by Ahmet Sacan.

Introduction to Symbolic Math.

"Symbolic" or "algebric" computations are carried out exactly, expressing the answer to a mathematical problem in a closed formula. The main task of a symbolic math system is to manipulate and simplify mathmetical expressions. These computations are carried out according to the rules of algebra, instead of approximate floating-point arithmetic used in "numerical" computations.

In Matlab, you can use sym() function to enforce symbolic computation rather than numerical computation.

Unfortunately, Matlab has decided to violate some programming language rules, such as independent and immediate evaluation of expressions, when implementing symbolic computation. e.g., in sym(1/3), matlab does not evaluate 1/3 first; it lets sym() function take 1/3 as is. And the rules Matlab follows seem arbitrary, so you should check small expressions to study the behavior of Matlab before incorporating them into larger expressions.

Use sym() function to convert a number to a symbolic value.

```
sym(1/3)  % Matlab delays calculation of "/"
```

```
ans =
```

$$\frac{1}{3}$$

```
sym(10^1000) % Matlab does not delay calculation of "^"
```

```
ans = ∞
```

```
% to enforce symbolic computation reliably, enclose each number with sym().
sym(10)^sym(100)
```

```
ans = 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Examples Demonstrating Symbolic Computation is Correct, but numerical Computation may not be

```
a = 1/3;
fprintf('%.50f\n',a)
```

```
0.33333333333333331482961625624739099293947219848633
```

```
sym(1) / sym(3)
```

ans =

$$\frac{1}{3}$$

```
10^1000 + 1 - 10^1000
```

ans = NaN

```
sym( 10^1000 + 1 - 10^1000 ) % this does not give correct symbolic result.
```

ans = NaN

```
sym(10)^sym(1000) + sym(1) - sym(10)^sym(1000)
```

ans = 1

```
sin(pi)  % this is the regular sin() function, which produces an approximate numerical value.
```

ans = 1.2246e-16

```
sin(sym('pi'))  %this actually calls a different sin() function that is designed for symbolic m
```

ans = $\sin(\pi)$

```
factorial(1000)
```

ans = Inf

```
factorial(sym(100))
```

ans = 933262154439441526816992388562667004907159682643816214685929638952175999932299 15

Use syms() function to define a symbolic variable and create mathematical expressions with it

```
% You can define a variable to be symbolic. After that, the expressions
% involuving such symbolic variables become "mathmetical expressions",
% rather than numerical expressions that need to evaluate to a number.
syms x
sqrt(x)
```

ans = $\sqrt{x}$

```
% Compare with a regular numerical variable:
t = 5;
sqrt( t )
```

ans = 2.2361

Storing mathematical expresions in variables.

You can store a mathematical expresion in a variable. The variable you store into automatically becomes a symbolic variable (You do not need to declare it using syms()).

```
syms x
f = 5*x^2 + 3*x + 10
```

f = $5x^2 + 3x + 10$

```
% You can now use f, which will essentially be replaced with the
% mathematical expression it contains.
diff(f, x)
```

ans = $10x + 3$

Converting a symbolic value to a numerical value: vpa() and double()

```
v = x * sqrt(56)
```

v = $2\sqrt{14}\,x$

```
% Use vpa() to convert every occurence of a numerical expression to it
% calculated value.
vpa( v )
```

ans = $7.483314773547882771167497464631x$

```
vpa( v, 4 ) % You can specifiy the number of significant digits (default is 32)
```

ans = $7.483x$

```
% if a symbolic expression does not contain any symbols, you can also use
% double() to convert it into a final numerical value.
v = sqrt(sym(56))
```

v = $2\sqrt{14}$

```
double( v )
```

```
ans = 7.4833
```

Calculus functions

```
int(x, x)  % indefinite integral
```

```
ans =
```

$$\frac{x^2}{2}$$

```
int(x, x, sym(1), sym(4))  % definite integral
```

```
ans =
```

$$\frac{15}{2}$$

```
%sym() above was used just to be safe, but you'd get the same result
%without it for this example
int(x, x, 1, 4)
```

```
ans =
```

$$\frac{15}{2}$$

```
% Derivative: diff()
diff( sin(x^5) , x)
```

```
ans =
```

$$5\,x^4\cos(x^5)$$

```
% Second, third, etc. derivative can be taken by repeatedly calling diff()
diff( diff( sin(x^5), x), x)
```

```
ans =
```

$$20\,x^3\cos(x^5) - 25\,x^8\sin(x^5)$$

```
% or by providing additional arugments to diff():
diff( sin(x^5) , x, x)
```

```
ans =
```

$$20\,x^3\cos(x^5) - 25\,x^8\sin(x^5)$$

```
diff( sin(x^5) , x, x, x)
```

```
ans =
```

$$60\,x^2\cos(x^5) - 125\,x^{12}\cos(x^5) - 300\,x^7\sin(x^5)$$

```
% Limit
% e.g., find the limit of (1+1/n)^n while n goes to infinity:
syms n
limit((1 + 1/n)^n, n, sym(inf))
```

```
ans =
```

$$e$$

It may not be possible to reduce the expression to elementary functions (or Mupad may not be able to find it).

```
f = int(inv(exp(x^2) + 1), x)
```

```
f =
```

$$\int \frac{1}{e^{x^2}+1}\,dx$$

But the differentiator recognizes that its derivative is the integrand:

```
diff(f, x)
```

```
ans =
```

$$\frac{1}{e^{x^2}+1}$$

Definite integrals may also be returned "unreduced"/"unsimplified"

```
v = int(inv(exp(x^2) + 1), x, sym(0), sym(1))
```

```
v =
```

$$\int_0^1 \frac{1}{e^{x^2}+1}\,dx$$

You can force calculation of a numerical value using vpa() or double()

```
double(v)
```

```
ans = 0.4195
```

Series Sum using **symsum()**

```
syms i n
symsum(i, i, sym(1), n)
```

```
ans =
```

$$\frac{n\,(n+1)}{2}$$

```
symsum(i^2, i, sym(1), n)
```

```
ans =
```

$$\frac{n\,(2\,n+1)\,(n+1)}{6}$$

Simplifying / manipulating expressions

Use simiplifyFraction() to find the common denominator for rational expressions and combine into a single ratio

```
f = x/(1 + x) - 2/(1 - x)
```

```
f =
```

$$\frac{x}{x+1}+\frac{2}{x-1}$$

```
simplifyFraction(f)
```

ans =

$$\frac{x^2+x+2}{(x-1)\ (x+1)}$$

simplifyFraction() will cancel out common factors between numerator and denominator in the result

```
f = x^2/(x + y) - y^2/(x + y)
```

f(t) =

$$\frac{x^2}{x+y(t)}-\frac{y(t)^2}{x+y(t)}$$

```
simplifyFraction(f)
```

ans(t) = $x-y(t)$

partfrac() (short for “partial fraction”) decomposes a rational expression into a sum of rational terms with simple denominators

```
f = (x^2 + x + 2) / (x^2-1)
```

f =

$$\frac{x^2+x+2}{x^2-1}$$

```
partfrac(f)
```

ans =

$$\frac{2}{x-1}-\frac{1}{x+1}+1$$

simplify() tries to find a representation that is as "simple" as possible.

```
f = (exp(x) - 1) / (exp(x/2) + 1)
```

f =

$$\frac{e^x-1}{e^{x/2}+1}$$

```
simplify(f)
```

ans = $e^{x/2}-1$

factor() decomposes an expression into product of simpler ones:

```
factor(x^3 + 3 * x^2 + 3 * x + 1)
```

ans = $(x+1 \quad x+1 \quad x+1)$

```
factor(2 * x * y + -2 * x + -2 * y + x^2 + y^2)
```

ans(t) = $(x+y(t)-2 \quad x+y(t))$

Defining your own symbolic "functions" that can take in variables.

```
syms x a b
F = @(x) x^2
```

```
F = function_handle with value:
    @(x)x^2
```

```
F(x)
```

ans = x^2

```
F(a + b)
```

ans = $(a+b)^2$

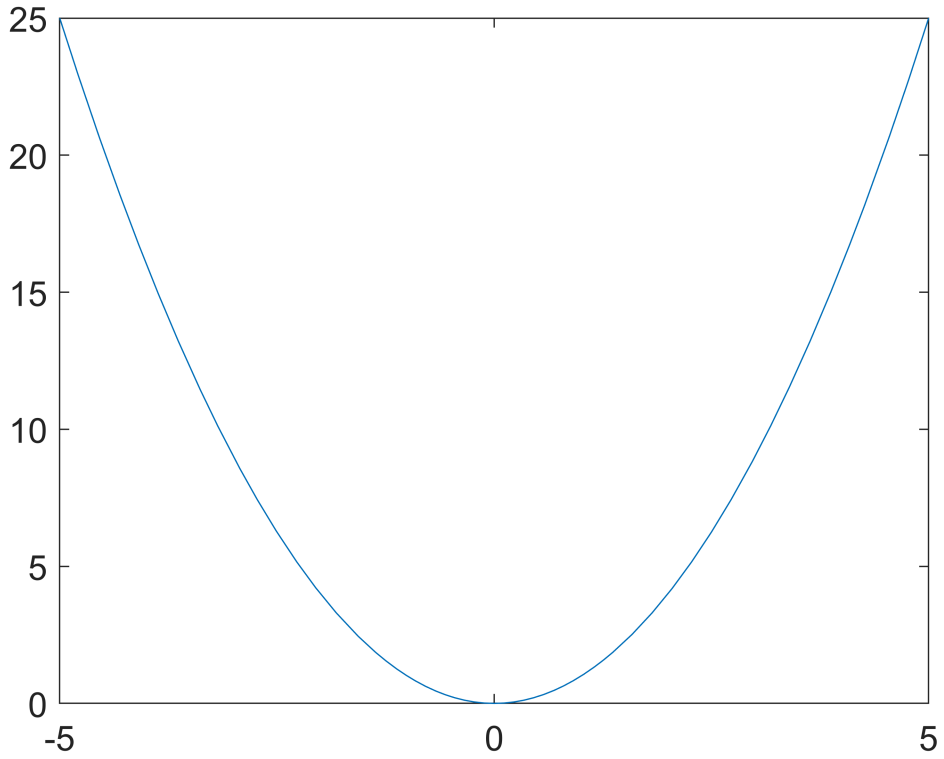
```
syms apple
F( apple )
```

ans = apple^2

Plotting with fplot()

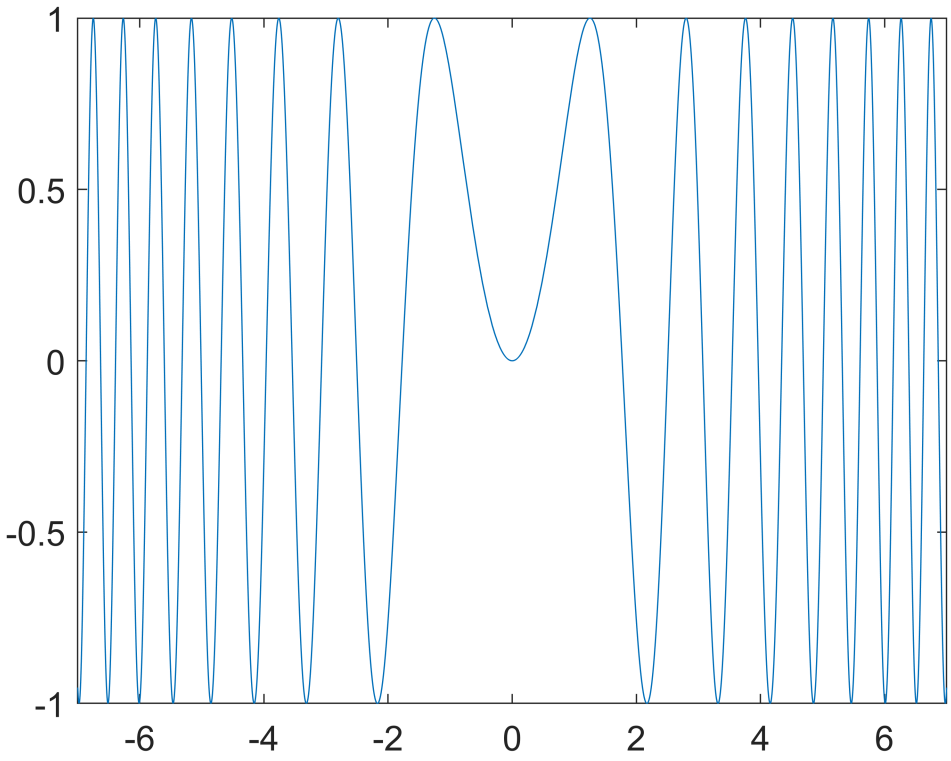
The best part of plotting in Symbolic Math is you don't need to worry about providing x values. Matlab will use a range of [-5...5] by default.

```
F = @(x) x.^2;
fplot(F)
```



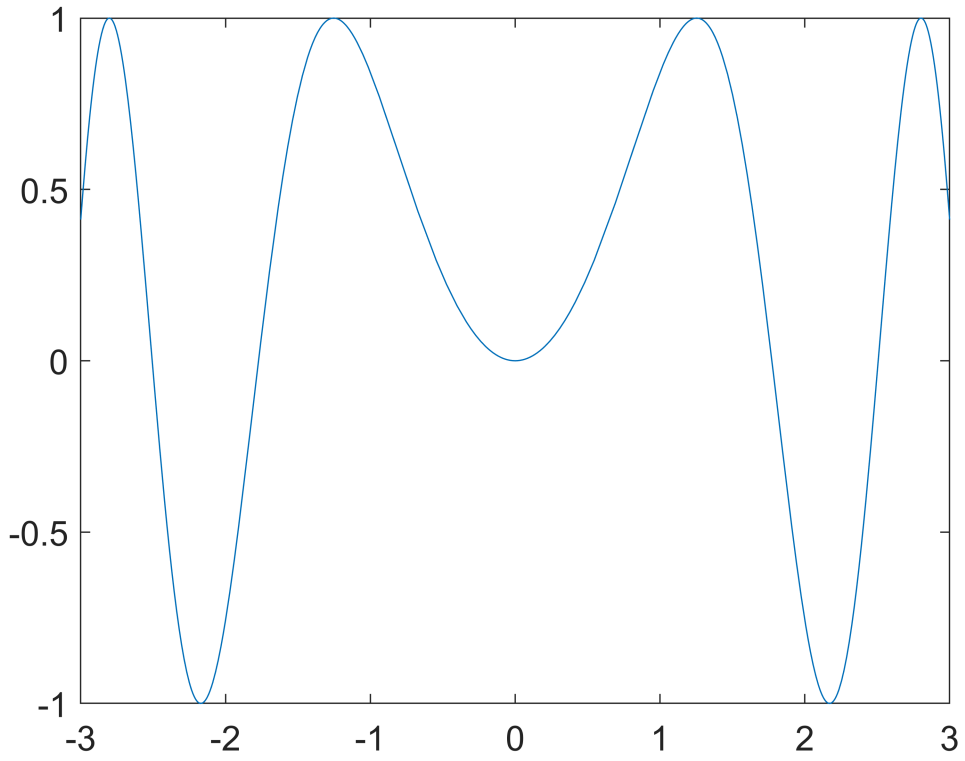
You can specify the x axis range after fplot() is called. Matlab will automatically update the fplot.

```
syms x
fplot(sin(x^2)) %a plot with x=[-5...5] would have been generated after this step.
axis([-7, 7, -Inf, Inf])
```



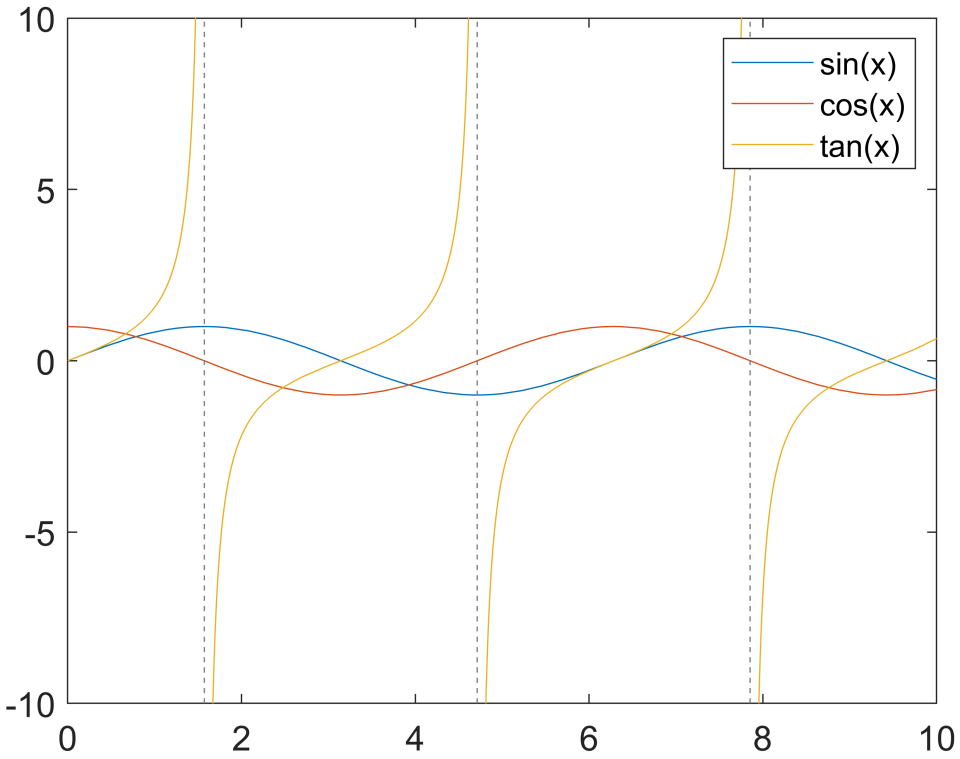
But the better method is to provide the range of x values when you call fplot()

```
fplot(sin(x^2), [-3 3])
```



hold on/ hold off still work to create multiple plots on the same figure.

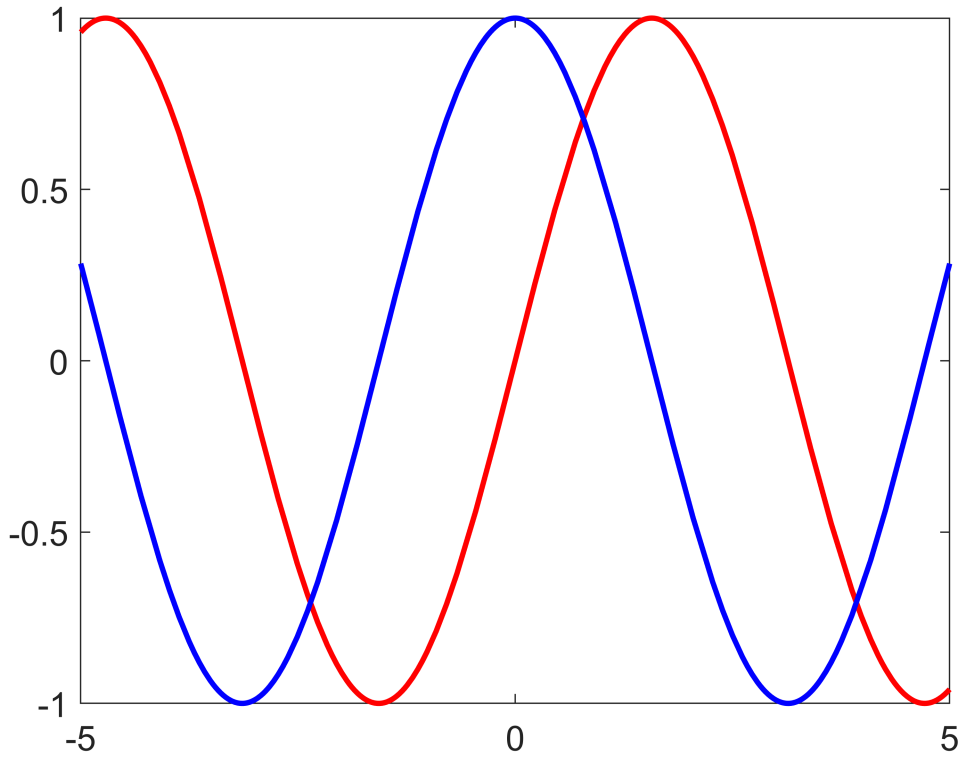
```
fplot(sin(x), [0, 10])
hold on
fplot(cos(x), [0, 10])
fplot(tan(x), [0, 10])
hold off;
% Matlab will automatically use the symbolic expressions as legends
legend show
```



If you don't like the default legends, use the "displayname" option.

Other usual options that you would use with plot() are still available with fplot().

```
fplot(sin(x), 'displayname','the sin function', 'color',[1 0 0], 'linewidth',2)
hold on
fplot(cos(x), 'displayname','the cos function', 'color',[0 0 1], 'linewidth',2)
hold off
```



Replacing variables in a symbolic expressions using subs()

```
syms a b c x
f = a * x^2 + b * x + c
```

f = $a\,x^2+b\,x+c$

```
g = subs(f, {a}, {4})
```

g = $4\,x^2+b\,x+c$

```
g = subs(f, {a, b, c}, {4, 5, -9})
```

g = $4\,x^2+5\,x-9$

Solving Mathematical Equations using solve()

```
syms a b c
% Remeber to use "==" when you express a symbolic equation.
solve(a * x + b == sym(0), x)
```

ans =

$-\frac{b}{a}$

```
% "equals zero" is implied when it is not explicitly given
```

```
solve(a * x + b , x)
```

ans =

$$-\frac{b}{a}$$

```
% if you omit what you are solving for, Matlab will guess it. Avoid this,
% as it can be reliable. It is better if you explicitly state what you are
% solving for.
solve(a * x + b )  % Avoid this.
```

ans =

$$-\frac{b}{a}$$

```
% When solution contains multiple roots, the results will be available as a
% list. You can retrieve each root individually.
sol = solve(a * x^2 + b * x + c , x)
```

sol =

$$\begin{pmatrix} -\frac{b+\sqrt{b^2-4ac}}{2a} \\ -\frac{b-\sqrt{b^2-4ac}}{2a} \end{pmatrix}$$

```
sol(1)
```

ans =

$$-\frac{b+\sqrt{b^2-4ac}}{2a}$$

```
sol(2)
```

ans =

$$-\frac{b-\sqrt{b^2-4ac}}{2a}$$

The roots can be real & complex.

```
solve(x^4 + x^3 + x * -1 == sym(1), x)
```

ans =

$$\begin{pmatrix} -1 \\ 1 \\ -\frac{1}{2}-\frac{\sqrt{3}\,\mathrm{i}}{2} \\ -\frac{1}{2}+\frac{\sqrt{3}\,\mathrm{i}}{2} \end{pmatrix}$$

You can omit any non-real solutions:

```
solve(x^4 + x^3 + x * -1 == sym(1), x, 'Real', true)
```

```
ans =
```

$$\begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Ignore unlikely edge cases with the IgnoreAnalyticConstraints option.

```
solve(log(y) + log(x + 1) == sym(1), y)
```

```
Warning: Unable to find explicit solution. For options, see help.
ans = struct with fields:
    t: [0×1 sym]
    x: [0×1 sym]
```

```
solve(log(y) + log(x + 1) == sym(1), y, 'IgnoreAnalyticConstraints', true)
```

```
Warning: Unable to find explicit solution. For options, see help.
ans = struct with fields:
    t: [0×1 sym]
    x: [0×1 sym]
```

Sometimes there are infinite number of solutions:

```
solve(sin(x) == sym(1), x)
```

```
ans =
```

$$\frac{\pi}{2}$$

To get only the "main" solution from the set, you can use the PrincipalValue option:

```
solve(sin(x) == sym(1), x, 'PrincipalValue', true)
```

```
ans =
```

$$\frac{\pi}{2}$$

If you don't know which options to use Ignore and you are being lazy, try to use all. :)

```
solve(sin(x) == sym(1), x, 'Real', true, 'PrincipalValue', true, 'IgnoreAnalyticConstraints', t
```

```
ans =
```

$$\frac{\pi}{2}$$

Solving systems of equations

The solve function can also be used to solve systems of equations. Simply specify the system of equations and the variables to solve for as vectors

```
% Multiple Equations can be specified as a list and solved simultaneously
syms x y
equations = [x + 2*y == 10,  x - y == 4]
```

```
equations = (x + 2 y = 10   x − y = 4)
```

```
sol = solve(equations, [x y])
```

```
sol = struct with fields:
    x: [1×1 sym]
    y: [1×1 sym]
```

```
% You can retrieve individual variables:
sol.x
```

```
ans = 6
```

```
sol.y
```

```
ans = 2
```

Another Example

```
equations = [x + y == a ,  x - a * y == b]
```

```
equations = (x + y = a   x − a y = b)
```

```
sol = solve(equations, [x y])
```

```
sol = struct with fields:
    x: [1×1 sym]
    y: [1×1 sym]
```

```
sol.x
```

```
ans =
```

$$\frac{a^2+b}{a+1}$$

```
sol.y
```

```
ans =
```

$$\frac{a-b}{a+1}$$

Working with matrices

We follow Matlab conventions for row and column matrices: Use space or comma to separate within a row. Use semicolon to move onto the next row.

```
a = [1 x]
```

a =

$$\begin{pmatrix} 1 & x \end{pmatrix}$$

```
b = [1; x]
```

b =

$$\begin{pmatrix} 1 \\ x \end{pmatrix}$$

```
m =  [1 x; -x 1]
```

m =

$$\begin{pmatrix} 1 & x \\ -x & 1 \end{pmatrix}$$

Linear Algebra operations can be done symbolically

```
syms a y z
A = [1 2; a, 4]
```

A =

$$\begin{pmatrix} 1 & 2 \\ a & 4 \end{pmatrix}$$

```
B = [y, 3; z, 5]
```

B =

$$\begin{pmatrix} y & 3 \\ z & 5 \end{pmatrix}$$

```
A + B
```

ans =

$$\begin{pmatrix} y+1 & 5 \\ a+z & 9 \end{pmatrix}$$

```
A * B
```

ans =

$$\begin{pmatrix} y+2z & 13 \\ 4z+ay & 3a+20 \end{pmatrix}$$

```
A^-1
```

ans =

$$\begin{pmatrix} -\frac{2}{a-2} & \frac{1}{a-2} \\ \frac{a}{2\,(a-2)} & -\frac{1}{2\,(a-2)} \end{pmatrix}$$

```
% let's replace the constant a with 0.5
subs(A^-1, a, 0.5)
```

ans =

$$\begin{pmatrix} \frac{4}{3} & -\frac{2}{3} \\ -\frac{1}{6} & \frac{1}{3} \end{pmatrix}$$

```
det(A) %determinant of matrix X
```

ans = $4-2\,a$

Solving system of equations in a matrix form.

In some cases, you may be able to express the system of equations in matrix form, .For these cases, you can use the linalg::matlinsolve function to solve the equations.

```
A = [1 3; 2 4]
```

```
A = 2×2
     1     3
     2     4
```

```
b = [5; 6]
```

```
b = 2×1
     5
     6
```

Let's solve Ax = b for x using mldivide "\" operation.

```
%use mldivide '\' operation instead of solve()
sol = A\b
```

```
sol = 2×1
    -1
     2
```

Let's check if the answer is correct.

```
A * sol
```

```
ans = 2×1
     5
     6
```

Solving Ordinary Differential Equations (ODEs)

You can solve various types of ordinary differential equations using the standard solve function (i.e. the same function that is used to solve algebraic equations) or the ode::solve function. If you use the standard solve function, you must first define your equation as an ODE. The same underlying algorithm is used regardless of which approach you choose.

First order differential equation example

```
% You must define y(t) as a symbol, rather than y and t separately.
syms y(t) a
eqn = diff(y, t) == a*y
```

```
eqn(t) =
```

$$\frac{\partial}{\partial t}\, y(t) = a\, y(t)$$

```
dsolve(eqn)
```

```
ans =
```

$$C_1\, \mathrm{e}^{a\, t}$$

2nd order linear ODE example

```
syms y(t) a
eqn = diff(y,t,2) + diff(y, t) + y(t) == 0
```

```
eqn(t) =
```

$$\frac{\partial^2}{\partial t^2}\, y(t) + \frac{\partial}{\partial t}\, y(t) + y(t) = 0$$

```
dsolve( eqn )
```

```
ans =
```

$$C_1\, \mathrm{e}^{-\frac{t}{2}} \cos\left(\frac{\sqrt{3}\, t}{2}\right) - C_2\, \mathrm{e}^{-\frac{t}{2}} \sin\left(\frac{\sqrt{3}\, t}{2}\right)$$

You can specify initial conditions and boundary conditions along with an ODE, which is often critical for real world problems in engineering and physics.

```
syms y(t)
eqn = diff(y,t,2) + diff(y,t) + y(t) == 0
```

```
eqn(t) =
```

$$\frac{\partial^2}{\partial t^2}\, y(t) + \frac{\partial}{\partial t}\, y(t) + y(t) = 0$$

```
% in order to specify y'(0)=1, create Dy function first, and then specify
```

```
% Dy(0)==1 as a condition.
Dy = diff(y,t)
```

```
Dy(t) =
```

$$\frac{\partial}{\partial t}\, y(t)$$

```
cond = [y(0) == 0,  Dy(0) == 1];
sol = dsolve(eqn, cond )
```

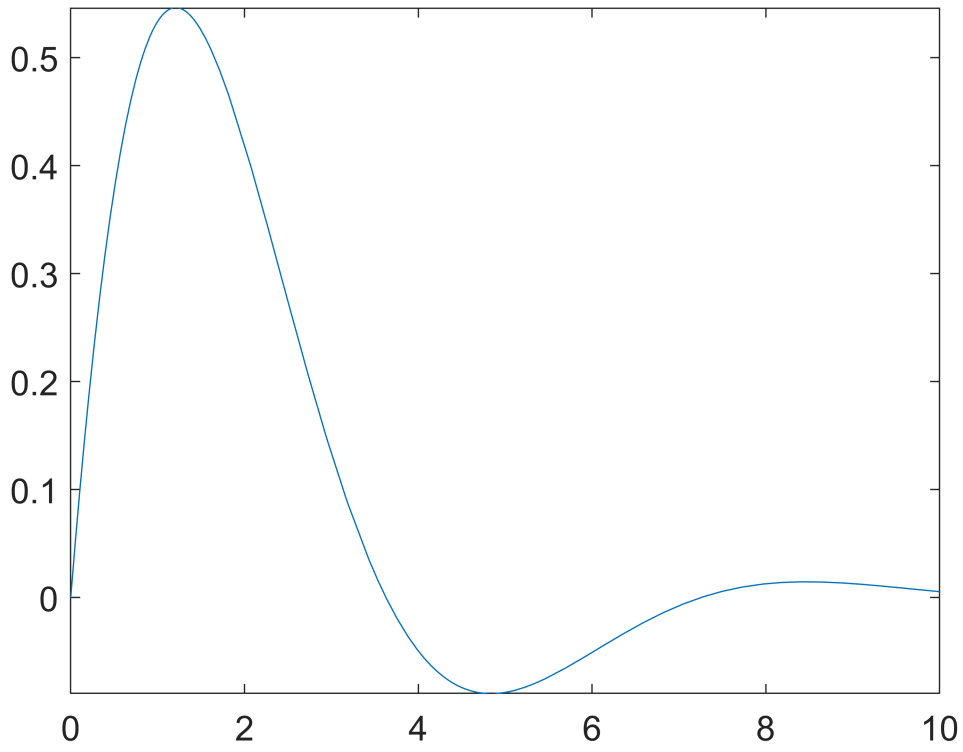
```
sol =
```

$$\frac{2\sqrt{3}\, e^{-\frac{t}{2}} \sin\left(\frac{\sqrt{3}\, t}{2}\right)}{3}$$

```
vpa(sol)
```

```
ans = 1.154700538379251529018297561039e-05t sin(0.866025403784438646763723170752)
```

```
fplot( sol, [0, 10])
```



Another 2nd order ODE example

```
syms y(t)
Dy = diff(y,t)
```

```
Dy(t) =
```

$$\frac{\partial}{\partial t}\, y(t)$$

```
D2y = diff(y,t,2)
```

```
D2y(t) =
```

$$\frac{\partial^2}{\partial t^2}\, y(t)$$

```
eqn = D2y == 2*t/Dy
```

```
eqn(t) =
```

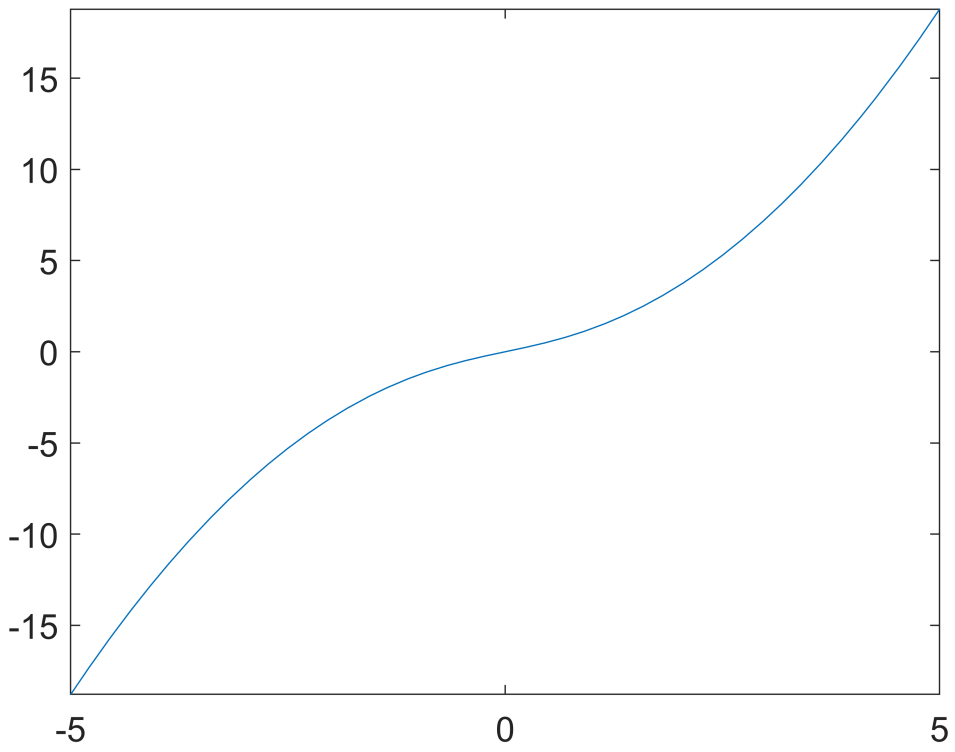
$$\frac{\partial^2}{\partial t^2}\, y(t) = \frac{2\,t}{\frac{\partial}{\partial t}\, y(t)}$$

```
cond = [y(0)==0, Dy(0)==1];
sol = dsolve(eqn, cond)
```

```
sol =
```

$$\frac{\sqrt{2}\,\log\left(t+\sqrt{t^2+\frac{1}{2}}\right)}{4} - \frac{\sqrt{2}\,\log\left(\frac{\sqrt{2}}{2}\right)}{4} + \frac{\sqrt{2}\,t\,\sqrt{t^2+\frac{1}{2}}}{2}$$

```
fplot(sol)
```



```
fplot( sol, [0,10])
```