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

Two of Maple's most useful commands are **solve**, which solves equations symbolically, and **fsolve**, which solves equations numerically. The first section of this worksheet gives an overview of working with these two commands. The rest of this worksheet goes into the details of using the **solve** command to solve single equations and systems of equations. We introduce Maple's RootOf expressions, which are used throughout Maple and are often used in the results returned by **solve**, and we consider the **allvalues** command for interpreting RootOf expressions. We also consider what kinds of equations **solve** can and cannot solve. Finally, we show how the numerical solving command **fsolve** can be used when **solve** does not return a result.

```
>
```

3.2. The basics of using solve and fsolve

Recall that an equation is a made up of an equal sign with an expression on either side of it. An equation can either be true, like $2 + 2 = 4$, or false, like $2 + 2 = 5$. Most of the time, equations contain one or more unknowns in them, as in $x^2 + 2 = 4$. When an equation contains an unknown, then we can ask for which values of the unknown is the equation true. For example, the equation $x^2 + 2 = 4$ is true when the unknown x is given either the value $\sqrt{2}$ or $-\sqrt{2}$. The values that make an equation true are called the **solutions** of the equation. Maple has two commands that can be used to find the solutions to an equation, **solve** and **fsolve**. The **solve** command attempts to solve an equation symbolically.

```
> solve( x^2+2=4, x);
```

The **fsolve** command attempts to solve an equation numerically.

```
> fsolve( x^2+2=4, x);
```

The rest of this section is an overview of using **solve** and **fsolve**. The rest of the sections in this worksheet go into the details of using these two commands.

We use the **solve** command by giving it 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 (in terms of the other 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 each solution from the last result really does solve the equation for **x** (notice the use of indexed names).
```
> subs( x=%[1], 1=a*x^2-b );
> subs( x=%%[2], 1=a*x^2-b );
>
```

Exercise: For each of the following two **solve** commands, use the **subs** command to verify that the solution really does solve the equation.
```
> solve( 1=a*x^2-b, a );
> solve( 1=a*x^2-b, b );
>
```

Exercise: Let us give the equation $1 = a\,x^2 - b$ a name.
```
> eqn := 1=a*x^2-b;
```
Now reuse the **solve** command to solve this equation for each unknown, as was done above, but always refer to the equation by its name. Also, use the **subs** command to verify each solution, but once again, always refer to the equation by its name.
```
>
```

If we put a pair of braces around the unknown we wish 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 each solution from the last result really does solve the equation for **x**. Notice how the form of the **subs** command changes slightly because of the braces around **x** in the **solve** command (since the result from **solve** is already in the form of an equation, the **subs** command does not need its first entry to be an equation).
```
> subs( %[1], 1=a*x^2-b );
> subs( %%[2], 1=a*x^2-b );
>
```

Exercise: For the following **solve** command, there is one equal sign in the input to the command and another equal sign in the result.
```
> b=a*x^2-1;
> solve( %, {a} );
```
What can you say about each of these two uses of an equal sign. Do both equal signs have the same meaning?
```
>
```

Exercise: After we solve an equation like the following one

```
> solve( a=(1+b)/x^2, b );
```

we can use the **subs** command to check Maple's solution.

```
> subs( b=-1+a*x^2, a=(1+b)/x^2 );
```

What can you say about each of the two uses of an equal sign in the last command and the third use of an equal sign in the command's result. Do each of these equal signs have the same meaning?

```
>
```

One reason for putting braces around the unknown being solved for by **solve** is so that we can use the **assign** command (covered in the last worksheet) to assign a solution given by **solve** to the unknown variable. Here is an example.

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

The next command assigns the first of the two solutions to **x**.

```
> assign( %[1] );
> x;
```

The next command tries to assign the second of the two solutions to **x**.

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

That did not work because **x** now has a value, so it is no longer an "unknown", and so the **assign** command was trying to execute the assignment **2:=-1**. We need to unassign **x** before we can use **assign** to give **x** the second solution from the **solve** command.

```
> x := 'x';
> assign( %%%[2] );
> x;
```

To avoid problems later on, let us unassign **x** again.

```
> x := 'x';
>
```

It is not always the case that **solve** can find symbolic solutions to an equation. The equation $\frac{x^5}{2} = \cos(x^9)$ has three (real) roots, but **solve** cannot find symbolic expressions for any of them and so it does not return a result.

```
> solve( (x^5)/2-cos(x^9)=0, x );
```

We can show that the equation has three roots by using a graph.

```
> plot( [(x^5)/2, cos(x^9)], x=-1.2..1.1, -1.1..1.1 );
>
```

Exercise: In the last **plot** command, change the comma just after the **2** to a minus sign. What does this show?

```
>
```

The last two graphs show one way that we can discover solutions to an equation when **solve**

cannot do it for us. But solving equations graphically has the disadvantage of not being very accurate, plus it is very labor intensive and cannot be done automatically.

Exercise: The equation $\frac{x^5}{2} = \cos(x^9)$ has a solution near $x = 1$. Use the **plot** command to zoom in on this solution as much as you can. Using your zoomed in graphs, what is the limit to the number of decimal places that you can determine for this solution.

```
[ >
```

If **solve** cannot find a symbolic solution to an equation, and if finding a solution graphically is too time consuming and inaccurate, then what should we do? Maple has another kind of solve command called **fsolve** that finds decimal, rather than symbolic, solutions to equations. The advantage of **fsolve** is that it can usually find a decimal approximation to a solution even when **solve** cannot find a symbolic solution. The disadvantage of **fsolve** is that decimal solutions are sometimes less useful, less informative, than symbolic solutions. But in situations where **solve** cannot find any symbolic solutions, **fsolve**'s decimal solutions are better than nothing.

The **fsolve** command will try to return a decimal approximation for *one* root of the equation being solved. Even if the equation has several roots, **fsolve** will only approximate one root at a time. In order to find all the solutions of an equation, it is usually necessary to use **fsolve** several times, once for each solution. In the following example, we see how to use **fsolve** to find several solutions to an equation.

We know that the equation $\frac{x^5}{2} = \cos(x^9)$ has three real roots and **solve** cannot find any of them. The **fsolve** command will readily return one of these roots.

```
[ > fsolve( x^5/2-cos(x^9)=0, x );
```

Let us check this result.

```
[ > subs( x=%, x^5/2-cos(x^9)=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 two 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 has three solutions).

```
[ > plot( x^5/2-cos(x^9), x=-2..2, -2..2 );
```

From the graph we see that the two other two solutions are 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^5/2-cos(x^9)=0, x, -2..-1 );
```

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. To find a more specific range, we need to zoom in on our graph near the two solutions.

```
> plot( x^5/2-cos(x^9), x=-1.4..-1, -1..1 );
```

We need to direct **fsolve** towards the right most of the two solutions (it has already found the left most one).

```
> fsolve( x^5/2-cos(x^9)=0, x, -1.12..-1.1 );
```

So now we have approximate values for all three (real) solutions of the equation. This example is typical of how we use **fsolve** with an equation. We need an appropriate graph of the equation to give us an idea of how many 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.

```
>
```

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

```
>
```

Part (b): How many solutions does the equation $z = 7\cos(z)$ have? Use **fsolve** to find approximations to all of the solutions.

```
>
```

The **solve** and **fsolve** commands can also be used to solve systems of equations. A **system of equations** is made up of two or more equations in one or more unknowns. A solution to a system of equations finds values for all of the unknowns that simultaneously solve each of the equations.

Here is an example of using **solve** to solve a system of two (linear) equations in two unknowns.

```
> solve( { 3*u+5*v=2, 4*u-v=1 }, {u,v} );
```

Notice how the equations are put inside of a pair of braces and the variables to solve for are placed inside of another pair of braces.

Now let us look at a system of two nonlinear equations in two unknowns. Nonlinear systems are much harder to solve than linear systems and **solve** often cannot solve them. In that case we need to use **fsolve**. Consider the two equations

$$y = 2\cos(5x) \quad \text{and} \quad x^2 + y^2 = 1$$

The solution of this system is the intersection of a circle with a graph of a cosine function. Maple can easily draw a graph of these equations and give us a rough idea of the solutions.

```
> plots[implicitplot]( {y=2*cos(5*x), x^2+y^2=1}, x=-1..1,
  y=-2..2);
```

The **solve** command cannot solve this system. But **fsolve** returns one decimal solution.

```
> fsolve( {y=2*cos(5*x), x^2+y^2=1}, {x,y});
```

From the graph we know that there are 8 solutions. To find the other solutions, we have to give **fsolve** hints about where to look. Since there are two variables to solve for, we can give give **fsolve** a hint about the value of either or both of the unknowns. Using the above graph of the solutions to get the hint, here is how we have **fsolve** find one of the two solutions with x between

0 and 0.5.

```
> fsolve( {y=2*cos(5*x), x^2+y^2=1}, {x,y}, x=0..0.5 );
```

fsolve found the right most of these two solutions. Here is how we can add a hint about *y* so that **fsolve** finds the left most solution with *x* between 0 and 0.5.

```
> fsolve( {y=2*cos(5*x), x^2+y^2=1}, {x,y}, x=0..0.5, y=0..1 );
```

Notice that if we just give **fsolve** the *y* hint, it finds a solution that has a negative *x* value.

```
> fsolve( {y=2*cos(5*x), x^2+y^2=1}, {x,y}, y=0..1 );
```

With appropriate hints, all of the other solutions can be found.

Exercise: Find decimal approximations to all of the solutions of the above system of equations.

```
>
```

Exercise: Given a number y_0 between 0 and 1, draw a graph of $\sin(x)$ and a parabola $p(x)$ with x between 0 and π so that $p(x_0) = \sin(x_0)$ and $p\left(\frac{\pi}{2}\right) = 1$ where x_0 solves $\sin(x) = y_0$.

```
>
```

Exercise: Given a number y_0 between 0 and 1, draw a graph of $\sin(x)$ and a parabola $p(x)$ with x between 0 and π so that $p(x_0) = \sin(x_0)$ and $\left(\frac{d}{dx} p(x)\right)\Big|_{x=x_0} = \left(\frac{d}{dx} \sin(x)\right)\Big|_{x=x_0}$ where x_0 solves $\sin(x) = y_0$.

```
>
```

```
>
```

3.3. Solving a single polynomial equation

If we ask **solve** to solve a polynomial equation (in one variable) of degree *n* with *n* equal to 1, 2, or 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 given explicitly. Instead, some of the solutions may be represented by a **RootOf** expression. Here is a degree five example with five "solutions".

```
> 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** expressions in the result. A **RootOf** expression usually (but not always)

means that Maple does not know how to find an explicit symbolic solution for the root (or roots) of the equation. But a **RootOf** expression can also be used by Maple as an abbreviation for a very large symbolic solution, as, for example, in the case of 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 solves 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 large and complicated that it is of limited usefulness (and that is why it was represented by a **RootOf** expression). Let us apply **allvalues** to one of the above **RootOf** expressions.

```
> allvalues( %[1] );
```

As we said, this symbolic result is so complicated that it is almost useless. We use the **evalf** command to get a decimal approximation of this solution.

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

We can also use **evalf** to get decimal approximations of all four of the results returned by the last **solve** command.

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

When a **RootOf** expression solves a polynomial of degree 5 or more, then the **allvalues** command applied to the **RootOf** expression may return unevaluated. In this case, Maple cannot actually 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 );
```

Let us apply **allvalues** to one of the **RootOf** expressions.

```
> allvalues( %[1] );
```

We can apply **evalf** to all five of the **RootOf** expression in just one command.

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

Notice that we have four complex numbers and one real number. What if we would like to convince ourselves that these really are correct solutions? The next command substitutes the five decimal 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 $0 = 0$ but that is not what we got for any of our five decimal numbers. For the single solution that is a real number, we get the equation $.6\ 10^{(-8)} = 0$ when that 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. The equation $.6\ 10^{(-8)} = 0$ approximates the equation $0 = 0$, so this tells us that 1.965948237 is an approximate solution of the original equation, not an exact solution. Similarly, the above substitutions tell us that the four

complex solutions are also correct approximations of exact solutions (why?).

```
> 
```

With a polynomial equation, **solve** (along with **allvalues** and **evalf**) 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 a non polynomial equation, there is no guarantee that **solve** will return all solutions of the equation (for example, there could be an infinite number of them), 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.

```
> 
```

Exercise: In the previous examples we saw that Maple can represent a number with a RootOf expression. Maple also allows us to convert any number into a RootOf expression. Here is a simple example.

```
> convert( sqrt(2), RootOf );
```

Part (a): Let us do a slightly more comlicated example. Let us take one of the roots of $x^3 - 1 = 0$ and convert it from a number into a RootOf expression.

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

Let us use the root with a positive imaginary part.

```
> -1/2+I*sqrt(3)/2;
> convert( %, RootOf );
```

Notice that, even though this number is a root of $x^3 - 1 = 0$, that is not how Maple converted it into a RootOf expression. What did Maple do to get the last result?

Part (b): The **allvalues** command converts RootOf expressions back into numbers.

```
> convert( sqrt(2), RootOf );
> allvalues( % );
```

Let us check the result of **allvalues** with the example from Part (a).

```
> -1/2+I*sqrt(3)/2:
> convert( %, RootOf );
> allvalues( % );
```

Notice that there is an index option inside of the RootOf expressions. Here is the last RootOf expression with the index options removed.

```
> -1/2+1/2*RootOf(_Z^2+1)*RootOf(_Z^2-3);
```

Explain the result from the following **allvalues** command. How many distinct numbers are in the result?

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

The following RootOf expression has one of the index options put back into it. Explain the result of the following **allvalues** command.

```
> -1/2+1/2*RootOf(_Z^2+1, index=1)*RootOf(_Z^2-3);
> allvalues(%);
```

You may want to look in the RootOf help page for the paragraph that describes "root selectors".

```
```

```
> ?RootOf
>

>
```

3.4. Solving a single nonlinear equation

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

We know from Section 3.2 of this worksheet that the equation $\frac{x^5}{2} = \cos(x^9)$ has three real roots, but **solve** cannot find symbolic expressions for any of them and **solve** does not return a result.

```
> solve( (x^5)/2-cos(x^9)=0, x );
>
```

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 of the way that **solve** approaches this equation. The equation **sin(x^2-1)=0** is true if **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

$$\sin(x^2+1+3.1415)=-1$$

are almost 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 also get **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 (and, strangely enough, that decimal solution is given inside the **RootOf** expression). We can get that same approximate solution by applying **evalf**, instead of **allvalues**, to the **RootOf** expression returned by **solve** (or we could apply **evalf** to the result from **allvalues**).

```
> evalf( %% );
> evalf( %% ); # which result does this %% refer to?
```

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 more about the values that are represented by

the **RootOf** expression.

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

Notice that **allvalues** returned two results. Notice also that each result is partly symbolic (the square root symbol), partly numeric, and still uses a **RootOf** expression. These results from **allvalues** are not so easy to use. What if we just apply **evalf** to the original **RootOf** expression returned by **solve**?

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

We only get one numeric result. If we instead apply **evalf** to the result from **allvalues**, then we get two numeric results.

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

The following graph shows us that these are the only two real solutions to the equation, so **solve** (with the help of **allvalues** and **evalf**) found all of the real solutions.

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

Let us go back to the nested **RootOf** expression returned by the **solve** command and try to see exactly how the **RootOf** expression 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 );
> evalf( %% );
> evalf( %% );
```

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: Each of the following four RootOf expressions represents solutions to the equation $x^2 - \sin(x^4) = 1$.

```
> RootOf(RootOf(u-sin(u)^2-2*sin(u)-1)-u^4);
> RootOf(sin(RootOf(_Z-sin(_Z)^2-2*sin(_Z)-1))+1-u^2);
> RootOf(RootOf(sqrt(u)-sin(u)-1)-u^4);
> RootOf(RootOf(u-sin(u^2)-1)-u^2);
```

Part (a): Two of the above RootOf expressions were derived by making the substitution $u = x^2$ in the original equation. The other two RootOf expressions were derived by making the substitution $u = x^4$. Show how these substitutions can be used to derive these RootOf expressions.

Part (b): Two of the above RootOf expressions have a small advantage over the other two. What is the advantage?

```
> 
```

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** and **evalf**, give us a list of decimal approximations of solutions (which may or may not be a complete list of all the 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 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 (look carefully at the result from **allvalues** and verify this). The following two 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 numeric values with symbolic results.

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

Let us check that this is a valid solution.

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

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 another example. 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 (draw a graph), and they are not very hard to find using paper and pencil. Are there any other solutions? Here is a way to find out using Maple.

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

We have four solutions, two of them real and two complex. Here is an easy way to see this.

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

The next two commands checks all four of the solutions and show that they are correct.

```
> seq( subs( solutions[i], equations ), i=1..4 );
> simplify( [%] );
```

It is interesting to note that earlier versions of Maple returned incorrect answers for this simple problem. If you have access to a copy of Maple V Release 5, try this problem with it and see what you get.

```
> 
```

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): Show how the nested RootOf expressions returned by **solve** can be derived from the system of equations. Which of the two approaches mentioned in Part (a) did the **solve** command use?

```
> 
```

Part (c): 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 that your RootOf expression represents the same four solutions found above.

```
[ >
```

Part (d): Use the next command to read about the keywords **dependent** and **independent** in the online documentation for **allvalues**.

```
[ > ?allvalues
```

Try using the **independent** keyword when applying **allvalues** to your RootOf expressions from Part (c). Are the "solutions" returned by **allvalues** in this case really solutions? Exactly how did **allvalues** arrive at them?

```
[ >
```

Exercise: Use **solve** to try and solve the equation $x^2 = \cos(x^9)$. Convert that equation to the equivalent system of equations $y = x^2$ and $y = \cos(x^9)$ and use **solve** to try solving this system.

Then convert the equation to the system $y = x^2$ and $y = \cos(\sqrt[9]{y})$ and try to use **solve** to solve this system. If **solve** returns a solution for any one of the systems, test the solution in each of the other systems.

```
[ >
```

```
[ >
```

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 of solutions to 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 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. We showed earlier that the equation $\frac{x^5}{2} = \cos(x^9)$ has three real roots and **solve** can not find any of them. The **fsolve** command will readily return one of the real roots.

```
> fsolve( x^5/2-cos(x^9)=0, x );
```

To find the other two real roots, we use the option for **fsolve** that allows us to give a hint about where it should look for a solution. We give **fsolve** a hint by giving **fsolve** a range that we know contains a solution.

```
> fsolve( x^5/2-cos(x^9)=0, x, -2..-1 );
```

The range that we just gave **fsolve** actually included two solutions. To find the third solution, we need to give **fsolve** a more specific range to look in.

```
> fsolve( x^5/2-cos(x^9)=0, x, -1.12..-1.1 );
```

Recall that we can get the ranges that we give to **fsolve** from appropriate graphs of the equation we are trying to solve.

```
>
```

There is another way to give **fsolve** a hint about where to find a solution. Instead of giving **fsolve** a range within which to search for the unknown, we can give **fsolve** a "starting value" for the unknown in the equation.

```
> fsolve( x^5/2-cos(x^9)=0, x=-1 );
> fsolve( x^5/2-cos(x^9)=0, x=-1.1 );
> fsolve( x^5/2-cos(x^9)=0, x=-1.2 );
```

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 of the equation's solutions **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. For an example of the unpredictability of **fsolve** when using a starting value, look at the first of the last three commands. Out of the three real roots of the equation, **fsolve** found the one that is furthest from the starting value.

```
>
```

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^5/2-cos(x^9)=0, x, complex );
```

Notice that the last result is one of the real solutions that we have already found. 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^5/2-cos(x^9)=0, x=1+I, complex );
```

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

```
> fsolve( x^5/2-cos(x^9)=0, x=2+I, complex );
```

We showed above that **fsolve** can be a bit unpredictable when we use starting values. To get more control over **fsolve** we can use a range with the **complex** option. Two complex numbers in a range are interpreted as the lower left hand corner and the upper right hand corner of a rectangle in the complex plane. For example, the range **1+I..3+2*I** describes a rectangle in the complex plane that is two units long horizontally, one unit high vertically, and has its lower left hand corner at the point **1+I**.

```
> fsolve( x^5/2-cos(x^9)=0, x=1+I..3+2*I, complex );
```

Be sure to convince yourself that this solution lies in the rectangle we just described.

```
>
```

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 these solutions. How many solutions does **solve**, along with **allvalues** and **evalf**, 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 (f): 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 am not sure why, when there is no solution within the given range, **fsolve** sometimes does not return a value and sometimes returns unevaluated. (The difference might be that when **fsolve** does not return a result, then Maple knows that there is

no solution in the given range, but when **fsolve** returns unevaluated, it means that Maple could not find a solution but it does not know how to prove that no solution exits.)

```
> 
```

```
> 
```

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 we are 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 only 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. Solving inequalities

Solving inequalities is not as common with Maple as solving equations, but it is interesting to see that the **solve** command can be used to solve many inequalities and it is also interesting to see in what form Maple returns the results of these kinds of problems.

The solution of the following inequality is an open interval. Notice how Maple denotes an open interval in the real line.

```
> solve( x^2-2*x-15<0, x );
```

The solution of the following inequality is a closed interval.

```
> solve( x^2-2*x-15<=0, x );
```

The next solution is the complement of the previous one. Notice that this solution is made up of two infinite, open intervals.

```
> solve( x^2-2*x-15>0, x );
```

Here are the last three inequalities again, but using a different notation for the results (because of the braces around **x**).

```
> solve( x^2-2*x-15<0, {x} );
> solve( x^2-2*x-15<=0, {x} );
> solve( x^2-2*x-15>0, {x} );
```

Here is a slightly more complicated inequality.

```
> exp(-x^2)>1/sqrt(2);
> solve( %, x );
```

Here is the same result in the other notation.

```
> solve( %%, {x} );
```

Here is a simple inequality. Notice that the form of the result is not really consistent with the form of the previous results (why not?).

```
> solve( exp(x)>0, x );
> solve( exp(x)>0, {x} );
```

Even some very simple inequalities cannot be solved by **solve**.

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

```
>
```

3.9. 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" and another section called "Solving Inequalities".

```
> ?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 are two help pages that provide some information about these expressions.

```
> ?RootOf
> ?RootOf,indexed
```

The **allvalues** command is used to find out more information about the values represented by a **RootOf** expression.

```
> ?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
```