```
> restart;
```

MATH 4503/6503 Worksheet 1: An introduction to Maple

Sanjeev S. Seahra

revised: February 2, 2014

This worksheet is meant to serve as a brief introduction to the Maple symbolic computation package. This is a powerful piece of softawre that have many applications outside the scope of this course. Here is a non-exclusive list of what I find it useful for:

- Originally, the package was developed with the purpose of performing (tedious) algebraic manipulation of complicated mathematical expressions. So, it can save you pages and pages of writing when trying to make sense of hideous equations, including finding their solutions. And it doesn't make mistakes with minus signs, or factors of 2.
- You can use the software to plot functions in two or three dimensions, construct movies, or plot datapoints.
- Maple knows calculus. That means it can *analytically* differentiate anything, integrate lots of things, and solve some differential equations.
- However, many of the integrals, DEs, and ordinary equations you will meet during your research will be too hard for Maple to solve in closed form. But all is not lost, since the package includes many facilities for constructing numeric solutions to problems.
- Lastly, but not least, Maple can act as a fully-functional programming language like FORTRAN or C.

I will attempt to give a brief introduction to some of these topics, but the best way to learn is to play with the software yourself, and ***use the help pages***.

Fundamentals

Basic syntax

Maple worksheets are broken up into individual components called execution groups. Here is an example of an execution group:

```
> sin(0.1);
```

$$0.09983341665 \quad (1.1.1)$$

The user input in this case is what appears in front of the > sign, while the output is the centred text. Notice that the user input here is terminated by a semi-colon "**;**". Whenever you want Maple to execute a command and print an answer, you must type in the command, end the line with a semi-colon, and press ENTER. In this case, I have asked Maple to calculate **sin(0.1)** and print out the answer. It is possible to have Maple execute a command and *not* display the answer by terminating the line with a colon "**:**" instead. For example, by typing in

```
> sin(0.1):
```

and pressing return, I get Maple to calculate **sin(0.1)**, but it does not display the result. One can have multiple commands in an execution group. For example,

```
> sin(0.1); cos(0.4); ln(0.2);
```

$$0.09983341665$$
$$0.9210609940$$
$$-1.609437912 \tag{1.1.2}$$

You can see that the output of each command is put on a separate line. For clarity, we may want to have each command in an execution group on its own line, which is accomplished by pressing SHIFT+ENTER after the colons or semi-colons. This looks like:

```
> sin(0.1);
  cos(0.4);
  ln(0.2);
```

$$0.09983341665$$
$$0.9210609940$$
$$-1.609437912 \tag{1.1.3}$$

Defining quantities

To define a quantity, you use the symbol "**:=**". That is, a colon followed by an equal sign. For example,

```
> a := sin(x^2);
  b := cos(x^2);
```

$$a := \sin(x^2)$$
$$b := \cos(x^2) \tag{1.2.1}$$

Now, whenever a appears again in the worksheet, Maple will automatically substitute in its definition.

```
> c := a^2 + b^2;
```

$$c := \sin(x^2)^2 + \cos(x^2)^2 \tag{1.2.2}$$

Notice here that c has been *recursively* defined in terms of a and b. As in most programming languages, you can re-define a quantity in terms of its current value:

```
> c := sqrt(c);
```

$$c := \sqrt{\sin(x^2)^2 + \cos(x^2)^2} \tag{1.2.3}$$

Sometimes, you want Maple to forget something you have previously defined. To "undefine" a, we enter in the following:

```
> a := 'a';
```

$$a := a \tag{1.2.4}$$

Now, whenever a appears, Maple will treat it as an unknown quantity. Finally, you can define a bunch of stuff at once using commas:

```
> theta,Theta,q := x,y,z;
  theta^2 + Theta^2 + q^2;
```

$$\theta, \Theta, q := x, y, z$$
$$x^2 + y^2 + z^2 \tag{1.2.5}$$

Multi-element objects: tables, sets, lists, etc...

```
> restart;
```

Several types of Maple objects can consist of collection of individual elements. The most rudimentary of these is an table, which we can create as follows:

```
> a[1] := 1;
  a[2] := 4;
  a[3] := 0;
  print(a);
```

$$a_1 := 1$$

$$a_2 := 4$$

$$a_3 := 0$$

$$\mathit{table}([1 = 1, 2 = 4, 3 = 0]) \tag{1.3.1}$$

The general rule is that components on multi-element objects are labeled by quantities in square brackets. Note that "indices" are not necessarily numeric:

```
> a := 'a':
  a[t] := r^2;
  a[r] := sin(omega*t);
  a[theta] := exp(t-r);
  a[phi] := L/r^2;
```

$$a_t := r^2$$

$$a_r := \sin(\omega\, t)$$

$$a_\theta := \mathrm{e}^{t - r}$$

$$a_\phi := \frac{L}{r^2} \tag{1.3.2}$$

Different types of data structures are sets and lists. Both of these are a one-dimensional collection of objects, and their chief difference is whether or not ordering is important. A list is a comma-delimited sequence of objects enclosed is square-brackets:

```
> a := 'a':
  a := [1,sin(x^2),int(f(q),q=-infinity..infinity),1];
  a[3];
```

$$a := \left[1, \sin(x^2), \int_{-\infty}^{\infty} f(q)\, \mathrm{d}q, 1\right]$$

$$\int_{-\infty}^{\infty} f(q)\, \mathrm{d}q \tag{1.3.3}$$

On the other hand, a set is a sequence of *distinct* elements where the order is not important. It is defined as a list, but with curly brackets:

```
> a := 'a':
  a := {1=x+(y+z)^3,0=sin(x*z/y),3,2,1,3};
```

$$a := \left\{1, 2, 3, 0 = \sin\left(\frac{x\, z}{y}\right), 1 = x + (y + z)^3\right\} \tag{1.3.4}$$

Notice that Maple did not preserve the ordering of elements given in the input, and that the duplicate entry **3** was deleted. Finally, you can define n-dimensional objects known as arrays. For

example, here is a 3-dimensional object:

```
> a := array(1..3,-5..5,0..2000);
  a[1,-5,1999] := Pi;
  a[0,0,0] := 5;
```

$$a := array(1..3, -5..5, 0..2000, [\ ])$$

$$a_{1, -5, 1999} := \pi$$

```
Error, 1st index, 0, smaller than lower array bound 1
```

The first line here defined a 3-dimensional array whose first index runs 1 to 3, second runs from -5 to 5 and last runs from 0 to 2000. In the third line, I tried to assign a value to a component with indices outside of these ranges, and hence generated an error.

Useful things to know

Special symbols

Certain symbols have reserved meanings. 3.14... is known as **Pi**, while Euler's constant is **gamma**. To evaluate expressions involving these to floating point accuracy, use **evalf**:

```
> a := [Pi/2,gamma];
  evalf(a);
```

$$a := \left[\frac{1}{2}\pi, \gamma\right]$$

$$[1.570796327, 0.5772156649] \quad \textbf{(1.4.1.1)}$$

We have already seen above how **infinity** refers to mathematical infinity. A capital **I** refers to the square root of minus 1; i.e. $\sqrt{-1}$:

```
> [I,I^2,I^3];
```

$$[I, -1, -I] \quad \textbf{(1.4.1.2)}$$

You can't assign values to any of these symbols:

```
> I := 2;
  Pi := 4;
  gamma := 7;
Error, illegal use of an object as a name
Error, attempting to assign to `Pi` which is protected.
Try declaring `local Pi`; see ?protect for details.
Error, attempting to assign to `gamma` which is protected.
Try declaring `local gamma`; see ?protect for details.
```

One thing that is **not** a special symbol is **e**; that is, **e** does not represent the base of the natural logarithm. Rather, **e** is just an ordinary variable:

```
> e := 2;
  e^7;
```

$$e := 2$$

$$128 \quad \textbf{(1.4.1.3)}$$

If you actually want the base of the natural logarithm, use the function **exp**:

```
> exp(1);
  evalf(exp(1));
```

$$\mathrm{e}$$

$$2.718281828 \quad \textbf{(1.4.1.4)}$$

Some "essential" commands

- Issuing a **restart** command undefines all variables, which makes it a useful command to have at the top of a worksheet
- Under the Edit menu, there is an ``execute entire worksheet'' item, which is self-explanatory
- The echo character is **%**, which inserts the last line of output into the current line of input. To wit:

```
> tanh(1/2);
  evalf(%);
```

$$\tanh\left(\frac{1}{2}\right)$$

$$0.4621171573 \tag{1.4.2.1}$$

- The comment character is **#**. When it appears on a line, the complier ignores everything after it
- On Windows and Linux, CTRL+J (CTRL+K) inserts a new execution group after (before) the position of the cursor
- On Macs, CMD+J (CMD+K) inserts a new execution group after (before) the position of the cursor
- Perhaps most importantly, a question mark followed by a topic opens a help-page (no colon/semi-colon needed). Try **?evalf**
- By default, Maple calcuates floating point numbers to 10 digit accuracy. If you want more, re-define **Digits**:

```
> evalf(exp(1));
  Digits := 30;
  evalf(exp(1));
  Digits := 10;
```

$$2.718281828$$

$$Digits := 30$$

$$2.71828182845904523536028747135$$

$$Digits := 10 \tag{1.4.2.2}$$

- Rule of thumb for Digits: you can set it up to 14 without sacrificing speed or memory, but more than that will increase the CPU time for numeric calculations.

Manipulating expressions

Simplification

```
> restart;
```

First and foremost, Maple is a symbolic computation language, so what it created to do is algebraically simplify complex expressions by applying various algorithms. The easiest way to access this functionality is to use the **simplify** command.

```
> expr := sin(x)^2+cos(x)^2;
  simplify(expr);
```

$$expr := \sin(x)^2 + \cos(x)^2$$

(2.1.1)

$$1 \quad \textbf{(2.1.1)}$$

In and of itself, **simplify** is a bit of a brute force command that might not give you what you want. The alternatives **expand** and **factor** perform more specific tasks:

```
> expr_1 := (a+c*b)^3*(a-c);
  simplify(expr_1);
  expr_2 := expand(expr_1);
  factor(expr_2);
```

$$expr_1 := (b\,c + a)^3\,(a - c)$$

$$(b\,c + a)^3\,(a - c)$$

$$expr_2 := a\,b^3\,c^3 - b^3\,c^4 + 3\,a^2\,b^2\,c^2 - 3\,a\,b^2\,c^3 + 3\,a^3\,b\,c - 3\,a^2\,b\,c^2 + a^4 - a^3\,c$$

$$(b\,c + a)^3\,(a - c) \quad \textbf{(2.1.2)}$$

The **collect** and **coeff** commands are convenient for the manipulation of polynomials like **expr_2** above:

```
> collect(expr_2,a);
  expr_3 := collect(expr_2,a,factor);
  coeff(expr_3,a^2);
```

$$a^4 + (3\,b\,c - c)\,a^3 + \left(3\,b^2\,c^2 - 3\,b\,c^2\right)a^2 + \left(b^3\,c^3 - 3\,b^2\,c^3\right)a - b^3\,c^4$$

$$expr_3 := a^4 + c\,(3\,b - 1)\,a^3 + 3\,b\,c^2\,(b - 1)\,a^2 + b^2\,c^3\,(b - 3)\,a - b^3\,c^4$$

$$3\,b\,c^2\,(b - 1) \quad \textbf{(2.1.3)}$$

The second command reads in words: ``rewrite **expr_2** such that each terms contains a single power of $a$, and then factor each coefficient in the sum," while the third command answers the question ``what is the coefficient of a^2 in **expr_3**?".

There are many simplification strategies/commands in Maple, and I can't review them all here. Have a look at **?simplify** and its sub-categories for more information. Here are a few tips:

- Often applying **expand** and **factor** sequentially to an expression gives more satisfactory results than just **simplify**
- by default, Maple assumes everything is complex-valued unless told otherwise. This can cause it to refuse to perform what look like trivial simplifications. One way around this is to use the **assume** command:

```
> x := 'x':
  expr := sqrt(x^2)/sqrt(-x^2);
  simplify(expr);
  assume(x,real);
  simplify(expr);
```

$$expr := \frac{\sqrt{x^2}}{\sqrt{-x^2}}$$

$$\frac{\mathrm{csgn}(x)\,x}{\sqrt{-x^2}}$$

$$-\mathrm{I} \quad \textbf{(2.1.4)}$$

The assume command will ensure that whenever Maple does a calcution involving x, it will assume

that x is real. If you would rather Maple just make the assumption for one calculation, then you can use **assuming**:

```
> simplify(cos(n*Pi));
  simplify(cos(n*Pi)) assuming(n,integer);
  simplify(sin(n*Pi));
  simplify(sin(n*Pi)) assuming(n,integer);
```

$$\cos(n\,\pi)$$

$$(-1)^n$$

$$\sin(n\,\pi)$$

$$0 \qquad \textbf{(2.1.5)}$$

Working with and solving equations

```
> restart;
```

A Maple variable can actually be an equation, or set of equations. Generally, any command that can be applied to an expression without an equal sign will work on an equation by operating separately on the left and right hand sides:

```
> eq := (a^2 - b^2)/(a-b) = tan(theta)/sin(theta);
  eq := simplify(eq);
```

$$eq := \frac{a^2 - b^2}{a - b} = \frac{\tan(\theta)}{\sin(\theta)}$$

$$eq := a + b = \frac{1}{\cos(\theta)} \qquad \textbf{(2.2.1)}$$

But certain commands only work on equations, like the closely related **solve** and **isolate**:

```
> solve(eq,theta);
  isolate(eq,theta);
```

$$\arccos\left(\frac{1}{a+b}\right)$$

$$\theta = \arccos\left(\frac{1}{a+b}\right) \qquad \textbf{(2.2.2)}$$

The difference between the two is that **isolate** returns an equation with the desired variable on the left, while solve gives only the solution for the desired variable. Actually, the first argument of solve can be any expression that is understood to be equal to zero:

```
> eq := a*x^2+b*x+c;
  sol := solve(eq,x);
  sol := [sol];
```

$$eq := a\,x^2 + b\,x + c$$

$$sol := \frac{1}{2}\,\frac{-b + \sqrt{-4\,a\,c + b^2}}{a}, -\frac{1}{2}\,\frac{b + \sqrt{-4\,a\,c + b^2}}{a}$$

$$sol := \left[\frac{1}{2}\,\frac{-b + \sqrt{-4\,a\,c + b^2}}{a}, -\frac{1}{2}\,\frac{b + \sqrt{-4\,a\,c + b^2}}{a}\right] \qquad \textbf{(2.2.3)}$$

Notice that here, **solve** returned the two possible solutions for x separated by a comma. The

third line takes this output and puts it into a list. We can check that we really have solutions by using the **subs** command to substitute our solutions back into the original expression:

```
> subs(x=sol[1],eq);
  simplify(%);
```

$$\frac{1}{4}\,\frac{\left(-b+\sqrt{-4\,a\,c+b^2}\right)^2}{a}+\frac{1}{2}\,\frac{b\left(-b+\sqrt{-4\,a\,c+b^2}\right)}{a}+c$$

$$0 \tag{2.2.4}$$

Some care must be used with the isolate command for equations with multiple solutions. We see that when we use **solve** on a quadratic equation, we sucessfully find both roots, but when we use **isolate**, we only get one root:

```
> eq := x^2 + 7*x + 12 = 0;
  solve(eq);
  isolate(eq,x);
```

Here is another example where we solve three equations (given as a set), for the three variables (which are returned as a set). The subs command is used to pick out the particular solution for x, y and z, which we call **x_sol**, etc...

```
> eqs := {2*a = x + y + z, 4*b=2*x + z, -4*c = z+y};
  sol := solve(eqs,{x,y,z});
  x_sol := subs(sol,x);
  y_sol := subs(sol,y);
  z_sol := subs(sol,z);
```

$$eqs := \{2\,a=x+y+z,\,4\,b=2\,x+z,\,-4\,c=z+y\}$$

$$sol := \{x=2\,a+4\,c,\,y=4\,c-4\,b+4\,a,\,z=4\,b-4\,a-8\,c\}$$

$$x_sol := 2\,a+4\,c$$

$$y_sol := 4\,c-4\,b+4\,a$$

$$z_sol := 4\,b-4\,a-8\,c \tag{2.2.5}$$

If **solve** fails to find an analytic solution to your problem, **fsolve** can sometimes find a numeric one:

```
> eq := sin(x)-x+1/sqrt(2);
  solve(eq);
  sol := fsolve(eq);
  simplify(subs(x=sol,eq));
```

$$eq := \sin(x)-x+\frac{1}{2}\sqrt{2}$$

$$RootOf\left(2\,_Z-2\sin(_Z)-\sqrt{2}\right)$$

$$sol := 1.698911280$$

$$-7.\;10^{-10} \tag{2.2.6}$$

Here, **fsolve** gives $x = 1.69\ldots$, which satisfies **eq = 0** to numeric accuracy. Finally, here are some examples of the useful **lhs**, **rhs** and **(lhs-rhs)** commands:

```
> eq := r^2 + 35 = 75*n + arctanh(r^4);
  lhs(eq);
  rhs(eq);
  (lhs-rhs)(eq);
```

$$eq := r^2+35=75\,n+\operatorname{arctanh}\left(r^4\right)$$

$$r^2 + 35$$

$$75\, n + \operatorname{arctanh}(r^4)$$

$$r^2 + 35 - 75\, n - \operatorname{arctanh}(r^4) \qquad \textbf{(2.2.7)}$$

Functions and calculus

Defining functions as mappings

A mapping in Maple is an object that takes a finite number of arguments, applies some operations to them, and returns a final result. Hence, it is the Maple equivalent of a mathematical function. Mappings are defined as follows:

```
> f := x -> x^2;
```

$$f := x \rightarrow x^2 \qquad \textbf{(3.1.1)}$$

This mapping is called **f**. It takes a single argument **x** and returns its square. I can call **f** with all sorts of arguments:

```
> f(x);
  f(4);
  f(sin(x));
  expand(f(5+I*y));
```

$$x^2$$

$$16$$

$$\sin(x)^2$$

$$25 + 10\, \mathrm{I}\, y - y^2 \qquad \textbf{(3.1.2)}$$

Mappings can also be defined with several arguments, and in terms of other mappings:

```
> g := (x,y) -> x/y*sin(x*y) + f(y);
  g(4,p);
```

$$g := (x, y) \rightarrow \frac{x \sin(x\, y)}{y} + f(y)$$

$$\frac{4 \sin(4\, p)}{p} + p^2 \qquad \textbf{(3.1.3)}$$

Note that one can also define sort of ``pseudo-functions" by indescriminate use of **:=**, as shown here

```
> h(x) := x^3;
  h(4);
```

$$h(x) := x^3$$

$$h(4) \qquad \textbf{(3.1.4)}$$

As you can see, $h(x)$ is not really a function at all, because Maple doesn't know how to evaluate $h(4)$. **Moral:** *to define a function, use mapping notation.*

Differentiating and integrating

Differentiating with the "diff" command

```
> restart;
```

The derivative of $f(x)$ with respect to x is simply given by:

```
> a := diff(f(x),x);
```

$$a := \frac{d}{dx} f(x) \quad (3.2.1.1)$$

Partial and higher order derivatives are straightforward generalizations of this:

```
> b := diff(g(u,v),u,u,u,v,v,u,u);
```

$$b := \frac{\partial^7}{\partial v^2 \, \partial u^5} g(u, v) \quad (3.2.1.2)$$

Derivatives are calculated explicitly if the quantity being differentiated is known explicitly:

```
> f := x -> exp(sin(x) + sin(tanh(x)));
  simplify(a);
  diff(tan(x),x,x);
```

$$f := x \rightarrow e^{\sin(x) + \sin(\tanh(x))}$$

$$\left(-\cos(\tanh(x)) \tanh(x)^2 + \cos(x) + \cos(\tanh(x))\right) e^{\sin(x) + \sin(\tanh(x))}$$

$$2 \tan(x) \left(1 + \tan(x)^2\right) \quad (3.2.1.3)$$

You can even differentiate equations:

```
> eq := x^2 = tan(x)^(1/4);
  diff(eq,x);
```

$$eq := x^2 = \tan(x)^{1/4}$$

$$2\, x = \frac{1}{4} \frac{1 + \tan(x)^2}{\tan(x)^{3/4}} \quad (3.2.1.4)$$

Differentiating with the D operator

The **D** operator differentiates an expression with respect to it's argument(s), while **diff** differentiates with respect to variables. This best illustrated by example:

```
> f := x -> sin(x);
  a := diff(f(q(x)),x);
  b := D(f)(q(x));
```

$$f := x \rightarrow \sin(x)$$

$$a := \cos(q(x)) \left(\frac{d}{dx} q(x)\right)$$

$$b := \cos(q(x)) \quad (3.2.2.1)$$

In this example, **diff** uses the chain rule to evaluate $\frac{d}{d\,x} f(q(x)) = f'(q(x)) q'(x)$, while $D(f)\,(q(x)) = f'(x)$ essentially returns "f prime" evaluated at $q(x)$. It is the difference between substituting in $q(x)$ for x in $f(x)$ and then differentiating, and differentiating $f(x)$ first and then substituting $q(x)$ for x. The notation for partial and multiple derivatives is

```
> U := (x,y) -> exp(I*omega*x) + exp(beta*y);
  D[1](U)(x+4,y);
  D[2](U)(x,y^35);
  D[1,2](U)(x,y);
```

$$U := (x, y) \rightarrow e^{I \omega x} + e^{\beta y}$$

$$I\,\omega\,e^{I\,\omega\,(x+4)}$$

$$\beta\,e^{\beta\,y^{35}}$$

$$0 \tag{3.2.2.2}$$

Note that **D** is another protected symbol:

```
> D := 4;
Error, attempting to assign to `D` which is protected.  Try
declaring `local D`; see ?protect for details.
```

Integrating

Integration is implemented using the **int** command. Like **diff**, the command needs at least two arguments, the first is the function to be integrated, and the second is the integration variable. First, we give some examples of indefinite integration:

```
> restart;
  int(exp(I*beta*q),q);
  f := x -> sin(x)^3*cos(2*x)/(1-2*tan(x))^2;
  simplify(int(f(x),x));
```

$$-\frac{I\,e^{I\,\beta\,q}}{\beta}$$

$$f := x \rightarrow \frac{\sin(x)^3\cos(2\,x)}{(1-2\tan(x))^2}$$

$$\frac{1}{9375}\,\frac{1}{\cos(x)-2\sin(x)}\Big(750\cos(x)^6+1500\cos(x)^5\sin(x)-475\cos(x)^4 \tag{3.2.3.1}$$

$$-1950\cos(x)^3\sin(x)$$

$$+168\,\mathrm{arctanh}\left(\frac{1}{5}\,\frac{\sqrt{5}\,(-1+\cos(x)-2\sin(x))}{\sin(x)}\right)\cos(x)\sqrt{5}$$

$$-336\,\mathrm{arctanh}\left(\frac{1}{5}\,\frac{\sqrt{5}\,(-1+\cos(x)-2\sin(x))}{\sin(x)}\right)\sin(x)\sqrt{5}$$

$$-425\cos(x)^2-950\cos(x)\sin(x)+910\cos(x)-1820\sin(x)+1060\Big)$$

Note that indefinite integrals omit the "arbitrary constant", just as most tables do. Now, some definite integrals:

```
> int(F(x),x=a..b);
  int(sin(omega)^2,omega=0..3*Pi);
  A := int(exp(-a*x^2),x=-infinity...infinity);
  assume(a>0);
  simplify(A);
```

$$\int_a^b F([x])\,dx$$

$$\frac{3}{2}\,\pi$$

$$A := \begin{cases} \frac{\sqrt{\pi}}{\sqrt{a}} & \operatorname{csgn}(a) = 1 \\ \infty & \mathit{otherwise} \end{cases}$$

$$\frac{\sqrt{\pi}}{\sqrt{a\sim}} \tag{3.2.3.2}$$

We can even define functions in terms of integrals using this mapping notation. The following function $x(r, M)$ satisfies the differential equation $\frac{\partial}{\partial r}x(r, M) = \frac{1}{f(r, M)}$ and appears prominently in the theory of black holes:

```
> restart;
  # define the function [normalized such that x(3*M)=0]

     f := (r,M) -> 1 - 2*M/r;
     x := (r,M) -> int(1/f(u,M),u=3*M..r);

  # define the ODE that it satisfies

     ode := diff(X(r,M),r) - 1/f(r,M) = 0;

  # verfify that it really satisfies the ODE

     subs(X(r,M)=x(r,M),ode);
     simplify(%);

  # numerically evaluate the function when M = 4 and r = 10

     evalf(x(10,4));
```

$$f := (r, M) \to 1 - \frac{2\,M}{r}$$

$$x := (r, M) \to \int_{3\,M}^{r} \frac{1}{f(u, M)}\,\mathrm{d}u$$

$$ode := \frac{\partial}{\partial r}\,X(r, M) - \frac{1}{1 - \frac{2\,M}{r}} = 0$$

$$\frac{\partial}{\partial r}\left(\int_{3\,M}^{r} \frac{1}{1 - \frac{2\,M}{u}}\,\mathrm{d}u\right) - \frac{1}{1 - \frac{2\,M}{r}} = 0$$

$$0 = 0$$

$$-7.545177445 \tag{3.2.3.3}$$

Finally, there are plenty of integrals Maple cannot do. When this happens, int returns then integral printed in a pretty form. You can evaluate such annoyances numerically by using `evalf`:

```
> gnat := int(sqrt(sin(x)+exp(x)),x=3..Pi);
  gnat := evalf(int(sqrt(sin(x)+exp(x)),x=3..Pi));
```

$$gnat := \int_{3}^{\pi} \sqrt{\sin(x) + e^{x}}\, dx$$

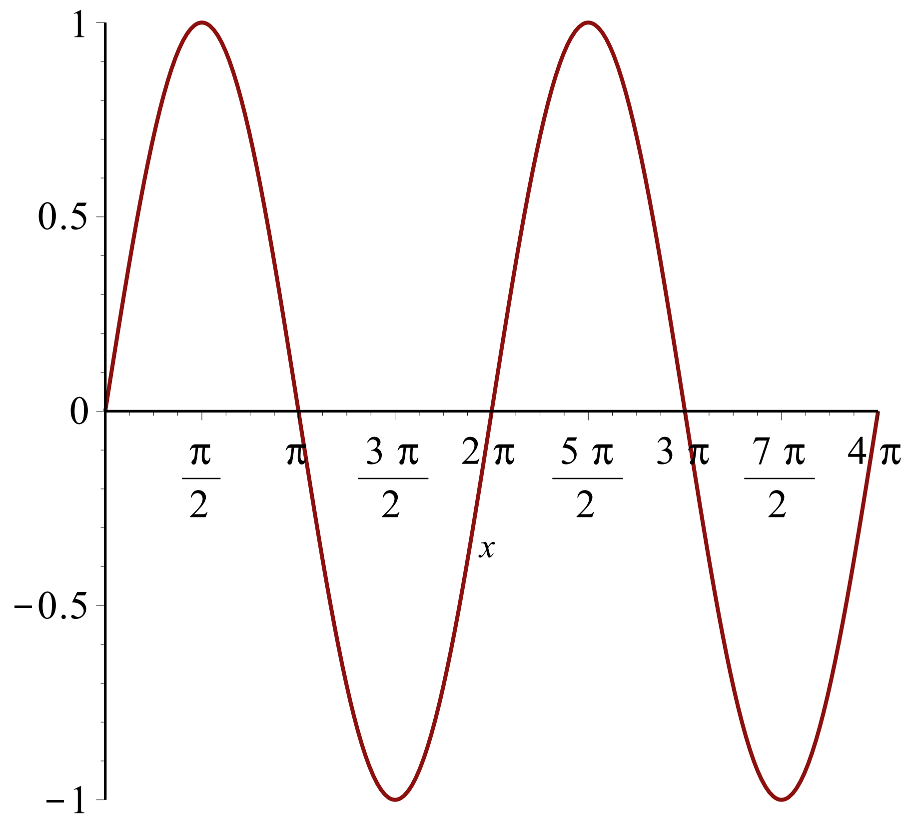
$$gnat := 0.6586659653 \qquad \textbf{(3.2.3.4)}$$

Plots

2-D plotting

Simple plots of functions are achieved with the **plot** command. Here is a plot of $\sin(x)$ for $x \in [0, 4\pi]$:

```
> plot(sin(x),x=0..4*Pi);
```

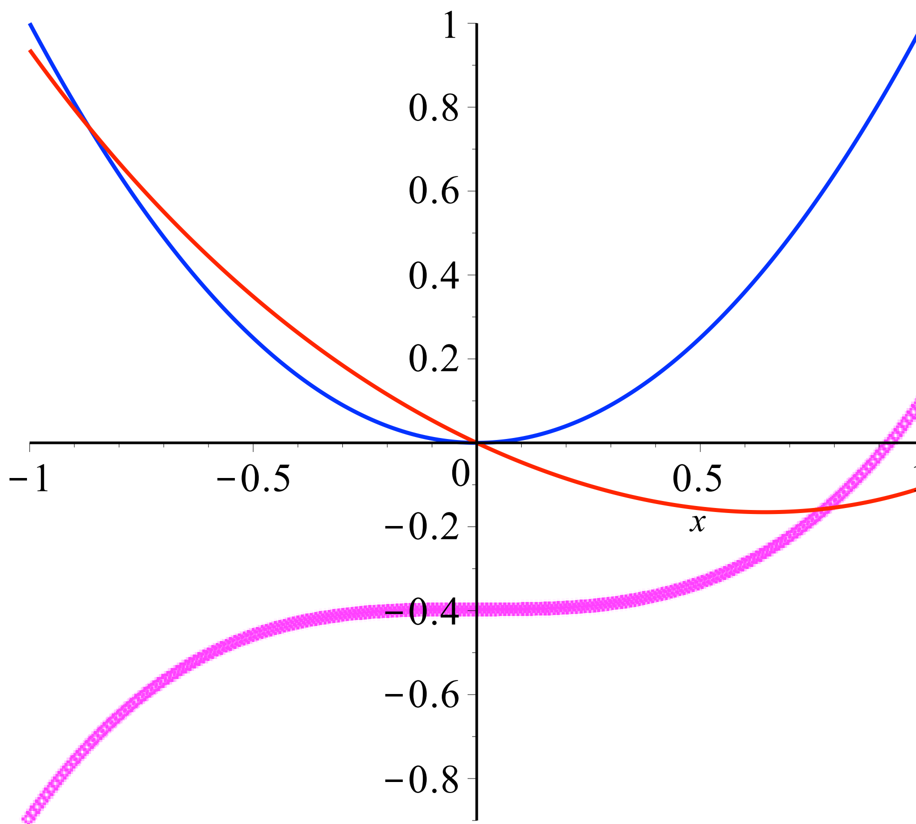


If you want to plot multiple functions of the same variable, include them as a list:

```
> f,g,h := x -> x^2, x -> (5*x^3-4)/10, x -> cosh(x)-exp(x/2);
  functions := [f(x),g(x),h(x)];
  plot(functions,x=-1..1,color=[blue,magenta,red],style=[line,
  point,line]);
```

$$f, g, h := x \rightarrow x^2, x \rightarrow \frac{1}{2}x^3 - \frac{2}{5}, x \rightarrow \cosh(x) - e^{\frac{1}{2}x}$$

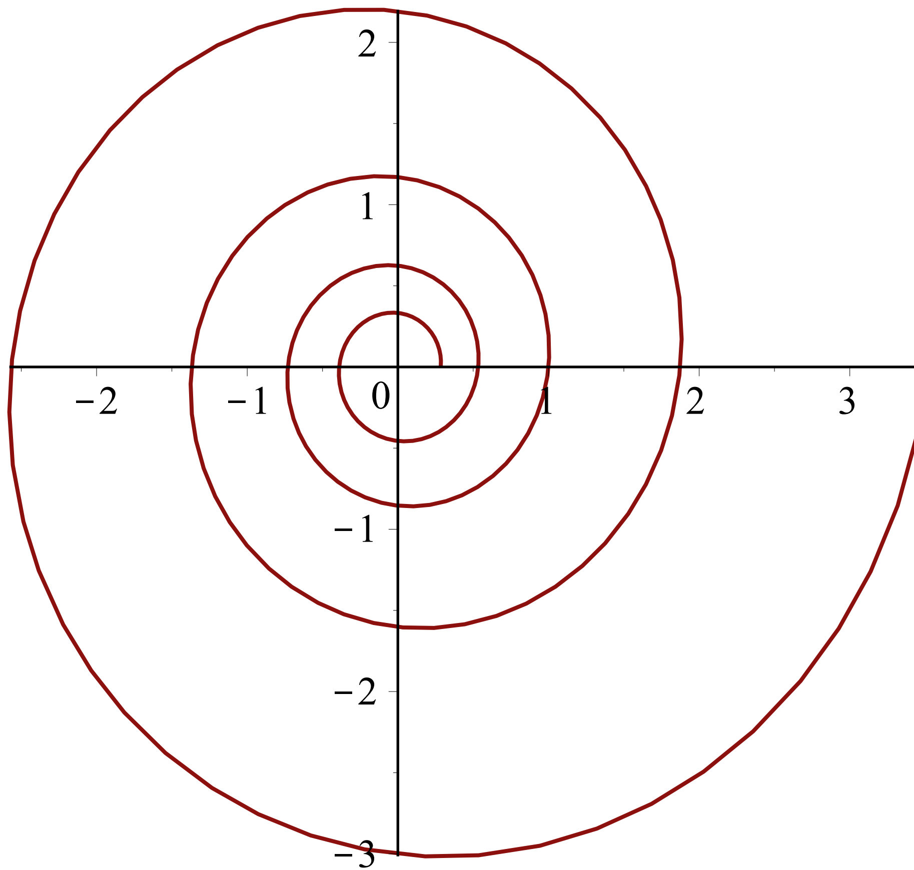
$$functions := \left[x^2, \frac{1}{2}x^3 - \frac{2}{5}, \cosh(x) - e^{\frac{1}{2}x}\right]$$



Parametric plots are defined by a list structure `[x(t),y(t),t=t1..t2]`:

```
> Curve := [exp(t/10)*cos(t),exp(t/10)*sin(t),t=-4*Pi..4*Pi];
  plot(Curve);
```

$$Curve := \left[e^{\frac{1}{10} t} \cos(t), e^{\frac{1}{10} t} \sin(t), t = -4\,\pi\,..4\,\pi \right]$$

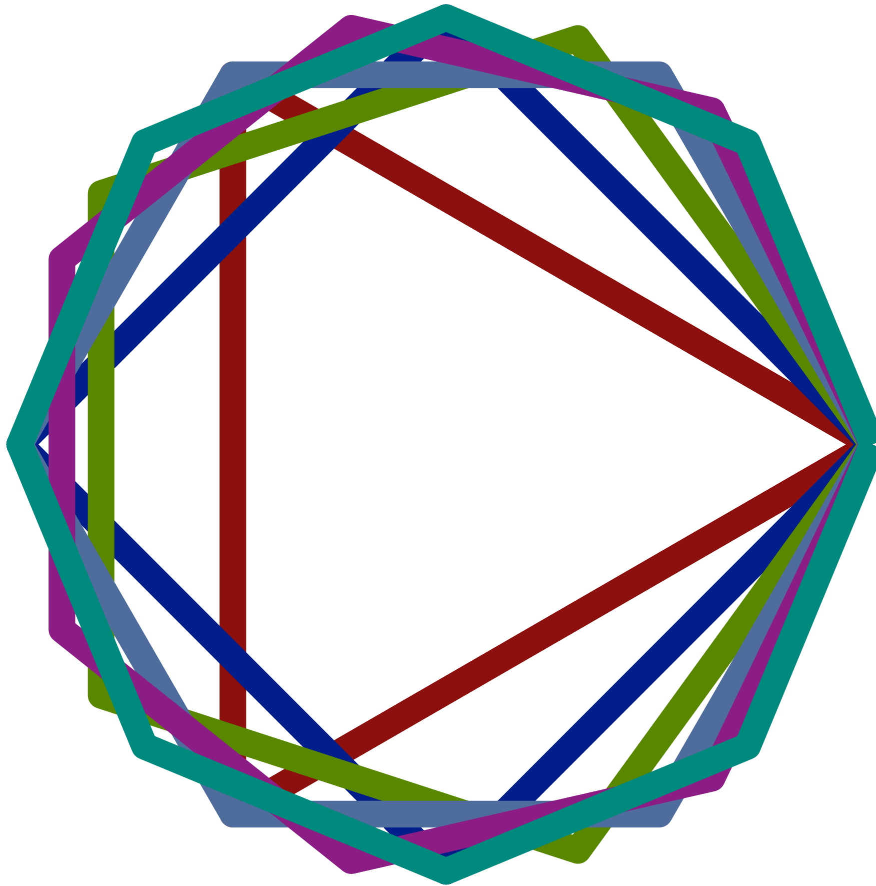


You can also plot 2-dimensional datapoints if they are represented by a list of lists: **`[[x1,y1],[x2,y2],[x3,y3],...]`**. By default, Maple draws a straight line between successive points. If you just want the points, use the **`style = point`** option. In this example, we use the **`seq`** command in a mapping to generate the datapoints of an *N*-sided regular polygon inscribed on a circle:

```
> Polygon := N -> [seq([cos(2*Pi*n/N),sin(2*Pi*n/N)],n=0..N)];
  Polygon(5);
  plot([seq(Polygon(N),N=3..8)],axes=none,thickness=10,scaling=
  constrained);
```

$$Polygon := N \to \left[seq\left(\left[\cos\left(\frac{2\,\pi\,n}{N}\right), \sin\left(\frac{2\,\pi\,n}{N}\right)\right], n = 0..N\right)\right]$$

$$\left[[1, 0], \left[\cos\left(\frac{2}{5}\,\pi\right), \sin\left(\frac{2}{5}\,\pi\right)\right], \left[-\cos\left(\frac{1}{5}\,\pi\right), \sin\left(\frac{1}{5}\,\pi\right)\right], \left[-\cos\left(\frac{1}{5}\,\pi\right), -\sin\left(\frac{1}{5}\,\pi\right)\right], \left[\cos\left(\frac{2}{5}\,\pi\right), -\sin\left(\frac{2}{5}\,\pi\right)\right], [1, 0]\right]$$



Here, we plotted six separate datasets by nested use of the **seq** command. The optional argument supressed the drawing of axes, set the line thickness to 10 point, and adjusted the vertical and horizontal scales to make sure a square looks like a square. In and of itself, this simple **plot** command is capable of much more than mentioned here, see **?plot** for details. Even more plotting capability is achieved by loading the **plots** or **plottools** packages:

```
> with(plots);
  with(plottools);
```

[*animate*, *animate3d*, *animatecurve*, *arrow*, *changecoords*, *complexplot*, *complexplot3d*, *conformal*, *conformal3d*, *contourplot*, *contourplot3d*, *coordplot*, *coordplot3d*, *densityplot*, *display*, *dualaxisplot*, *fieldplot*, *fieldplot3d*, *gradplot*, *gradplot3d*, *implicitplot*, *implicitplot3d*, *inequal*, *interactive*, *interactiveparams*, *intersectplot*, *listcontplot*, *listcontplot3d*, *listdensityplot*, *listplot*, *listplot3d*, *loglogplot*, *logplot*, *matrixplot*, *multiple*, *odeplot*, *pareto*, *plotcompare*, *pointplot*, *pointplot3d*, *polarplot*, *polygonplot*, *polygonplot3d*, *polyhedra_supported*, *polyhedraplot*, *rootlocus*, *semilogplot*, *setcolors*, *setoptions*, *setoptions3d*, *spacecurve*, *sparsematrixplot*, *surfdata*, *textplot*, *textplot3d*, *tubeplot*]

[*annulus*, *arc*, *arrow*, *circle*, *cone*, *cuboid*, *curve*, *cutin*, *cutout*, *cylinder*, *disk*, *dodecahedron*, *ellipse*, *ellipticArc*, *getdata*, *hemisphere*, *hexahedron*, *homothety*, **(4.1.1)**

hyperbola, *icosahedron*, *line*, *octahedron*, *parallelepiped*, *pieslice*, *point*, *polygon*, *prism*, *project*, *rectangle*, *reflect*, *rotate*, *scale*, *sector*, *semitorus*, *sphere*, *stellate*, *tetrahedron*, *torus*, *transform*, *translate*]

The stuff in the square brackets are the new commands that are available after the packages are loaded. There are a lot of them, and quite frankly the effects of some of them are easy to achieve with more elementary commands. That said, some are quite useful, such as **logplot** which generates log-linear plots, etc. Some plotting comments:

- All 2D Maple plots consist of lists of data points that are graphically interpreted by the front-end software. Whenever possible, an adaptive plotting scheme is used that concentrates the datapoints wherever function derivatives are large. Usually this gives satisfactory results. However, if you find a particular plot looks bad, try the optional **numpoints** argument, which forces Maple to use more points to render the plot.
- The **display** command in the plots package has a lot of uses, including a rapid way to choose plot parameters without recalcuting the data. See **?plots/display**
- The range of the axes can be altered in a number of ways. The following three commands generate the same plot of $\sin(x)$:

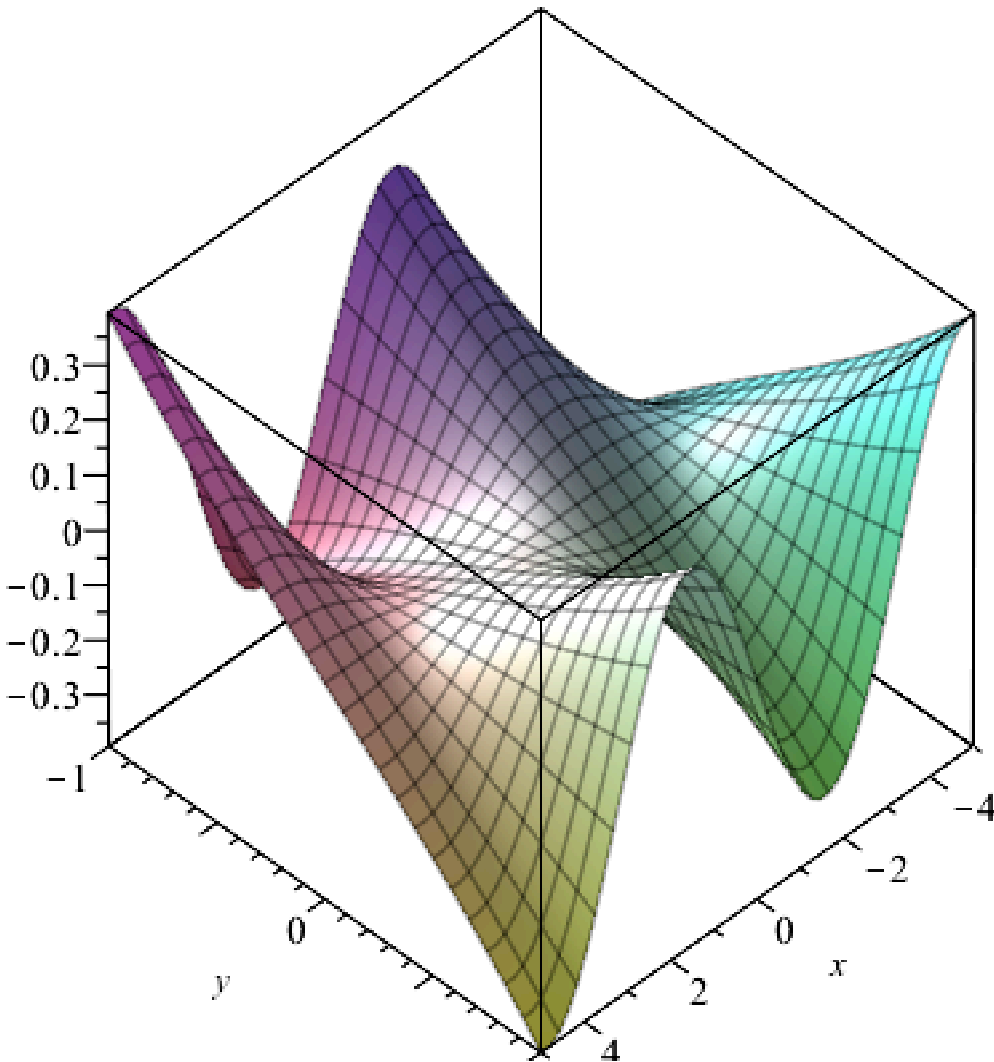
```
> plot(sin(x),x=-Pi..Pi,-3..3):
  plot(sin(x),x=-Pi..Pi,view=[default,-3..3]):
  plot(sin(x),x=-500..500,view=[-Pi..Pi,-3..3]):
```

3-D plotting

3D plots are generated with the "plot3d" command. Here is a plot of the imaginary part of the 2nd order Bessel function of the first kind in a region of the complex plane defined by $-5 < \mathrm{Re}(z) < 5$ and $-1 < \mathrm{Im}(z) < 1$.

```
> f := (x,y) -> Im(BesselJ(2,x+I*y));
  plot3d(f(x,y),x=-5..5,y=-1..1,axes=boxed);
```

$$f := (x, y) \rightarrow \Im(\mathrm{BesselJ}(2, x + \mathrm{I}\, y))$$



If you are viewing these notes in the worksheet form, by clicking and dragging on the above plot you will be able to rotate and resize it. Also, some buttons will appear in the above toolbar that let you change the axes, style of rendering, etc. Most of this functionality is also available from the command line, see **?plot3d**. One can accomplish truly useless things with the animate facilities, execute the following command to see some sort of circular wave propagating on a disk (remove the colon and replace with a semi-colon):

```
> animate3d([r*cos(phi),r*sin(phi),cos(t-r)],r=0..10,phi=0..2*
  Pi,t=0..20,frames=50,scaling=constrained,axes=none):
```

In this instance, I have defined a surface parametrically as $x = x(r, \phi, t)$, $y = y(r, \phi, t)$, and $z = z(r, \phi, t)$, where t is time. Click on the plot and press play in the toolbar to see the movie.

Procedures and programming

Basic procedure syntax

```
> restart;
```

Procedures are the like the Maple version of subroutines in a traditional programming language. A mapping is a rudimentary procedure. For example, f and g here accomplish the same thing, even though the former is a mapping and the latter is a procedure:

```
> f := x -> x^2 + sin(x)^2;
  g := proc(x)
       x^2 + sin(x)^2
  end proc;
  f(5);
  g(5);
```

$$f := x \rightarrow x^2 + \sin(x)^2$$

$g :=$ **proc**(x) x^2 + $\sin(x)$^2 **end proc**

$$25 + \sin(5)^2$$

$$25 + \sin(5)^2 \quad \textbf{(5.1.1)}$$

As a matter of style, it is useful to format procedures as above, with different commands on different lines. But this is not necessary:

```
> g := proc(x) x^2 + sin(x)^2 end proc;
```

$g :=$ **proc**(x) x^2 + $\sin(x)$^2 **end proc** **(5.1.2)**

Of course, we can have more complicated procedures. This procedure take a mapping and an integer N as its argument and returns the Nth order Fourier approximation:

```
> Fourier := proc(f,N,x)
     local a,b:
     a := n -> 1/Pi*int(f(x)*cos(n*x),x=-Pi..Pi):
     b := n -> 1/Pi*int(f(x)*sin(n*x),x=-Pi..Pi):
     1/2*a(0) + sum(a(n)*cos(n*x),n=1..N) + sum(b(n)*sin(n*x),n=
  1..N):
  end proc;
```

Fourier := **proc**(f, N, x) **(5.1.3)**

local a, b;

$a := n \rightarrow int\big(f(x) * \cos(n * x), x = -\pi..\pi\big) / \pi$;

$b := n \rightarrow int\big(f(x) * \sin(n * x), x = -\pi..\pi\big) / \pi$;

$1/2 * a(0) + sum(a(n) * \cos(n * x), n = 1..N) + sum(b(n) * \sin(n * x), n = 1..N)$

end proc

Several matters of syntax arise in this example:

- The second line declares that `a` and `b` are local variables in this procedure. This means that even though I define them within the procedure, the Maple environment outside the procedure does not know about them. Practically, this means that if I define `a := 5` and then run/call the procedure, `a` will still be equal to 5 after the procedure is run.
- As in the ordinary worksheet environment, separate commands must be separated by colons or semi-colons. The difference is that it doesn't matter which --- whether or not a given procedure produces printed output depends on whether or not is called with a colon or semi-colon. As usual, if you mess-up the punctuation you will generate an error when you execute the group in which the procedure definition occurs
- All procedures end with an "end proc". The nature of the character after this command determines if Maple prints out the procedure definition.
- The output of a procedure is the last line executed before end proc. This behaviour can be

modified by the use of an explicit **return** command, but this is usually unnecessary (not always, see **?return**)

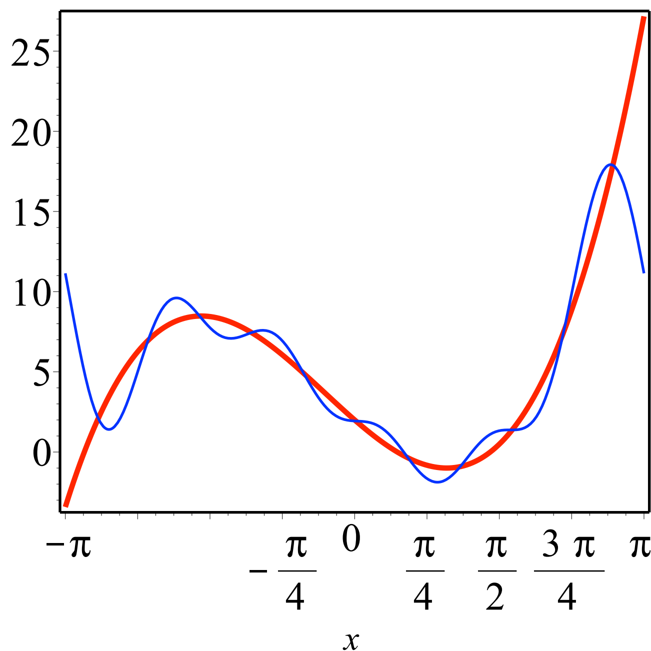
Let's explore the behaviour of the procedure defined above:

```
> # choose a function to decompose
    a := x -> (x+Pi)^4;
  # generate a Fourier approximation accurate to 2nd order (and
  simplify the output of Fourier)
    simplify(Fourier(a,2,q));
  # choose a different function to analyze
    b := x -> (x^3+x^2-5*x+2);
  # this time, create a plot of the function (red, thick line)
  and the 5th order approximation (blue, thin line)
    plot([b(x),Fourier(b,5,x)],x=-Pi..Pi,axes=boxed,color=[red,
  blue],thickness=[2,0]);
```

$$a := x \to (x+\pi)^4$$

$$-16\pi^3 \sin(q)\cos(q) + 16\pi^3 \sin(q) + 12\pi \sin(q)\cos(q) - 48\pi \sin(q) + \frac{16}{5}\pi^4 + 16\pi^2 \cos(q)^2 - 32\pi^2 \cos(q) - 8\pi^2 - 6\cos(q)^2 + 48\cos(q) + 3$$

$$b := x \to x^3 + x^2 - 5x + 2$$



Conditional if...then..else statements

Conditional **if..then..else** statements are ubiquitous in procedures. Here is a procedure that tests if its argument is a half-integer (-1/2,1/2,3/2,etc..):

```
> HalfInteger := proc(x):
      if (type(x,integer)) then:
          false:
      elif (type(2*x,integer)) then:
          true:
      else:
```

```
            false:
        fi:
    end proc:
    HalfInteger(3/2+Pi);
    HalfInteger(-105/2);
    HalfInteger(200/2);
    HalfInteger(blueberry);
```

false

true

false

false **(5.2.1)**

In this procedure, the last line of output depends on what type of argument we have. The **true** and **false** quantities are special in Maple because there are the arguments of if/then structures. Indeed, **true/false** is what the **type** function returns:

```
> type(5,prime);
  type(6,prime);
```

true

false **(5.2.2)**

Because our **HalfInteger** procedure is **true/false** valued, it can be used in further **if/then** statements:

```
> PositiveHalfInteger := proc(x):
      if (HalfInteger(x) and x > 0) then:
          `Congratulations! You have found a positive half-
  integer! Call Sweden!`:
      else:
          `I am sorry. This is not what we are looking for.
  Please don't give up.`:
      fi:
  end proc:
  PositiveHalfInteger(-5/2);
  PositiveHalfInteger(5/2);
```

I am sorry. This is not what we are looking for. Please don't give up.

Congratulations! You have found a positive half-integer! Call Sweden! **(5.2.3)**

Repetitive statements (loops)

Also ubiquitous in many procedures are repetition loops. Like the conditional statements above, they don't actually have to appear in procedures:

```
> for i from 40 to -11 by -7 do:
     print(i):
  od:
```

40

33

26

19

12

5

$$-2$$

$$-9 \tag{5.3.1}$$

The **for..while..do** structure is implemented in this procedure, which finds the smallest prime number in a given range of integers:

```
> FindSmallestPrime := proc(low,high)
      local CONTINUE,i:
      CONTINUE := true:
      for i from low to high while (CONTINUE) do:
         if (type(i,prime)) then:
             print(i):
             CONTINUE := false:
         fi:
      od:
      if (CONTINUE) then print(`no primes in range`) fi:
  end proc:
```

In this case, the **while** statement causes the for loop to terminate as soon as a prime number is found.

```
> FindSmallestPrime(23211432,432450443590);
  FindSmallestPrime(152,155);
```

$$23211437$$

no primes in range **(5.3.2)**