Now that you know the basics of Beamer, we encourage you to explore. For example, you can learn how to do each of the following:

- Enhance the look of your slides with "Beamer blocks."
- Jump forward and backward in your presentation, and link to Web documents.
- Generate a Table of Contents for your talk.

11.5 How to learn more

You can learn more about Beamer at:

`ctan.org/tex-archive/macros/latex/contrib/beamer/doc/beameruserguide.pdf`

There are many other good guides to Beamer on the World Wide Web. You can search for guides by entering "Beamer tutorial" in a Web browser. The LaTeX Beamer Class Homepage is located at:

`latex-beamer.sourceforge.net`

You may want to create a simple Beamer presentation, perhaps using the examples in this chapter, and then add to your presentation by following some of the more detailed advice that you can find on the Web.

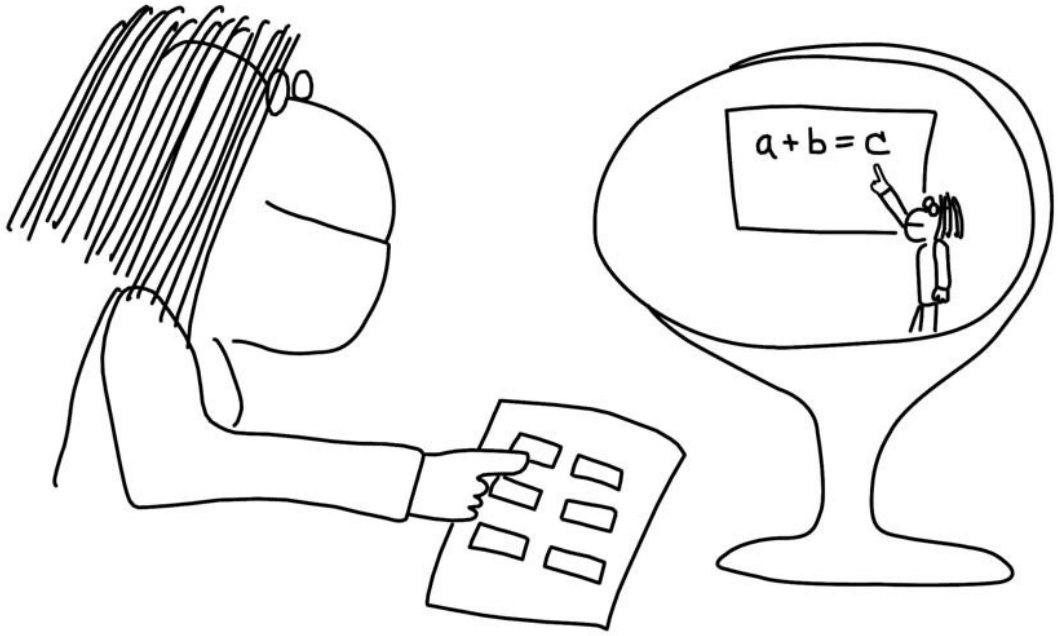
Exercises

Use Beamer to give presentations on the following topics.

1. Give a presentation on the Pythagorean Theorem. Be sure to include diagrams of right triangles.
2. Give a presentation on Pascal's triangle.
3. Give a presentation on the Fundamental Theorem of Calculus. Include diagrams.
4. Give a presentation on Linear Programming optimization methods. You may want to include a discussion of the Simplex Algorithm.
5. Give a presentation on Numerical Analysis used in solving equations.
6. Give a presentation on some theorems of Graph Theory. Include diagrams.
7. Give a presentation on fast Fourier transforms.
8. Give a presentation on a mathematician chosen from [5] or [6]. Give biographical background on the person and describe some of the mathematics that the person created.
9. Give a presentation on a topic in mathematics and art, chosen from [28], [13], or [41].

Chapter 12

Getting Started with Mathematica®, Maple®, and Maxima



Arguably the most important software tool for a mathematics student is the computer algebra system (CAS). No other software puts so much mathematical potential into a single tool.

12.1 What is a computer algebra system?

A computer algebra system is a program with which you can perform calculations, evaluate functions, create graphics, and develop your own programs. The key feature of computer algebra systems is the ability to manipulate expressions symbolically. Typical manipulations possible in a CAS include simplifying expressions, factoring, taking derivatives, computing integrals (symbolically and numerically), and solving systems of equations. This chapter explores the basics of three popular computer algebra systems, and it contains simple examples for you to try.

Mathematica, created by Stephen Wolfram, is probably the world's most recognized computer algebra system. It was originally released in 1988 and is still being developed and

improved. In addition to its raw power, one notable feature of Mathematica is its use of the "notebook." Mathematica notebooks allow a user to combine written text with calculations in one integrated document.

Maple, created by Waterloo Maple under the trade name Maplesoft, dates back to the early 1980s. It is one of the dominant commercial computer algebra systems and favored by many institutions. Like Mathematica, it can create integrated documents, called worksheets, that combine text, calculations, and hyperlinks.

Maxima is a free software computer algebra system. It is derived from an early computer program, Macsyma, which dates back to the 1960s and was made available under an open source license starting in 1998. Because it is free, it is especially attractive to students. Maxima can create documents that combine text with calculations through a graphical front end called wxMaxima.

12.2 How to use a CAS as a calculator

When using any computer algebra system as a calculator, it is important to understand that it is a bit difficult to translate mathematical writing directly to the computer. Humans instinctively adapt to ambiguity, but software is less flexible. For example, mathematicians use parentheses for grouping, as in $(x-1)(x+2)$. But they also use parentheses to indicate ordered pairs, like $(1,3)$, and to denote functions $f(x)$.

Another ambiguity that is perhaps a bit more subtle occurs with "equals." Humans have little trouble understanding that sometimes we intend equals to assign values, as with "let $x = 2$." At other times, we mean to assert equality; a circle is the set of points (x, y) such that $x^2 + y^2 = 1$.

Each computer algebra system addresses the job of translating mathematical syntax into unambiguous "computer syntax" in its own way. To a first-time software user this can feel somewhat unintuitive, even quirky, but mastering the language of your favorite CAS is an important part of using it effectively.

Mathematica as a calculator

Since Mathematica notebooks are used for text as well as calculations, you will almost immediately notice one idiosyncrasy when you try to use Mathematics as a calculator. The ENTER key does not run a calculation (it ends a paragraph or makes it possible to enter multi-line computations). To use Mathematica as a calculator, type the expression you wish to evaluate and press SHIFT+ENTER. (The special ENTER key on the lower-right corner of the keypad of an extended keyboard also works.)

Example 12.1. We add 2 and 2 to get ... 4.

```
In[1]:= 2 + 2

Out[1]= 4
```

Note. Mathematica assigns line numbers to the input and output, e.g., the "`In[1]:=`" and "`Out[1]=`" above. You do not type them yourself.

To multiply two numbers, type the numbers with a space between them. Use a caret (`^`) for exponentiation. Notice that Mathematica can handle very large numbers easily, even numbers with hundreds of digits.

Example 12.2. A product, and the value of 3^{100}.

```
In[2]:= 1024 59049

Out[2]= 60466176

In[3]:= 3^100

Out[3]= 515377520732011331036461129765621272702107522001
```

The values of important mathematical constants (such as π, e, and i) are stored in Mathematica. To distinguish them from variables you might create yourself, Mathematica's internal constants are capitalized (`Pi`, `E`, `I`, etc.).

The built-in constants are handled algebraically, but we can request the numerical value of an expression with the `N` function.

Example 12.3. Calculations with π, e, and i.

```
In[4]:= Pi

Out[4]= Pi

In[5]:= N[Pi]

Out[5]= 3.14159

In[6]:= N[E]

Out[6]= 2.71828

In[7]:= I I

Out[7]= -1
```

If you want a numerical result given to a high degree of accuracy, use the command `N[_,_]`. The first argument of this function is the number to be calculated. The second argument is the number of decimal places to which the number is computed.

Example 12.4. We calculate π to 100 decimal places.

```
In[8]:= N[Pi,100]

Out[8]= 3.14159265358979323846264338327950288419716939937510
058209749445923078164062862089986280348253421170 68
```

Note. You can obtain information about a specific command by typing a question mark followed by the name of the command. For instance, to find out about the function `N`, type:

```
In[9]:= ? N

N[expr] gives the numerical value of expr. N[expr, n]
attempts to give a result with n-digit precision.
```

In addition to processing numerical calculations, Mathematica performs algebraic operations. If a variable has not been assigned a value, Mathematica will work with it algebraically.

Example 12.5. We set a equal to 17, and then calculate with a and the (undefined) variable b.

```
In[10]:= a = 17

Out[10]= 17

In[11]:= -b (a^3 + a - 15)

Out[11]= -4915 b
```

To work with the output of the previous command, use the special variable `%` (percentage sign).

Example 12.6. We compute the square of the output of the previous example.

```
In[12]:= %^2

                    2
Out[12]= 24157225 b
```

Mathematica also performs matrix calculations. Matrices are entered with braces and are stored as lists of lists.

Example 12.7. We define two matrices:

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \text{ and } N = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

```
In[13]:= m = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

In[14]:= n = {{0, 1, 0}, {0, 0, 1}, {1, 0, 0}};
```

Note. The `;` (semicolon) symbol is used to separate commands, allowing you to perform more than one calculation on a line. If you end a command with a semicolon, the output will not be displayed.

Example 12.8. We add and multiply the matrices.

```
In[15]:= m + n

Out[15]= {{1, 3, 3}, {4, 5, 7}, {8, 8, 9}}

In[16]:= m . n

Out[16]= {{3, 1, 2}, {6, 4, 5}, {9, 7, 8}}
```

We may want to use the same variables (e.g., a, m, and n in the above computations) later in a different context. Therefore, it is a good idea to "clear" the values of variables when we are finished using them. We can then check that the values of these variables have disappeared.

```
In[17]:= Clear[a,m,n]

In[18]:= {a, m, n}

Out[18]= {a, m, n}
```

Maple as a calculator

Maple syntax is different from Mathematica syntax, but some things are only slightly different. For instance, to form the product of two expressions, we must explicitly use an asterisk ($*$) in Maple. Other things are exactly opposite. For instance, in Maple, ENTER evaluates an expression, whereas SHIFT+ENTER merely breaks the line (i.e., inserts a soft break).

Historically, expressions in Maple had to be "terminated" with a semicolon (;). Newer versions no longer have this requirement, although the semicolon is still allowed. We have kept with the old convention to stay compatible with all versions of Maple.

Example 12.9. We add 2 and 2 to get ... 4.

```
> 2 + 2;
                                    4
```

To multiply two numbers, type the numbers with an asterisk (`*`) between them. Use a caret (`^`) for exponentiation. Notice that Maple can handle very large numbers easily, even numbers with hundreds of digits.

Example 12.10. A product and the value of 3^{100}.

```
> 1024*59049;
                                 60466176
> 3^100;
          515377520732011331036461129765621272702107522001
```

The values of important mathematical constants (such as π and i) are stored in Maple. They are capitalized to distinguish them from variables you might create yourself (e.g., `Pi`, and `I`). Notably, e is *not* represented by `E` in Maple; you must use `exp(1)` instead.

The built-in constants are handled algebraically, but we can request the numerical value of expressions with the `evalf()` function, which evaluates them as "floating point" values.

Example 12.11. Calculations with π, e, and i.

```
> Pi;
                                   Pi
> evalf( Pi );
                               3.141592654
> exp( 1 );
                                    e
> I*I;
                                   -1
```

If you want a numerical result given to a high degree of accuracy, include a second argument in the `evalf()` function. The first argument is the number to be calculated. The second argument is the number of decimal places to which the number is computed.

Example 12.12. We calculate π to 100 decimal places.

```
> evalf( Pi, 100 );
3.14159265358979323846264338327950288419716939937510 58 \
   20974944592307816406286208998628034825342117068
```

Note. You can obtain information about a specific command by typing a question mark (`?`) followed by the name of the command. For instance, to find out about the function `evalf()`, type `? evalf`.

In addition to processing numerical calculations, Maple performs algebraic operations. If a variable has not been assigned a value, Maple will work with it algebraically.

Example 12.13. We set a equal to 17, and then calculate with a and the (undefined) variable b.

```
> a := 17;
                                  a:=17
> -b*(a^3 + a - 15);
                                -4915 b
```

If you want to work with the output of the previous command, use the special variable `%` (percentage sign).

Example 12.14. We compute the square of the output of the previous example.

```
> %^2;
                                      2
                              24157225 b
```

Maple can also perform matrix calculations. Many of Maple's matrix functions are part of a package called `linalg`, which must be loaded first.

Example 12.15. Loading Maple's linear algebra package.

```
> with( linalg ):
```

Note. Lines in Maple may be terminated with a colon (`:`) to suppress the output of the calculation.

Matrices are entered as lists of lists (in square brackets) using the `matrix()` function. Matrix operations must occur inside of `evalm` if we expect them to be evaluated in the usual way. Notice that Matrix multiplication is somewhat special; it uses the `&*` operator and not the usual `*`.

Example 12.16. Matrices in Maple.

```
> m := matrix( [ [ 1, 2, 3], [4, 5, 6], [7, 8, 9 ] ] ):
> n := matrix( [ [ 0, 1, 0], [ 0, 0, 1], [ 1, 0, 0 ] ] ):
> m;
                                      m
> evalm( m );
                                [ 1 2 3 ]
                                [ 4 5 6 ]
                                [ 7 8 9 ]
> evalm( m+n );
                                [ 1 3 3 ]
                                [ 4 5 7 ]
                                [ 8 8 9 ]
```

```
> evalm( m &* n );
                                  [ 3 1 2 ]
                                  [ 6 4 5 ]
                                  [ 9 7 8 ]
```

We may want to use the same variables (e.g., a) later in a different context. Therefore, it's a good idea to "unassign" the values of variables when we're finished using them. We can then check that the values of these variables have disappeared.

```
> a := 17;
                                    a:=17
> unassign( 'a' );
> a;
                                     a
```

Maxima as a calculator

Maxima expects all expressions to end with a semicolon (;), and if you use "command line Maxima," it will continue to prompt for input (without evaluating) until you enter a semicolon. In this respect, Maxima is not unlike many computer languages such as C or Java. This behavior can be convenient as well, since long computations may be split over several lines. The graphical front end, wxMaxima, uses SHIFT+ENTER to run a calculation, like Mathematica does, and will supply a terminating semicolon if you forget one.

Example 12.17. Here we do some simple addition and subtraction. The subtraction is split over two lines.

```
(%i1) 2+2;
(%o1)                                  4
(%i2) 3 -
1 ;
(%o2)                                  2
```

As you use Maxima, you will notice that the input prompt increments each time you enter a calculation: first `(%i1)`, then `(%i2)`. Each piece of output is also marked, beginning with `(%o1)`. As long as the program remains running, these names can be used as variables in later computations. In a similar spirit, the percent sign alone `(%)` always refers to "the last answer."

Example 12.18. Compute with the last answer, or with an arbitrary previous answer.

```
(%i1) 2+2;
(%o1)                                  4
(%i2) 3 * %;
(%o2)                                 12
(%i3) 5 * %i1;
(%o3)                                 20
```

Maxima can handle very large numbers, larger than may be convenient to work with on a hand-held calculator.

Example 12.19. We evaluate 3^{100}.

```
(%i4) 3^100;
(%o4)          515377520732011331036461129765621272702107522001
```

Many standard constants, such as π, e, and the imaginary unit i, are part of Maxima. Maxima uses a syntax with percent signs to signify the built-in constants, very much like the special variables that represent the results of previous calculations. So `%pi`, `%e`, and `%i` denote π, e, and i.

Maxima handles constants algebraically when possible, but we can force numerical presentation with the special symbol `numer`.

Example 12.20. Calculations with π, e, and i.

```
(%i1) %pi;
(%o1)                                %pi
(%i2) %pi,numer;
(%o2)                          3.141592653589793
(%i3) %e,numer;
(%o3)                          2.718281828459045
(%i4) %i * %i;
(%o4)                                - 1
```

To specify arbitrary "big floating point" precision, there is also a special symbol called `bfloat`.

Example 12.21. The number π calculated to 100 decimal places.

```
(%i1) %pi,bfloat,fpprec=100;
(%o1) 3.14159265358979323846264338327950288419716939937\
5105820974944592307816406286208998628034825342117068b0
```

Note. Big floating point results are always presented in scientific notation, so do not forget to read the exponent following the "b". The result above is in fact $3.14\ldots \times 10^0$.

Note. You can obtain information about a specific symbol (or function) by typing a question mark (and space) followed by the name of the symbol. This is one of the few times Maxima does *not* want a terminating semicolon. For example, to find out about the special symbol `numer`:

```
(%i1) ? numer

 -- Special symbol: numer
     'numer' causes some mathematical functions (including
     exponentiation) with numerical arguments to be evaluated
     in floating point. It causes variables in 'expr' which
     have been given numerals to be replaced by their values.
     It also sets the 'float' switch on.

  There are also some inexact matches for 'numer'.
  Try '?? numer' to see them.

(%o1)                                true
```

In addition to numerical calculations, Maxima can work with variables and do algebraic operations. The colon (`:`) is the assignment operator. We use it to define variables. If a variable has not been assigned a value, Maxima will work with it algebraically.

Example 12.22. We set a equal to 17, and then calculate with a and the (undefined) variable b.

```
(%i1) a : 17;
(%o1)                                17
(%i2) -b * (a^3 + a - 15);
(%o2)                             - 4915 b
```

We can tell Maxima that we wish it to handle a variable name algebraically, even if it knows how to evaluate it, by using the single quote (`'`) symbol. For example, in an expression, `'x` will not be expanded, even if a value of x has already been defined. When we are ready to have Maxima evaluate an expression, we can use the `ev()` function.

Example 12.23. Quoting a variable prevents evaluation.

```
(%i1) x : 2;
(%o1)                                 2
(%i2) y : x^2;
(%o2)                                 4
(%i3) z : 'x^2;
                                      2
(%o3)                                 x
(%i4) ev(z);
(%o4)                                 4
```

Maxima can handle vectors and matrices that contain values or variables, and it can do the usual mathematical operations on them. Simple row vectors may be entered as lists in square brackets. Matrices with more than one row are defined using the `matrix()` function, which takes the rows of the matrix as its arguments.

Note. The asterisk (`*`) and caret (`^`) operators both work component-wise on matrices (i.e., on each entry). Use dot (`.`) and double caret (`^^`) for matrix multiplication and matrix exponentiation, respectively.

Example 12.24. Matrices in Maxima.

```
(%i1) M : matrix( [ 17, b], [1, 17 ] );
                                  [ 17  b  ]
(%o1)                             [        ]
                                  [ 1   17 ]
(%i2) M . M;
                             [ b + 289   34 b   ]
(%o2)                        [                  ]
                             [   34     b + 289 ]
(%i3) M + M;
                                  [ 34  2 b ]
(%o3)                             [         ]
                                  [ 2   34  ]
(%i4) [ 1, 0] . M;
(%o4)                              [ 17  b ]
```

```
(%i5) M ^^ 2;
                              [ b + 289    34 b   ]
(%o5)                         [                   ]
                              [   34     b + 289  ]
```

The `kill()` function is used to clear variables. For example, if we wish to clear the variable M to use it in some other way, we can kill it.

Example 12.25. Clearing (killing) the variable M.

```
(%i6) kill( M );
(%o6)                                done
(%i7) M;
(%o7)                                  M
```

12.3 How to compute functions

Functions in Mathematica

The operator `N`, which we saw earlier, is actually a function. Mathematica contains many such built-in functions, and you can usually guess the names of common functions. For instance, `Sin[x]` computes $\sin x$.

Note. In Mathematica, every built-in function name begins with a capital letter. Arguments of functions follow in square brackets.

Example 12.26. We calculate $\sin(\pi/2)$ and the binomial coefficient $\binom{7}{2}$.

```
In[1]:= Sin[Pi/2]

Out[1]= 1

In[2]:= Binomial[7,2]

Out[2]= 21
```

Some functions have outputs that are lists.

Example 12.27. The command `FactorInteger` determines the prime factorization of an integer. Here we find the prime factorization of the number 60466176.

```
In[3]:= FactorInteger[60466176]

Out[3]= {{2, 10}, {3, 10}}
```

The output tells us that $60466176 = 2^{10} \cdot 3^{10}$.

In the next example, we calculate and display a table of function values.

Example 12.28. The function `Prime[n]` gives the nth prime number. Using this function, we construct a table of the first 100 prime numbers.

Command	Meaning	Example Input	Meaning
`Sqrt[]`	square root	`Sqrt[5]`	$\sqrt{5}$
`Exp[]`	exponential	`Exp[x]`	e^x
`Log[]`	natural logarithm	`Log[10]`	$\ln 10$
`Log[,]`	logarithm	`Log[10,5]`	$\log_{10} 5$
`Sin[]`	sine	`Sin[x]`	$\sin x$
`Cos[]`	cosine	`Cos[x]`	$\cos x$
`Tan[]`	tangent	`Tan[x]`	$\tan x$
`Sum[,]`	sum	`Sum[a[i],{i,1,n}]`	$\sum_{i=1}^{n} a_i$
`Product[,]`	product	`Product[a[i],{i,1,5}]`	$\prod_{i=1}^{5} a_i$
`Mod[,]`	modulus	`Mod[10,3]`	$10 \bmod 3$

TABLE 12.1: Some Mathematica functions.

```
In[4]:= Table[Prime[n], {n, 1, 100}]

Out[4]= {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
> 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107,
> 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
> 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239,
> 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311,
> 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383,
> 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457,
> 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541}
```

Mathematica can evaluate functions both arithmetically and symbolically.

Example 12.29. The sums $\sum_{i=1}^{10} i^2$ and $\sum_{i=1}^{n} i^2$.

```
In[5]:= Sum[i^2, {i, 1, 10}]

Out[5]= 385

In[6]:= Sum[i^2, {i, 1, n}]

        1
Out[6]= - n (1 + n) (1 + 2 n)
        6
```

As we can see, Mathematica knows that $\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$.

Table 12.1 displays some useful Mathematica functions.

You can define your own functions. To create a function $f(x)$, write `f[x_] :=` followed by the definition of f.

Example 12.30. We define a function $f(x) = x^3 + \sin x$.

```
In[7]:= f[x_] := x^3 + Sin[x]
```

```
In[8]:= f[Pi/2]

              3
            Pi
Out[8]= 1 + ---
             8
```

We can differentiate and integrate our function.

```
In[9]:= D[f[x],x]

           2
Out[9]= 3 x  + Cos[x]

In[10]:= Integrate[f[x],x]

          4
         x
Out[10]= -- - Cos[x]
         4


In[11]:= Integrate[f[x], {x, 0, Pi}]

         1         4
Out[11]= - ( 8 + Pi  )
         4
```

Note. Mathematica does not supply an additive constant ($+C$) for indefinite integrals.

You can define functions recursively (in terms of previous values), as with the function below. Notice the use of `=` for the assignment of initial values in contrast with `:=` for the definition of the iteration.

Example 12.31. We define the Fibonacci sequence.

```
In[1]:= f[0] = 1;

In[2]:= f[1] = 1;

In[3]:= f[n_] := f[n] = f[n-2] + f[n-1]

In[4]:= Table[f[n], {n, 0, 10}]

Out[4]= {1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89}
```

You may wonder at the construction in line 3. In Mathematica, `:=` is the "delayed" assignment operator as opposed to `=`, which does immediate assignment. When we use delayed assignment, Mathematica will wait to fully evaluate an expression.

To explore this idea, consider the first time we ask Mathematica for the value `f[3]`. Since we used `:=`, the value of 3 will be substituted for n on the right side of the definition, which will evaluate to the expression `f[3] = f[1] + f[2]` (which is actually an assignment itself). Mathematica already knows the values of `f[1]` and `f[2]` and consequently sets `f[3] = 2`

using the immediate assignment operator. The = also "returns a value," the value that we see.

In more detail, the steps performed by Mathematica to do the evaluation of `f[3]` are:

1. `f[3] = f[3-2] + f[3-1]` (this is the right-hand side of the `:=`, with 3 substituted for n in all places)
2. `f[3] = f[1] + f[2]`
3. `f[3] = 1 + 1` (from previously defined values)
4. `f[3] = 2` (this is ready to perform immediate assignment)
5. `2` (the return value of the `=` assignment).

Of course, all we see is the final result:

```
In[5]:= f[3]

Out[5] = 2
```

The consequence of doing the computation this way is that Mathematica now knows permanently that `f[3]` has the value 2 and will never have to evaluate it again (say, when we ask for `f[4]` or any other value). This becomes important for larger values, like `f[100]`, which would evaluate too slowly if we created the function less carefully.

Functions in Maple

Some of the commands we've seen, like `matrix()` and `evalf()`, are actually functions. Maple contains many built-in functions, including common mathematical functions like `sin()` and `cos()`.

Example 12.32. We calculate $\sin(\pi/2)$ and the binomial coefficient $\binom{7}{2}$.

```
> sin( Pi/ 2 );
                                    1
> binomial( 7, 2 );
                                   21
```

Maple also has functions related to Number Theory. For example, `ifactor()` determines the prime factorization of an integer.

```
> ifactor( 60466176 );
                                  10    10
                               (2)   (3)
```

Maple can easily generate sequences (i.e., a table or list of values). Using the `seq()` function and the `ithprime()` function (which returns the ith prime number), we construct a list of the first 100 prime numbers.

```
> seq( ithprime(n), n=1..100 );
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
  67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
  139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199,
```

Command	Meaning	Example Input	Meaning
`sqrt()`	square root	`sqrt(5)`	$\sqrt{5}$
`exp()`	exponential	`exp(x)`	e^x
`ln()` or `log()`	natural logarithm	`ln(10)` or `log(10)`	$\ln 10$
`log10()`	common logarithm	`log10(5)`	$\log_{10} 5$
`sin()`	sine	`sin(x)`	$\sin x$
`cos()`	cosine	`cos(x)`	$\cos x$
`tan()`	tangent	`tan(x)`	$\tan x$
`sum(,)`	sum	`sum(i^2,i=1..n)`	$\sum_{i=1}^{n} i^2$
`mul(,)`	product	`mul(i^2,i=1..5)`	$\prod_{i=1}^{5} i^2$
`mod(,)`	modulus	`mod(10,3)`	$10 \bmod 3$

TABLE 12.2: Some Maple functions.

```
211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277,
281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359,
367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439,
443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521,
523, 541
```

Maple has functions that compute sums and products, which it can evaluate both arithmetically and symbolically.

Example 12.33. The sums $\sum_{i=1}^{10} i^2$ and $\sum_{i=1}^{n} i^2$.

```
> sum( i^2, i=1..10 );
                                385
> sum( i^2, i=1..n );
                                3         2
                           (n+1)     (n+1)     n   1
                           -----  - -----  + - + -
                             3         2       6   6
```

Maple knows in its own way that $\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$.

Table 12.2 displays some useful Maple functions.

You can create your own functions. To create a function, use the `:=` and `->` operators.

Example 12.34. We define a function $f(x) = x^3 + \sin x$.

```
> f := x -> x^3 + sin(x);
                                    3
                          f := x -> x  + sin(x)
> f( Pi/2 );
                                  3
                                Pi
                                --- + 1
                                 8
```

We can differentiate and integrate our function.

```
> diff( f(x), x );
                                    2
                                3 x + cos(x)
> integrate( f(x), x );
                                  4
                                 x
                                --- - cos(x)
                                 4
> integrate( f(x), x=0..Pi );
                                    4
                                  Pi
                                  ---  + 2
                                   4
```

Note. Maple does not supply an additive constant ($+C$) for indefinite integrals.

Functions in Maxima

Some of the commands we've seen, like `matrix()` and `kill()`, are actually functions. Maxima contains many built-in functions, including common mathematical functions like `sin()` and `cos()`.

Example 12.35. We compute a couple of trigonometric functions.

```
(%i1) sin( %pi/2 );
(%o1)                                  1
(%i2) cos( %pi/4 );
                                       1
(%o2)                                -------
                                     sqrt(2)
```

Maxima also contains functions for counting things, like binomial coefficients, as well as for dealing with numbers and factorizations.

Example 12.36. Computing with some of Maxima's other functions:

```
(%i1) binomial( 7, 3 );
(%o1)                                  35
(%i2) factor( 210 );
(%o2)                               2 3 5 7
(%i3) next_prime(1);
(%o3)                                   2
(%i4) next_prime(8);
(%o4)                                  11
(%i5) prev_prime(2009);
(%o5)                                 2003
```

Some functions must be loaded before they are available. For example, by default Maxima does not load the `permutation()` function. It is contained in a packaged called `functs`. Similarly, it does not load all of the functions you might use to do descriptive statistics (like means or standard deviations).

Example 12.37. Loading functions into Maxima:

```
(%i8) load( functs );
(%o8)      /usr/share/maxima/5.13.0/share/simplification/functs.mac
(%i9) permutation( 10, 3 );
(%o9)                                720
(%i10) load (descriptive)$
(%i11) mean( [ 1, 2, 3, 4, 5 ] );
(%o11)                                3
```

Note. The dollar sign (`$`) at the end of a line serves exactly as a semicolon, but will suppress the output. Notice that there is no `(%o10)` above.

Maple has functions that compute sums and products, and it can evaluate them both arithmetically and symbolically.

Example 12.38. Maxima can compute products and sums:

```
(%i17) prod( sqrt(i), i, 1, 4 );
                                    3/2
(%o17)                             2    sqrt(3)
(%i18) sum( i^2, i, 1, n );
                                     n
                                    ====
                                    \       2
(%o18)                               >     i
                                    /
                                    ====
                                    i = 1
```

Note. The variable `simpsum` controls whether Maxima will perform simplifications to sums. It is false by default. Setting it to true will make Maxima return a closed form for the sum above.

Example 12.39. Maxima will simplify sums when `simpsum` is true:

```
(%i19) simpsum : true;
(%o19)                               true
(%i20) sum( i^2, i, 1, n );
                                    3      2
                                 2 n  + 3 n  + n
(%o20)                           ---------------
                                        6
```

Maxima knows that $\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$.

Maxima can also apply a condition to a calculation. For example, if we are interested in the previous sum when $n = 10$, we could reevaluate it with that condition placed after the comma (,) operator.

Command	Meaning	Example Input	Meaning
`sqrt()`	square root	`sqrt(5)`	$\sqrt{5}$
`exp()`	exponential	`exp(x)`	e^x
`log()`	natural logarithm	`log(10)`	$\ln 10$
`sin()`	sine	`sin(x)`	$\sin x$
`cos()`	cosine	`cos(x)`	$\cos x$
`tan()`	tangent	`tan(x)`	$\tan x$
`sum(,,,)`	sum	`sum(i^2,i,1,n)`	$\sum_{i=1}^{n} i^2$
`prod(,,,)`	product	`prod(i^2,i,1,5)`	$\prod_{i=1}^{5} i^2$
`mod(,)`	modulus	`mod(10,3)`	$10 \bmod 3$

TABLE 12.3: Some Maxima functions.

Example 12.40. Evaluating an expression at a particular value:

```
(%i21) %, n=10;
(%o21)                                385
```

Table 12.3 displays some useful Maxima functions.

To define your own functions, use the `:=` operator.

Example 12.41. We define a function $f(x) = x^3 + \sin x$.

```
(%i1) f(x) := x^3 + sin(x);
                                      3
(%o1)                        f(x) := x  + sin(x)
(%i2) f( %pi/2 );
                                      3
                                   %pi
(%o2)                              ---- + 1
                                    8
```

We can differentiate and integrate our function.

```
(%i3) diff( f(x), x );
                                              2
(%o3)                              cos(x) + 3 x
(%i4) integrate( f(x), x );
                                    4
                                   x
(%o4)                              -- - cos(x)
                                   4
(%i5) integrate( f(x), x, 0, %pi );
                                       4
                                    %pi  + 8
(%o5)                               --------
                                       4
```

Note. Maxima does not supply an additive constant ($+C$) for indefinite integrals.

Functions in Maxima may also be defined recursively (in terms of previous values). Perhaps the easiest way is to code the function as a small program. For example, the famous Fibonacci sequence might be defined like this:

Example 12.42. Defining and evaluating the Fibonacci sequence.

```
(%i1) f(n) := if n<2 then 1 else f(n-1)+f(n-2);
(%o1)          f(n) := if n < 2 then 1 else f(n - 1) + f(n - 2)
(%i2) makelist( f(n), n, 1, 10 );
(%o2)                 [1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Although technically correct, this is a very inefficient way to compute Fibonacci numbers. The problem is that the same values of the Fibonacci sequence must be computed over and over again (the evaluation of `f(3)` computes `f(1)` twice, for example). Large values like $f(100)$ will take much too long to compute using this naive definition. From a performance point of view, it would have been better to use a loop (or some even better method) instead of recursion for this particular computation.

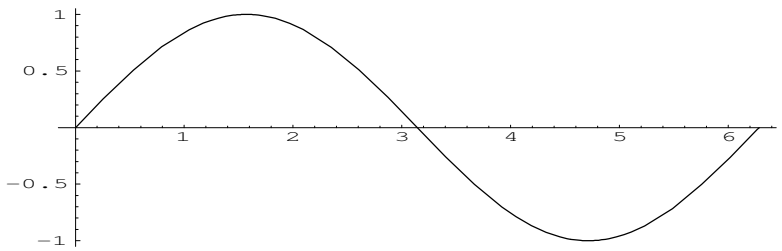
12.4 How to make graphs

Graphs in Mathematica

Mathematica offers many graphing options. We show a few examples here. You can create graphs of functions using Mathematica's `Plot` command.

Example 12.43. A graph of the function $y = \sin x$, for $0 \le x \le 2\pi$.

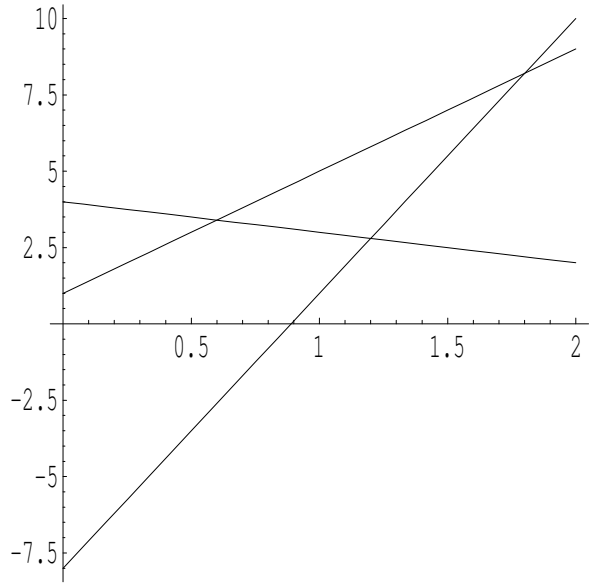
```
In[1]:= Plot[Sin[x], {x, 0, 2 Pi}]
```



You can graph several curves together.

Example 12.44. A graph of three lines, $y = 4x + 1$, $y = -x + 4$, and $y = 9x - 8$, for $0 \le x \le 2$.

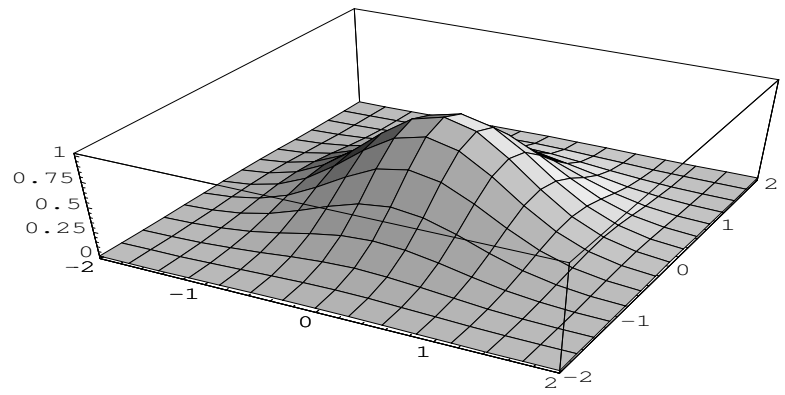
```
In[1]:= f[x_] := 4 x + 1;

In[2]:= g[x_] := -x + 4;

In[3]:= h[x_] := 9 x - 8;

In[4]:= Plot[{f[x], g[x], h[x]}, {x, 0, 2}]
```



You can create 3-dimensional graphs of surfaces using the `Plot3D` command.

Example 12.45. A graph of the surface $z = e^{-(x^2+y^2)}$, for $-2 \leq x,\, y \leq 2$.

```
In[1]:= Plot3D[E^(-(x^2 + y^2)), {x, -2, 2}, {y, -2, 2}]
```



Example 12.46. We plot a sphere using parametric equations.

```
In[1]:= ParametricPlot3D[{Sin[phi] Cos[theta],
 Sin[phi] Sin[theta], Cos[phi]},
 {phi, 0, Pi}, {theta, 0, 2 Pi}]
```

See Figure 2 in the color insert.

If you want an Encapsulated PostScript (EPS) version of your image, use an `Export` command.

```
In[2]:= Export["newgraph.eps", %]
```

The graphics file, `newgraph.eps`, is stored in a "working directory," which you can identify using the `Directory` command.

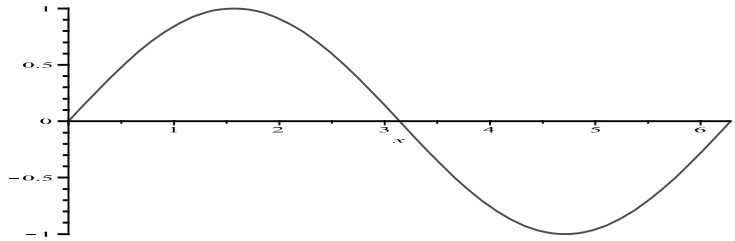
```
In[3]:= Directory[]

Out[3]= C:\\Program Files\\Wolfram Research\\Mathematica\\7
```

Graphs in Maple

Maple can create beautiful graphs, and a few examples are shown here. You can create graphs of functions using the `plot()` function.

Example 12.47. A graph of the function $y = \sin x$, for $0 \le x \le 2\pi$.

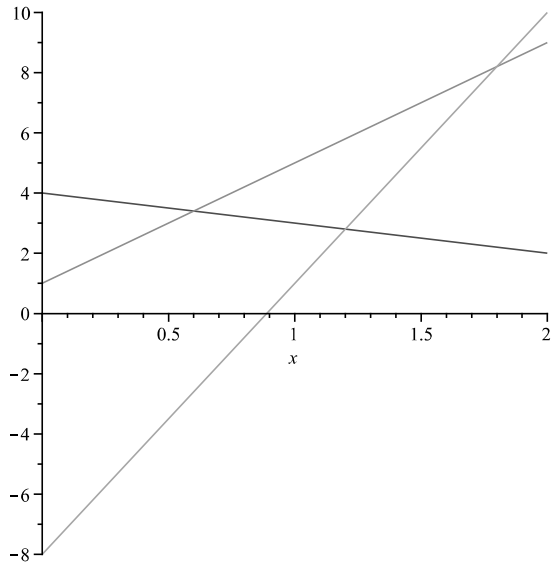
```
> plot( sin(x), x=0..2*Pi );
```



You can graph several curves together by passing a set of functions to the `plot()` function.

Example 12.48. A graph of three lines, $y = 4x + 1$, $y = -x + 4$, and $y = 9x - 8$, for $0 \le x \le 2$.

```
> f := x -> 4*x+1:
> g := x -> -x + 4:
> h := x -> 9*x - 8:
> plot( { f(x), g(x), h(x) }, x=0..2 );
```



You can create 3-dimensional graphs of surfaces using the `plot3d()` function.

Example 12.49. A graph of the surface $z = e^{-(x^2+y^2)}$, for $-2 \le x,\, y \le 2$.

```
> plot3d( exp( -(x^2 + y^2)), x=-2..2, y=-2..2 );
```

See Figure 3 in the color insert.

The `plot3d()` function can also perform parametric plots if you pass a list (in square brackets) of the (x, y, z) coordinates of your surface.

Example 12.50. A parametric plot of a torus.

```
> plot3d([(2+cos(v))*cos(u), (2+cos(v))*sin(u), sin(v)],
u = 0 .. 2*Pi, v = 0 .. 2*Pi,
axes = framed, labels = [x, y, z], scaling = constrained);
```

See Figure 4 in the color insert.

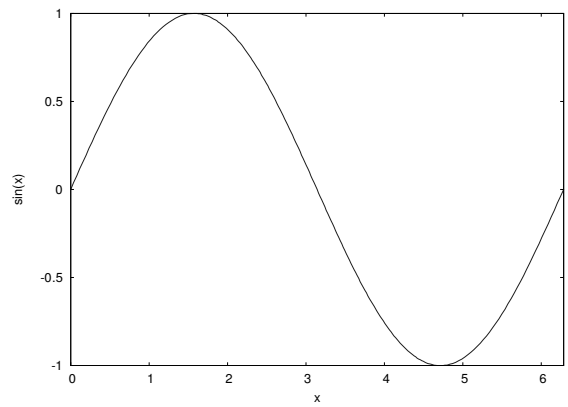
Images generated in Maple may be saved in several different formats. Simply right-click the image you wish to save and choose Export.

Graphs in Maxima

Maxima can graph both curves and surfaces. You can create graphs of curves with the `plot2d()` command.

Example 12.51. A graph of the function $y = \sin x$, for $0 \le x \le 2\pi$.

```
(%i1) plot2d( sin(x), [x, 0, 2*%pi ] )$
```



Note. If you use wxMaxima, substitute `wxplot2d()` for `plot2d()` to create "inline" graphics that are integrated into your document.

You can graph several curves together by combining them in a list (i.e., in square brackets).

Example 12.52. A graph of three lines, $y = 4x + 1$, $y = -x + 4$, and $y = 9x - 8$, for $0 \le x \le 2$

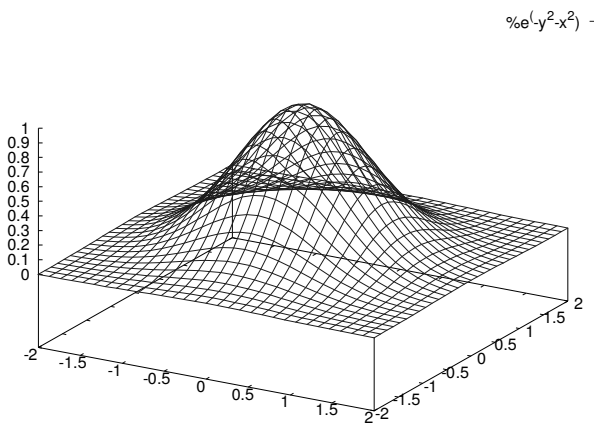
```
(%i1) f(x) := 4*x + 1;
(%o1)                          f(x) := 4 x + 1
(%i2) g(x) := -x + 4;
(%o2)                          g(x) := - x + 4
(%i3) h(x) := 9*x - 8;
(%o3)                          h(x) := 9 x - 8
(%i4) plot2d( [f(x), g(x), h(x)], [x,0,2] )$
```

See Figure 5 in the color insert.

You can create 3-dimensional graphs of surfaces using the `plot3d()` command.

Example 12.53. A graph of the surface $z = e^{-(x^2+y^2)}$, for $-2 \le x, y \le 2$.

```
(%i1) plot3d( %e^( -(x^2+y^2) ), [x,-2,2], [y,-2,2] )$
```



Note. In wxMaxima documents, use `wxplot3d()` to create inline 3D plots.

The plotting engine that lies beneath Maxima is called gnuplot, and it is a very functional program of its own. By adding extra arguments to the `plot3d()` command, we can ask gnuplot to enhance the graphs that we construct in Maxima. For example, we can choose different coloring algorithms, we can draw contour lines on the surface (or beneath it), we can specify a particular viewing angle, or we can label the axes.

Example 12.54. A nicely colored surface with contour lines and labeled axes.

```
(%i1) plot3d( %e^( -(x^2+y^2) ), [x,-2,2], [y,-2,2],
[gnuplot_preamble, "set pm3d; set hidden3d; set contour both;\
 set xlabel 'x'; set ylabel 'y'; set zlabel 'z';\
 set key top left"] )$
```

See Figure 6 in the color insert.

Gnuplot settings that you may wish to include:

- `set contour base`

- `set contour surface`
- `set contour both`
- `set cntrparam levels 10`
- `set cntrparam levels discrete 1,2,4,8` (sets contour lines at specific values)
- `set hidden3d` (makes the surface opaque)
- `set nohidden3d` (makes the surface transparent)
- `set key` (moves the legend key to different places)
- `set pm3d` (colors the surface according to height)
- `set surface` (draws the surface)
- `set nosurface` (hides the surface, but contour lines and pm3d will still show)
- `set xlabel 'x axis'`
- `set ylabel 'y axis'`
- `set zlabel 'z axis'`
- `set view 45,115` (rotates the viewpoint around x-axis and z-axis)

Maxima doesn't have a command for printing a plot, but it does have options for saving a plot to a file, which you can then print. The most useful file formats are probably PostScript and PNG (Portable Network Graphics). Since gnuplot actually does the plotting, we simply have Maxima tell gnuplot what we want. If we don't specify, plots will be saved with the name `maxplot.ps` (for PostScript files) or `maxplot.png` (for PNG files).

Example 12.55. Saving a plot to a file.

```
(%i1) plot2d( x^2, [x,0,3], [gnuplot_term, ps] )$

(%i2) plot2d( x^2, [x,0,3], [gnuplot_term, png] )$
```

Maxima will try to guess the best place to save your files, usually your "home" directory on GNU/Linux or Unix systems and in the "My Documunts" folder on Windows systems. The variable `maxima_tempdir` determines where images will be saved. Notice that even on systems running Microsoft Windows, Maxima uses forward slashes to indicate folders (just as browsers do with Web addresses).

Example 12.56. Changing the default location for saved images.

```
(%i3) maxima_tempdir;
(%o3)                 C:/Documents and Settings/dbindner
(%i4) maxima_tempdir : "C:/";
(%o4)                                 C:/
```

We can also specify a file name if we wish. For example, if we want a plot of $y = x^2$ saved as (encapsulated) PostScript with the file name `parabola.eps`, we could do something like this:

Example 12.57. Naming the file that a Maxima plot is saved to.

```
(%i5) plot2d( x^2, [x,0,3],
      [gnuplot_term, ps], [gnuplot_out_file, "parabola.eps" ] )$
```

12.5 How to do simple programming

Programming in Mathematica

Mathematica supports a full spectrum of programming paradigms, including procedural, functional, transformational, and object-oriented approaches. We give a sampling here.

A `Do` loop is a simple kind of program that performs a calculation some fixed number of times.

Example 12.58. A calculation related to the fractal known as the Mandelbrot set. We set $c = -0.5 + 0.5i$ and $z = 0 + 0i$. Then we iterate the function $f(z) = z^2 + c$ ten times.

```
In[1]:= c = -0.5 + 0.5 I;

In[2]:= z = 0 + 0 I;

In[3]:= Do[z = z^2 + c, {10}];

In[4]:= z

Out[4]= -0.11932 + 0.219608 I

In[5]:= Clear[c,z]
```

This is fine for a one-time computation. But perhaps we wish to run the same program several times, with different values for c and different numbers of iterations. To do this, we create a module, which is a procedure containing local variables.

Example 12.59. We define a module containing the local variable z. The values of c and i (the number of iterations) are input when the module is called.

```
In[1]:= f[c_, i_] := Module[{z}, z = 0 + 0 I;
          Do[z = z^2 + c, {i}];
          z
        ]

In[2]:= f[-0.5 + 0.5 I, 10]

Out[2]= -0.11932 + 0.219608 I
```

Notice that z has no value outside the module.

```
In[3]:= z

Out[3]= z
```

It is good programming practice to use modules, and to make them small and easy to understand.

One useful feature of functions in Mathematica is that they are "threaded" over lists automatically and applied to each list item.

Example 12.60. We thread addition and cubing operations over the list $\{a, b, c\}$.

```
In[1]:= a := 6; c := 2+I;

In[2]:= 1000 + {a,b,c}^3

                          3
Out[2]= {1216, 1000 + b , 1002 + 11 I}
```

Sometimes functions are complex enough to be called programs.

Example 12.61. We define a function

$$f(n) = \frac{1}{n} \sum_{k|n} \phi(k) 2^{n/k}.$$

Note. From the Pólya theory of counting, $f(n)$ is the number of distinct (up to rotation and flipping) necklaces formed by n beads of two types.

The summation is over a set of numbers, namely, the set of positive divisors of n. This set is obtained in Mathematica as `Divisors[n]`. We need to apply the summand, $\phi(k)2^{n/k}$, to each element of this set. The summand contains a "dummy variable," k. To define the summand as a Mathematica function, we replace each instance of the dummy variable with the marker `#` (number sign).

```
EulerPhi[#]2^(n/#)&
```

The `&` (ampersand) identifies the function as a "pure function" in which the argument is denoted by `#`.

Then we apply the function to the set `Divisors[n]` as follows.

```
EulerPhi[#]2^(n/#)&/@Divisors[n]
```

(The construction `f/@s` applies a function `f` to a set `s`.)

Finally, we add the elements of the set produced by this process. The expression `Plus@@s` adds the elements of the set `s`. Thus, we can now define our function in Mathematica.

```
In[1]:= f[n_] := (1/n)Plus@@(EulerPhi[#]2^(n/#)&/@Divisors[n])
```

We test our function.

```
In[2]:= f[4]

Out[2]= 6
```

It is easy to verify by inspection that there are exactly six different necklaces made of four beads of two types.

And now we compute a large value of the function.

```
In[3]:= f[100]

Out[3]= 1267650600228230527396681356O
```

Programming in Maple

Programming in Maple is similar to programming in Mathematica. In Maple, programs are called procedures. Let's create a procedure to find the greatest common divisor (gcd) of two positive integers. Recall that the Euclidean algorithm for finding the gcd of integers a and b is based on the relation

$$\gcd(a, b) = \gcd(b, r),$$

where $a = bq + r$, with $0 \leq r < b$. In Maple, the remainder r is given by `modp(a,b)`.

Here is our procedure, which we call `ourgcd`.

```
> ourgcd := proc(a,b)
  local atemp, btemp;
  (atemp, btemp) := (a, b);
  while btemp > 0 do
    (atemp, btemp) := (btemp,modp(atemp,btemp));
  end do;
  atemp;
end proc:
```

We use the construction `proc()` and `end proc` to begin and end a procedure. The inputs in the procedure are `a` and `b`. In the second line, we declare the variables to be used in the procedure (`atemp` and `btemp`). In the third line, we set values for these variables. The fourth line begins with a `while` loop, which has a test condition (`btemp > 0`) and a command to be performed (the key step). The command is bracketed by `do` and `end do`. Finally, an output (the value of `atemp`) is output.

```
> ourgcd(15,24);
                                3
```

Programming in Maxima

Interesting and powerful programs may be expressed in Maxima. In fact, Maxima can be programmed in two different ways. Not only is Maxima a full-fledged programming language, but Maxima itself is written in Lisp, and it is possible to graft custom Lisp programs into Maxima. Both simple and elaborate programs can generally be written without resorting to Lisp, however, building on the syntax that we have already learned.

A `thru-do` loop is a simple kind of program that performs a calculation some fixed number of times.

Example 12.62. A calculation related to the fractal known as the Mandelbrot set. We set $c = -0.5 + 0.5i$ and $z = 0$. Then we iterate $f(z) = z^2 + c$ ten times.

```
(%i1) c : -0.5 + 0.5*%i;
(%o1)                          0.5 %i - 0.5
(%i2) z : 0;
(%o2)                                0
(%i3) thru 10 do z : z^2 + c;
(%o3)                              done
(%i4) expand(z);
(%o4)           0.21960760831735 %i - 0.11932015744635
```

This is fine for a one-time computation. But perhaps we wish to run the same program several times, using different values of c and for a different number of iterations. To do this, we can create a function to perform the computation.

An essential part of getting complex functions to work well is the `block()` function, which is used in Maxima to encapsulate a program. The `block()` function does two important tasks that help protect our code.

The first useful thing that `block()` does is provide a new variable name space. If the first argument in a block is a list of variable names (or variable initializations), then those variables will be local to the block. A local variable z, for example, used in a program will not overwrite a variable z you may have been using in a calculation elsewhere.

The second task that `block()` does is to evaluate a compound expression and return the value of the last computation performed. Whatever value is calculated by the last step of a program becomes the return value for the function.

Example 12.63. A function with one local variable, z, encapsulated in a block. Notice that outside the program z has not been changed or defined.

```
(%i1) f(c,i) := block(
 [ z : 0 ],
 thru i do z : z^2 + c,
 expand(z)
);
(%o1) f(c, i) := block(z : 0, thru i do z : z  + c, expand(z))
(%i2) f( -0.5+0.5*%i, 10 );
(%o2)               0.21960760831735 %i - 0.11932015744635
(%i3) z;
(%o3)                                  z
```

Note. The expressions that make up a compound expression are separated by commas, not semicolons, and the last expression in the block is not followed by anything. The only semicolon comes at the end of the block.

Programs in Maxima can be as elaborate as you can imagine and code. Here is a bit longer example. A common task for Calculus students is to learn and compute Newton's method to find zeros of a function. Maxima has native ways to solve for zeros, but for this example we program Newton's method into Maxima directly.

Example 12.64. A hand-coded Newton's method program used to find the square root of 81 (a solution to $x^2 - 81 = 0$).

```
(%i1) newt( f, x, err ) := block(
 [ df, old, i ],          /* local variables */
 df : diff( f('x), 'x ),  /* differentiate algebraically */

 old : x,
 x : ev(x-f(x)/df, numer),/* evaluate numerically */

 for i:1 thru 20 do (
  if abs(x-old) < abs(err) then return(x),

  old : x,
  x : ev(x-f(x)/df, numer)
 )
```

```
 /* if loop runs out without a return(), */
 /* no value will be returned */
)$

(%i2) f(x) := x^2 - 81;
                                        2
(%o2)                          f(x) := x  - 81
(%i3) newt( f, 1, .001 );
(%o3)                         9.000000000007093
```

12.6 How to learn more

Learning more about Mathematica

There are many aspects of Mathematica not discussed in this introduction, such as standard and add-on packages, sound, and animated graphics. Here are some resources for you to investigate to learn more.

The definitive book about Mathematica is [59]. Good beginning books are [16] and [56]. For informative examples of Mathematica in a wide variety of settings, see [11], [36], [48], and [58]. The books [27] and [26] show many applications of Mathematica to Calculus. For a comprehensive guide to add-on packages, see [37].

For complete and up-to-date information describing Mathematica, you may want to visit the Web site www.wolfram.com. Another interesting site, concerning the *Mathematica in Education and Research* journal, is www.telospub.com/journal/MIER/.

Learning more about Maple

Maple is produced by Waterloo Maple, Inc. You can obtain more information at www.maplesoft.com.

A favorite source to learn more about Maple is the MaplePrimes discussion site at www.mapleprimes.com.

Learning more about Maxima

The Maxima project page is `maxima.sourceforge.net`. The project page contains documentation in various languages for Maxima as well as links to sites for downloading Maxima. Most notably, a version of Maxima that works on Microsoft Windows is available as a free download from `sourceforge.net/projects/maxima/files/`.

Exercises

1. Graph each of the functions. Experiment with different domains or viewpoints to display the best images.

(a) $f(x) = \dfrac{x}{1+x^2}$

(b) $y = x\sin(1/x)$

(c) $g(x,y) = \cos(x) + \sin(y)$

(d) $z = \dfrac{xy}{x^2+y^2}$

2. Let $f(x) = \dfrac{x}{1+x^2}$.

   (a) Find $f'(x)$ and $f''(x)$.

   (b) Find $f'(-1)$ and $f'(0)$.

   (c) Find $f''(0)$ and $f''(1)$.

3. Find the prime factorization of each integer.

   (a) 3,527,218,133,309,949,276,293

   (b) 471,945,325,930,166,269

   (c) 471,945,325,930,166,281

4. Compute each expression. Do you notice a pattern?

   (a) $3^6 \mod 7$

   (b) $6^{10} \mod 11$

   (c) $7^{20} \mod 21$

   (d) $7^{22} \mod 23$

5. In 1976, Whitfield Diffie and Martin Hellman published a way for two people to share a secret number when communicating over an insecure medium (like the Internet).

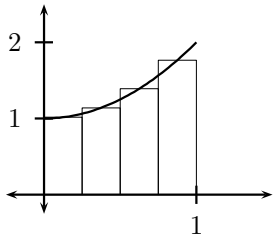
   If Alice and Bob want to communicate, Alice picks a random prime p and another random number g smaller than p. To create a shared secret, Alice and Bob each pick part of the secret. Alice picks a random number a and Bob picks a random b. Alice computes $A = g^a \mod p$, which she sends to Bob, and Bob computes $B = g^b \mod p$ to send to Alice. The actual secret is $g^{ab} \mod p$, which both Alice and Bob can compute, but which someone watching only the communication would find difficult to reproduce (because it's difficult to figure out what a and b are).

   (a) Play the part of Alice. We'll use $p = 36479$ for our prime and a random number $g = 14$, which she shares with Bob (and potentially with the world). Then Alice needs a secret value, and we'll use $a = 5013$. Alice sends Bob the value $A = g^a \mod p$. Compute $14^{5013} \mod 36479$ to send to Bob.

   (b) Play the part of Bob, who has received p, g, A from Alice. Bob needs a secret value, and we'll use $b = 29252$. Bob sends Alice the value $B = g^b \mod p$. Compute $14^{29252} \mod 36479$ to send to Alice.

   (c) Verify that when Alice computes $S = B^a \mod p$ she gets the same answer that Bob does when he computes $S = A^b \mod p$. This is their shared secret that no one else knows.

6. A good estimate for the area under a curve can be obtained using the Midpoint rule, which approximates the exact area using rectangles. Consider the curve $y = 1 + x^2$ on the interval $[0, 1]$. Using Calculus, we can verify that the exact area is $4/3$, but even without Calculus we can approximate the area with rectangles.



Each rectangle has width 1/4 and touches the function at the midpoint of its top side, giving rectangle heights of $1 + (1/8)^2$, $1 + (3/8)^2$, $1 + (5/8)^2$, and $1 + (7/8)^2$.

(a) Estimate the area under the curve by computing the sum:

$$\sum_{i=0}^{3} 0.25 \left(1 + \left(\frac{1+2i}{8}\right)^2\right)$$

(b) Compute an expression for the area estimated by n rectangles:

$$\sum_{i=0}^{n-1} \frac{1}{n} \left(1 + \left(\frac{1+2i}{2n}\right)^2\right)$$

7. To win the jackpot of the Missouri lottery, a ticket holder must correctly match 6 of 40 numbered balls (in any order).

(a) Compute $\binom{40}{6}$, the number of combinations of balls that may be a winning lottery number.

(b) Each ticket holder picks two (different) sets of six numbers for the ticket and wins if either set of six is an exact match. What is the probability that an individual ticket wins the jackpot?

(c) If a person plays every month for 30 years, compute the probability of winning at least one time: $1 - \left(1 - 2/\binom{40}{6}\right)^{360}$.

8. Let $M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$.

(a) Find M^2, M^3, ..., M^{10}.

(b) Do your answers suggest a way to compute Fibonacci numbers? Find the 100th Fibonacci number.

(This method is much more efficient than the computation in Example 12.42 and probably faster than the computation in Example 12.31.)

9. Find solutions to the following equations or systems of equations. Hint: If you use Mathematica, the command you are looking for is `Solve`. Maple and Maxima both use `solve()` to find algebraic solutions. Check the help for examples and syntax.

(a) Find x, if $x^2 + x = 1$.

(b) Find x, if $x^2 + x = -1$.

(c) Find x and y.

$$4x - 3y = 5$$

$$6x + 2y = 14$$

(d) Find x, y, z, and t.

$$\begin{aligned} -2x - 2y + 3z + t &= 8 \\ -3x + 0y - 6z + t &= -19 \\ 6x - 8y + 6z + 5t &= 47 \\ x + 3y - 3z - t &= -9 \end{aligned}$$

10. Some equations are difficult or impossible to solve explicitly, even with software. In such situations, we often resort to numerical methods. Mathematica uses `FindRoot`, Maple uses `fsolve()`, and Maxima uses `find_root()` to find numerical solutions to equations. Here is an example where a numerical approach works well.

    Assume that I invest \$250 at the beginning of the year, \$300 at the beginning of the second quarter, \$350 at the beginning of the third quarter, and \$400 at the beginning of the fourth quarter. At the end of the year, I have \$1365 (because my investments grow). To find my (continuous) rate of return, solve this equation for r:

$$250e^{1.0r} + 300e^{0.75r} + 350e^{0.5r} + 400e^{0.25r} = 1365.$$

11. If n is a positive number, and $g > 0$ is any "guess" for the square root of n, then a better estimate of $\sqrt{n}$ is the average of g and n/g, i.e., $\dfrac{g + n/g}{2}$. Write a function called `mysqrt` that accepts one argument, begins with an initial guess of 1.0, finds 20 new guesses, and returns the answer.

12. The Collatz conjecture states that if we start from any natural number $a_0 = n$ and form a sequence by the rule

$$a_{i+1} = \begin{cases} a_i/2 & \text{if } a_i \text{ is even} \\ 3a_i + 1 & \text{if } a_i \text{ is odd,} \end{cases}$$

    then the sequence eventually contains the value 1. For example, starting from $a_0 = 6$, we get the sequence 6, 3, 10, 5, 16, 8, 4, 2, 1 (we reached 1 after eight steps).

    (a) Write a (recursive) function called `collatz` that accepts a single argument, n, and returns:

    - 0 if n is equal to 1
    - `1+collatz(n/2)` if n is even
    - `1+collatz(3*n+1)` if n is odd

    Thus, `collatz(n)` is the number of steps needed to go from n to 1.

    (b) Verify the values:

n	`collatz(n)`
1	0
2	1
6	8
27	111