đạo hàm cấp hai của f, nhập vào

```
>> diff(f,2)
ans =
  (-25)*sin(5*x)
```

Ta có thể nhận được cùng kết quả trên bằng cách dùng lệnh

```
>> diff(diff(f))
ans =
  (-25)*sin(5*x)
```

Chú ý 2.1

Khi lấy đạo hàm của một hằng số, trước hết ta phải định nghĩa hằng số đó như là một biểu thức `symbolic`. Ví dụ

```
>> c = sym('5');
>> diff(c)
ans =
0
```

Nếu ta chỉ nhập

```
>> diff(5)
```

sẽ trả về

```
ans =
     []
```

Bởi vì 5 không phải là một biểu thức `symbolic`.

Để tính đạo hàm riêng của một hàm nhiều biến, ta phải xác định biến muốn lấy đạo hàm. Ví dụ, cho biểu thức `symbolic`

```
>> syms s t
>> f = sin(s*t)
```

Khi đó, lệnh

```
>> diff(f,t)
```

sẽ tính đạo hàm riêng của f theo đối t. Kết quả là

```
ans =
s*cos(s*t)
```

Để tính đạo hàm của f theo đối s, nhập vào

```
>> diff(f,s)
```

sẽ trả về

```
ans =
t*cos(s*t)
```

Nếu ta không chỉ rõ biến lấy đạo hàm, Matlab sẽ chọn biến mặc định được xác định bởi lệnh `symvar`:

```
>> symvar(f, 1)
ans =
t
```

Để tính đạo hàm riêng cấp hai theo đối t, nhập vào

```
>> diff(f, t, 2)
```

sẽ trả về

```
ans =
-s^2*sin(s*t)
```

Chú ý rằng lệnh `diff(f,2)` sẽ cho cùng kết quả vì t là biến mặc định.

Symbolic Math Toolbox cho phép tính giới hạn của các hàm số một cách trực tiếp. Các lệnh

```
>> syms h n x
>> limit((cos(x+h) - cos(x))/h, h, 0)
```

sẽ trả về

```
ans =
-sin(x)
```

và

```
>> limit((1 + x/n)^n, n, inf)
```

sẽ trả về

```
ans =
exp(x)
```

Để tính giới hạn trái $\lim\limits_{x\to 0^-} \dfrac{x}{|x|}$, nhập vào

```
>> limit(x/abs(x),x,0,'left')
ans =
-1
```

Để tính giới hạn phải $\lim\limits_{x\to 0^+} \dfrac{x}{|x|}$, nhập vào

```
>> limit(x/abs(x),x,0,'right')
ans =
1
```

Vì các giới hạn trái khác giới hạn phải nên giới hạn 2 phía không tồn tại. Trong trường hợp này, Matlab trả về giá trị trừu tượng `NaN` (not a number). Ví dụ,

```
>> limit(x/abs(x),x,0)
ans =
NaN
```

Nếu f là một biểu thức `symbolic` thì

```
>> int(f)
```

sẽ trả về tích phân bất định hay nguyên hàm của f. Tương tự như phép tính đạo hàm,

```
>> int(f,v)
```

sử dụng đối tượng `symbolic v` như là biến lấy tích phân.

Ta có thể xem hoạt động của lệnh `int` ở bảng sau:

Hàm toán học	**Lệnh Matlab**
$\int x^n dx = \begin{cases} \log x & n = -1 \\ \dfrac{x^{n+1}}{n+1} & \text{ngược lại} \end{cases}$	`int(x^n)` hay `int(x^n,x)`
$\int_0^{\pi/2} \sin(2x)dx = 1$	`int(sin(2*x),0,pi/2)` hay `int(sin(2*x),x,0,pi/2)`
$g = \cos at + b$ $\int g(t)dt = \dfrac{1}{a}\sin(at+b)$	`g=cos(a*t+b)` `int(g)` hay `int(g,t)`

Một trong các vấn đề của tích phân `symbolic` đó là "giá trị" của các tham số.

Ví dụ, nếu ta muốn tính tích phân $I = \int_{-\infty}^{+\infty} e^{-ax^2} dx$ mà không gắn dấu cho a,

Matlab sẽ coi như a là một số phức và do đó sẽ cho kết quả dưới dạng phức. Nếu ta chỉ quan tâm trường hợp a là số thực dương, ta có thể tính tích phân trên như sau:

```
>> syms a positive;
```

Bây giờ ta có thể sử dụng các lệnh

```
>> syms x;
>> f = exp(-a*x^2);
>> int(f,x,-inf,inf)
```

sẽ trả về

```
ans =
1/(a)^(1/2)*pi^(1/2)
```

Nếu ta muốn tính tích phân trên với a là một số thực bất kỳ:

```
>> syms a real
>> f=exp(-a*x^2);
>> F = int(f, x, -inf, inf)
F =
piecewise([1/a^(1/2)*pi^(1/2), signum(a) = 1],[Inf, otherwise])
```

Ta có thể dùng lệnh `pretty(F)` để nhận được dạng dễ đọc hơn:

```
{   1/2
{ pi
{ ----- signum(a~) = 1
{   1/2
{ a~
{
{ Inf otherwise
```

Ký hiệu ~ sau a cho ta biết a là một số thực và `signum(a~)` là dấu của a.

Để tính tích phân $I = \int_{-\infty}^{+\infty} e^{-ax^2} dx$ với a là một số phức, nhập vào

```
>> syms a x unreal
>> f = exp(-a*x^2);
>> F = int(f, x, -inf, inf)
```

sẽ trả về

```
F =
  piecewise([a < 0, Inf],
  [(0 <= Re(a) or abs(arg(a)) <= pi/2) and a <> 0, pi^(1/2)/a^(1/2)],
  [Otherwise, int(1/exp(a*x^2), x = -Inf..Inf)])
```

Ta có thể sử dụng lệnh `int` để tính tích phân nhiều lớp. Ví dụ, để tính tích phân 2 lớp $I = \int_0^1 \int_{1-x}^{1-x^2} xydydx$ ta dùng các lệnh sau

```
>> syms x y;
>> firstint=int(x*y,y,1-x,1-x^2)
firstint =
(x^2*(x - 1)^2*(x + 2))/2
>> answer=int(firstint,x,0,1)
answer =
 1/24
```

hoặc gọn hơn

```
>> int(int(x*y,y,1-x,1-x^2),x,0,1)
 ans =
 1/24
```

Hoàn toàn tương tự ta có thể tính tích phân 3 lớp, ví dụ

$$I = \int_0^3 \int_0^{3-x} \int_0^{3-x-y} xyz dz dy dx$$

bởi các lệnh

```
>> syms x y z;
>> I=int(int(int(x*y*z,z,0,3-x-y),y,0,3-x),x,0,3)
 I =
 81/80
```

Ta có thể tính các tổng symbolic bằng cách sử dụng lệnh symsum. Ví dụ, chuỗi

$$1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots$$

có tổng là $\frac{\pi^2}{6}$, trong khi chuỗi hàm

$$1 + x + x^2 + ...$$

có tổng là $\frac{1}{(1-x)}$, với $|x| < 1$. Các tổng trên có thể được tính bởi các lệnh

```
>> syms x k
>> s1 = symsum(1/k^2,1,inf)
>> s2 = symsum(x^k,k,0,inf)
s1 =
1/6*pi^2
s2 =
-1/(x-1)
```

Các câu lệnh

```
>> syms x
>> f = 1/(5+4*cos(x))
>> T = taylor(f,8)
```

cho kết quả

```
T =
1/9+2/81*x^2+5/1458*x^4+49/131220*x^6
```

chứa tất cả các số hạng có bậc nhỏ hơn 8 trong khai triển chuỗi Taylor tại lân cận $x = 0$ (khai triển Maclaurin) của hàm $f(x)$:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n.$$

Các câu lệnh

```
>> syms x
>> g = exp(x*sin(x))
>> t = taylor(g,12,2);
```

tạo ra 12 số hạng đầu khác 0 của chuỗi Taylor tại lân cận $x = 2$ của g. Tiếp theo, ta vẽ đồ thị của cả hai hàm g và xấp xỉ Taylor t của nó:

```
xd = 1:0.05:3; yd = subs(g,x,xd);
ezplot(t, [1,3]); hold on;
plot(xd, yd, 'r-.')
title('Taylor approximation vs. actual function');
legend('Taylor','Function')
```

Rút gọn

Xét các biểu thức dạng symbolic khác nhau của cùng một hàm toán học:

```
>> syms x
>> f = x^3-6*x^2+11*x-6
>> g = (x-1)*(x-2)*(x-3)
>>h = -6+(11+(-6+x)*x)*x
```

Nhập vào các lệnh

```
>> pretty(f), pretty(g), pretty(h)
```

ta nhận được

```
 3     2
x - 6 x + 11 x - 6
(x - 1) (x - 2) (x - 3)
-6 + (11 + (-6 + x) x) x
```

Rút gọn

collect

Câu lệnh

```
>> collect(f)
```

xem f như là một đa thức đối với biến `symbolic` x, và gộp tất cả các hệ số cùng bậc của x. Sau đây là một vài ví dụ

f	collect(f)
`(x-1)*(x-2)*(x-3)`	`x^3-6*x^2+11*x-6`
`x*(x*(x-6)+11)-6`	`x^3-6*x^2+11*x-6`
`(1+x)*t + x*t`	`2*x*t+t`

Rút gọn

expand

Câu lệnh

```
>> expand(f)
```

khai triển biểu thức f bằng cách áp dụng tính chất phân phối của phép nhân lên phép cộng và các đồng nhất thức đối với phép cộng được mô tả bởi bảng sau:

f	expand(f)
a*(x+y)	a*x+a*y
(x-1)*(x-2)*(x-3)	x^3-6*x^2+11*x-6
x*(x*(x-6)+11)-6	x^3-6*x^2+11*x-6
exp(a+b)	exp(a)*exp(b)
cos(x+y)	cos(x)*cos(y)-sin(x)*sin(y)
cos(3*acos(x))	4*x^3-3*x

Rút gọn

horner

Câu lệnh

```
>> horner(f)
```

biến đổi một đa thức thành dạng Horner hay biểu diễn lồng nhau. Một số ví dụ

f	horner(f)
x^3-6*x^2+11*x-6	x*(x*(x - 6) + 11) - 6
1.1+2.2*x+3.3*x^2	x*((33*x)/10 + 11/5) + 11/10

Rút gọn

factor

Nếu f là một đa thức với các hệ số hữu tỷ, lệnh

```
>> factor(f)
```

biểu diễn f thành tích các đa thức hữu tỷ bậc nhỏ hơn. Nếu f là bất khả qui, Matlab sẽ trả về kết quả chính là f. Sau đây là một vài ví dụ

f	factor(f)
x^3-6*x^2+11*x-6	(x-1)*(x-2)*(x-3)
x^3-6*x^2+11*x-5	x^3-6*x^2+11*x-5
x^6+1	(x^2+1)*(x^4-x^2+1)

Rút gọn

simplify

Hàm `simplify` rất hữu dụng trong việc rút gọn các biểu thức nói chung. Sau đây là một số ví dụ:

f	simplify(f)
x*(x*(x-6)+11)-6	x^3-6*x^2+11*x-6
(1-x^2)/(1-x)	x+1
(1/a^3+6/a^2+12/a+8)^(1/3)	((2*a+1)^3/a^3)^(1/3)
syms x y positive log(x*y)	log(x)+log(y)
exp(x) * exp(y)	exp(x+y)
cos(x)^2 + sin(x)^2	1

Rút gọn

simple

Hàm `simple` trả về dạng ngắn nhất có thể của một biểu thức. `Simple` có rất nhiều dạng, mỗi dạng lại cho một kết quả khác nhau. Dạng

```
simple(f)
```

hiển thị kết quả ngắn nhất, ví dụ câu lệnh

```
simple(cos(x)^2 + sin(x)^2)
```

cho kết quả:

```
ans =
1
```

Rút gọn

simple

Đôi khi, hàm `simple` cải tiến kết quả cho bởi hàm `simplify`. Ví dụ, khi áp dụng các ví dụ cho bởi `simplify`, `simple` cho kết quả đơn giản hơn (hoặc ít nhất ngắn hơn). Xét các ví dụ sau:

`f`	`simplify(f)`	`simple(f)`
`(1/a^3+6/a^2+12/a+8)^(1/3)`	`((2*a+1)^3/a^3)^(1/3)`	`(2*a+1)/a`
`syms x y positive` `log(x*y)`	`log(x)+log(y)`	`log(x*y)`

Rút gọn

simple

Hàm `simple` đặc biệt hiệu quả trong việc rút gọn các biểu thức lượng giác. Sau đây là một số ví dụ:

f	simple(f)
cos(x)^2+sin(x)^2	1
2*cos(x)^2-sin(x)^2	3*cos(x)^2-1
cos(x)^2-sin(x)^2	cos(2*x)
cos(x)+(-sin(x)^2)^(1/2)	cos(x)+i*sin(x)
cos(x)+i*sin(x)	exp(i*x)
cos(3*acos(x))	4*x^3-3*x

Các phép toán đại số cơ bản trên các đối tượng `symbolic` cũng giống như đối với lớp `double`. Ví dụ, các lệnh

```
>> syms t;
>> G = [cos(t) sin(t); -sin(t) cos(t)]
```

tạo ra

```
G =
[ cos(t), sin(t) ]
[ -sin(t), cos(t) ]
```

Cả hai câu lệnh

```
>> A = G*G
```

và

```
>> A = G^2
```

tạo ra

```
A =
[cos(t)^2-sin(t)^2, 2*cos(t)*sin(t)]
[ -2*cos(t)*sin(t), cos(t)^2-sin(t)^2]
```

Áp dụng hàm `simple`

```
A = simple(A)
A =
[ cos(2*t), sin(2*t)]
[-sin(2*t), cos(2*t)]
```

Ma trận G là ma trận trực giao ($G' = G^{-1}$), có thể kiểm chứng điều này bởi

```
>> I = G.' *G
```

sẽ tạo ra

```
I =
[cos(t)^2+sin(t)^2, 0]
[ 0, cos(t)^2+sin(t)^2]
```

Đơn giản hóa:

```
>> I = simple(I)
I =
[1, 0]
[0, 1]
```

Lệnh

```
>> H = hilb(3)
```

tạo ra ma trận Hilbert cấp 3

```
H =
    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000
```

Biến đổi H thành ma trận `symbolic`

```
>> H = sym(H)
H =
  [   1, 1/2, 1/3]
[ 1/2, 1/3, 1/4]
[ 1/3, 1/4, 1/5]
```

Tính nghịch đảo:

```
>> inv(H)
ans =

[   9,  -36,   30]
[ -36,  192, -180]
[  30, -180,  180]
```

và định thức

```
>> det(H)
ans=
1/2160
```

Ta có thể sử dụng toán tử \ để giải hệ đại số tuyến tính:

```
>> b = [1 1 1]'
>> x = H\b % Giải hệ Hx = b
x=
 3
-24
 30
```

Các giá trị riêng dạng `symbolic` của ma trận vuông `A` hoặc giá trị riêng và vector riêng dạng `symbolic` của `A` được tính bằng các lệnh tương ứng sau

```
>> E = eig(A)
>> [V,E] = eig(A)
```

Các giá trị riêng của `A` là nghiệm của đa thức đặc trưng `det(A-x*I)`, được tính bởi

```
>> poly(A)
```

Xét ma trận H

```
H =
[8/9, 1/2, 1/3]
[1/2, 1/3, 1/4]
[1/3, 1/4, 1/5]
```

Ma trận này là suy biến, do đó có ít nhất một giá trị riêng bằng 0. Câu lệnh

```
>> [T,E] = eig(H)
```

tạo ra các ma trận T và E. Các cột của T là các vector riêng của H

```
T =
[ 1, 28/153+2/153*12589^(1/2), 28/153-2/153*12589^(12)]
[ -4, 1, 1]
[ 10/3, 92/255-1/255*12589^(1/2), 292/255+1/255*12589^(12)]
```

Tương tự, các thành phần trên đường chéo chính của E là các trị riêng của H:

```
E =
[0, 0, 0]
[0, 32/45+1/180*12589^(1/2), 0]
[0, 0, 32/45-1/180*12589^(1/2)]
```

Sẽ dễ dàng hiểu được cấu trúc của các ma trận T và E nếu ta chuyển chúng về dạng thập phân:

```
>> Td = double(T)
>> Ed = double(E)
```

sẽ cho

```
Td =                          Ed =
1.0000 1.6497 -1.2837         0 0 0
-4.0000 1.0000 1.0000         0 1.3344 0
3.3333 0.7051 1.5851          0 0 0.0878
```

Giá trị riêng đầu tiên bằng 0. Vector riêng tương ứng (cột đầu tiên của Td). Hai giá trị riêng còn lại là kết quả của việc áp dụng công thức toàn phương đối với

```
x^2-64/45*x+253/2160
```

là nhân tử bậc hai trong khai triển `factor(poly(H))`:

```
>> syms x
>> g = simple(factor(poly(H))/x);
>> solve(g)
ans =
[ 32/45+1/180*12589^(1/2)]
[ 32/45-1/180*12589^(1/2)]
```

Các lệnh trong Symbolic Math Toolbox

```
>> syms t
>> A = sym([0 1; -1 0]);
>> G = expm(t*A)
```

trả về

```
G=
[ cos(t), sin(t)]
[ -sin(t), cos(t)]
```

Tiếp theo, lệnh

```
>> g = eig(G)
g =
[ cos(t)+(cos(t)^2-1)^(1/2)]
[ cos(t)-(cos(t)^2-1)^(1/2)]
```

Ta có thể dùng `simple` để rút gọn g nhiều lần:

```
for j = 1:4
[g,how] = simple(g)
end
```

sẽ cho ta kết quả tốt nhất:

```
g =
[ cos(t)+(-sin(t)^2)^(1/2)]
[ cos(t)-(-sin(t)^2)^(1/2)]
how =
mwcos2sin
g =
[ cos(t)+i*sin(t)]
[ cos(t)-i*sin(t)]
how =
radsimp
```

```
g =
[ exp(i*t)]
[ 1/exp(i*t)]
how =
convert(exp)
g =
[ exp(i*t)]
[ exp(-i*t)]
how =
simplify
```

Dạng Jordan chuẩn tắc nhận được từ việc chéo hóa một ma trận bằng các phép biến đổi đồng dạng. Với ma trận đã cho A, tìm một ma trận không suy biến V sao cho `inv(V)*A*V` hay gọn hơn `J=V\A*V` "càng gần với ma trận đường chéo càng tốt". Với hầu hết các ma trận, dạng Jordan chính tắc là ma trận đường chéo của các giá trị riêng và các cột của ma trận chuyển vị của ma trận các vector riêng. Điều này luôn đúng nếu A là ma trận đối xứng hoặc có các giá trị riêng phân biệt. Một số ma trận không đối xứng cùng các giá trị riêng bội không thể chéo hóa được.
Câu lệnh

```
>> J = jordan(A)
```

tính dạng Jordan chuẩn tắc của A. Câu lệnh

```
>> [V,J] = jordan(A)
```

trả về thêm ma trận V có các cột là các vector riêng mở rộng của A.

Dạng chính tắc Jordan rất "nhạy cảm" với các nhiễu. Điều này gây khó khăn cho việc tính dạng Jordan với kết quả dạng dấu chấm động. Điều này cũng đòi hỏi phải biết chính xác ma trận A. Các phần tử của A phải là các số nguyên hoặc tỷ số của các số nguyên nhỏ. Ví dụ:

```
>> A = sym([12,32,66,116;-25,-76,-164,-294;
21,66,143,256;-6,-19,-41,-73])
A =
[ 12, 32, 66, 116]
[ -25, -76, -164, -294]
[ 21, 66, 143, 256]
[ -6, -19, -41, -73]
```

Khi đó

```
>> [V,J] = jordan(A)
V =
[ 4, -2, 4, 3]
[ -6, 8, -11, -8]
[ 4, -7, 10, 7]
[ -1, 2, -3, -2]
J =
[ 1, 1, 0, 0]
[ 0, 1, 0, 0]
[ 0, 0, 2, 1]
[ 0, 0, 0, 2]
```

Ta thấy A có giá trị riêng bội tại 1, chỉ có hai vector riêng `V(:,1)` và `V(:,3)`. Chúng thỏa mãn

```
A*V(:,1) = 1*V(:,1)
A*V(:,3) = 2*V(:,3)
```

Hai cột còn lại của V là các vector riêng mở rộng của 2:

```
A*V(:,2) = 1*V(:,2) + V(:,1)
A*V(:,4) = 2*V(:,4) + V(:,3)
```

Nếu A là một ma trận `symbolic` dạng dấu chấm động hay biến số chính xác thì

```
>> S = svd(A)
```

tính các giá trị kỳ dị của A và

```
>> [U,S,V] = svd(A);
```

tạo ra hai ma trận trực giao U, V, và ma trận đường chéo S sao cho

$$A = U * S * V';$$

Xét ma trận cấp n với các phần tử `A(i,j) = 1/(i-j+1/2)`. Với $n = 5$ ta có

```
[ 2 -2 -2/3 -2/5 -2/7]
[2/3 2 -2 -2/3 -2/5]
[2/5 2/3 2 -2 -2/3]
[2/7 2/5 2/3 2 -2]
[2/9 2/7 2/5 2/3 2]
```

Điều này dẫn đến việc rất nhiều giá trị kỳ dị của A gần với π. Cách rõ nhất để tạo ra ma trận này:

```
for i=1:n
     for j=1:n
          A(i,j) = sym(1/(i-j+1/2));
     end
end
```

Cách thuận tiện nhất để tạo ra ma trận A là

```
[J,I] = meshgrid(1:n);
A = sym(1./(I - J+1/2));
```

Vì các phần tử của A là các tỷ lệ của các số nguyên nhỏ nên hàm `vpa(A)` tạo ra một biểu diễn biến số chính xác. Do đó

```
S = svd(vpa(A))
```

sẽ tính các giá trị kỳ dị một cách hoàn toàn chính xác. Với $n = 16$ và `digits(30)`, kết quả sẽ là

```
S =
[ 1.2096813760566898533245568535 7 ]
[ 2.6916215868606660677478276359 4 ]
[ 3.0779029723111974865842472735 4 ]
[ 3.1350405439974465484389890126 1 ]
[ 3.1410604466347006380521837192 4 ]
[ 3.1415575435991808369105065826 0 ]
[ 3.1415907545860584872898257711 9 ]
[ 3.1415925692549230647028486310 2 ]
[ 3.1415926505265488081556947961 3 ]
[ 3.1415926534996105314385683856 4 ]
[ 3.1415926535876736171239261238 4 ]
[ 3.1415926535897543920684990722 0 ]
[ 3.1415926535897927034263555905 1 ]
[ 3.1415926535897932332529014278 1 ]
[ 3.1415926535897932384306684671 2 ]
[ 3.1415926535897932384625503597 4 ]
```

Nếu `S` là một biểu thức `symbolic` thì lệnh

```
>> solve(S)
```

sẽ tìm giá trị của các biến `symbolic` có trong `S` (có thể xác dịnh bởi lệnh `findsym`) sao cho $S = 0$. Ví dụ

```
syms a b c x
S = a*x^2 + b*x + c;
solve(S)
```

sẽ cho kết quả là một vector `symbolic` mà các thành phần là 2 nghiệm của phương trình $S = 0$:

```
ans =
[1/2/a*(-b+(b^2-4*a*c)^(1/2))]
[1/2/a*(-b-(b^2-4*a*c)^(1/2))]
```

Nếu ta muốn giải phương trình với biến định trước, ta phải chỉ rõ biến đó, ví dụ nếu ta giải $S = 0$ theo đối b

```
>> b = solve(S,b)
b =
-(a*x^2+c)/x
```

Chú ý 2.2

Chú ý rằng các ví dụ trên đều giả thiết phương trình có dạng $f(x) = 0$. Nếu muốn giải phương trình dạng $f(x) = q(x)$, ta phải sử dụng một chuỗi trích dẫn. Nói riêng, lệnh

```
>> s = solve('cos(2*x)+sin(x)=1')
```

cho ta một vector chứa 4 nghiệm:

```
s =
[ 0]
[ pi]
[ 1/6*pi]
[ 5/6*pi]
```

Giả sử cần tìm nghiệm (x, y) của hệ phương trình

$$\begin{cases} x^2y^2 & = 0 \\ x - \dfrac{y}{2} & = \alpha \end{cases}$$

```
>> syms x y alpha

>> [x,y] = solve(x^2*y^2, x-y/2-alpha)
x =
[ 0]
[ 0]
[ alpha]
[ alpha]
y =
[ -2*alpha]
[ -2*alpha]
[ 0]
[ 0]
```

Do đó, vector nghiệm

```
>> v = [x, y]
v=
```

xuất hiện các thành phần dư thừa. Nguyên nhân là do phương trình ban đầu $x^2y^2 = 0$ có 2 nghiệm $x = \pm 0, y = \pm 0$. Thay đổi hệ thành

```
eqs1 = 'x^2*y^2=1, x-y/2-alpha'
[x,y] = solve(eqs1)
```

tạo ra 4 nghiệm phân biệt:

```
x =                                          y =
[ 1/2*alpha+1/2*(alpha^2+2)^(1/2)]           [ -alpha+(alpha^2+2)^(1/2)]
[ 1/2*alpha-1/2*(alpha^2+2)^(1/2)]           [ -alpha-(alpha^2+2)^(1/2)]
[ 1/2*alpha+1/2*(alpha^2-2)^(1/2)]           [ -alpha+(alpha^2-2)^(1/2)]
[ 1/2*alpha-1/2*(alpha^2-2)^(1/2)]           [ -alpha-(alpha^2-2)^(1/2)]
```

Cách làm trên chỉ thích hợp với hệ có ít phương trình. Rõ ràng, nếu ta xét một hệ 10 phương trình, 10 ẩn, sẽ phải nhập vào

```
[x1,x2,x3,x4,x5,x6,x7,x8,x9,x10] = solve(...)
```

Điều này thật bất tiện và tốn thời gian. Nhằm tránh khó khăn này, `solve` sẽ trả về một cấu trúc mà các trường của nó chính là các nghiệm. Nói riêng, ta xét hệ $u^2 - v^2 = a^2, u + v = 1, a^2 - 2a = 3$, lệnh

```
S = solve('u^2-v^2 = a^2','u + v = 1','a^2-2*a = 3')
S =
a: [2x1 sym]
u: [2x1 sym]
v: [2x1 sym]
```

Các nghiệm được lưu trong các trường của `S`. Ví dụ

```
S.a
ans =
[ 3]
[ -1]
```

Tương tự đối với các nghiệm `u, v`. Cấu trúc `S` bây giờ có thể xử lý bằng trường và các chỉ số để truy cập tới các nghiệm cụ thể. Ví dụ, nếu ta muốn kiểm tra nghiệm thứ hai, có thể dùng lệnh sau

```
s2 = [S.a(2), S.u(2), S.v(2)]
s2 =
[ -1, 1, 0]
```

Câu lệnh sau

```
M = [S.a, S.u, S.v]
```

tạo ra một ma trận nghiệm

```
M =
[ 3, 5, -4]
[ -1, 1, 0]
```

có các dòng chứa các nghiệm của hệ.

Hệ tuyến tính có thể giải bằng lệnh `solve` bằng cách sử dụng phép chia ma trận. Ví dụ

```
clear u v x y
syms u v x y
S = solve(x+2*y-u, 4*x+5*y-v);
sol = [S.x;S.y]
```

và

```
A = [1 2; 4 5];
b = [u; v];
z = A\b
```

cho các kết quả

```
sol =
[ -5/3*u+2/3*v]
[ 4/3*u-1/3*v]
```

```
z =
[ -5/3*u+2/3*v]
[ 4/3*u-1/3*v]
```

Do đó `sol` và `z` tạo ra cùng một nghiệm, mặc dù chúng được gắn với các biến khác nhau.

- Hàm `dsolve` tìm các nghiệm `symbolic` của phương trình vi phân thường.
- Các phương trình được mô tả bằng các biểu thức `symbolic`, trong đó chữ cái `D` dùng để ký hiệu đạo hàm. Các ký hiệu `D2, D3, ..., Dn` tương ứng với các đạo hàm các cấp $2, 3, \ldots, n$ tương ứng. Do đó `D2y` sẽ tương đương với $\frac{d^2y}{dt^2}$.
- Các biến phụ thuộc sẽ đi sau `D` và biến độc lập mặc định là `t`. Chú ý rằng tên của biến `symbolic` không được chứa ký tự `D`. Có thể dùng biến độc lập khác bằng cách nhập nó như là thông số cuối của `dsolve`.
- Các điều kiện đầu có thể được mô tả như là các phương trình phụ, nếu không có điều kiện đầu, các nghiệm sẽ chứa các hằng số $C_1, C_2, \ldots$.

Cú pháp của `dsolve` có thể được mô tả trong bảng sau:

Cú pháp	**Phạm vi**
`y=dsolve('Dy=y0*y')`	Một phương trình, một nghiệm
`[u,v]=dsolve('Du=v','Dv=u')`	Hai phương trình, hai nghiệm
`S=dsolve('Df=g','Dg=h','Dh=-f')`	Ba phương trình, nghiệm cấu trúc

Ví dụ 7

Câu lệnh

```
>> dsolve('Dy=1+y^2')
```

sử dụng y như là biến phụ thuộc và t là biến độc lập mặc định. Kết quả:

```
ans =
tan(t+C1)
```

Để thêm điều kiện đầu, ta dùng

```
>> y = dsolve('Dy=1+y^2','y(0)=1')
y =
tan(t+1/4*pi)
```

Ví dụ 8

Các phương trình phi tuyến có thể có nhiều nghiệm, ngay cả khi điều kiện đầu đã cho:

```
>> x = dsolve('(Dx)^2+x^2=1','x(0)=0')
x =
[ sin(t)]
[ -sin(t)]
```

Ví dụ 9

Xét phương trình vi phân cấp hai với hai điều kiện đầu. Các lệnh

```
>> y = dsolve('D2y=cos(2*x)-y','y(0)=1','Dy(0)=0', 'x');
>> simplify(y)
```

tạo ra

```
ans =
4/3*cos(x)-2/3*cos(x)^2+1/3
```

Ví dụ 10

Điều mấu chốt trong ví dụ này là bậc của phương trình và các điều kiện đầu. Để giải phương trình vi phân thường

$$\begin{cases} \dfrac{d^3y}{dx^3} = u \\ u(0) = 1, \quad u'(0) = -1, \quad u''(0) = \pi \end{cases}$$

ta sử dụng các lệnh

```
>> u = dsolve('D3u=u','u(0)=1','Du(0)=-1','D2u(0) = pi','x')
u =
  (pi*exp(x))/3 - (cos((3^(1/2)*x)/2)*(pi/3 - 1))/exp(x/2)
  - (3^(1/2)*sin((3^(1/2)*x)/2)*(pi + 1))/(3*exp(x/2))
```

Hàm `dsolve` có thể giải hệ phương trình vi phân có hoặc không có điều kiện đầu. Ví dụ, xét hệ hai phương trình tuyến tính cấp 1

```
>> S = dsolve('Df = 3*f+4*g', 'Dg = -4*f+3*g')
```

Nghiệm tính được sẽ có dạng cấu trúc `S`. Ta có thể xác định các giá trị của `f` và `g` bằng cách nhập

```
f = S.f
f =
exp(3*t)*(C1*sin(4*t)+C2*cos(4*t))
g = S.g
g =
exp(3*t)*(C1*cos(4*t)-C2*sin(4*t))
```

Nếu ta muốn thêm điều kiện đầu:

```
>> [f,g] = dsolve('Df=3*f+4*g, Dg =-4*f+3*g', 'f(0) = 0, g(0) = 1')
f =
exp(3*t)*sin(4*t)
g =
exp(3*t)*cos(4*t)
```

Một số ví dụ khác :

Phương trình vi phân	**Lệnh Matlab**
$\frac{dy}{dt} + 4y(t) = e^{-t}$ $y(0) = 1$	`y=dsolve('Dy+4*y=exp(-t)','y(0)=1')`
$\frac{d^2y}{dx^2} + 4y(x) = e^{-2x}$ $y(0) = 0, \quad y(\pi) = 0$	`y=dsolve('D2y+4*y=exp(-2*x)','y(0)=1','y(pi)=0','x')`

Cú pháp

```
F = fourier(f)
F = fourier(f,v)
F = fourier(f,u,v)
```

Mô tả

- `F=fourier(f)` là biến đổi Fourier của biểu thức `symbolic` vô hướng f với biến mặc định là x. Kết quả trả về mặc định là hàm theo biến w:

$$f = f(x) \Longrightarrow F = F(w).$$

Hàm $F(w)$ được định nghĩa bởi

$$F(w) = \int_{-\infty}^{+\infty} f(x)e^{-iwx}dx$$

Mô tả (tiếp)

Nếu $f = f(w)$ thì kết quả trả về là một hàm theo t : $F = F(t)$.

- `F = fourier(f,v)` tạo một hàm F của đối `symbolic v` thay vì biến mặc định `w`

$$F(v) = \int_{-\infty}^{+\infty} f(x)e^{-iwx}dx$$

- `F = fourier(f,u,v)` tạo f là hàm theo u và F là hàm theo v thay vì các biến mặc định tương ứng là x và w

$$F(v) = \int_{-\infty}^{+\infty} f(x)e^{-iwu}du$$

Ví dụ 11

Biến đổi Fourier	Lệnh Matlab
$f(x) = e^{-x^2}$	`f=exp(-x^2);`
$F[f](w) = \int_{-\infty}^{+\infty} f(x)e^{-iwx}dx = \sqrt{\pi}e^{-w^2/4}$	`fourier(f)` *trả về*
	`pi^(1/2)/exp(w^2/4)`
$g(w) = e^{-\|w\|}$	`exp(-abs(w))`
$F[g](t) = \int_{-\infty}^{+\infty} g(w)e^{-iwt}dt = \frac{2}{1+t^2}$	`fourier(g)` *trả về*
	`2/(v^2 + 1)`

Ví dụ (tiếp)

Biến đổi Fourier	Lệnh Matlab

$f(x) = x.e^{-x}$
$F[f](u) = \int_{-\infty}^{+\infty} f(x)e^{-xu}dx$
	`syms u x`
	
	`f=x*exp(-abs(x))` trả về
	
	`-(4*u*i)/(u^2 + 1)^2`

$f(x,v) = e^{-x^2\|v\|}\frac{\sin v}{v}$
$F[f(v)](u) = \int_{-\infty}^{+\infty} f(x,v)e^{-ivu}dv$
$= -\arctan\frac{u-1}{x^2} + -\arctan\frac{u+1}{x^2}$
	`syms v u; syms x real`
	
	`f = exp(-x^2*abs(v))*sin(v)/v;`
	
	`fourier(f)`
	`piecewise([x <> 0, atan((u + 1)/x^2) -`
	`atan(1/x^2*(u - 1))])`

Cú pháp

```
f = ifourier(F)
f = ifourier(F,u)
f = ifourier(F,v,u)
```

Mô tả

`f = ifourier(F)` là biến đổi Fourier ngược của biểu thức `symbolic` vô hướng F với biến mặc định là w. Kết quả trả về mặc định là một hàm của x

$$F = F(w) \Longrightarrow f = f(x).$$

Mô tả (tiếp)

Nếu $F = F(x)$, `ifourier` trả về hàm theo đối t : $f = f(t)$. Bằng cách định nghĩa

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(w) e^{iwx} dw,$$

`f = ifourier(F,u)` tạo ra hàm $f(u)$ thay vì theo biến mặc định x

$$f(u) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(w) e^{iwu} dw.$$

`f = ifourier(F,v,u)` tạo một hàm f theo đối u và F là hàm theo đối v theay vì các biến mặc định là x và w.

Biến đổi tích phân (Tự đọc help)

Biến đổi Laplace và biến đổi Laplace ngược

Biến đổi tích phân (Tự đọc help)

Biến đổi Z và biến đổi $Z-$ ngược