October 23, 1992
Draft 0.5.1

Contents

We would appreciate any comments you have regarding errors, improvements, or additions, the latter consistent with the goal of keeping these notes rather short.

1. Introduction

These notes are intended as an introduction to Maple, a computer language for helping with mathematical calculations. With it we can write out numerical expressions like this:

```
> (1 + 2)*(6 + 7) - 12/7;
```

and Maple will will respond with:

$$261/7$$

We can use all the standard mathematical functions provided that we know (or can guess) their names:

```
> sin(sin(20)) + arctan(sqrt(2)) + log(abs(-3)) + 2^3.1 + exp(1.1);
```

$$\sin(\sin(20)) + \arctan(2^{1/2}) + \log(3) + 2^{3.1} + 3.004166024$$

Whenever possible Maple tries to compute exact quantities (1.414213562 is, of course, an approximation to $\sqrt{2}$). To get results in decimal (floating point) form, we use the function *evalf*:

```
> evalf(");
```

```
14.42359058
```

The quotation mark stands for the result of the immediately preceding computation. We can also do algebra:

```
> (a+b)^2;
```

$$(a+b)^2$$

```
> expand(");
```

$$a^2 + 2ab + b^2$$

```
> factor(");
```

$$(a + b)^2$$

Values can be assigned to variables, and substitions made:

```
> p := ":  b := 1: p;
```

$$(a + 1)^2$$

We use colons instead of semicolons to supress the display of expressions like p and $b := 1$ — we already know what they are. To expand the result of the last computation, we do this:

```
> expand(");
```

$$a^2 + 2a + 1$$

Variables can be returned to their original symbolic (unasssigned) state:

```
> b := 'b':
> p;
```

$$(a + b)^2$$

Note that in the above examples, the symbol used for assigning values, sometimes called "gets", is different from the equal sign "=." Note also how we "unassign" a variable, so that it becomes a pure symbol once again.

Maple knows much more than arithmetic and algebra — with it you can define new functions and plot them, solve equations in one or more unknowns, do calculus (compute derivatives, integrals, and solve some differential equations), work with vectors and matrices, etc. These facilities will help us to concentrate on the ideas and applications of mathematics rather than its mechanics. We will be able to look at more (and more complicated) examples without losing our way (and our enthusiasm) in lengthy but routine calculations.

The best way to learn Maple is by using it. For this reason the notes below are intended to be read while at the computer. You should try each example as you read it, to see that it really does what it should. Sometimes Maple will misbehave. Often this is because you mistyped something — perhaps a function name is misspelled, or a semicolon is missing, or you used = instead of :=. Sometimes it is because you and Maple have slightly different ideas about the meaning of what you typed in. These errors are more difficult to deal with. However, as you develop an understanding of what you are doing, they will be less common and more easily solved.

We have included exercises at the end of each section. You are encouraged to try some of these, as well as to think up your own. Most important, you should begin, starting now, to use Maple in your own work.

For more information about Maple see the references in the annotated bibliography at the end of these notes.

2. Basics

To start Maple, type the command *Maple* or *xMaple* on the command line. *XMaple*, if available, will give better graphics. Leave Maple by typing *quit*. Below is a sample session, in which we do some simple arithmetic:

```
% Maple
> 2 + 2;
                                4

> quit
%
```

Note the Maple's command prompt ">". This is where you type what you want Maple to compute. Once you have typed your "command", press the return key to tell Maple to execute it. Commands must be properly punctuated: they end with either a colon or a semicolon. For now we will use semicolons. If you forget to end a command with a semicolon you can just type it in on the next line. It is the semicolon and not the return key that tells Maple you have finished a command. Try starting up Maple now and compute $2+2$ as in the example (this is your first exercise). Then try something more difficult, maybe an expression with parentheses.

Note: Sometimes Maple will take too long to do a computation, in which case you need to interrupt it. There are different ways to do this on different systems. For plain *Maple* on a Unix system press ctrl-C. This is a universal way to abort a running program. For *xMaple* press the *interrupt* button in the Maple window.

Addition, subtraction, etc., are standard, and parentheses are used in the usual way. Be sure to use an asterix for multiplication: $2*3=6$. To take powers, use the caret symbol:

```
> 2^10;
```

Note that Maple carries out exact rational arithmetic. Thus, if you compute 31416/10000, which is a way of representing an approximate value of π, Maple gives a result of 3927/1250. To get a decimal result, you then have to say *evalf*("). *Evalf* is a function which turns whatever you give it into a floating point or decimal number. The quote symbol stands for the result of the immediately preceding computations. Here are some examples (remember — do them now):

```
> 123/111;
> evalf(");
> 129/127;
> evalf(", 20);
> Pi;
> evalf(Pi,100);
```

Note that *evalf* can take a second, optional input which determines the precision of the output. (Inputs, which in mathematics are called independent variables, are known as *arguments* in the computer jargon). Another way to control the precision used by *evalf* is to set the `Digits` variable, e.g., `Digits := 2` for a low precision.

Warning: Some commands, like *fsolve*, which is used to solve equations, will give strange answers if `Digits` is set too low.

Functions work in Maple as they do in mathematics. For example, to compute $2+14*8+*-14*-\sqrt{\sin(1)}$, say

```
> 2 + 14*8 + abs(-14)-16/4 - sqrt(sin(1));
> evalf(");
```

To see whether trig functions are computed in radians or degrees, we do an experiment:

```
> evalf(sin(1));
```

What do you conclude? Here is another thing to try:

```
> arctan(tan(x))
```

The result is x, as expected, since $arctan$ undoes what tan does: these two functions are inverse to each other.

What we just did illustrates an important point: you can often guess the correct form of a Maple expression even if you haven't looked it up in the manual. Thus, $tan(1)$ does indeed compute the tangent, $artan(1)$ computes the arctangent, and

```
> 20!;
```

computes a factorial. If you need additional information on something, e.g., arctan, try this:

```
> ?arctan
```

Note that there is no semicolon this time.

Let's now take a big step forward and begin using variables. We assign integer values to variables a and b and then do some computations:

```
> a:=25;
> b:=15;
> (a/b)^17;
```

Note the special symbol used to assign a value to a variable. You can read $a := 25$ as "a gets 25". To get a decimal approximation for a result just computed, recall that we use $evalf$. Another way of getting decimal results is to assign decimal values to at least one of the variables a and b. This way Maple knows that all computations are to be done in decimal form:

```
> a:=15.0;  # or 15.00 or just 15. (etc.)
> b:=25;
> (a/b)^17;
```

The idea here is that if one of the numbers in the expression is in decimal format, Maple computes and presents the result in the same format. Note the comment in the first line of our examples. Comments always begin with the character #, which just tells Maple to ignore the # and everything which comes after it, up to the end of the line.

Exercises

1. A rational number — the quotient of two integers — has a decimal expansion which either terminates or eventually settles down to a repeating pattern. (Consider the expansion of 1/3, for example). Use Maple to find the repeating part of 23/11, 23/13, and 713/211. Try also
$$\frac{524767338531}{3030303030303}$$
and
$$\frac{192414574797}{1111111111111}$$
2. For the sake of argument, let $P = 4$ be the human population of the earth today, measured in billions. Also for the sake of argument, assume that the annual growth rate is $r = .04$ (four percent). Then the population n years later will be
$$Q := (1+r)^n P.$$
Compute the population in billions for $n = 10, 25, 100, 1000$. What conclusions can you draw? Redo your calculations with different values of r. (Use variables in this problem).
3. Warning: Before you try this problem you may want to review the procedure for interrupting a computation that takes too long. (see Note at the beginning of this section.)

   Maple has a function called *ifactor* which factors integers. It takes one integer as input and produces the factorization as output. Try factoring some reasonably large numbers, like 123456789. Try also some really large numbers like $2^{89}-1$ and $2^{211}-1$. You might want to evaluate these before factoring them so that you can enjoy the decimal expansions. Comment on the time necessary to perform these calculations. How might it depend on the size of the integers one tries to factor?

   There is a function for determining whether a number is prime without finding its factors, as in $isprime(2^{211}-1)$. Try this function on the numbers above. How does its speed compare with *ifactor*?
4. Maple also has a function called *nextprime*. It takes an integer n as input and returns the smallest prime $p > n$ as output. Use it to find some very large prime numbers.

3. Algebraic expressions

Let us now try working with algebraic expressions. We begin by defining a binomial:

```
> p := x+y;
```

Note that Maple does not expand automatically, as you will see when you try these:

```
> (x + 2*y)*p; # Note that Maple does not understand 2y--the
>              # asterisk is necessary.
> p*p*p;
> p^15;
```

Thus, to get the expanded form, we do this:

```
> expand(p^15);
```

To continue working, we store the expanded form in a variable *pp*:

```
> pp:= ";
```

Note how much space the expanded form takes. If we don't want it to appear on the screen, remember we use a colon instead of a semicolon:

```
> pp:
```

To get one of the coefficients of pp, we do this:

```
> coeff(pp,x,12);
```

The reverse of expansion is factorization:

```
> factor(pp);
```

Notice what happens when we try to get the coefficient of something that is not expanded:

```
> coeff(",x,12);
```

In order to be sure that we get the coefficient as we want it, we can tell Maple to collect everything in powers of x:

```
> collect(pp,x): coeff(",x,12);
```

As we learned in the last section, variables can be assigned values. Thus, we can do the following:

```
> p := (x+y)^3;
> expand(");
> y := 1;
> p;
> expand(");
```

Our substitution works as expected. To get back our original polynomial we "unassign" y:

```
> y := 'y';
> p;
```

Be careful about the kind of quotes you use here — there are three different kinds on the keyboard. Note that assignments don't have to be numerical, and in fact can be anything you like. Experiment with $y := a + b$.

Exercises

1. Compute $(x + 1)(x + 2)(x + 3)(x + 4)$ in expanded form, then find the coefficient of x.
2. Factor (and so find the roots of) $x^4 + 8x^3 - 10x^2 - 104x + 105$.
3. Find the coefficient of x^{10} in $p := (x + ay)^{14}$. For this problem use something like $collect(p, x)$.

4. Graphing

Maple can plot graphs for you:

```
> plot( sin(x*3), x = -Pi..Pi );
```

It is also possible to plot more than one curve at a time:

```
> plot( { sin(x), sin(3*x)/3 }, x = -Pi..Pi );
```

The first argument to *plot* is a set of expressions to be graphed. For more information, use

```
> ?plot
```

Parametric plots

For parametric plots, given by $x = f(t)$, $y = g(t)$, imitate the following, which is an attempt to plot an ellipse:

```
> plot( [2*cos(t), sin(t), 0..2*Pi] );
```

You will notice that it looks like a circle because Maple did not choose to use the same horizontal and vertical scales. To fix this, we do the following:

```
> plot( [2*cos(t), sin(t), 0..2*Pi], -2..2, -2..2 );
```

You might enjoy plotting *Bowditch curves*, named for Nathaniel Bowditch (1815), and also called *Lissajous curves*. These are defined parametrically by $x = a\sin(nt + c)$, $y = b\sin(t)$. Here is a nice one:

```
> plot( [ sin(3*t), sin(2*t), 0..2*Pi ] );
```

Polar plots

For plots of curves given parametrically in polar coordinates by $r = f(t)$, $\theta = g(t)$, use the form

```
plot( [ f(t), g(t), a..b ], coords = polar
```

For example, let's plot the cardioid, a heart-shaped figure defined by $r = 2(1 + \cos\theta)$. To do this we have to parametrize the cardioid, which we do by setting $\theta = t$, so that $r = 2(1 + \cos t)$. In other words, $f(t) = 2(1 + \cos t$, $g(t) = t$:

```
> plot( [ 2*(1+cos(t)), t, 0..2*Pi ], coords = polar );
```

Here are some other things to try: The *logarithmic spiral*, given by $r = ke^{a\theta}$, and the spirals defined by $r^m = a^m\theta$. For example, let's graph the spiral with $m = -2$, $a = 1$, which has equation $r^{-2} = \theta$. One possible parametric representation is $r = t$, $\theta = g(t) = t^2$, which translates into Maple as

```
>  plot( [ t, t^2, 0..4 ], coords = polar );
```

More information

For more information type

```
> ?plot
```

There you will find reference to 3-D plots, parametric plots, polar plots, etc. To find out about them, use queries like

```
> ?plot3d
> ?plot[parametric]
> ?plot[polar]
```

etc.

Printing plots

This is easy or difficult, depending on the design of the system. Ideally, there is a "print" command in some menu or in a button in the plot window (Macs, NeXT's and some other Unix systems). If you are using an X-windows Unix system that does not have these features, proceed as follows.

1. Play with your plot until it looks right.
2. Type `plotsetup(postscript);`
3. Redo the plot command, e.g., `plot( sin(x), x=-1..1 );` The plot will not appear on the screen, but instead will be sent to a postscript file called `postscript.out`.
4. Print the file `postscript.out`. You can do this by typing `print -p postscript.out`. For more information on the print command, type `print`.
5. To go back to on-screen plotting, type `plotsetup(x11)` .

Exercises

1. Make a graph of $sin(x) + sin(3x)/3$ on the interval $[-\pi, \pi]$.
2. Make a graph of $sin(x)$ and $x - x^3/6$ on the interval $[-\pi, \pi]$. Both curves should appear on the same plot. The polynomial $x - x^3/6$ is a Taylor approximation to $sin(x)$. For what could it be used?
3. Plot the cardiod, $r = 2(1 + \cos(\theta)$, $\theta \in [0, 2 * \pi]$.

5. Roots and equations

Maple knows how something about how to find the roots of equations. Let p be the polynomial $18x^4 + 69x^3 - 40x^2 - 124x - 48$. If we factor it, we get $(2x - 3)(x + 4)(3x + 2)^2$, from which we can find the roots. A more direct way, however, is this:

```
> p := 18*x^4 + 69*x^3 - 40*x^2 - 124*x - 48;
> roots(p);
```

The result gives the roots and their multplicities ($-2/3$ is a root of multiplicity 2, the other roots are simple). We could also do this:

```
> solve( p = 0 );  # solve( p ); also works
```

This form is convenient because we can then solve other equations:

```
> solve( p = 1 );
```

If you did this you might have been surprised: the answer is given in exact form and is rather hard to read. To understand it, let us begin by assigning all the solutions to a variable s:

```
> s := ";
```

We can then pick out the individual solutions that we want to study (there are four of them, since p has degree four)

```
> s[1];
```

Then, to get a floating point answer we do this:

```
> evalf( s[1] );
```

Note that the answer is a complex number. You could also have tried `simplify( s[1] )`. This will give you an exact answer in what Maple thinks is the simplest possible form.

If you want all the solutions in floating point form at once, you can say this:

```
> map( evalf, [s] );
```

In the previous expression $[s]$ is a *list*, namely a list of solutions. Whenever you give *map* a function and a list, it applies that function to each member of the list. The result is a list of function values. For example, to get a short list of square roots, we can do this:

```
> map( sqrt, [1, 2, 3, 4, 5] );
> map(evalf, ");
```

Here is one last example:

```
> Digits := 2: map( evalf@sqrt, [1, 2, 3, 4, 5] );
```

Note that we set the precision before doing the calculation. The really new part is the construction $evalf@sqrt$, which stands for the function obtained by composing $evalf$ and $sqrt$. The mathematical notation for composition is $evalf \circ sqrt$, and it works like this:

$$evalf \circ sqrt(x) = evalf(sqrt(x))$$

If the equation you give Maple contains floating point numbers, then you will automatically get an answer in floating point form:

```
p := 18*x^4 + 69*x^3 - 40*x^2 - 124*x - 48.0;
```

Numerical Methods

Another way to solve equations is with the function *fsolve*. It gives decimal approximations to the solutions of an equation, but, unlike *solve*, looks only for real solutions unless we explicitly request that it find complex solutions. A word of warning: *fsolve* sometimes misses roots. This is because it uses numerical techniques rather than symbolic ones. Because these usually depend on the way we phrase the problem, we must use *fsolve* with caution and a certain amount of skepticism. Let's look at some examples. We will try to solve $x^2+x+1=0$, which has complex roots:

```
> Digits := 2;
> q := x^2+x+1;
> fsolve( q );
```

The nonsensical answer is because `Digits`, which determines the precision with which *fsolve* looks for roots, is set too low. We try it with a better value:

```
> Digits := 8: fsolve( q );
```

That didn't work either, because it turns out we must instruct *fsolve* to look for complex roots:

```
> fsolve( q, x, complex);
```

In this last example, we have to specify that x is a variable because the syntax for *fsolve* is

$$solve(\ equation,\ variables,\ options),$$

We can omit the options, or we can omit both the variables and the options, but if so, we accept the consequences. Our previous examples worked because Maple could guess what the variables (the unknowns) were. For more about *fsolve*, including its options, do

```
> ?fsolve
```

Moral: Don't blindly trust what the computer gives you as an answer. It may be nonsense. You can usually tell by thinking about the problem. For example, we know that a quadratic polynomial has two roots, counting multiplicities. So if it gives you none, something is wrong.

It is important to be aware that Maple's solving functions frequently have trouble with square roots:

```
> solve(sqrt(x)=-1);
> solve(sqrt(2*x+9)=x-13,x);
> fsolve(sqrt(2*x+9)=x-13,x);
```

If we check the solutions returned by Maple we see that there is a problem here. By squaring both sides of the equation above we can rephrase the second question without square roots:

```
> fsolve(2*x+9=(x-13)^2,x);
```

Note that now we are thinking about two related equations at the same time: $\sqrt{2x+9}=x-13$ and $\sqrt{2x+9}=-(x-13)$. Maple has no problem solving this equation but we must check that the solutions returned are really solutions to the question we asked originally.

Note that *fsolve* can find solve nonpolynomial equations:

```
> fsolve( sin(x) + x = 0.2 )
```

Of course, the solutions it does find are approximate ones, where the precision is controlled by `Digits`. To get a feel for where these roots might be, it helps to make a graph. In our example, the root is the place where the graph of $y = \sin x + x$ crosses the line $y = 0.2$. We can visualize this by making a little sketch or by typing something like

```
> plot( { sin(x) + x, 0.2 }, x = -4..4);
```

You can play with the range for x to home in on the place where the action is (near the root).

The command *solve* sometimes works for inequalities and at other times Maple complains that it has not yet learned to solve inequalities:

```
> solve(x^2-3*x>=0);
> solve(sqrt(x^2-3*x)-x=0>=0);
```

We should mention that you can use symbolic coefficients as well as numeric ones:

```
> solve( x^2 + a*x + 1, x);
> solve( x^2 + a*x + 1, a);
```

Note that since two symbols appear, we have to tell Maple, which does not have telepathic powers, which one is the unknown. You will get some kind of answer for polynomials of degree less than five. For degree five and higher there is (by a theorem of Galois) no algebraic formula for the roots in terms of the coefficients. The quadratic formula was known to the ancients, e.g., the Babylonians, while the cubic and quartic cases had to wait until the Renaissance for solution. Galois' work, done around 1830, relied on the theory of groups.

To end our discussion of equation solving, let us consider systems, e.g.,

$$\begin{aligned} 2x + 3y &= 1 \\ 3x + 2y &= 2 \end{aligned}$$

To solve it using Maple, we say this:

```
> solve( { 2*x + 3*y = 1, 3*x + 2*y = 2 } );
```

Note that we write the system as a *set* of equations. Sets consist of a sequence of elements separated by commas, the whole thing enclosed in curly braces. If we had a symbolic parameter, we would have to specify the variables:

```
> solve( { a*x + 3*y = 1, 3*x + 2*y = 2 }, { x, y } );
```

Of course, we are not limited to linear equations.

```
> solve( { 2*x + 3*y = 1, x^2 + 3*y^2 = 2 }, { x, y} );
> evalf(");
```

We get one solution. Is this reasonable? How many do we expect on algebraic grounds? What does geometry tell us about this problem? Can a picture help? (As we saw in the previous section, Maple can help you with pictures).

Exercises

1. Solve $x^2 + 2 * x + 1 = a$ for $a = 0,\ 0.1, -0.1$. As we change the coefficients by a small amount, how do the roots change? (By a small amount or a large amount?) Try to explain what happens by using appropriate graphs, either sketched by hand or done with Maple.
2. Solve $x^3 - 27 = 0$, $x^3 + x - 27 = 0$, and comment on the relation between the solutions in the two cases. Does a small change in the coefficients lead to a small change in the roots? Give a graphic explanation if possible.
3. Solve $x^n + a * x + 1 = 0$ for $n = 2, 3, 4, 5$.
4. Solve $\tan x + x = 0.2$. How is the solution found different from that of $\sin x + x = 0.2$? Does this have a graphic explanation? (Do a plot — perhaps with both $y = \tan x + x$ and $y = \sin x + x$ on the same axes). Are there other solutions?

6. Functions

With Maple, just as in mathematics, one can define new functions. For example, to define the function

$$p1(x) = (2x - 3)(x + 4)(3x + 2)^2$$

we say this:

```
> p1 := x -> (2*x-3)*(x+4)*(3*x+2)^2;
```

Think of the symbol $->$ in the definition as an arrow: x "goes to" the expression $(2x-3)(x+4)(3x+2)^2$. We can now proceed with the usual kind of computation, e.g.,

```
> p1(-2);
> p1(y);
> p1(a+b);
```

If you don't like the format of some of the results that $p1(*)$ gives you, try *expand*(").

It is important to keep in mind that functions and expressions are different kinds of mathematical objects. Mathematicians know this, and so does Maple: compare the results of the following.

```
> p1;        # function
> p1(x);     # expression
> p1(y);     # expression
> p1(3);     # expresssion
```

As further proof, let us try to "factor $p1$":

```
> factor(p1);
> factor(p1(x));
> factor(p1(y));
```

Which of these worked? Does the *factor* function work on functions or expressions?

We can define functions of many variables just as easily as we can functions of a single variable:

```
> f := (x,y) -> exp(-x)*sin(x);
> f(1, 2);
> g := (x,y) -> A*exp(-k*x)*sin(w*x);
> g(1,1);
> A := 2: k := 1.4: g(1,1);
> w := 3.7: g(1,1);
> A := 'A'; g(1,1);
```

Exercises

1. Define a function *inv* which sends a number x to its reciprocal; define another function f which sends x to $\frac{\sin x}{x}$. Study the behavior of f on the sequence $x = 1, 1/2, 1/3, ...1/10$ using *map* and *inv*.
2. Define a function $log(x, b)$ which computes the logarithm of of x to the base b.

7. Loops

To do a sequence of repetitive calculations on a computer, you use a *loop*. Let us try this to print out a short table of square roots:

```
> for i from 1 to 10 do sqrt(i) od;
```

We modify this slightly to get answers in floating point form:

```
> Digits := 4: for i from 1 to 10 do evalf(sqrt(i)) od;
```

Results can be accumulated inside a loop. We illustrate this by computing the sum of the first ten cubes,

$$1^3 + 2^3 + \cdots + 10^3$$

```
> S := 0:
  for i from 1 to 10
    do
      S := S + i^3
    od;
```

If you don't want to see all the intermediate results, do this:

```
>  S := 0:
  for i from 1 to 10
    do
      S := S + i^3
    od:
  S;
```

For more about loops, use

```
?for
```

There we find out about the *while* construction. As an example of how to use it we find the number of terms such that the sum

$$H = 1 + 1/2 + 1/3 + \cdots + 1/n$$

exceeds a fixed limit L:

```
> H := 0;
  n := 0;
  L := 6;
  while (H < L)
    do
      n := n+1;
      H := H + 1/n;
    od:
  n;
```

Note that the loop body (the part between `do` and `od`) can contain more than one statement.

Exercises

1. In the previous example we looked at the sum
$$H = 1 + 1/2 + 1/3 + \cdots + 1/n.$$
We know from calculus that as we take larger and larger values of n this sum gets arbitrarily large. If you choose a number L then no matter how large L is I can find a number n so that the sum H is bigger than your number L. In the example above we computed how large n must be if you choose $L = 6$. Find the value of n needed so that H will be bigger than L if $L = 16, 26, 36, 46$.
2. Write a loop that computes the value of $m(n) = n^2 + n + 41$ for n from 0 to 25. In each case use the function *isprime* to check that $m(n)$ is prime. Now extend your loop to compute the values of $m(n)$ for n from 0 to 100. What pattern do you see?
3. In the previous exercise we might easily be mislead into the conclusion that $n^2 + n + 41$ will produce a list of larger and larger prime numbers as we let n increase since this was true for the first 40 values of n. This illustrates the danger of drawing a general conclusion from checking a few cases. Consider the sequence of odd counting numbers:
$$1, \quad 3, 5, \quad 7, 9, 11, \quad 13, 15, 17, 19, \quad 21, 23, 25, 27, 29, \quad \cdots.$$
Notice that the first term is just 1^3. The sum of the next two terms is 8 or 2^3. The sum of the next three terms is 27 or 3^3. The sum of the next four terms is 64 or 4^3. Does this pattern continue? Use Maple to check that this pattern continues up to at least 15^3. How could one be sure that this pattern continues forever?

8. Calculus

Maple also does calculus. With it you can compute derivatives and integrals, both definite and indefinite, find Taylor series, and even solve differential equations (or systems of them).

Differentiation

Suppose we are given a function g defined by

```
> g := x -> x^3 - 2*x + 9;
```

It seem natural to ask for its derivative. but we must be careful how we phrase our command to Maple:

```
> diff( g, x ); # Maple interprets g as a constant.
> diff( g(x), x ); # Now Maple agrees with us on the meaning of g
```

The subtlety here is that *diff* operates on expressions, not on functions: g is a function while $g(x)$ is an expression. Thus, we could also say

```
> diff( x^3 - 2*x + 9, x );
```

To compute the second derivative (the derivative of the derivative) we do this:

```
> diff( x^3 - 2*x + 9, x, x );
```

An alternative way is to say

```
> diff( x^3 - 2*x + 9, x$2 );
```

This works because the expression $x\$2$ gets translated as the sequence x, x. By analogy, $x\$3$ means x, x, x, so that

```
> diff( x^3 - 2*x + 9, x$3 );
```

computes the third derivative — the derivative of the derivative of the derivative.

We can also compute partial derivatives:

```
> q:=sin( x*y );  # expression, not function
> diff( q, x );
> diff( p/(q^2), x, y );
```

If it is not obvious what this expression is compare it to the following:

```
> diff( p/(q^2), x ); diff(", y);
```

Integration

To find the indefinite integral of a function, we use *int*. For example, to compute

$$\int x^n \, dx$$

we do this:

```
> int( x^n, x );
```

Here are some more examples:

```
> int( 1/x, x );
> int( sin(x), x );
> int( x*sin(x), x );   # Maple knows integration by parts
> int( sin(3*x + 1), x );  # And substitution
> int( (x^2 - 1)/(x^2 + 5*x + 4), x );  # And partial fractions
> int( sqrt(1-x^3), x );  # but it can't do everything
```

Of course, we can also compute definite integrals, e.g.,

$$\int_0^1 x^3 \, dx,$$

which we do this way:

```
> int( x^3, x = 0..1 );
```

The only difference is that we give the interval of integration. You should be aware of the fact that we can compute definite integrals numerically even when there is no integral in closed form. For example, consider

$$\int_0^1 \sqrt{1 - x^3} \, dx$$

The corresponding indefinite integral cannot be expressed in terms of elementary functions — the functions you make by doing the usual operations of algebra, together with extraction of roots on polynomials, logs, exponentials, trig functions, and their inverses. Thus $e^{\arcsin x^2}$ is an elementary function. However, we can certainly find an approximate value for the definite integral above, accurate to any precision we want:

```
> int( sqrt(1 - x^3), x = 0..1 );
> evalf(");
```

Notice that (unlike the previous definite integral) we had to use *evalf* to force numerical evaluation.

The fundamental theorem of calculus

Now let's look at the relationship between differentiation and integration.

```
> diff( x^2/(x+1), x );
> int(", x);
```

We would expect these to match but they do not. Have we found an error? Let's investigate further.

```
> d:=diff( x^2/(x+1), x); simplify( d );
> i:=int( d, x ); simplify( i );
```

Now perhaps it is clear that there is no problem. We have verified in one case that integrating the derivative of a function gives the original function back again (up to a constant). Said in fewer words, "integration undoes what differention does." **Question**: does differentiation undoe what integration does?

Other features

We can compute limits:

```
> limit( g(x), x=infinity );
> limit( g(x), x=-infinity );
> limit( sin(x)/x, x=0 );  # limit
```

We can also compute taylor series and do sums:

```
> taylor( exp(x), x=0, 4 );  # Taylor series of order 4
> sum(i^2, i=1..100);  # sum of series
> sum(x^n, n=1..infinity);
```

Differential equations

To solve the diffferential equation

$$y'' + y = 0$$

we do this:

```
> deq:=diff( y(x),x ,x ) + y(x) =0;
> dsolve(deq, y(x) );
```

For more information, consult `?dsolve`. There you will find out how to solve initial value problems, systems, do series, numerical, and Laplace transform solutions.

There is a special package called ODE which allows you to plot line fields, vector fields integral curves, etc. This package is courtesy of Daniel Schwalbe. The main routines are

```
fieldplot:   create a direction field plot for a first order ODE

phaseplot:   create a phase plot with flow lines for two first order ODE's

irsteuler:   solves a first order ODE using Eulers method

impeuler:    solves a first order ODE using the "improved Euler method"

rungekutta:  numerical solution to a system of first order ODEs using
             a 4'th order Runge-Kutta method.  sion of rungekutta
```

To use the package (which resides in /u/cl/maple), type

```
> read `ODE`;  # those are supposed to be backquotes, even
               # if they don't show as backquotes
```

Once this is done you can get more information about the above routines by typing ?*fieldplot*, ?*phaseplot*, etc. Pay special heed to the examples at the end of each help screen. They may help you more than the general description does.

9. Vectors and Matrices

Maple can solve systems of linear equations and perform most of the computations you need to do with vectors and matrices. You can add, multiply and invert matrices, compute determinants, do row reduction, find eigenvalues and eigenvectors, etc.

Linear equations

It is easy to solve systems of linear equations:

```
> solve( { 2*x + 3*y = 1, 3*x + 2*y = 2 }, {x, y} );
> s := ";
> s[1];
```

In the second line we assign the solutions to a variable s, and in third line we display the first component of the solution set, which is its x-value.

Basic vector and matrix operations

To do more linear algebra, we load the linear algebra package:

```
> with(linalg);  # load the package
```

As the package loads, the functions defined in it are displayed. Next, we define and display a vector:

```
> v := vector( [1, -1] );
> v;  # try to display it
> print(v);  # try to display it another way
```

We define another vector and then do some computations — we add v and w, and we compute the dot product $v \cdot w$:

```
> w := vector( [1, 1] ):
> add(v,w);
> dotprod(v,w);
```

Let's now define two matrices

```
> A := matrix( [[2, 3], [1, 2]] );
> B := matrix(  [[1, 1], [0, 1]] );
```

To display them we use *print*:

```
> print(A); print(B);
```

Next, we do some computations:

```
> add(A,B);
> multiply(A,B);
```

We check to see if $AB = BA$:

```
> multiply(B,A);
```

It is tiresome to type "multiply" all the time, so we introduce an alias:

```
> alias(m=multiply);
```

We could also have typed `m := multiply` , but aliasing is more efficient. (To unalias, if necessary, we type `alias(m=m)`). Now we can do the following to multiply a matrix times a vector:

```
> m(A,v); # matrix times vector
```

Note that Maple treats vectors as columns, even though it displays them as rows (confusing, isn't it?):

```
> m(v,A); # doesn't work: Maple treats our vector as a column
> m(transpose(v), A); # transpose row to column
```

The latter works, but note that result is a column vector, and so is displayed in a funny way.

We can also work with determinants (square matrices only!):

```
> det(A);
```

Of course, we can also use symbolic matrices:

```
> A := matrix( [[a, b], [c, d]] );
> det(A);
> a := 2; print(A);
```

Maple didn't reflect the definition `a := 2`, but it really is there, as we see when we do computations:

```
> det(A);
> a := 'a';  # unassign value
> det(A);  # check determinant
```

If the determinant of a square matrix is nonzero, the matrix has an inverse A^{-1}. This is a matrix with the property

$$AA^{-1} = A^{-1}A = I$$

where I is the $n \times n$ identity matrix (one's on the diagonal, zeroes elsewhere). The identity matrix satisfies

$$Ix = x$$

for any vector x. The inverse is useful in solving equations:

$$\begin{aligned} Ax &= b \\ A^{-1}Ax &= A^{-1}b \\ Ix &= A^{-1}b \\ x &= A^{-1}b \end{aligned}$$

Here is an example:

```
> A := randmatrix(4, 4, entries = rand(-2..2)); # create random matrix
> B := inverse(A);
> multiply(A,B); multiply(B,A);  # check result
```

It is an interesting fact that a matrix with random entries is invertible with probability one: the odds favor generation of invertible matrices. Note that you can also say

```
> A := randmatrix(4, 4);
```

In that case the entries are random integers in the range $[-99,\ 99]$

Row operations and row reduction

The functions *addrow*, *mulrow*, *swaprow* allow you to do row reduction. There are companion functions for column operations. To row reduce a matrix, use *gausselim* or *gaussjord*. To illustrate these, we work out an example. It illustrates the standard procedure for computing the inverse of a matrix by row reduction.

```
> A := matrix( [[3, 2, 2], [-3, -1, -2], [1, 1, 1]] );
> B := diag(1,1,1); # make 3x3 identity matrix
> C := augment(A,B); # make 3x6 matrix with A as left block, B as right block
> C := swaprow(C, 1, 3); # get a one in the (1,1) position
> C := addrow(C, 1, 2, 3); # start cleaning column 1
> C := addrow(C, 1, 3, -3); # finish cleaning column 1
> C := mulrow(C, 2, 1/2); # get a one in the (2,2) position
> C:= addrow(C, 1, 3, -3); # clean up lower part of column 2
> C := addrow(C, 2, 3, 3); # get a one in the (3,3) position
> C := addrow(C, 3, 2, -1/2); # start cleaning column 3
> C := addrow(C, 3, 1, -1); # finish cleaning column 3
> C := addrow(C, 2, 1, -1); # finish cleaning column 2
> D := submatrix(C, 1..3, 4..6); # extract right-hand block
> multiply(A,D); # check to see if D is inverse of A
```

Here is one more example:

```
> A := randmatrix( 3, 7, entries = rand(0..4) );
> gausselim(A);
> gaussjor(A);
```

Notice that *gausselim* gives the row echelon form, while *gaussjord* gives the completely reduced row echelon form.

Eigenvalues and eigenvectors

There will be more in this space later. For now consult *?eigenvals*, *?eigenvectors*, and *?charpoly*. However, we would like to mention now one numerical method which is quite easy to implement in Maple. Let A be a matrix, x a vector. Form the product $y = Ax$, then replace x by y/y, where y is the norm, or length. Repeat this process many times and call the resulting vector z. Then with probability one the z is very nearly an eigenvector of A, provided that A has eigenvectors. In fact, it approximates the eigenvector associated with the eigenvalue of largest absolute value. That eigenvalue (the dominant eigenvalue) is approximately

$$\lambda = \frac{Az \cdot z}{z \cdot z},$$

where the dot is the dot product. Here is a definition of a Maple function which does the necessary work:

```
deigenvector := proc(A,x,n)
 x[1] := x[1]/1.0; # force floating point calculations
 for i from 1 to n do
   x := multiply(A,x);
   x := scalarmul(x, 1/norm(x) );
   print(i, x );
 od;
 x;
end;
deigenval := proc(A,x)
  dotprod( multiply( A, x ), x )/dotprod( x, x );
end;
```

Here is an example:

```
> with(linalg):
> A := matrix( [[3,1], [1,2]] );
> v := vector( [1, 1]);
> w := deigenvector( A, v, 10);
> deigenval( A, w );
```

10. Saving your work

You can save your work to a file using the function "save", and you can read it from a file using "read". Type

```
> ?save
> ?read
```

for instructions on how to use these.

11. Getting help and more info

We can get a lot of info by typing commands like:

```
> ?
> ?intro
> ?library
> ?fsolve
> ?taylor
> ?packages
```

If you have tried these commands you see that they give a perhaps overwhelming amount of information. Much of it will be incomprehensible for you at this point. Each help entry ends with a list of examples and related topics. These are usually the most helpful and comprehensible parts.

Bibliography

1. Bruce W. Char et al., *First Leaves: A Tutorial Introduction to Maple*, WATCOM Publications 1988, pp. 115.
2. Bruce W. Char et al., *Maple Reference Manual*, 5th Edition, WATCOM Publications 1988, pp. 403.