Part II. Calculations and Calculus with *Mathematica*

Section 2. Symbolics: Equations and Assignments

Mathematica is an excellent calculator. However, calculus and differential equations applications require symbolic capabilities. Although *Mathematica* makes symbolic computations in a natural way (see the examples in Section 1), it is essential that users have a clear understanding of how it handles variables, equations, and functions.

■ Variables, equations and assignments

The equation x = 2, the expression y + x, and the expression y/x all involve the variables x and y. They are entered separately below.

```
x = 3
y + x
y/x
```

```
3
3 + y
y
─
3
```

Key observation: The equation x = 3 has assigned the value 3 to the x variable. This can be seen from the output for the second and third entries.

Once 3 has been assigned to the x variable, it stays that way. Examine carefully the three terms in the output for the following list..

```
{y + x, x^3 - x*z, x^3}
```

```
{3 + y, 27 - 3 z, 27}
```

There are two ways to undo the assignment of 3 to the x variable. One way is to enter

Clear[x]

And the other is to enter

x = .

as shown below.

```
x = .
y + x
```

```
x + y
```

Note that the entry **x = .** has no output. Neither does **Clear[x]**.

Multiple assignments: Full evaluation

Mathematica does not mind if several assignments are made to the same variable. It only remembers the last one. Consider the following example.

```
x=1; x=2; x=3
x
 3
 3
```

Now clear x and check that it is free.

```
Clear[x]; x
 x
```

A chain of functions

Suppose now that several variables are defined in a "chain" like this.

```
y=x^2; x=2z; z=3Log[w]; w=Exp[1]
 ⅇ
```

Then w is assigned the number *e*

```
{w, N[w]}
{ⅇ, 2.71828}
```

and *Mathematica* will dig down all the way to *e* when asked to evaluate y, x, z and w. This can be referred to as "full evaluation".

```
{y, x, z, w}
{36, 6, 3, ⅇ}
```

Make sure that you understand the output. What is Log(w)? Why did x evaluate to 36?

If the assignment to w is subsequently changed:

```
w=Sin[3]
 Sin[3]
```

This change will be reflected in the evaluation of y, x, and z as well.

```
{y, x, z, w}
```

$\{36\,\mathrm{Log}[\mathrm{Sin}[3]]^2,\ 6\,\mathrm{Log}[\mathrm{Sin}[3]],\ 3\,\mathrm{Log}[\mathrm{Sin}[3]],\ \mathrm{Sin}[3]\}$

Before proceeding we clear x, y, z, and w.

```
Clear[x,y,z,w]
```

Some remarks on input

Before proceeding it would be appropriate to explain some things about how *Mathematica* handles input.

1. Regarding multiplication. Although multiplication can always to denoted with an asterisk, it is sometimes not required. A space can be used instead and not even a space is required for a product like 4x. For example, the entry

x^2 + 3x - x y

with a space between the last x and y and no space between 3 and x will be interpreted by *Mathematica* as the same as

x^2 + 3*x - x*y.

See below.

```
Poly = x^2 + 3x - x y
Factor[Poly]
```

$3\,x + x^2 - x\,y$

$x\,(3 + x - y)$

2. If several computations (in sequence) are needed to obtain some particular output consider using semicolons to suppress (and compress) the preliminary inputs, and finish with no semicolon to show the last output.

Solving equations exactly: The Solve function

Mathematica's "Solve" procedure is very powerful. Ask for exact solutions for x in "equation" by entering the following:

Solve[lhs = = rhs, x]

Note that the equation to be solved must be in the form rhs == lhs. Having named the solution "solns" the entry solns[[k]] can be used to obtain the kth solution in the sequence. If exact solutions are found, *Mathematica* returns them in a list.

```
Solve[ x^3 - x == x^2, x]
```

$\{\{x \to 0\}, \{x \to \frac{1}{2}\,(1 - \sqrt{5})\}, \{x \to \frac{1}{2}\,(1 + \sqrt{5})\}\}$

If the solution values are to be used in a subsequent input, then assign a name to the Solve entry as shown in the next example. The same name will be assigned to the output.

In the next example a name has also been assigned to the equation. This is recommended or two reasons:

1. The equation can be checked for correctness by examining the output.

2. If the equation has to be used again, simply type its name.

```
eqn = x^3 - 5x^2 == 2 - 6x
solns = Solve[ eqn, x]
```

$-5\,x^2 + x^3 == 2 - 6\,x$

$\{\{x \to 1\}, \{x \to 2 - \sqrt{2}\}, \{x \to 2 + \sqrt{2}\}\}$

Having named the solution "solns" the entry solns[[k]] can be used to obtain the kth solution in the sequence.

```
solns[[1]]
```

$\{x \to 1\}$

```
solns[[2]]
```

$\{x \to 2 - \sqrt{2}\}$

```
solns[[3]]
```

$\{x \to 2 + \sqrt{2}\}$

Check by substitution (the arrow makes it easy)

An expression like "eqn" can be evaluated for various x values using an input of the form

eqn/.x->value

The **/.** notation is described by *Mathematica* as "Replace all". It can also be interpreted as "substitute for the x value indicated". Here is a simple example. Note that no simplification is applied.

```
Sin[3(x+y)]/.x->Pi
```

$\text{Sin}[3\ (\pi + y)]$

And here is an example showing how to check the second solution above.

```
eqn/.solns[[2]]
```

N::meprec : Internal precision limit \$MaxExtraPrecision = 50.\` reached while evaluating $-2 + 6\left(2 - \sqrt{2}\right) - 5\left(2 - \sqrt{2}\right)^2 + \left(2 - \sqrt{2}\right)^3$. More…

$-5\left(2 - \sqrt{2}\right)^2 + \left(2 - \sqrt{2}\right)^3 == 2 - 6\left(2 - \sqrt{2}\right)$

Mathematica did not simplify the equation and cannot, using numerical evaluation, verify if the left side is identical to the right side. Applying Simplify will clarify the situation.

```
Simplify[%]
```

```
True
```

Note that the output is the word "True" indicating that the output is an identity.

If numerical solutions are needed apply the N function to solns.

```
N[solns]
```

$\{\{x \to 1.\}, \{x \to 0.585786\}, \{x \to 3.41421\}\}$

Approximate solutions using NSolve and FindRoot

The Solve function often fails to return satisfactory solutions because *Mathematica* cannot find an exact solution formula. When this happens, floating point solutions to polynomial equations can be obtained by using the NSsolve procedure. The syntax for NSolve is

NSolve[lhs = = rhs, x]

If a particular solution is required use FindRoot using the syntax

FindRoot[lhs = = rhs, {x, x0}]

where x0 is the approximate value of the desired solution.

The next input requests a solution to "eqn" (defined above) near to x =0.5. Compare the output to the middle term in the last output list.

```
FindRoot[ eqn, {x,0.5} ]
```

```
{x → 0.585786}
```

In the next input we have

1. Entered and named the polynomial equation x^5 - x^4 + x^3 + x - 1 = = 0.

2. Used NSolve to generate a sequence of floating point solutions named solns.

```
eqn = x^5 - x^4 + x^3 + x - 1 == 0
solns = NSolve[ eqn, x ]
```

$-1 + x + x^3 - x^4 + x^5 == 0$

```
{{x → -0.579723 - 0.73932 ⅈ}, {x → -0.579723 + 0.73932 ⅈ},
 {x → 0.712453}, {x → 0.723496 - 1.03282 ⅈ}, {x → 0.723496 + 1.03282 ⅈ}}
```

Observe that the third solution in the solution sequence is purely real.

```
solns[[3]]
```

```
{x → 0.712453}
```

The other four solutions are complex, appearing in conjugate pairs (the symbol ⅈ is used by *Mathematica* to denote $i = \sqrt{-1}$).

```
{solns[[1]],solns[[2]]}
```

```
{{x → -0.579723 - 0.73932 ⅈ}, {x → -0.579723 + 0.73932 ⅈ}}
```

The next entry checks the third solution. Because a float has been substituted, the output is in floating point form and automatically simplified. The output is "True" indicating that, to 50 digits, the equation is true.

```
eqn/.solns[[3]]
```

```
True
```

More about Solve and FindRoot

When the Solve function cannot find exact solutions to an equation, it returns an explanation with some indication of why no solution is forthcoming.

```
eqn = Tan[x] == x
solns = Solve[ eqn, x]
```

```
Tan[x] == x
```

```
Solve::tdep : The equations appear to involve the variables
   to be solved for in an essentially non-algebraic way. More…
```

```
Solve[Tan[x] == x, x]
```

We will take this is a signal to use FindRoot as shown below.

```
FindRoot[ eqn, {x,4} ]
FindRoot[ eqn, {x,7} ]
```

```
{x → 4.49341}
```

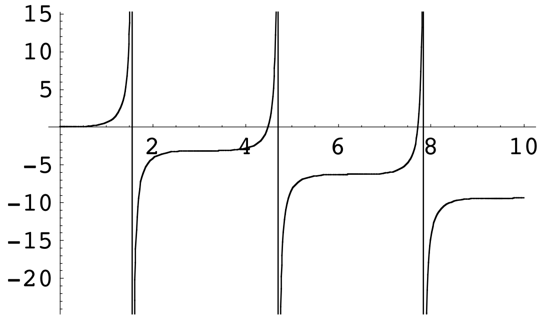
```
{x → 7.72525}
```

In order to search successfully it would be helpful to have the graph of the function tan(x) - x. This is easy to obtain using *Mathematica*'s Plot function. It has the following basic form.

Plot(f[x], {x,a,b})

The next entry asks for the graph of tan(x) - x from x = 0 to x = 10.

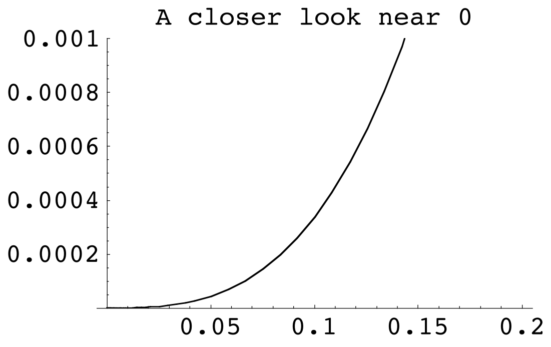
```
Plot[ Tan[x] - x, {x,0,10} ]
```



This is a pretty good picture. Observe that the four branches of the graph are connected with vertical lines that appear to be the asymptotes. This is because *Mathematica* made the graph by plotting lots of points and connecting the dots. This is the default plot behavior.

It appears that the only solutions to tan(x) = x in the interval from 0 to 10, other than x = 0, are the two that were just found using FindRoot. However, it may be worth looking more closely at the graph near x = 0. The next input shows how to control the vertical plot range and add a title to the plot.

```
Plot[ Tan[x] - x, {x,0,0.2}, PlotRange->{0,0.001},
                              PlotLabel->"A closer look near 0" ]
```



Using variables as functions

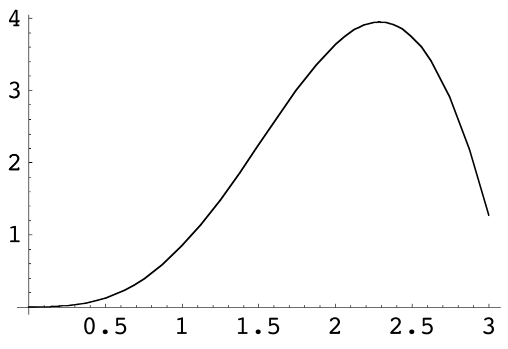
A typical calculus book is full of problems like the following.

Graph the function $y = x^2 \sin(x)$ *over the interval from* $x = 0$ *to* $x = 3$. *Find its maximum and minimum values on the interval.*

To handle this problem in *Mathematica* we can define the variable *y* as a function of *x* using the equals operator. This is done in the first entry. The second entry uses the Plot function to display the graph of the function over the given interval.

```
y = x^2 Sin[x]
Plot[ y, {x,0,3} ]
```

x^2 Sin[x]



Clearly the minimum *y* value on this interval is $y = 0$, attained at the left endpoint where $x = 0$. To find the maximum *y* value we will compute the derivative function and then find its zero near $x = 2.5$. The D function can be used to obtain the derivative. The syntax for differentiating variable y with respect to variable x is simply

D[y,x]

Following the usual calculus convention we name the derivative function *y* ' .

```
y' = D[y,x]
```

x^2 Cos[x] + 2 x Sin[x]

Now use FindRoot to find the zero of *y*' near to 2.5. Solve will not work here; try it. We name the zero of y' xmax and the corresponding y value ymax. Note the use of /., twice.

```
FindRoot[ y' == 0, {x,2.5} ]
```

```
{x → 2.28893}
```

```
xmax = x/.%
```

```
2.28893
```

```
ymax = y/.%%
```

```
General::spell1 : Possible spelling error: new symbol
   name "ymax" is similar to existing symbol "xmax". More…
```

```
3.9453
```

Now that the two coordinates of the high point have been found, it can be displayed on the graph by adding the point (xmax,ymax). *Mathematica* uses the notation {xmax,ymax} to denote a point and the ListPlot function to plot lists of points.

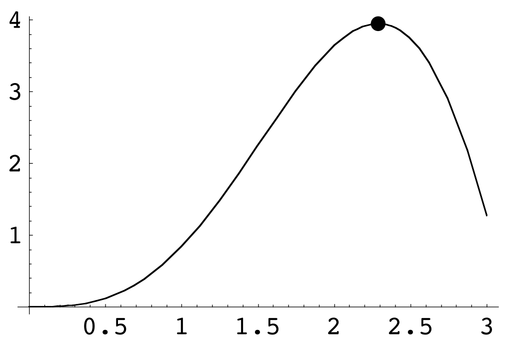
The next input uses the Show function to display the original plot and the ListPlot of the high point. The actual output consisted of three plots. The first two were deleted. Note how spaces and carriage returns were used to clarify the Show input which has the general syntax

Show[{ P1, P2, ... , Pn }]

where P1, P2, ... , Pn are Plot functions.

```
HighPoint = {xmax,ymax}
Show[ { Plot[ y, {x,0,3}],
        ListPlot[ {HighPoint},
                  PlotStyle->PointSize[0.03] ]
      }
    ]
```

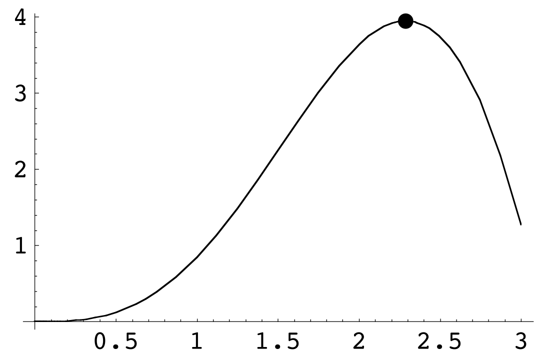
```
{2.28893, 3.9453}
```



Unfortunately, it appears that, in addition to showing the plots together, Show will also plot each of the component graphs. These were deleted, leaving only the composite plot in view above.

Note. It is possible to suppress the component plots by creating them separately and adding the option DisplayFunction -> Identity. One then adds the option DisplayFunction -> $DisplayFunction to the Show input. See below.

```
P1 = Plot[ y, {x,0,3}, DisplayFunction → Identity];
P2 = ListPlot[ {HighPoint},
     PlotStyle → PointSize[0.03],
     DisplayFunction → Identity];
Show[ {P1,P2}, DisplayFunction → $DisplayFunction ]
```



4
3
2
1
0.5
1
1.5
2
2.5
3