
Symbolic Solutions

Table of Contents

The Symbolic Approach	1
Solving One Equation Symbolically	1
Which Variable to Solve For?	2
Solving When Not Equal to Zero	2
Solving a System of Equations Symbolically	2
Order of The Variables in the Solution	4

We can ask MATLAB to try to solve equations two different ways. MATLAB can sometimes obtain a symbolic solution by manipulating the symbols in the equation(s) much like you would do with pencil and paper in an introductory math class. Another way is to find a numerical answer which may be approximate.

The Symbolic Approach

This approach works well for some problems. Unfortunately there is a large class of problems that will defy attempts to solve them this way. For example try solving (by hand) the equation

$$\ln(x) + x + 1 = 0$$

It looks easy to solve but it isn't! An equation like this one can be solved using a numerical method which will yield a very good approximation to the precise answer. The numerical approach usually involves taking an initial estimate of the correct solution and then repeating some kind of calculation over and over to gradually improve the estimate.

Solving One Equation Symbolically

Suppose you want to find the solutions to the quadratic equation

$$x^2 - 5x = 0$$

Matlab can solve this with the `solve` command. By default this solves the symbolic expression equal to zero. First we symbolically define our variable `x` and then apply the command. Try this:

```
syms x
solve (x^2 - 5 * x)
```

ans =

0
5

That was easy! MATLAB found both of the solutions. You can also ask MATLAB to solve equations that involve arbitrary constants. A nice example is to try to find the general solutions to the generalized form of the quadratic equation $a*x^2 + b*x + c = 0$. What do you think the solutions should look like? Let's see if you're right...

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

ans =

$$\begin{aligned} & -(b + (b^2 - 4*a*c)^{(1/2)})/(2*a) \\ & -(b - (b^2 - 4*a*c)^{(1/2)})/(2*a) \end{aligned}$$

Do those solutions look familiar? You may have to mentally re-arrange them to make them look the way you're used to seeing them!

Which Variable to Solve For?

How does Matlab know which variable to solve for? It tries to follow common-sense respecting what we usually solve for. Since the variable x is so common it solves for that one first. Suppose we wished to solve $a*x^2+b*x+c=0$ for a ; we'd have to tell Matlab as follows:

```
syms a b c x
solve(a*x^2+b*x+c, a)
```

ans =

$$-(c + b*x)/x^2$$

Solving When Not Equal to Zero

If we're interested in solving an equation which is not equal to zero then we have to enclose the expression in apostrophes (single quotes):

```
syms x
solve('2*x+3 = 7')
```

ans =

2

Solving a System of Equations Symbolically

You can use the `solve` command for a whole system of equations as well. For example, suppose we are trying to find the solution to the following system of equations:

$$\begin{aligned} 3*x+4*y+z-7&=0 \\ x-y-15&=0 \\ 6*x-2*y-5*z+11&=0 \end{aligned}$$

We can try to use the `solve` command to do this by feeding it all of the equations at once, separating them with commas:

```
syms x y z
solve (3*x+4*y+z-7,x-y-15,6*x-2*y-5*z+11)
```

```
ans =

      x: [1x1 sym]
      y: [1x1 sym]
      z: [1x1 sym]
```

Hmmmm. That output is a bit disappointing! To enable us to see the actual answers conveniently, we can introduce an intermediate variable that will store the whole "vector" of answers. Then we can view the values individually. Here's how that might look:

```
syms x y z
MyAnswers = solve (3*x+4*y+z-7,x-y-15,6*x-2*y-5*z+11)
```

```
MyAnswers =

      x: [1x1 sym]
      y: [1x1 sym]
      z: [1x1 sym]
```

Now we can ask to see each of the three values individually, as follows:

```
MyAnswers.x
```

```
ans =

98/13
```

```
MyAnswers.y
```

```
ans =

-97/13
```

```
MyAnswers.z
```

```
ans =

185/13
```

Another way to accomplish the same thing is by assigning the value of the computation to a vector comprised of variables like this:

```
[x y z] = solve (3*x+4*y+z-7,x-y-15,6*x-2*y-5*z+11)
```

$x =$ $98/13$ $y =$ $-97/13$ $z =$ $185/13$

Notice that the answers are displayed as fractions rather than decimal expansions. Does that surprise you? It actually makes sense because Matlab solved the equations symbolically. If you solved these equations by hand, the last operation you would do in calculating z would be to take 185 and divide it by 13. Although you could try to evaluate this as a decimal why should you? The expression $185/13$ is exact. If this were written as a decimal it would be $14.230769230769230769230769\dots$. We could only write down an approximation this way and besides -- it looks really hideous!

Order of The Variables in the Solution

The `solve` command returns a vector with entries in alphabetical order. For example

```
solve(x+y,x-y+2)
```

will return the vector whose first entry is x and whose second entry is y . If you do something like

```
[y x]=solve(x+y,x-y+2)
```

then what happens is that y gets assigned to the x solution and x gets assigned to the y solution. This is not what you want.

To unconfuse things make sure that your entries in the vector on the left are in alphabetical order! For example if your variables are `dogs` and `cats` then *don't* do:

```
[dogs,cats] = ...
```

instead do

```
[cats,dogs] = ...
```

Published with MATLAB® 8.0