Maple for Math Majors

Roger Kraft
Department of Mathematics, Computer Science, and Statistics
Purdue University Calumet
roger@calumet.purdue.edu

3. Solving Equations

3.1. Introduction

The **solve** command for solving equations symbolically is one of Maple most useful commands. In this worksheet we go into the details of using the **solve** command to solve single equations and systems of equations. We also consider more carefully what kinds of equations **solve** can and cannot solve. We introduce Maple's RootOf expressions, which are used throughout Maple and are often used in the results returned by **solve**, and the **allvalues** command for interpreting RootOf expressions. Finally, we show how the numerical solving command **fsolve** can be used when **solve** does not work.

```
> 
```

3.2. The basics of solve

We give the **solve** command an equation in some unknowns and also one specific unknown that it should solve the equation for. For example, an equation like $1 = a\,x^2 - b$ has three unknowns in it. Each of the following **solve** commands solves this equation for one of the unknowns.

```
> solve( 1=a*x^2-b, a );
> solve( 1=a*x^2-b, b );
> solve( 1=a*x^2-b, x );
```

The next two commands check that the last solution really does solve the equation for **x**.

```
> subs( x=%[1], 1=a*x^2-b );
> subs( x=%%[2], 1=a*x^2-b );
```

If we put a pair of braces around the unknown to solve for, then **solve** returns the solution written as an equation.

```
> solve( 1=a*x^2-b, {a} );
> solve( 1=a*x^2-b, {b} );
> solve( 1=a*x^2-b, {x} );
```

The next two commands once again check that the last solution solves the equation for **x**. Notice how the form of the **subs** command changes slightly because of the **braces** around **x** in the solve command.

```
> subs( %[1], 1=a*x^2-b );
> subs( %%[2], 1=a*x^2-b );
```

In each of these examples, **solve** is solving for one variable in terms of the other two variables,

and so we expect **solve** to return an expression. In the rest of this worksheet we emphasize equations in just one variable (or systems of two equations in two variables) where we expect **solve** to return specific numbers that **solve** the equation.

```
> 
```

```
> 
```

3.3. Solving a single polynomial equation

If we ask **solve** to solve a polynomial equation (in one variable) of degree n with $n \le 3$, then **solve** will always produce n explicit solutions. Some of the solutions may be complex numbers. Here are a few simple examples.

```
> solve( x^2+1=0, x );
```

```
> solve( x^2+x-1=0, {x} );
```

```
> solve( x^3-4*x^2+8*x-8=0, x );
```

When the degree of the polynomial is 3, the **solve** command can produce some pretty impressive looking answers.

```
> solve( x^3-x^2-x-1=0, {x} );
```

If we give **solve** a polynomial equation of degree n with $4 \le n$, then **solve** will always produce n solutions, but some of the solutions may not be explicitly given. Instead, some of the solutions may be represented by a **RootOf** expression. Here is an example.

```
> solve( x^5+2*x^4-4*x^3-4*x^2-4*x^1-3=0, {x} );
```

One of the solutions of the equation is given explicitly, –3, and the other four solutions are expressed by the **RootOf** part of the result. A **RootOf** expression usually (but not always) means that Maple does not know how to find explicit symbolic solutions for the roots of some polynomial, in this case the polynomial **x^4-x^3-x^2-x-1**.

```
> solve( x^4-x^3-x^2-x-1=0, {x} );
```

When **solve** returns a **RootOf** expression, we can use the **allvalues** command to find out more about the solutions represented by the **RootOf** expression. In the case where a **RootOf** expression contains a polynomial of degree 4, the **allvalues** command will in fact return symbolic values for the roots, but as the next command shows, the symbolic result may be so incredibly large and complicated that it is almost useless (and that is why it was represented by a **RootOf** expression).

```
> allvalues( % );
```

As we said above, the last symbolic result is so complicated that it is almost useless. So we use the **evalf** command to get decimal approximations of these solutions.

```
> evalf( % );
```

When a **RootOf** expression contains a polynomial of degree 5 or more, then the **allvalues** command applied to the **RootOf** expression may return decimal approximations of the roots represented by the **RootOf** expression. In this case, Maple actually cannot return symbolic results and decimal approximations are the only option. Here is an example.

```
> solve( x^5-x^4-x^3-x^2-x-1=0, x );
```

```
> allvalues( % );
```
What if we would like to convince ourselves that these really are correct solutions? The next command substitutes the above results into the original polynomial.
```
> for i in % do subs( x=i, x^5-x^4-x^3-x^2-x-1=0 ) od;
```
When we substitute a solution into the original equation, we expect to get the equation $0 = 0$, and that is what we get with two of the solutions above. But for the single solution that is a real number, we get the equation $.6\ 10^{(-8)} = 0$ when the solution is substituted into the original equation. Does this mean that we do not have a correct solution? Yes and no. The "solution" in question is 1.965948237. This numerical result is meant to be an approximation of an exact solution, so we should not expect it to be a correct, exact solution. When this result is substituted into the original equation, we should not expect to get exactly $0 = 0$. But we should expect to "almost" get $0 = 0$, and that is what we do get, since $.6\ 10^{(-8)}$ is a very small number. So the equation $.6\ 10^{(-8)} = 0$ tells us that 1.965948237 is an approximate solution to the original equation, not an exact solution. Similarly, the above substitutions tell us that the other two complex solutions are also correct approximations of exact solutions.
```
>
```

With polynomial equations, **solve** can always return all of the solutions to the equation, some of the solutions as exact symbolic results and some possibly as decimal approximations. On the other hand, with non polynomial equations there is no guarantee that **solve** will return all solutions of the equation, or even any solutions. In the next section we look at some examples of the kinds of results that **solve** can return for non polynomial equations.
```
>
```

```
>
```

3.4. Solving a single nonlinear equation

In the last section we saw that with a polynomial equation, **solve** will always returns all the solutions to the equation. On the other hand, with non polynomial equations there is no guarantee that **solve** will return all solutions of the equation, or even any solutions. Here are some examples of the kinds of results that **solve** can return for non polynomial equations.

The equation $x^3 - \cos(3\,x^2) = 0$ three real roots, but **solve** cannot find symbolic expressions for any of them, and so it does not return a result.
```
> solve( x^3-cos(3*x^2)=0, x );
```
We can show that the equation has three real roots by using a graph.
```
> plot( [x^3, cos(3*x^2)], x=-2..2, -2..2 );
```
```
>
```

Exercise: In the last **plot** command, change the comma just after **x^3** to a minus sign.
```
>
```

We know that the equation $\sin(x) = 0$ has an infinite number of solutions, but if we solve it with **solve**, we only get one solution.

```
> solve( sin(x)=0, x );
>
```

The equation $\sin(x^2 - 1) = 0$ also has an infinite number of solutions, which we can verify by graphing the function $\sin(x^2 - 1)$.

```
> plot( sin(x^2-1), x=-8..8 );
```

But if we solve the equation with **solve**, we get only two solutions, 1 and -1. This is because to **solve**, **sin(x^2-1)=0** is true when **x^2-1=0**, which is solved by either **1** or **-1**.

```
> solve( sin(x^2-1)=0, x );
>
```

The equation $\sin(x^2 + 1) = 0$ also has an infinite number of real solutions, which we can again verify using a graph.

```
> plot( sin(x^2+1), x=-8..8 );
```

But if we solve the equation using **solve**, we get two imaginary solutions.

```
> solve( sin(x^2+1)=0, x );
>
```

Exercise: Explain why **solve** returned two imaginary solutions for the equation $\sin(x^2 + 1) = 0$.

```
>
```

Similarly, for the two equations $\sin(x^2 + 1) = 1$ and $\sin(x^2 + 1) = -1$, both of which have an infinite number of real solutions, **solve** will return two real solutions and two imaginary solutions, respectively.

```
> solve( sin(x^2+1)=1, x );
> solve( sin(x^2+1)=-1, x );
>
```

Exercise: The number π to five decimal places is 3.1415. So the two equations

$$\sin(x^2 + 1 + \pi) = -1$$

and

$$\sin(x^2 + 4.1415) = -1$$

are almost exactly the same equation. Explain why the following command produces two real solutions

```
> solve( sin(x^2+1+Pi)=-1, x );
> evalf( % );
```

but the following command produces two imaginary solutions. (In the following command **4.1415** is written as **41415/10000** in order to prevent **solve** from returning a decimal answer.)

```
> solve( sin(x^2+41415/10000)=-1, x );
> evalf( % );
>
```

Recall that with polynomial equations, **solve** sometimes represents solutions with a **RootOf** expression. We can get also**RootOf** expressions as a result of solving non polynomial equations. Here is a simple example.

```
> solve( x=cos(x), x );
```

As with the case of polynomial equations, we can use the **allvalues** command to find out what values are represented by the **RootOf** expression.

```
> allvalues( % );
```

So we got a decimal approximation to one solution of the equation. The following graph shows us that this is the only real solution to the equation.

```
> plot( [x, cos(x)], x=-3*Pi/2..3*Pi/2, -2..2 );
>
```

With non polynomial equations, we can even get results from **solve** that contain nested **RootOf** expressions. Here is an example.

```
> solve( x^2=cos(x^2), x );
```

Again we use the **allvalues** command to find out what values are represented by the **RootOf** expression.

```
> allvalues( % );
```

We got decimal approximations to two solutions of the equation. The following graph shows us that these are the only two real solutions to the equation.

```
> plot( [x^2, cos(x^2)], x=-3..3 );
```

Let us go back to the nested **RootOf** expression returned by the **solve** command and try to see exactly how it represents the solutions to the equation $x^2 = \cos(x^2)$. Here is the **RootOf** expression again.

```
> solve( x^2=cos(x^2), x );
```

To derive this **RootOf** expression, let us start with the equation $x^2 = \cos(x^2)$ and make a substitution $\_Z = x^2$ in the equation. So then we have the equation $\_Z = \cos(\_Z)$. Let the expression RootOf(_Z-cos(_Z)) denote a solution of the equation $\_Z - \cos(\_Z) = 0$. Given the solution RootOf(_Z-cos(_Z)) of the equation $\_Z - \cos(\_Z) = 0$, we plug this solution back in for _Z in the substitution $\_Z = x^2$ and we get the equation $\text{RootOf}(\_Z - \cos(\_Z)) = x^2$. Solving this equation for x gives us a solution to our original equation. Let the expression $\text{RootOf}(-\text{RootOf}(\_Z - \cos(\_Z)) + x^2)$ denote a solution to the equation $x^2 - \text{RootOf}(\_Z - \cos(\_Z)) = 0$. Then $\text{RootOf}(-\text{RootOf}(\_Z - \cos(\_Z)) + x^2)$ also represents a solution to our original equation $x^2 = \cos(x^2)$. To get the nested **RootOf** expression returned by **solve**, the only thing left to do is replace the variable x in our last nested RootOf expression with the variable _Z. We can do this because changing the name of the variable used in an equation does not affect the solution values of the equation. Notice that this means that the variable _Z that appears

in the outer RootOf expression is *not* the same variable _Z that appears in the inner RootOf expression (the inner _Z is the unknown in the equation _Z-cos(_Z)=0, and the outer _Z is the unknown in the equation $r - \_Z^2 = 0$ where r is a solution of the former equation). The fact that there can be two distinct variables both called **_Z** in a nested **RootOf** expression is one of the things that can make nested **RootOf** expressions hard to understand.

```
> 
```

Exercise: Explain how each of the following three **solve** commands derived its **RootOf** expression.

```
> solve( x-sin(x^2)=0, x );
```

(Hint: First take the arcsin of both sides of the equation $x = \sin(x^2)$, and then let $x = \sin(\_Z)$.)

```
> solve( x-sin(x^2)=1, x );
```

(Hint: First make a substitution $u = x - 1$. Later, let $u = \sin(\_Z)$.)

```
> solve( x^2-sin(x^2)=1, x );
```

(Hint: Let $w = x^2$.)

```
> 
```

Exercise: First, explain how **solve** derived the following nested **RootOf** expression from the equation $x^2 = \sin(x^2 + 1)$.

```
> solve( x^2-sin(x^2+1)=0, x );
```

Now explain how the following nested **RootOf** expression can also be derived from the equation $x^2 = \sin(x^2 + 1)$.

```
> RootOf( -RootOf(_Z-sin(_Z)-1)+1+_Z^2 );
```

To prove that the last two nested **RootOf** expressions both represent the same solutions to the equation, let us evaluate both of them using **allvalues**.

```
> soln1 := solve( x^2-sin(x^2+1)=0, x );
> soln2 := RootOf( -RootOf(_Z-sin(_Z)-1)+1+_Z^2 );
> allvalues( soln1 );
> allvalues( soln2 );
```

The following graph shows that these are in fact the only two real solutions to the equation.

```
> plot( [x^2, sin(x^2+1)], x=-3..3, -1..2 );
> 
```

Exercise: One of the amazing things that Maple can do is algebra with RootOf expressions. Here is an example (from *A Guide to Maple*, by E. Kamerich, page 174).

```
> r := solve( x^5-t*x^2+p=0, x );
> simplify( r^7 );
> simplify( r^8 );
```

Explain why these results are correct. (Hint: Derive the first result from the equation in the **solve** command. Derive the second result from the first one.) Also, explain why the following result is correct.

```
> simplify( r^11 );
>
```

The last several examples have shown what kind of results we can get from the **solve** command with non polynomial equations. In summary, we can get no solution, a list of solutions that may or may not be complete, or we can get **RootOf** expressions that, when evaluated using **allvalues**, give us a list of decimal approximations of solutions. And the solutions that we get can be either real or complex numbers.

```
>
```

```
>
```

3.5. Solving a system of equations

We can use the **solve** command to solve systems of equations, that is, two or more equations that we want solved simultaneously. When we work with systems of equations we put the equations to be solved inside of a pair of braces and we put the variables to be solved for inside another pair of braces. Here is an example with two (nonlinear) equations in two unknowns.

```
> solve( {x^2-y^2-y=0, x+y^2=0}, {x, y} );
```

As with solving a single equation, sometimes we can get the answer given implicitly in terms of RootOf expressions. As before, we can use the **allvalues** command to find out more about the solutions represented by the RootOf expressions. Sometimes **allvalues** will return symbolic solutions and sometimes it will return decimal approximations. In the case of the last example, **allvalues** returns a list of very complicated symbolic solutions.

```
> allvalues( %[2] );
```

We can get decimal approximations of these solutions by using **evalf**.

```
> evalf( % );
```

Notice that the RootOf expressions represented three solutions, one real solution and two complex solutions. The following commands check that these really are solutions.

```
> seq( subs(%%[i], {x^2-y^2-y=0, x+y^2=0}), i=1..3 ):
> simplify( [%] );
>
```

Here is an interesting result where **solve** mixes a partly symbolic solution with a RootOf expression.

```
> solve( {x^2+y^2=9, x^y=2}, {x,y} );
```

And **allvalues** mixes symbolic results with numeric values.

```
> allvalues( % );
```

Let us check that this is a valid solution.

```
> subs( %, {x^2+y^2=9, x^y=2} );
> simplify( % );
>
```

Exercise: How many real solutions does the system of equations $x^2 + y^2 = 9$ and $x^y = 2$ have? (Hint: Graph the equations.)

```
> 
> 
```

Here is an example that shows how important it is to check that the results from **solve** and **allvalues** are correct. In this example, **allvalues** will produce incorrect results. Let us find the intersection of a circle and a parabola. The system of equations $x^2 + y^2 = 1$ and $y = x^2$ obviously has exactly two real solutions, and they are not very hard to find using paper and pencil. Here is one way to solve them using Maple.

```
> equations := {x^2+y^2=1, y-x^2=0};
> solve( equations, {x,y} );
> solutions := allvalues( % );
```

It looks like we have eight solutions. Some of them are complex. Here is an easy way to see this.

```
> evalf( solutions );
```

Notice that only four of the eight solutions are complex. This means that we have four real solutions, which is clearly two too many. Close inspection of the numerical values shows which two real solutions are incorrect (the ones with negative *y* values). But what about the complex solutions? Are any of them correct? (It is easy to see that two of the complex values are incorrect; which ones?) The next command checks all of the solutions to see how many are correct.

```
> seq( subs( solutions[i], equations ), i=1..8 );
> simplify( [%] );
```

We can now clearly see that four of the results returned by **allvalues** are incorrect; they do not solve the original system of equations. Never take the results from Maple at face value! Learn how to verify and double check all of Maple's results.

```
> 
```

Exercise: Go back and change the definition of **equations** from **{x^2+y^2=1, y-x^2=0}** to **{x^2+y^2=1, y=x^2}** and then re-execute the rest of the commands in the example. What affect does this minor change in the form of the equations have on the example? Is one form of the equations preferable?

```
> 
```

Exercise: Part (a): Solve the system of equations $x^2 + y^2 = 1$ and $y - x^2 = 0$ using paper and pencil. Pay close attention to your steps. Notice that there are two approaches to solving the problem; either substitute for the x^2 term in the equation $x^2 + y^2 = 1$ or substitute for the y^2 term.

```
> 
```

Part (b): Verify that the nested RootOf expressions returned by **solve** are correct, that is, show how they can be derived from the system of equations. Which of the two approaches mentioned in part (a) did the **solve** command choose?

```
> 
```

Part (c): Figure out exactly what mistake was made by the **allvalues** command when it interpreted the RootOf expressions returned by **solve**. (Hint: Part (d) of this exercise.)

```
> 
```

Part (d): Use the next command to read about the keywords **dependent** and **independent** in the online documentation for **allvalues**. Pay particular attention to the very last example in the section of examples (the last three commands in the section). Should these two keywords be relevant to our solution of the system $x^2 + y^2 = 1$ and $y - x^2 = 0$?

```
> ?allvalues
```

Part (e): Write a RootOf expression that represents solutions of the system $x^2 + y^2 = 1$ and $y - x^2 = 0$ but is different than the RootOf expression returned by **solve**. (Hint: Use the approach mentioned in part (a) that is not used by **solve**.) Use **allvalues** to check your RootOf expression.

```
> 
```

Exercise: Use **solve** to solve the equation $x^2 = 7\cos(x^2)$. Convert that equation to the equivalent system of equations $y = x^2$ and $y = 7\cos(x^2)$ and use **solve** to solve this system. Then convert the equation to the equivalent system $y = x^2$ and $y = 7\cos(y)$ and use **solve** to solve this system.

```
> 
```

```
> 
```

3.6. Using fsolve

We have seen that the **solve** command does not always produce all of the solutions of an equation or system of equations. When that happens, and we need to know the value of some solution that **solve** did not find, then we need to use the **fsolve** command. The **fsolve** command is used to find decimal approximations to solutions of equations.

For a single polynomial equation, **fsolve** will return decimal approximations for all of the real roots of the equation. Here is an example.

```
> fsolve( 1-x-2*x^3-x^4=0, x );
```

To also get decimal approximations for the complex roots of the polynomial equation we need to use the keyword **complex**.

```
> fsolve( 1-x-2*x^3-x^4=0, x, complex );
```

As with the **solve** command, if we put braces around the unknown that we are solving for, then **fsolve** returns the results as equations (which can make the results a bit easier to read).

```
> fsolve( 1-x-2*x^3-x^4=0, {x}, complex );
```

For a single polynomial equation, using **fsolve** does not provide us with anything that we could not get using **solve** together with **allvalues** and **evalf**. The real use for **fsolve** is with non polynomial equations and with systems of equations.

For a single, non polynomial equation, **fsolve** will try to return a decimal approximation for one real root of the equation. Even if the equation should have several real roots, **fsolve** only tries to

approximate one of the roots. Recall that earlier we showed that the equation $x^3 - \cos(3\,x^2) = 0$ has three real roots but **solve** could not find any of them. The **fsolve** command will readily return one of the roots.

```
> fsolve( x^3-cos(3*x^2)=0, x );
```

Let us check this result.

```
> subs( x=%, x^3-cos(3*x^2)=0 );
> simplify( % );
```

This is close enough to 0 = 0 to convince use that **fsolve** really has found an approximate solution. But what about the other real solutions? To find them, we need to use an option for **fsolve** that allows us to give **fsolve** a hint about where it should look for a solution. But how do we know where **fsolve** should look? We use a graph of the equation (which, you will recall, is also how we figured out that the equation actually had real solutions).

```
> plot( x^3-cos(3*x^2), x=-1..1 );
```

From the graph we see that there are two more real solutions near –1. The following **fsolve** command will find one of these two solutions. We give **fsolve** a hint of where to find a solution by giving **fsolve** a range that we know includes a solution.

```
> fsolve( x^3-cos(3*x^2)=0, x, -1..0 );
```

Since the range we gave **fsolve** included two solutions, we really could not predict which one of them **fsolve** would find. To find the other solution, we give **fsolve** a more specific range to look in.

```
> fsolve( x^3-cos(3*x^2)=0, x, -1..-0.9 );
```

So now we have approximate values for all three real solutions of the equation. This example is typical of how **fsolve** gets used with a single equation. We need a graph of the equation to give us an idea of how many (real) solutions there are and about where they are. Then we use this rough information from the graph to give **fsolve** hints so that it can find each of the solutions.

```
>
```

There is also another way to give **fsolve** a hint about where to find a solution. We can give **fsolve** a "starting value" for the unknown in the equation.

```
> fsolve( x^3-cos(3*x^2)=0, x=-1 );
> fsolve( x^3-cos(3*x^2)=0, x=-.8 );
```

This way of giving a hint to **fsolve** is not as good as the previous way. For a given starting value, it is often difficult to predict which solution **fsolve** will find. It is not uncommon for **fsolve** to find a solution that is far away from a starting value even when there is another solution very near to the starting value. Here is an example of the unpredictability of **fsolve** using a starting value. In the following command, out of the three real roots of the equation, **fsolve** finds the one that is furthest from the starting value.

```
> fsolve( x^3-cos(3*x^2)=0, x=0 );
>
```

What about complex solutions for the equation? To get **fsolve** to look for a complex solution we need to use the keyword **complex**. But the keyword **complex**, by itself, need not lead to a

complex solution.

```
> fsolve( x^3-cos(3*x^2)=0, x, complex );
```

In the next command we give **fsolve** a complex starting value along with the keyword **complex**, and then it finds a complex solution.

```
> fsolve( x^3-cos(3*x^2)=0, x=I, complex );
```

A different complex starting value leads to a different complex solution.

```
> fsolve( x^3-cos(3*x^2)=0, x=2+I, complex );
>
```

Exercise: What happens if you give **fsolve** a complex starting value but not the keyword **complex**?

```
>
```

Exercise: Part (a): How many real solutions does the equation $x^2 = 7\cos(x^2)$ have? Use **fsolve** to find approximations to all of the solutions. How many solutions does **solve** along with **allvalues** find?

```
>
```

Part (b): Can you find any complex solutions to the equation in part (a)? (Hint: Try many different complex starting values.)

```
>
```

Part (c): How many real solutions does the equation $z = 7\cos(z)$ have? Use **fsolve** to find approximations to all of the solutions. How many solutions does **solve** along with **allvalues** find?

```
>
```

Part (d): How are the solutions from part (c) related to the solutions from parts (a) and (b)?

```
>
```

Part (e): Can you find any complex solutions to the equation in part (c)?

```
>
```

Part (e): How are the solutions from part (e) related to the solutions from part (b)?

```
>
```

It is interesting to see what happens when we give **fsolve** a bad hint. In each of the following two commands, there is no solution of the equation within the range given to **fsolve**. In the first example, **fsolve** does not return any value.

```
> fsolve( 1-x-2*x^3-x^4=0, x, -2..0 );
```

But notice what **fsolve** does in the next example.

```
> fsolve( x^3-cos(3*x^2)=0, x, -2..-1 );
```

In this example, **fsolve** returned unevaluated. I do not know why, when there is no solution within the given range, **fsolve** sometimes does not return a value and sometimes returns unevaluated.

```
>
```

```
> 
```

3.7. Using `fsolve` with a system of equations

We can use **fsolve** to find numerical solutions for systems of equations. For example. the system of equations $x^2 + y^2 = 9$ and $x^y = 2$ has three real solutions, as the following graph shows.

```
> plots[implicitplot]( {x^2+y^2=9, x^y=2}, x=-4..4, y=-4..4,
>                      scaling=constrained );
```

But the **solve** command can only find one of the solutions.

```
> solve( {x^2+y^2=9, x^y=2}, {x,y} );
> allvalues( % );
> evalf( % );
```

To find approximations for the other two solutions, we need to use the **fsolve** command.

```
> fsolve( {x^2+y^2=9, x^y=2}, {x,y} );
```

Without any hints, the **fsolve** command found the same solution that **solve** and **allvalues** found. When solving a system of equations, we can give **fsolve** hints about either or both of the unknowns that it is solving for. The next command gives **fsolve** a hint about the **x** value of the solution that we wish to find.

```
> fsolve( {x^2+y^2=9, x^y=2}, {x,y}, x=0..2 );
```

Within that range for **x** there are two solutions of the system and **fsolve** found one of them. If we want to find the other, we can also give **fsolve** a range for **y**.

```
> fsolve( {x^2+y^2=9, x^y=2}, {x,y}, x=0..2, y=2..4 );
```

Or, we could just give **fsolve** a range for **y**.

```
> fsolve( {x^2+y^2=9, x^y=2}, {x,y}, y=2..4 );
> fsolve( {x^2+y^2=9, x^y=2}, {x,y}, y=-4..-2 );
```

As with single equations, we can give **fsolve** hints by using starting values instead of ranges.

```
> fsolve( {x^2+y^2=9, x^y=2}, {x=1,y=-3} );
```

We can try to find a complex solution to the system by using the keyword **complex** and giving **fsolve** complex starting values.

```
> fsolve( {x^2+y^2=9, x^y=2}, {x=I,y=I}, complex );
```

Let us check this last solution.

```
> subs( %, {x^2+y^2=9, x^y=2} );
```

Does that seem reasonable?

Exercise: Find another complex solution for the system of equations $x^2 + y^2 = 9$ and $x^y = 2$. Be sure to check any solution that you find.

```
> 
```

Exercise: Use **fsolve** to solve the equation $x^2 = 7\cos(x^2)$. Convert that equation to the equivalent system of equations $y = x^2$ and $y = 7\cos(x^2)$ and use **fsolve** to solve this system. Then convert the equation to the equivalent system $y = x^2$ and $y = 7\cos(y)$ and use **fsolve** to solve this system.

```
> 
```

[>

3.8. Online help for solving equations

The following command calls up the help page for **solve**.

[> **?solve**

The following help pages provide more details about how the **solve** command works. Each page discusses a special case of what the **solve** command can do.

[> **?solve,scalar**
[> **?solve,system**
[> **?solve,linear**
[> **?solve,radical**
[> **?solve,float**
[> **?solve,ineq**

The following command brings up a worksheet, from the New User's Tour, called "Algebraic Computations". Notice that this worksheet has a section called "Solving Equations and Systems of Equations".

[> **?newuser,topic04**

The next command brings up one of the "Examples Worksheets" called "Solving Equations".

[> **?examples,solve**

It is common for **solve** to return a **RootOf** expression. Here is a help page that provides some information about these expressions.

[> **?RootOf**

The **allvalues** command is used to find out more information about **RootOf** expressions.

[> **?allvalues**

When **solve** cannot find a symbolic solution for an equation, we need to use the **fsolve** command.

[> **?fsolve**

The **solvefor** command is closely related to **solve**.

[> **?solvefor**

The **eliminate** command also, in a sense, allows one to "solve" a set of equations.

[> **?eliminate**

The following two commands are special kinds of **solve** commands. The **isolve** commands solves equations for integer solutions. The **msolve** command solves equations for integers solutions mod m.

[> **?isolve**

```
> ?msolve

>
```