Maple Tutorial

to accompany
Partial Differential Equations: Analytical and Numerical Methods, 2nd edition
by Mark S. Gockenbach
(SIAM, 2010)

Introduction

In this introduction, I will explain the organization of this tutorial and give some basic information about *Maple* and *Maple* worksheets. I will also give a preliminary introduction to the capabilities of *Maple*.

About this tutorial

The purpose of this document is to explain the features of *Maple* that are useful for applying the techniques presented in my textbook. This really is a tutorial (not a reference), meant to be read and used in parallel with the textbook. For this reason, I have structured the tutorial to have the same chapter and section titles as the book. However, the purpose of the sections of this document is not to re-explain the material in the text; rather, it is to present the capabilities of *Maple* as they are needed by someone studying the text.

Therefore, for example, in Section 2.1 (Heat flow in a bar; Fourier's Law), I do not explain any physics or modeling (the physics and modeling are in the text). Instead, I explain the *Maple* command for integration, because Section 2.1 is the first place in the text where the student is asked to integrate a function. Because of this style of organization, some parts of the text have no counterpart in this tutorial. For example, there is no Chapter 7, because by the time you have worked through the first six chapters of the tutorial, you have learned all the capabilities of *Maple* that you need to address the material in Chapter 7 of the text. For the same reason, you will see that some individual sections are missing; Chapter 5, for example, begins with Section 5.2.

I should point out that my purpose in writing this tutorial is not to show you how to solve the problems in the text; rather, it is to give you the tools to solve them. Therefore, I do not give you a worked-out example of every problem type---if I did, your "studying" could degenerate to simply looking for an example, copying it, and making a few changes. At crucial points, I do provide some complete examples, since I see no other way to illustrate the power of *Maple* than in context. However, there is still plenty for you to figure out for yourself.

About *Maple*

At the heart of *Maple* is a computer algebra system, that is, a system for doing algebraic manipulations symbolically (and therefore exactly). However, *Maple* also incorporates numerics, graphics, and text processing. It is also a programming environment. We will touch on all of these capabilities in this tutorial.

Maple worksheets

This document you are reading is called a *Maple* worksheet; it combines text with *Maple* commands and their results, including graphics. (Here I assume that you are reading this file in *Maple*, not as a printed document. If you are reading a printed copy, you will have to ignore a few comments about how worksheets are manipulated.) It consists of both text and *Maple* input and output, organized in paragraphs. The input and output occur in execution groups, which I will explain below. The most important thing to understand about a worksheet is that it is interactive---at any time you can execute a *Maple* command and see what it does. This makes a *Maple* worksheet a tremendous learning environment: when you read an explanation of a *Maple* command, you can immediately try it out.

Getting help with *Maple* commands

Help on *Maple* commands is always available through the help menu. You can also get help at the *Maple* prompt by using the "?" operator. I will explain this below.

Getting started with *Maple*

As mentioned above, *Maple* has many capabilities, such as the fact that one can write programs made up of *Maple* commands. The simplest way to use *Maple*, though, is as an interactive computing environment---essentially, a very fancy graphing calculator. You enter a command and *Maple* executes it and returns the result. Here is an example:

```
> 2+2;
```


(1.5.1)

The ">" symbol is the *Maple* prompt; when the cursor is at the prompt, a *Maple* command can be entered and executed. A command can occupy several lines, and is terminated by a semicolon. The command is executed when you enter the semicolon followed by return or enter. *Return by itself just takes you to the next line; the command will not be executed if it is not terminated by a semicolon.* If you wish to enter a *Maple* command, and the prompt is not present, you can select "Execution Group" from the "Insert" menu, and a prompt will appear. (You can choose to put the prompt below or above the current cursor position.) Like

many *Maple* worksheet commands, there are shortcuts for these commands. Control-j insert an execution group below the current cursor position, while control-k inserts one above the cursor.

When *Maple* finishes a command, it displays the output on the next line, centered in the worksheet.

Now that you know how to enter commands and see the results, let's quickly go over some of the most basic capabilities of *Maple*. First of all, *Maple* can do arithmetic with integers and rational numbers, regardless of the number of digits involved.

```
> 123^45;
```

$$11110408185131956285910790587176451918559153212268021823627\
\ 90731998661110012427432839661270480 43 \quad (1.5.2)$$

```
> 115/39+727/119;
```

$$\frac{42038}{4641} \quad (1.5.3)$$

Maple knows the standard elementary functions, such as the trigonometric functions, logarithms, and exponentials, the square root function, and so forth. It also knows the constant p. Consider the following calculation:

```
> sin(Pi/4);
```

$$\frac{1}{2}\sqrt{2} \quad (1.5.4)$$

There are several important things to learn from this example. First of all, the constant p is typed with the first letter capitalized. A common mistake is to forget to capitalize Pi, in which case *Maple* regards "pi" as an arbitrary symbol rather than a known constant:

```
> sin(pi/4);
```

$$\sin\left(\frac{1}{4}\,\pi\right) \quad (1.5.5)$$

(Notice how *Maple* did not evaluate sin(pi/4), since it does not know the value of "pi".)

Another thing to learn from the preceding example is that *Maple* knows that the sine of p/4 is exactly $\frac{\sqrt{2}}{2}$; it does not return an estimate like 0.70710678, as a handheld calculator might. Now consider the following computations of the square root of 2.

```
> sqrt(2);
```

$$\sqrt{2} \tag{1.5.6}$$

```
> sqrt(2.0);
```

$$1.414213562 \tag{1.5.7}$$

The previous two examples show that *Maple* will return a symbolic output when the input is symbolic, and a numeric output when the input is numeric. If an explicit number contains a decimal point, it is regarded as numeric. *Maple* will not replace symbolic quantities with numeric ones unless specifically told to do so (below I will explain how to get a numeric result when desired). Here is another example:

```
> sin(Pi/4.0);
```

$$\sin(0.2500000000\,\pi) \tag{1.5.8}$$

Notice that the combination of the symbolic quantity Pi and the numeric quantity 4.0 is partially simplified (1/4.0 is replaced by 0.2500000000), but Pi is not evaluated numerically. Also, the sine is not evaluated as $\frac{\sqrt{2}}{2}$, since *Maple* does not regard 0.2500000000 as exactly equal to 1/4.
Here are a few more examples:

```
> (100-9)*(100+9);
```

$$9919 \tag{1.5.9}$$

```
> (-5+sqrt(5^2-4*1*4))/2;
```

$$-1 \tag{1.5.10}$$

```
> (-1)^2+5*(-1)+4;
```

$$0 \tag{1.5.11}$$

You should notice that the "^" symbol is used to indicate an exponent, "*" for multiplication, and "/" for division.

An important feature of *Maple* is that you can refer to the previous output using the "%" symbol (which *Maple* calls the "ditto" operator):

```
> 42^2;
```

$$1764 \tag{1.5.12}$$

```
> %-20;
```

$$1744 \tag{1.5.13}$$

You can also use "%%" to refer to the next-to-last output, and "%%%" to refer to

the second-to-last output. You cannot go any further back than this, however,. If you expect to need a certain result later in your *Maple* session, then you should assign it to a variable:

```
> a:=23;
```

$$a := 23 \qquad \textbf{(1.5.14)}$$

```
> a^2+2*a-1;
```

$$574 \qquad \textbf{(1.5.15)}$$

Alternatively, you can refer to previous results by the equation number automatically assigned by *Maple*. For instance, the last output was assigned number (1.5.15). To enter this in an expression, type control-L, followed by the number:

```
> (1.5.15)^2
```

$$329476 \qquad \textbf{(1.5.16)}$$

Important note: the assignment operator is ":=", not just "=". The equals sign by itself is used to represent equations, as explained below. *A common mistake is to use "=" when you should use ":="!*

You will often wish to obtain a numeric value from a symbolic quantity. *Maple* provides the **evalf** function for this purpose:

```
> evalf(Pi);
```

$$3.141592654 \qquad \textbf{(1.5.17)}$$

By default, all numeric computation is performed with 10 decimal digits. The evalf function will give a more precise result if desired; the form "evalf[n](x)" yields n digits of the decimal expansion of x:

```
> evalf[30](Pi);
```

$$3.14159265358979323846264338328 \qquad \textbf{(1.5.18)}$$

You can also reset the default number of digits used in numeric computations by changing the constant "Digits":

```
> Digits;
```

$$10 \qquad \textbf{(1.5.19)}$$

```
> Digits:=100;
```

$$Digits := 100 \qquad \textbf{(1.5.20)}$$

```
> evalf(Pi);
```

$$3.14159265358979323846264338327950288419716939937510582097\backslash \qquad \textbf{(1.5.21)}$$

$$4944592307816406286208998628034825342117068$$

```
> sqrt(2.0);
```

1.41421356237309504880168872420969807856967187537694807317\ **(1.5.22)**

667973799073247846210703885038753432764157 3

I will reset Digits to 10 for this tutorial:

```
> Digits:=10;
```

$$Digits := 10 \qquad \textbf{(1.5.23)}$$

Saving a worksheet

When you prepare a homework solution in *Maple*, or do some other work that you want to save for later reference, you must save the contents of the worksheet. The first time you save the document, go to the "File" menu, select the "Save As" option, and then enter a file name ending in ".mws". For example, "hw1.mws" would be a reasonable name for your first homework assignment. Thereafter, whenever you make changes to the worksheet, use the "Save" option under the "File" menu. As you work on your worksheet, you should frequently save it, so that, if something goes wrong, you will never lose much work.

As you work your way through this tutorial, you will want to stop at times and come back to it later. At those times, you will have to decide if you wish to save the changes you have made or not. You may wish to save the tutorial with your modifications under a different name, so that you keep a clean copy of the original tutorial.

Here is an important point about *Maple* worksheets: When you open an existing worksheet, the *Maple* kernel (the part of *Maple* that actually executes user commands) is not aware of any commands that appear in the worksheet. In particular, any variables that are initialized in the worksheet do not actually have values unless you cause the kernel to execute the commands appearing in the worksheet. For example, above I gave variable a the value of 23. If you were now to create an execution group (using control-j, for instance) and enter the variable name a, *Maple* would return the name 'a', not the value 23. If you want the kernel to be initialized, you must cause it to happen in one of two ways. First of all, you can execute the commands one-by-one. To do this, simply put the cursor on the first command line and press enter. The kernel will execute the first command and the cursor will automatically go to the next command line (skipping any intervening text). You can then press enter repeatedly until you come to the point in the tutorial at which you wish to continue.

Alternatively, you can cause *Maple* to execute every command in the notebook by choosing "Execute" and then "Worksheet" from the "Edit" menu. The kernel will execute each command, beginning with the first. This is very convenient if, for example, you are working on homework in a worksheet and you wish to pick up where you left off. However, it may not be very useful for working with this

tutorial; if you have not finished the tutorial, then you probably do not want to execute the commands that come after the point you have currently reached.

Chapter 1: Classification of differential equations

Maple allows us to define functions and compute their derivatives symbolically. Using these capabilities, it is usually straightforward to verify that a given function is a solution to a differential equation.

Example

Suppose you wish to verify that

$$u(t) = e^{at}$$

is a solution to the ODE

$$\frac{du}{dt} - au = 0.$$

First define the function u:

```
> u:=t->exp(a*t);
```

$$u := t \to e^{a\,t} \qquad \textbf{(2.1.1)}$$

As the above example shows, a function is defined in the form of a mapping. The syntax

t->exp(*a***t*)

states that the input variable *t* is mapped to the output value exp(*a***t*). (Also notice from this example that the natural exponential function e^x is denoted "exp(*x*)").

Having defined the function, I can now manipuate it in various ways: evaluate it, differentiate it, etc. The function is evaluated in the expected way:

```
> u(1);
```

$$e^{23} \qquad \textbf{(2.1.2)}$$

The expected result is e^a. Where did the number 23 come from? In fact, I have intentionally illustrated a common mistake. Earlier in the worksheet, I defined the variable a to have the value 23. Now I want to use a as an indeterminate parameter. It is necessary to clear the value of the variable before reusing it. *Failure to clear the values of variables is a common source of errors in using*

Maple!

Here is the value of *a*:

```
> a;
```

$$23 \tag{2.1.3}$$

The value of a variable is cleared by assigning to the variable its own name:

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

$$a := a \tag{2.1.4}$$

I can now use the variable as an indeterminate parameter:

```
> a;
```

$$a \tag{2.1.5}$$

In fact, I can now use the function u in the expected way, without redefining it:

```
> u(1);
```

$$e^{a} \tag{2.1.6}$$

This example illustrates an important feature of function definitions in *Maple*: When you define a function, *Maple* holds the definition as you give it (without trying to simplify it or evaluate any of the parameters in the expression defining the function). When you wish to evaluate the function, *Maple* uses the current value of any parameters in the function definition.

Another way to clear the value of a parameter is using the unassign command, which is convenient because you can use it to clear several variables at once. Here is an example:

```
> x:=1;
```

$$x := 1 \tag{2.1.7}$$

```
> y:=2;
```

$$y := 2 \tag{2.1.8}$$

```
> unassign('x','y');
> x;
```

$$x \tag{2.1.9}$$

```
> y;
```

$$y \tag{2.1.10}$$

It is good practice to clear the values of any parameters before you use them, in

case they have previously been assigned values. For example, here is how I would define the function u:

```
> unassign('t','a');
> u:=t->exp(a*t);
```

$$u := t \to e^{a\,t} \qquad \textbf{(2.1.11)}$$

There is no harm in clearing a variable that does not have a value, and this practice will eliminate certain errors that are difficult to find.

Now I want to get back to the example. I want to determine if

$$u(t) = e^{at}$$

is a solution of the differential equation

$$\frac{du}{dt} - au = 0.$$

The **diff** command computes derivatives symbolically:

```
> diff(u(t),t)-a*u(t);
```

$$0 \qquad \textbf{(2.1.12)}$$

Since the result is zero, the given function *u* is a solution of the differential equation.

The syntax for the **diff** command should be clear: "diff(expr,var)" differentiates the expression "expr" with respect to the variable "var". Here are some more examples:

```
> diff(x^3,x);
```

$$3\,x^2 \qquad \textbf{(2.1.13)}$$

```
> diff(x^3*y^2,y);
```

$$2\,x^3\,y \qquad \textbf{(2.1.14)}$$

As the previous example shows, **diff** can compute partial derivatives as well as ordinary derivatives. Consider the following function of two variables:

```
> unassign('x','y','a');
> w:=(x,y)->sin(y-a*x);
```

$$w := (x, y) \to \sin(y - a\,x) \qquad \textbf{(2.1.15)}$$

We have

```
> diff(w(x,y),x)+a*diff(w(x,y),y);
```

$$0 \qquad \textbf{(2.1.16)}$$

This shows that w is a solution of the PDE

$$\frac{\partial u}{\partial x} + a\frac{\partial u}{\partial y} = 0.$$

Is the same function a solution of the PDE

$$\frac{\partial^2 u}{\partial x^2} - a^2\frac{\partial^2 u}{\partial y^2} = 0?$$

To determine this, we must know how to compute higher order derivatives. One way is by simply iterating the **diff** command. For example, here is the second derivative of *w*(*x*,*y*) with respect to *x*:

```
> diff(diff(w(x,y),x),x);
```

$$\sin(-y + a\,x)\,a^2 \qquad \textbf{(2.1.17)}$$

However, there is a simpler way to obtain the same result:

```
> diff(w(x,y),x,x);
```

$$\sin(-y + a\,x)\,a^2 \qquad \textbf{(2.1.18)}$$

An alternate way to compute higher-order derivatives uses the "$" operator. The following command computes the fifth-order derivative of *w*(*x*,*y*) with respect to *x* twice and *y* three times:

```
> diff(w(x,y),x$2,y$3);
```

$$\cos(-y + a\,x)\,a^2 \qquad \textbf{(2.1.19)}$$

So here is the answer to the previous question: We have

```
> diff(w(x,y),x$2)-a^2*diff(w(x,y),y$2);
```

$$0 \qquad \textbf{(2.1.20)}$$

Since the result is zero, *w* also solves the above second-order PDE.

More about functions and derivatives

It is important to understand the difference between an expression and a function. In the previous example, *w*(*x*,*y*) is an expression (in the variables *x* and

y), while *w* itself is a function (of two variables). The **diff** command manipulates expressions, and it is not necessary to define a function to use it.

On the other hand, if I have defined a function, I cannot pass it directly to **diff**; I must evaluate it to form an expression. Here is the wrong way to do it:

```
> unassign('x');
> f:=x->x*sin(x);
```

$$f := x \to x \sin(x) \qquad \textbf{(2.2.1)}$$

```
> diff(f,x);
```

$$0 \qquad \textbf{(2.2.2)}$$

The result is zero because the variable "*x*" does not appear in the expression "*f*".

Here is the correct use of **diff**:

```
> diff(f(x),x);
```

$$\sin(x) + x\cos(x) \qquad \textbf{(2.2.3)}$$

The derivative of the function *f* is another function, and *Maple* can compute this other function directly using the **D** operator. Here is the derivative of *f*:

```
> D(f);
```

$$x \to \sin(x) + x\cos(x) \qquad \textbf{(2.2.4)}$$

Notice how the derivative of *f* is displayed as a mapping. This is because the derivative of *f* is a function, not an expression. On the other hand, I can evaluate the derivative function to get an expression:

```
> D(f)(x);
```

$$\sin(x) + x\cos(x) \qquad \textbf{(2.2.5)}$$

Partial derivatives can also be computed using the **D** operator. For example, consider the following function:

```
> unassign('x','y');
> w:=(x,y)->x^4*y^4+x*y^2;
```

$$w := (x, y) \to x^4 y^4 + x y^2 \qquad \textbf{(2.2.6)}$$

Here is the partial derivative of *w* with respect to the first input variable (which was called "*x*" above):

```
> D[1](w);
```

$$(x, y) \to 4 x^3 y^4 + y^2 \qquad \textbf{(2.2.7)}$$

Here is the partial derivative of *w* with respect to the second input variable:

```
> D[2](w);
```

$$(x, y) \to 4\, x^4\, y^3 + 2\, x\, y \qquad \textbf{(2.2.8)}$$

To compute higher-order partial derivatives, the indices of the variables can be listed in the brackets. For example, here is the second partial derivative with respect to y twice:

```
> D[2,2](w);
```

$$(x, y) \to 12\, x^4\, y^2 + 2\, x \qquad \textbf{(2.2.9)}$$

To differentiate repeatedly with respect to a given variable, use the "k$n" notation (thus D[1$4](f) computes the fourth partial derivative with respect to the first independent variable). Here is another way to give the previous command:

```
> D[2$2](w);
```

$$(x, y) \to 12\, x^4\, y^2 + 2\, x \qquad \textbf{(2.2.10)}$$

The following command computes

$$\frac{\partial^5 w}{\partial x^2 \partial y^3}.$$

```
> D[1$2, 2$3](w);
```

$$(x, y) \to 288\, x^2\, y \qquad \textbf{(2.2.11)}$$

If you have not already done so, now is a good time to try out the help facilities. You might start by entering "?diff" and then "?D" to compare the two differentiation commands.

Chapter 2: Models in one dimension

Section 2.1: Heat flow in a bar; Fourier's Law

Maple can compute both indefinite and definite integrals. The command for computing an indefinite integral (that is, an antiderivative) is exactly analogous to that for computing a derivative:

```
> unassign('x');
> int(sin(x),x);
```

$$-\cos(x) \qquad \textbf{(3.1.1)}$$

As this example shows, *Maple* does not add the "constant of integration". It simply returns one antiderivative (when possible). If the integrand is too

complicated, the integral is returned unevaluated:

```
> int(exp(cos(x)),x);
```

$$\int e^{\cos(x)}\,dx \qquad \textbf{(3.1.2)}$$

(Recall from calculus that some elementary functions do not have antiderivatives that can be expressed in terms of elementary functions.)

To compute a definite integral such as

$$\int_0^1 \sin(x)\,dx,$$

we must specify the interval of integration along with the integrand and the variable of integration. *Maple* has a special notation for specifying an interval or a range of values:

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

$$1-\cos(1) \qquad \textbf{(3.1.3)}$$

(Notice that **int**, like **diff**, operates on expressions, not functions.)

Maple has an "inert" version of the **int** command, which represents an integral as an expression without trying to evaluate it:

```
> Int(exp(cos(x)),x=0..1);
```

$$\int_0^1 e^{\cos(x)}\,dx \qquad \textbf{(3.1.4)}$$

The inert version, **Int**, is useful for evaluating integrals numerically (that is, approximately) when you do not want *Maple* to first try to find a symbolic result:

```
> evalf(Int(exp(cos(x)),x=0..1));
```

$$2.341574842 \qquad \textbf{(3.1.5)}$$

As this example shows, the combination of **evalf** and **Int** is useful for integrating functions for which no elementary antiderivative exists. (One could use **int** in place of **Int** in the previous example, but then *Maple* would waste time in first trying to find a symbolic result.)

As an example, I will use the commands for integration and differentiation to test Theorem 2.1 from the text. The theorem states that (under certain conditions)

$$\frac{\partial}{\partial x}\left(\int_c^d F(x,\,y)\,dy\right)=\int_c^d \frac{\partial F}{\partial x}(x,\,y)\,dy.$$

Here is a specific instance of the theorem:

```
> unassign('x','y');
```

```
> F:=(x,y)->x*y^3+x^2*y;
```

$$F:=(x,\,y)\to x\,y^3+x^2\,y \tag{3.1.6}$$

```
> unassign('c','d');
```

```
> diff(int(F(x,y),y=c..d),x);
```

$$\frac{1}{4}\,d^4-\frac{1}{4}\,c^4+x\,(d^2-c^2) \tag{3.1.7}$$

```
> int(diff(F(x,y),x),y=c..d);
```

$$\frac{1}{4}\,d^4-\frac{1}{4}\,c^4+x\,(d^2-c^2) \tag{3.1.8}$$

The two results are equal, as expected.

Solving simple BVPs by integration

Consider the following BVP

$$-\left(\frac{d^2u}{dx^2}\right)=1+x,\ 0<x,\ x<1,$$
$$u(0)=0,\ u(1)=0.$$

The solution can be found by two integrations (cf. Example 2.2 in the text). Remember, as I mentioned above, *Maple* does not add a constant of integration, so I do it explicitly. (I begin by clearing the variables I am going to use, in case I assigned values to any of them earlier.)

```
> unassign('x','u','C1','C2');
```

Here is the first integration:

```
> int(-(1+x),x)+C1;
```

$$-x-\frac{1}{2}\,x^2+C1 \tag{3.2.1}$$

And here is the second integration:

```
> int(%,x)+C2;
```

$$-\frac{1}{2}x^2-\frac{1}{6}x^3+C1\,x+C2 \qquad \textbf{(3.2.2)}$$

(Recall that the % symbol represents the last output.)

Now I have a common situation: I have an expression, and I would like to turn it into a function. *Maple* has a special function, called **unapply**, for doing this:

```
> u:=unapply(%,x);
```

$$u:=x\rightarrow -\frac{1}{2}x^2-\frac{1}{6}x^3+C1\,x+C2 \qquad \textbf{(3.2.3)}$$

Unapply takes an expression and one or more independent variables, and creates the functions mapping the variable(s) to the expression.

Here is another example of the use of **unapply**:

```
> f:=unapply(x*y^2,x,y);
```

$$f:=(x,y)\rightarrow x\,y^2 \qquad \textbf{(3.2.4)}$$

```
> f(2,3);
```

$$18 \qquad \textbf{(3.2.5)}$$

Returning to the BVP, I will now solve for the constants, using the *Maple* **solve** command:

```
> sols:=solve({u(0)=0,u(1)=0},{C1,C2});
```

$$sols:=\left\{C1=\frac{2}{3},\ C2=0\right\} \qquad \textbf{(3.2.6)}$$

The general form of the **solve** command is "solve(eqns,vars)", where eqns is an equation or a set of equations, and vars is an unknown or a set of unknowns. Notice that the equations are formed using the equals sign (whereas assignment uses the ":=" sign).

The above result has no effect on the values of C1 and C2; in particular, they do not now have the values 2/3 and 0, respectively:

```
> C1;
```

$$C1 \qquad \textbf{(3.2.7)}$$

```
> C2;
```

$$C2 \qquad \textbf{(3.2.8)}$$

I can cause *Maple* to make the assignment using the **assign** command:

```
> assign(sols);
> C1;
```

$$\frac{2}{3} \tag{3.2.9}$$

```
> C2;
```

$$0 \tag{3.2.10}$$

```
> u(x);
```

$$-\frac{1}{2}x^2-\frac{1}{6}x^3+\frac{2}{3}x \tag{3.2.11}$$

I now have the solution to the BVP. I will check it:

```
> -diff(u(x),x$2);
```

$$1+x \tag{3.2.12}$$

```
> u(0);
```

$$0 \tag{3.2.13}$$

```
> u(1);
```

$$0 \tag{3.2.14}$$

Both the differential equation and the boundary conditions are satisfied.

As another example, I will solve a BVP with a nonconstant coefficient:

$$-\frac{d}{dx}\left(\left(1+\frac{x}{2}\right)\frac{du}{dx}\right)=0,\ 0<x,\ x<1,$$
$$u(0)=20,\ u(1)=25.$$

Integrating once yields

$$\frac{\mathrm{d}}{\mathrm{d}x}u(x)=\frac{C1}{1+\frac{x}{2}}.$$

(It is easier to do this in my head than to type anything into *Maple*.) Integrating a second time yields

```
> unassign('C1','C2','x');
> int(C1/(1+x/2),x)+C2;
```

$$2\,C1\ln\left(1+\frac{1}{2}x\right)+C2 \tag{3.2.15}$$

```
> u:=unapply(%,x);
```

(3.2.16)

$$u := x \to 2\,C1 \ln\left(1 + \frac{1}{2}\,x\right) + C2 \qquad \textbf{(3.2.16)}$$

Now I solve for the constants of integration:

```
> solve({u(0)=20,u(1)=25},{C1,C2});
```

$$\left\{C1 = \frac{5}{2\ln\left(\frac{3}{2}\right)},\ C2 = 20\right\} \qquad \textbf{(3.2.17)}$$

```
> assign(%);
> u(x);
```

$$\frac{5\ln\left(1 + \frac{1}{2}\,x\right)}{\ln\left(\frac{3}{2}\right)} + 20 \qquad \textbf{(3.2.18)}$$

Check:

```
> -diff((1+x/2)*diff(u(x),x),x);
```

$$0 \qquad \textbf{(3.2.19)}$$

```
> u(0);
```

$$20 \qquad \textbf{(3.2.20)}$$

```
> u(1);
```

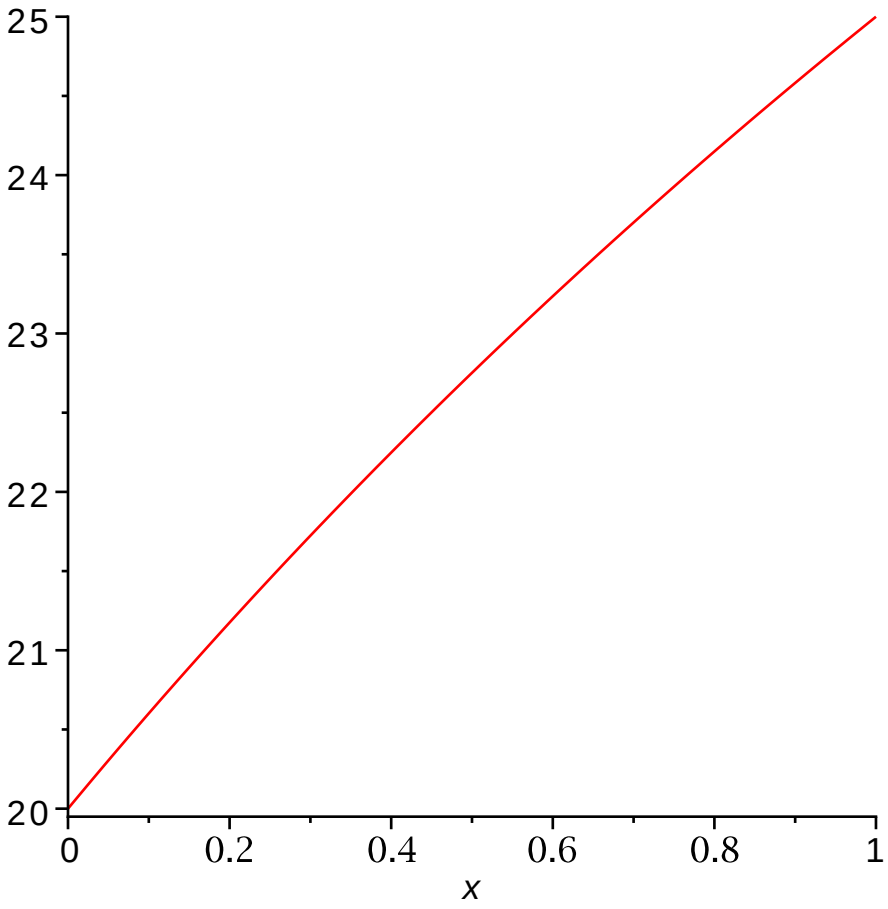
$$25 \qquad \textbf{(3.2.21)}$$

Thus *u* is the desired solution.

Simple plots

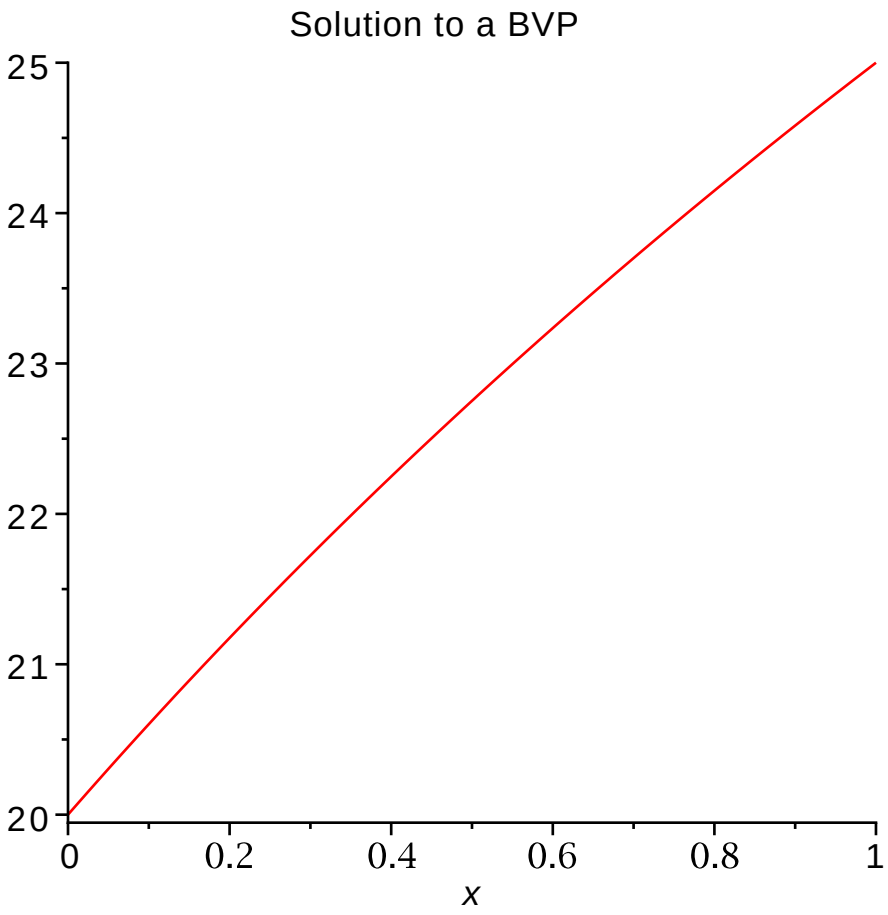
One of the most useful features of *Maple* is its ability to draw many kinds of graphs. Here I will show how to produce the graph of a function of one variable. The command is called **plot**, and we simply give it the expression to graph, the independent variable, and the interval. The following command produces the graph of the previous solution:

```
> plot(u(x),x=0..1);
```



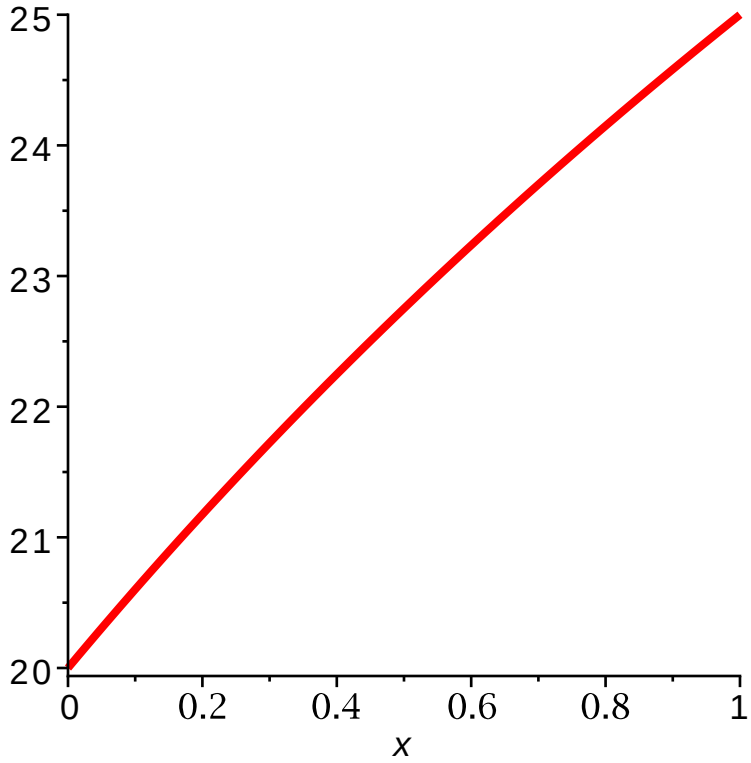
The plot command has various options; for example, you can add a title to a plot:

```
> plot(u(x),x=0..1,title="Solution to a BVP");
```



You can also increase the thickness of the curve using the "thickness=n" option. The default is n=0, and n must be an integer between 0 and 15.

```
> plot(u(x),x=0..1,thickness=3);
```



For more details about options to a plot, see ?plot[options].

Using the plot command, you can graph several functions in the same figure by listing the expressions in curly brackets. For example, suppose I wish to compare the solution computed above to the solution of

$$-\frac{d^2v}{dx^2} = 0, 0 < x, x < 1,$$
$$v(0) = 20, v(1) = 25.$$

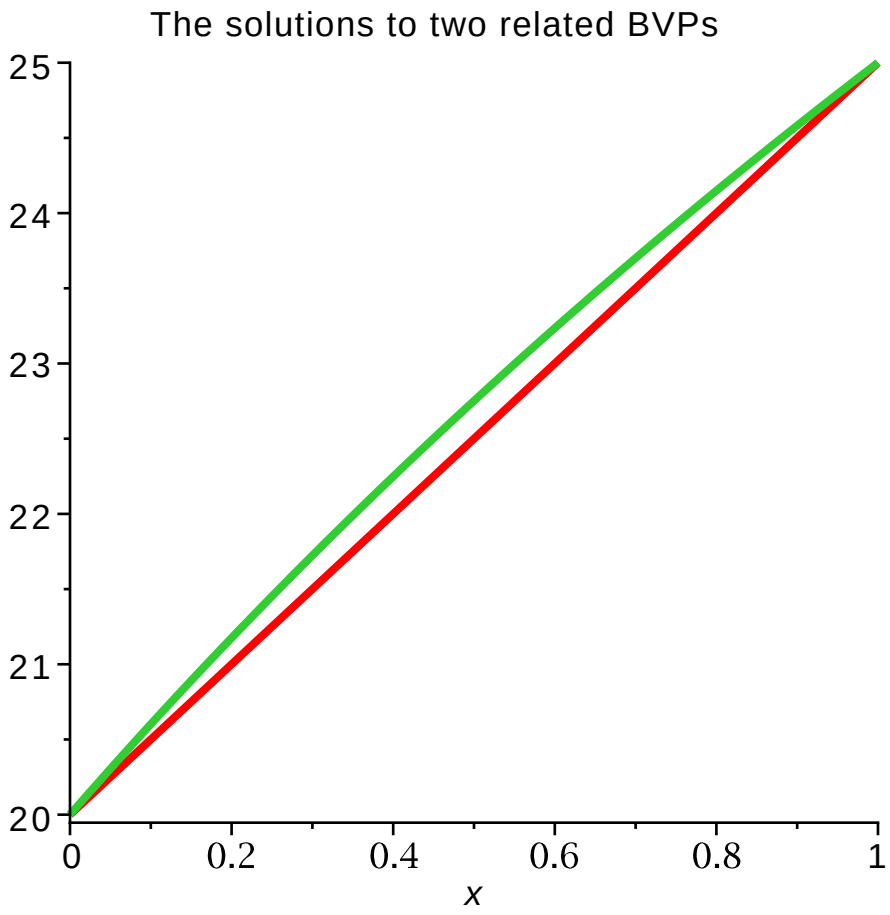
The solution is

```
> v:=x->20+5*x;
```

$$v := x \to 20 + 5\,x \qquad \textbf{(3.3.1)}$$

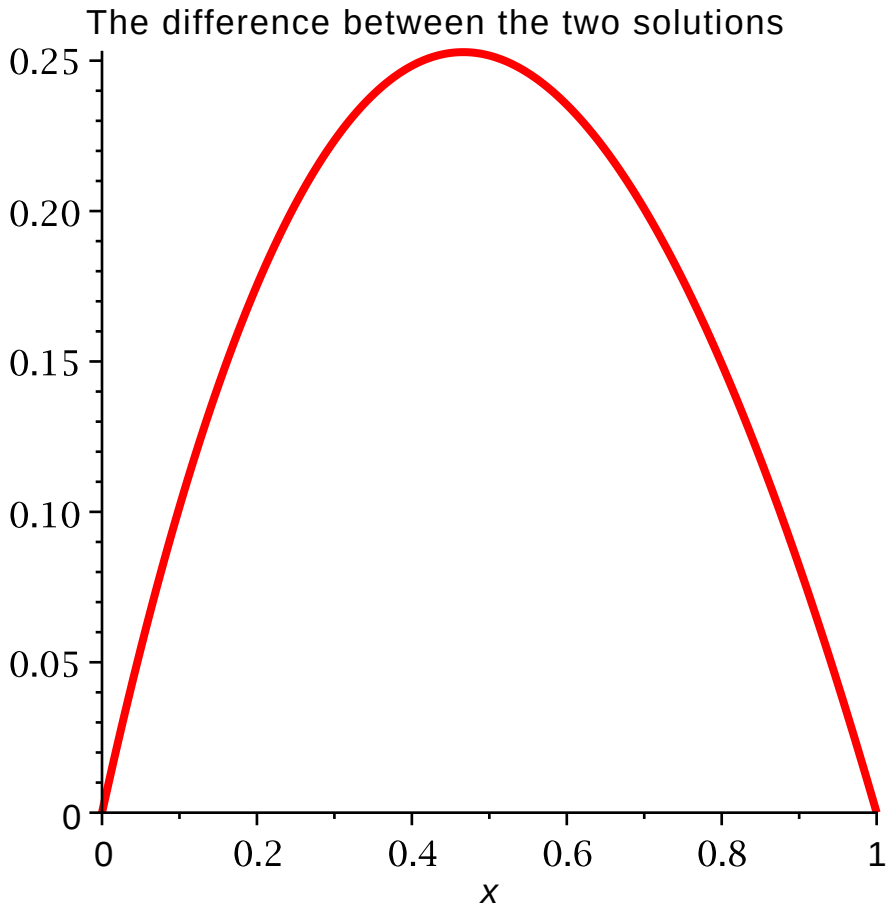
Here is a graph of the two solutions:

```
> plot({u(x),v(x)},x=0..1,title="The solutions to two related
  BVPs",thickness=3);
```



It is frequently useful to compare two functions by plotting their difference:

```
> plot(u(x)-v(x),x=0..1,title="The difference between the two
  solutions",thickness=3);
```



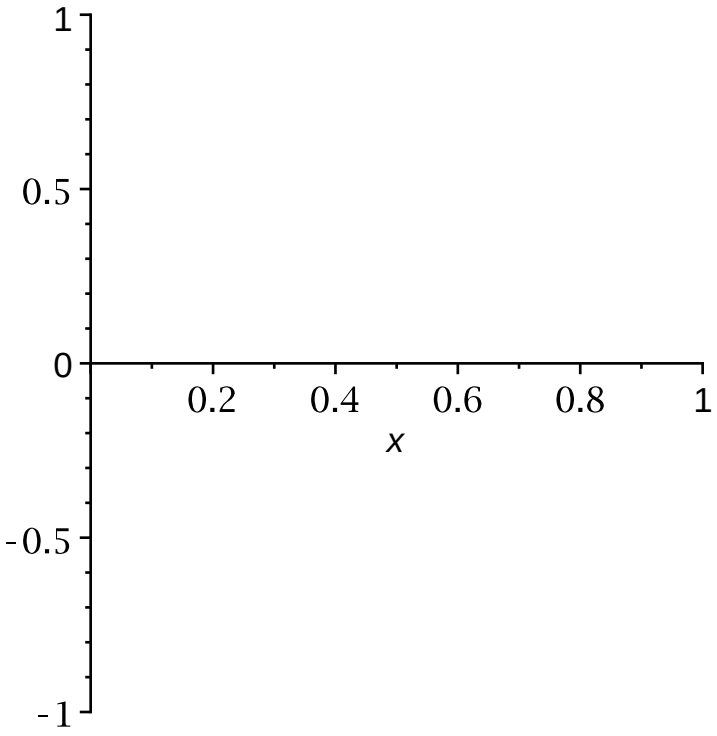
A common mistake

Consider the following attempt to graph a function:

```
> unassign('x');
> f:=x->sin(pi*x);
```

$$f := x \to \sin(\pi x) \qquad \textbf{(3.3.1.1)}$$

```
> plot(f(x),x=0..1);
```



Why did the curve not appear on the graph? The reason is that I did not define f correctly, and so *Maple* cannot evaluate the function:

```
> f(1.0);
```

$$\sin(1.0\,\pi) \qquad \textbf{(3.3.1.2)}$$

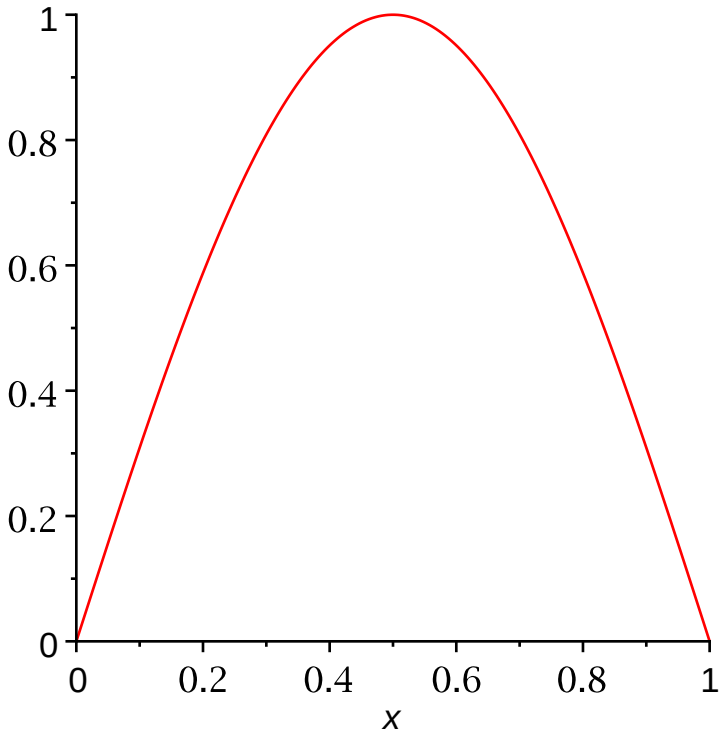
The problem is that I typed in "sin(pi*x)" rather than "sin(Pi*x)." The symbol "pi" is not defined and therefore *Maple* does not know its value.

Here is the correct command:

```
> f:=x->sin(Pi*x);
```

$$f := x \to \sin(\pi\,x) \qquad \textbf{(3.3.1.3)}$$

```
> plot(f(x),x=0..1);
```



A common mistake is to try to graph an expression that contains an indeterminant parameter.

Chapter 3: Essential linear algebra

Section 3.1: Linear systems as linear operator equations

Maple will manipulate matrices and vectors, and perform the usual computations of linear algebra. Both symbolic and numeric (that is, floating point) computations are supported. In order to take advantage of these capabilities, you must load the **LinearAlgebra** package by giving the following command:

```
> with(LinearAlgebra);
```

(4.1.1)

[*&x*, *Add*, *Adjoint*, *BackwardSubstitute*, *BandMatrix*, *Basis*, *BezoutMatrix*, *BidiagonalForm*, *BilinearForm*, *CARE*, *CharacteristicMatrix*, *CharacteristicPolynomial*, *Column*, *ColumnDimension*, *ColumnOperation*, *ColumnSpace*, *CompanionMatrix*, *ConditionNumber*, *ConstantMatrix*, *ConstantVector*, *Copy*, *CreatePermutation*, *CrossProduct*, *DARE*, *DeleteColumn*, *DeleteRow*, *Determinant*, *Diagonal*, *DiagonalMatrix*, *Dimension*, *Dimensions*,

DotProduct, *EigenConditionNumbers*, *Eigenvalues*, *Eigenvectors*, *Equal*, *ForwardSubstitute*, *FrobeniusForm*, *GaussianElimination*, *GenerateEquations*, *GenerateMatrix*, *Generic*, *GetResultDataType*, *GetResultShape*, *GivensRotationMatrix*, *GramSchmidt*, *HankelMatrix*, *HermiteForm*, *HermitianTranspose*, *HessenbergForm*, *HilbertMatrix*, *HouseholderMatrix*, *IdentityMatrix*, *IntersectionBasis*, *IsDefinite*, *IsOrthogonal*, *IsSimilar*, *IsUnitary*, *JordanBlockMatrix*, *JordanForm*, *KroneckerProduct*, *LA_Main*, *LUDecomposition*, *LeastSquares*, *LinearSolve*, *LyapunovSolve*, *Map*, *Map2*, *MatrixAdd*, *MatrixExponential*, *MatrixFunction*, *MatrixInverse*, *MatrixMatrixMultiply*, *MatrixNorm*, *MatrixPower*, *MatrixScalarMultiply*, *MatrixVectorMultiply*, *MinimalPolynomial*, *Minor*, *Modular*, *Multiply*, *NoUserValue*, *Norm*, *Normalize*, *NullSpace*, *OuterProductMatrix*, *Permanent*, *Pivot*, *PopovForm*, *QRDecomposition*, *RandomMatrix*, *RandomVector*, *Rank*, *RationalCanonicalForm*, *ReducedRowEchelonForm*, *Row*, *RowDimension*, *RowOperation*, *RowSpace*, *ScalarMatrix*, *ScalarMultiply*, *ScalarVector*, *SchurForm*, *SingularValues*, *SmithForm*, *StronglyConnectedBlocks*, *SubMatrix*, *SubVector*, *SumBasis*, *SylvesterMatrix*, *SylvesterSolve*, *ToeplitzMatrix*, *Trace*, *Transpose*, *TridiagonalForm*, *UnitVector*, *VandermondeMatrix*, *VectorAdd*, *VectorAngle*, *VectorMatrixMultiply*, *VectorNorm*, *VectorScalarMultiply*, *ZeroMatrix*, *ZeroVector*, *Zip*]

Whenever you load a package using the **with** command, *Maple* echos the names of all command defined by the package. (As with other *Maple* output, you can suppress this output by ending the **with** command with a colon instead of a semicolon.)

For example, here is the command for defining a vector:

```
> x:=Vector([1,0,-3]);
```

$$x := \begin{bmatrix} 1 \\ 0 \\ -3 \end{bmatrix} \qquad \textbf{(4.1.2)}$$

Having defined a vector, I can access its components using square brackets:

```
> x[2];
```

$$0 \tag{4.1.3}$$

A matrix is specified as a list of row vectors, using the **Matrix** command:

```
> A:=Matrix([[1,2,3],[4,5,6],[7,8,9]]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \tag{4.1.4}$$

(Notice the double square brackets; the input is a list of row vectors, each of which is a list of numbers.)

You can compute the transpose of a matrix:

```
> Transpose(A);
```

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \tag{4.1.5}$$

You can also extract the entries of the matrix:

```
> A[1,3];
```

$$3 \tag{4.1.6}$$

```
> A[3,3];
```

$$9 \tag{4.1.7}$$

Matrix-vector and matrix-matrix multiplication are indicated by the "." (dot) operator:

```
> A.x;
```

$$\begin{bmatrix} -8 \\ -14 \\ -20 \end{bmatrix} \tag{4.1.8}$$

```
> A.A;
```

(4.1.9)

$$\begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix} \quad \textbf{(4.1.9)}$$

Of course, you can only multiply two matrices if their sizes are compatible:

```
> B:=Matrix([[1,2],[3,4],[5,6]]);
```

$$B := \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad \textbf{(4.1.10)}$$

```
> A.B;
```

$$\begin{bmatrix} 22 & 28 \\ 49 & 64 \\ 76 & 100 \end{bmatrix} \quad \textbf{(4.1.11)}$$

```
> B.A;
```

Error, (in LinearAlgebra:-Multiply) first matrix column dimension (2) <> second matrix row dimension (3)

(By the way, the symbol "<>" that appears in the above error message is the "not-equals" symbol.)

Maple provides a special mechanism for creating vectors or matrices whose entries can be described by a function. For example, consider the vector whose ith entry is i^2. Here is how you create it automatically:

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

$$f := i \rightarrow i^2 \quad \textbf{(4.1.12)}$$

```
> y:=Vector(10,f);
```

(4.1.13)

$$y := \begin{bmatrix} 1 \\ 4 \\ 9 \\ 16 \\ 25 \\ 36 \\ 49 \\ 64 \\ 81 \\ 100 \end{bmatrix} \quad \textbf{(4.1.13)}$$

The syntax of this version of the **Vector** command requires the size of the vector and a function that computes the ith component from the index i. Thus "Vector (n,f)" creates a vector with n components, $f(1)$, $f(2)$, ..., $f(n)$.

You do not actually have to perform the first step above of defining the function (that is, of giving the function a name), since you can describe a function using the -> notation:

```
> z:=Vector(10,i->i^2);
```

$$z := \begin{bmatrix} 1 \\ 4 \\ 9 \\ 16 \\ 25 \\ 36 \\ 49 \\ 64 \\ 81 \\ 100 \end{bmatrix} \quad \textbf{(4.1.14)}$$

The same technique can be used to define a matrix; the only difference is that a function of two indices is required. Here is a famous matrix (the Hilbert matrix):

```
> H:=Matrix(5,5,(i,j)->1/(i+j-1));
```

$$H := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{bmatrix} \quad \textbf{(4.1.15)}$$

The form of the **Matrix** command just used is "Matrix(m,n,f)", which creates an m by n matrix whose i, j entry is $f(i, j)$.

Alternate notation for entering vectors and matrices

Here is an alternate, simplified syntax for defining a vector:

```
> x:=<1,2,3>;
```

$$x := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \textbf{(4.1.1.1)}$$

(Notice the use of the less than and great than symbols to create the "angled brackets".) Since this is simpler than using the **Vector** command, I will use it in the remainder of the tutorial.

Using the above notation, *Maple* allows us to enter matrices by columns, rather than by rows:

```
> A:=<<1,2>|<3,4>>;
```

$$A := \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \quad \textbf{(4.1.1.2)}$$

Notice the use of the vertical bar "|" to separate the columns of the matrix; this notation is also common in written mathematics. Just as a review, here is the command for entering the above matrix A by rows:

```
> A:=Matrix([[1,3],[2,4]]);
```

$$A := \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \quad \textbf{(4.1.1.3)}$$

Section 3.2 Existence and uniqueness of solutions to Ax=b

Maple can find a basis for the null space of a matrix:

```
> A:=Matrix([[1,2,3],[4,5,6],[7,8,9]]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \textbf{(4.2.1)}$$

```
> NullSpace(A);
```

$$\left\{ \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \right\} \quad \textbf{(4.2.2)}$$

In this example, the null space is one-dimensional, so the basis contains a single vector. If we wish to refer to the basis vector itself (instead of the set containing the vector), we can use the index notation:

```
> y:=%[1];
```

$$y := \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \quad \textbf{(4.2.3)}$$

Here I check that the result is correct:

```
> A.y;
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \textbf{(4.2.4)}$$

If the given matrix is nonsingular, then its null space is trivial (that is, contains only the zero vector) and therefore does not have a basis. In this case, the **NullSpace** command returns an empty list:

```
> A:=Matrix([[1,2,1],[-3,0,0],[1,2,2]]);
```

$$A := \begin{bmatrix} 1 & 2 & 1 \\ -3 & 0 & 0 \\ 1 & 2 & 2 \end{bmatrix} \tag{4.2.5}$$

```
> NullSpace(A);
```

$$\{\} \tag{4.2.6}$$

Here is an example with a two-dimensional null space:

```
> C:=Matrix([[1,3,-1,2],[0,1,4,2],[2,7,2,6],[1,4,3,4]]);
```

$$C := \begin{bmatrix} 1 & 3 & -1 & 2 \\ 0 & 1 & 4 & 2 \\ 2 & 7 & 2 & 6 \\ 1 & 4 & 3 & 4 \end{bmatrix} \tag{4.2.7}$$

```
> NullSpace(C);
```

$$\left\{ \begin{bmatrix} 4 \\ -2 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 13 \\ -4 \\ 1 \\ 0 \end{bmatrix} \right\} \tag{4.2.8}$$

Maple can find the inverse of a matrix, assuming, of course, that the matrix is nonsingular:

```
> MatrixInverse(A);
```

$$\begin{bmatrix} 0 & -\frac{1}{3} & 0 \\ 1 & \frac{1}{6} & -\frac{1}{2} \\ -1 & 0 & 1 \end{bmatrix} \tag{4.2.9}$$

```
> C:=Matrix([[1,0,1],[1,2,1],[0,1,0]]);
```

$$C := \begin{bmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.2.10}$$

```
> MatrixInverse(C);
Error, (in LinearAlgebra:-MatrixInverse) singular matrix
```

The matrix inverse can be used to solve a square system $Ax = b$, since the solution (when A is nonsingular) is $x = A^{-1}b$. However, it is more efficient to use the **LinearSolve** command, which applies Gaussian elimination directly:

```
> A:=Matrix([[1,0,1],[0,1,1],[1,1,1]]);
```

$$A := \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \textbf{(4.2.11)}$$

```
> b:=<1,0,2>;
```

$$b := \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \quad \textbf{(4.2.12)}$$

```
> x:=LinearSolve(A,b);
```

$$x := \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix} \quad \textbf{(4.2.13)}$$

Any nonsingular matrix times its inverse yields the identity matrix:

```
> A.MatrixInverse(A);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(4.2.14)}$$

Maple has a command for creating an identity matrix of a given size:

```
> Id:=IdentityMatrix(3);
```

$$Id := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(4.2.15)}$$

However, you cannot call your identity matrix I, as might seem natural, since, in *Maple*, **I** represents the imaginary unit (the square root of negative one):

```
> I^2;
```

$$-1 \quad \textbf{(4.2.16)}$$

Section 3.3: Basis and dimension

In this section, I will further demonstrate some of the capabilities of *Maple* by repeating some of the examples from Section 3.3 of the text.

Example 3.25

Consider the three vectors v1, v2, v3 defined as follows:

```
> v1:=<1/sqrt(3),1/sqrt(3),1/sqrt(3)>;
```

$$v1 := \begin{bmatrix} \frac{1}{3}\sqrt{3} \\ \frac{1}{3}\sqrt{3} \\ \frac{1}{3}\sqrt{3} \end{bmatrix} \qquad \textbf{(4.3.1.1)}$$

```
> v2:=<1/sqrt(2),0,-1/sqrt(2)>;
```

$$v2 := \begin{bmatrix} \frac{1}{2}\sqrt{2} \\ 0 \\ -\frac{1}{2}\sqrt{2} \end{bmatrix} \qquad \textbf{(4.3.1.2)}$$

```
> v3:=<1/sqrt(6),-2/sqrt(6),1/sqrt(6)>;
```

$$v3 := \begin{bmatrix} \frac{1}{6}\sqrt{6} \\ -\frac{1}{3}\sqrt{6} \\ \frac{1}{6}\sqrt{6} \end{bmatrix} \qquad \textbf{(4.3.1.3)}$$

I can use the "." (dot) operator to compute the ordinary dot product of two vectors:

```
> v1.v2;
```

$$0 \qquad \textbf{(4.3.1.4)}$$

```
> v1.v3;
```

$$0 \qquad \textbf{(4.3.1.5)}$$

```
> v2.v3;
```

$$0 \qquad \textbf{(4.3.1.6)}$$

Example 3.27

I will verify that the following three quadratic polynomials form a basis for P_2, the space of all polynomials of degree 2 or less.

```
> unassign('x');
> p1:=x->1;
```

$$p1 := x \to 1 \qquad \textbf{(4.3.2.1)}$$

```
> p2:=x->x-1/2;
```

$$p2 := x \to x - \frac{1}{2} \qquad \textbf{(4.3.2.2)}$$

```
> p3:=x->x^2-x+1/6;
```

$$p3 := x \to x^2 - x + \frac{1}{6} \qquad \textbf{(4.3.2.3)}$$

Suppose $q(x) = ax^2 + bx + c$ is an arbitrary quadratic polynomial:

```
> unassign('x','a','b','c');
> q:=x->a*x^2+b*x+c;
```

$$q := x \to a\,x^2 + b\,x + c \qquad \textbf{(4.3.2.4)}$$

(Notice how I clear any values that a, b, c might have, just to be sure. I want these to be indeterminate parameters.) Now I will try to write q in terms of p1, p2, and p3:

```
> unassign('c1','c2','c3');
> q(x)-(c1*p1(x)+c2*p2(x)+c3*p3(x));
```

$$a\,x^2 + b\,x + c - c1 - c2\left(x - \frac{1}{2}\right) - c3\left(x^2 - x + \frac{1}{6}\right) \qquad \textbf{(4.3.2.5)}$$

Next, I need to gather like terms in the above expression, which is accomplished with the **collect** command:

```
> collect(%,x);
```

$$(a - c3)\,x^2 + (b - c2 + c3)\,x + \frac{1}{2}\,c2 + c - c1 - \frac{1}{6}\,c3 \qquad \textbf{(4.3.2.6)}$$

Now I extract the coefficients of the above polynomial, set them equal to zero,

and solve for c1, c2, c3:

```
> solve({coeff(%,x,0)=0,coeff(%,x,1)=0,coeff(%,x,2)=0},{c1,
  c2,c3});
```

$$\left\{c1 = \frac{1}{2}\, b + \frac{1}{3}\, a + c,\ c2 = b + a,\ c3 = a\right\} \qquad \textbf{(4.3.2.7)}$$

Since there is a unique solution c1, c2, c3 for any a, b, c, that is, for any quadratic polynomial, this shows that {p1,p2,p3} is a basis for P_2.

Example

Here is a final example. Consider the following three vectors in R^3:

```
> u1:=<1,0,2>:
> u2:=<0,1,1>:
> u3:=<1,2,-1>:
```

I will verify that {u1,u2,u3} is a basis for R^3, and express the vector

```
> x:=<8,2,-4>;
```

$$x := \begin{bmatrix} 8 \\ 2 \\ -4 \end{bmatrix} \qquad \textbf{(4.3.3.1)}$$

in terms of the basis. As discussed in the text, {u1,u2,u3} is linearly independent if and only if the matrix A whose columns are u1, u2, u3 is nonsingular. Here I create the matrix A, using the syntax explained earlier for entering a matrix by columns:

```
> A:=<u1|u2|u3>;
```

$$A := \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 2 & 1 & -1 \end{bmatrix} \qquad \textbf{(4.3.3.2)}$$

Now I can verify that the matrix A is nonsingular by computing its determinant:

```
> Determinant(A);
```

$$-5 \qquad \textbf{(4.3.3.3)}$$

Since the determinant of A is nonzero, A is nonsingular. Note that, in general, computing the determinant is not a good way to decide if a matrix is nonsingular. If the matrix is large and the entries are floating point numbers, the result is likely to be misleading. A better way is to use the **Rank** command:

```
> Rank(A);
```

$$3 \qquad \textbf{(4.3.3.4)}$$

Since *A* has full rank, the matrix is nonsingular.

I can now express x as a linear combination of u1, u2, u3 by solving $Ac = x$:

```
> c:=LinearSolve(A,x);
```

$$c := \begin{bmatrix} \frac{18}{5} \\ -\frac{34}{5} \\ \frac{22}{5} \end{bmatrix} \qquad \textbf{(4.3.3.5)}$$

Finally, I will check the result:

```
> c[1]*u1+c[2]*u2+c[3]*u3;
```

$$\begin{bmatrix} 8 \\ 2 \\ -4 \end{bmatrix} \qquad \textbf{(4.3.3.6)}$$

```
> x;
```

$$\begin{bmatrix} 8 \\ 2 \\ -4 \end{bmatrix} \qquad \textbf{(4.3.3.7)}$$

Section 3.4: Orthogonal bases and projection

I have already shown to compute dot products and verify that vectors are orthogonal. For example, consider the vectors

```
> v1:=<1/sqrt(3),1/sqrt(3),1/sqrt(3)>:
> v2:=<1/sqrt(2),0,-1/sqrt(2)>:
> v3:=<1/sqrt(6),-2/sqrt(6),1/sqrt(6)>:
```

These vectors are orthogonal:

```
> v1.v2;
```

$$0 \tag{4.4.1}$$

```
> v1.v3;
```

$$0 \tag{4.4.2}$$

```
> v2.v3;
```

$$0 \tag{4.4.3}$$

They are also normalized:

```
> v1.v1;
```

$$1 \tag{4.4.4}$$

```
> v2.v2;
```

$$1 \tag{4.4.5}$$

```
> v3.v3;
```

$$1 \tag{4.4.6}$$

Therefore, I can easily express any vector as a linear combination of these three vectors, which form an orthonormal basis for R^3:

```
> unassign('a','b','c');
> x:=<a,b,c>;
```

$$x := \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{4.4.7}$$

```
> y:=(v1.x)*v1+(v2.x)*v2+(v3.x)*v3;
```

$$y := \left[\left[\frac{1}{3}\left(\frac{1}{3}\sqrt{3}\,a+\frac{1}{3}\sqrt{3}\,b+\frac{1}{3}\sqrt{3}\,c\right)\sqrt{3}+\frac{1}{2}\left(\frac{1}{2}\sqrt{2}\,a-\frac{1}{2}\sqrt{2}\,c\right)\sqrt{2}+\frac{1}{6}\left(\frac{1}{6}\sqrt{6}\,a-\frac{1}{3}\sqrt{6}\,b+\frac{1}{6}\sqrt{6}\,c\right)\sqrt{6}\right], \tag{4.4.8}$$

$$\left[\frac{1}{3}\left(\frac{1}{3}\sqrt{3}\,a+\frac{1}{3}\sqrt{3}\,b+\frac{1}{3}\sqrt{3}\,c\right)\sqrt{3}-\frac{1}{3}\left(\frac{1}{6}\sqrt{6}\,a-\frac{1}{3}\sqrt{6}\,b+\frac{1}{6}\sqrt{6}\,c\right)\sqrt{6}\right],$$

$$\left[\frac{1}{3}\left(\frac{1}{3}\sqrt{3}\,a+\frac{1}{3}\sqrt{3}\,b+\frac{1}{3}\sqrt{3}\,c\right)\sqrt{3}-\frac{1}{2}\left(\frac{1}{2}\sqrt{2}\,a-\frac{1}{2}\sqrt{2}\,c\right)\sqrt{2}\right.$$

$$+\frac{1}{6}\left(\frac{1}{6}\sqrt{6}\,a-\frac{1}{3}\sqrt{6}\,b+\frac{1}{6}\sqrt{6}\,c\right)\sqrt{6}\Bigg]\Bigg]$$

Notice that *Maple* did not produce the vector x, as expected. However, before we conclude that I (or *Maple*) made a mistake, we should ask it to simplify the result as much as possible:

```
> simplify(%);
```

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{4.4.9}$$

The **simplify** command causes the *Maple* kernel to apply a number of algebraic transformations to the input, in hopes of finding a simpler form. We will often find this command useful.

Working with the L^2 inner product

The L^2 inner product is not provided in *Maple* (unlike the dot product), so it is convenient to define a function implementing it. Suppose we are working on the interval $[0, 1]$. I define

```
> l2ip:=(f,g)->int(f(x)*g(x),x=0..1);
```

$$l2ip := (f, g) \to \int_0^1 f(x)\, g(x)\, \mathrm{d}x \tag{4.4.1.1}$$

For convenience, I also define a function implementing the L^2 norm:

```
> l2norm:=f->sqrt(l2ip(f,f));
```

$$l2norm := f \to \sqrt{l2ip(f, f)} \tag{4.4.1.2}$$

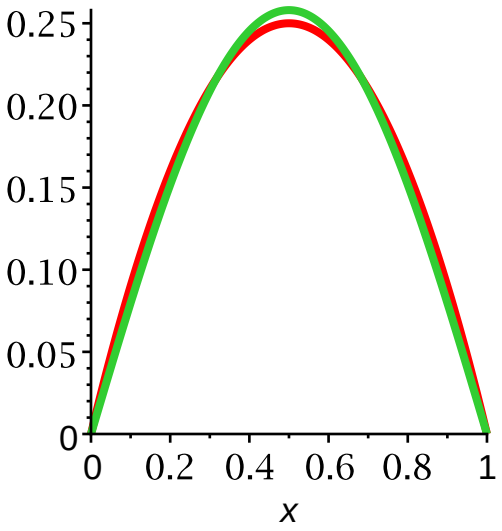
Example 3.35

Now consider the following two functions:

```
> unassign('x');
> f:=x->x*(1-x):
> g:=x->8/Pi^3*sin(Pi*x):
```

The following graph shows that the two function are quite similar:

```
> plot({f(x),g(x)},x=0..1,thickness=3);
```



By how much do the two functions differ? One way to answer this question is to compute the relative difference in the L^2 norm:

```
> l2norm(f-g)/l2norm(f);
```

$$\frac{\sqrt{-960+\pi^6}}{\pi^3} \qquad \textbf{(4.4.2.1)}$$

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

$$0.03801297659 \qquad \textbf{(4.4.2.2)}$$

The difference is less than 4%.

To further illustrate the capabilities of *Maple*, I will work through two more examples from Section 3.4.

Example 3.37

The data given in this example can be stored in two vectors:

```
> x:=<0.1,0.3,0.4,0.75,0.9>:
> y:=<1.7805,2.2285,2.3941,3.2226,3.5697>:
```

A useful command for working with discrete data like this is **pointplot**. It is part of the **plots** package. Before using pointplot, you must either load the entire plots package (using "with(plots)") or simply load pointplot itself as follows:

```
> with(plots,pointplot):
```

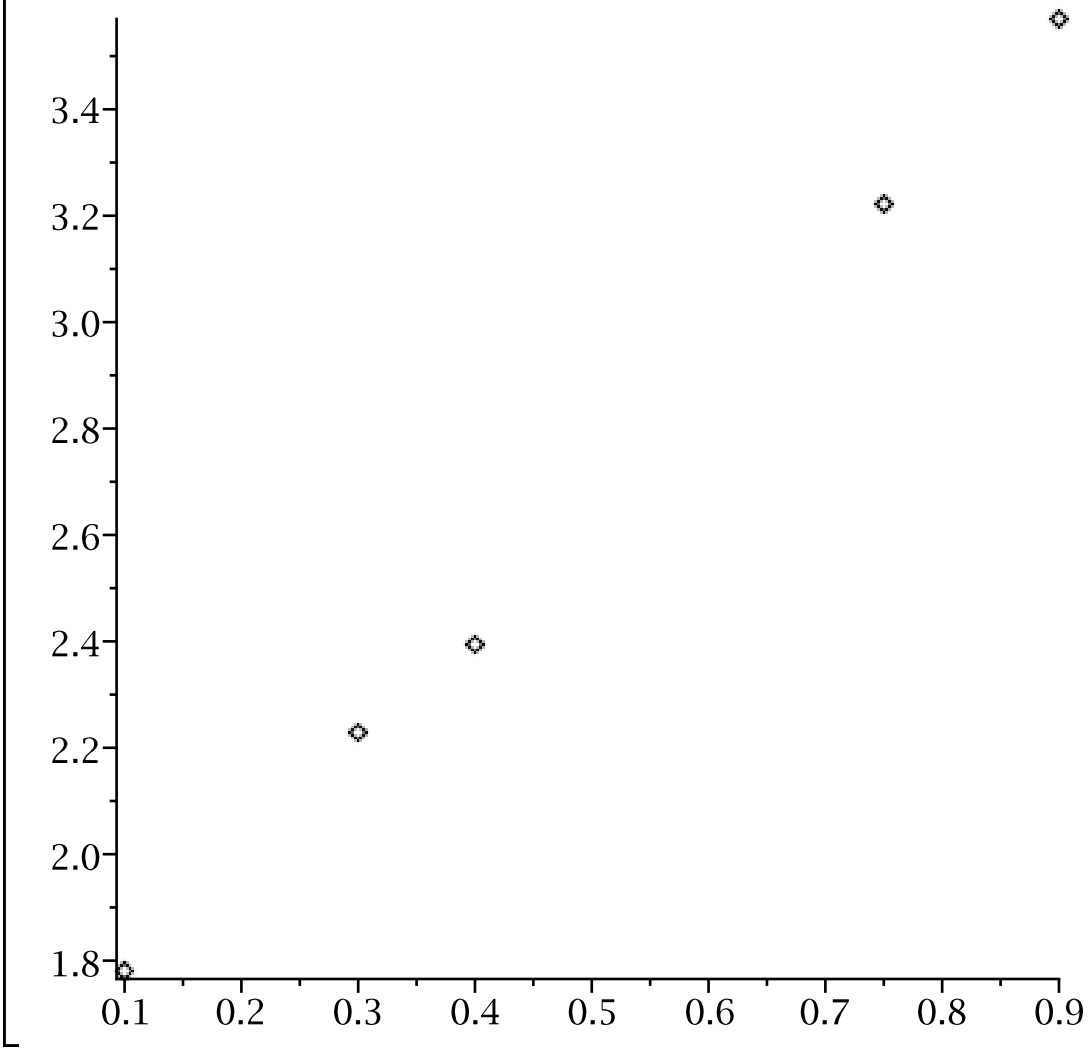
We must also make a list of points from the data; one way to do this is to create a matrix whose columns are x and y:

```
> d:=<x|y>;
```

$$d := \begin{bmatrix} 0.1 & 1.7805 \\ 0.3 & 2.2285 \\ 0.4 & 2.3941 \\ 0.75 & 3.2226 \\ 0.9 & 3.5697 \end{bmatrix} \quad \textbf{(4.4.3.1)}$$

Here is the pointplot command. Notice the use of the "symbolsize" option to make the markers larger.

```
> pointplot(d,symbolsize=15);
```



Now I will compute the first-degree polynomial $f(x) = mx + b$ that best fits this data. The Gram matrix and the right hand side of the normal equations

are computed as follows:

```
> e:=<1,1,1,1,1>;
```

$$e := \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{4.4.3.2}$$

```
> G:=Matrix([[x.x,x.e],[e.x,e.e]]);
```

$$G := \begin{bmatrix} 1.632500000000000006 & 2.450000000000000018 \\ 2.450000000000000018 & 5 \end{bmatrix} \tag{4.4.3.3}$$

```
> z:=<x.y,e.y>;
```

$$z := \begin{bmatrix} 7.43391999999999964 \\ 13.1953999999999994 \end{bmatrix} \tag{4.4.3.4}$$

Now I can solve for the coefficients in the best approximation:

```
> c:=LinearSolve(G,z);
```

$$c := \begin{bmatrix} 2.24114351851852 \\ 1.54091967592593 \end{bmatrix} \tag{4.4.3.5}$$

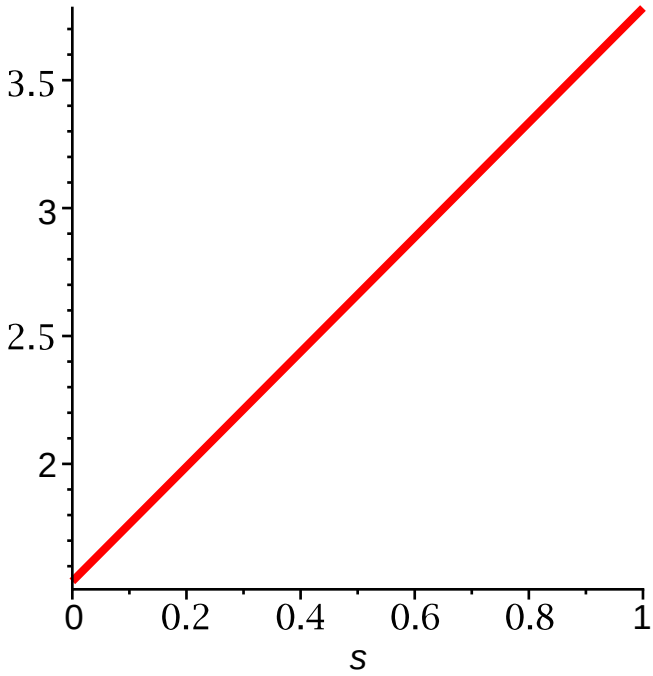
The solution is

```
> l:=s->c[1]*s+c[2];
```

$$l := s \rightarrow c_1\, s + c_2 \tag{4.4.3.6}$$

Here is a graph of the best fit polynomial:

```
> plot(l(s),s=0..1,thickness=3);
```



If we want to graph the line and the data on the same figure, then we have to use the **display** command, which displays two or more plots together (we have to use a special command because one plot is produced by **pointplot**, and the other by **plot**). The **display** command is part of the **plots** package, and must be loaded as follows:

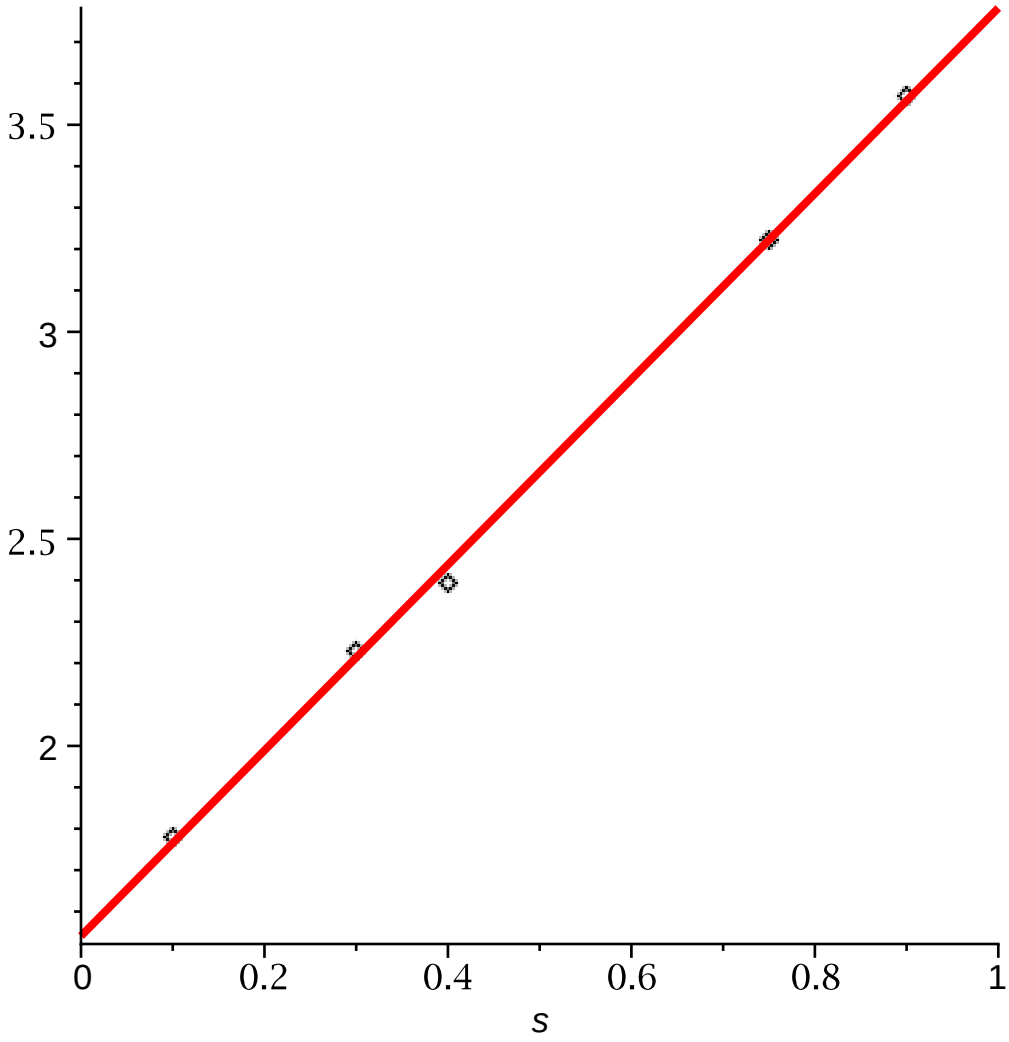
```
> with(plots,display):
```

Now I will recreate the plots, assigned the results to variables so that I can refer to them below. Notice that I use a colon to suppress the output. This is desirable, since I do not wish to see the plots separately, but only together. But, in fact, when a plot is assigned to a variable, *Maple* does not display the plot anyway, but rather the data structure representing it.

```
> plot1:=pointplot(d,symbolsize=15):
> plot2:=plot(l(s),s=0..1,thickness=3):
```

Now I can invoke the **display** command:

```
> display([plot1,plot2]);
```



The fit is not bad.

Example 3.38

In this example, I will compute the best quadratic approximation to the function $f(x) = \mathrm{e}^x$ on the interval $[0, 1]$. Here are the standard basis functions for the space P^2:

```
> unassign('x');
> p1:=x->1:
> p2:=x->x:
> p3:=x->x^2:
```

I will now compute the Gram matrix and the right hand side of the normal equations. Notice that the function $f(x) = \mathrm{e}^x$ is named **exp**.

```
> G:=Matrix([[l2ip(p1,p1),l2ip(p2,p1),l2ip(p3,p1)],[l2ip(p1,
```

```
p2),l2ip(p2,p2),l2ip(p3,p2)],[l2ip(p1,p3),l2ip(p2,p3),l2ip
(p3,p3)]]);
```

$$G := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix} \quad \textbf{(4.4.4.1)}$$

```
> b:=<l2ip(p1,exp),l2ip(p2,exp),l2ip(p3,exp)>;
```

$$b := \begin{bmatrix} -1+e \\ 1 \\ -2+e \end{bmatrix} \quad \textbf{(4.4.4.2)}$$

I now solve the normal equations and find the best quadratic approximation:

```
> c:=LinearSolve(G,b);
```

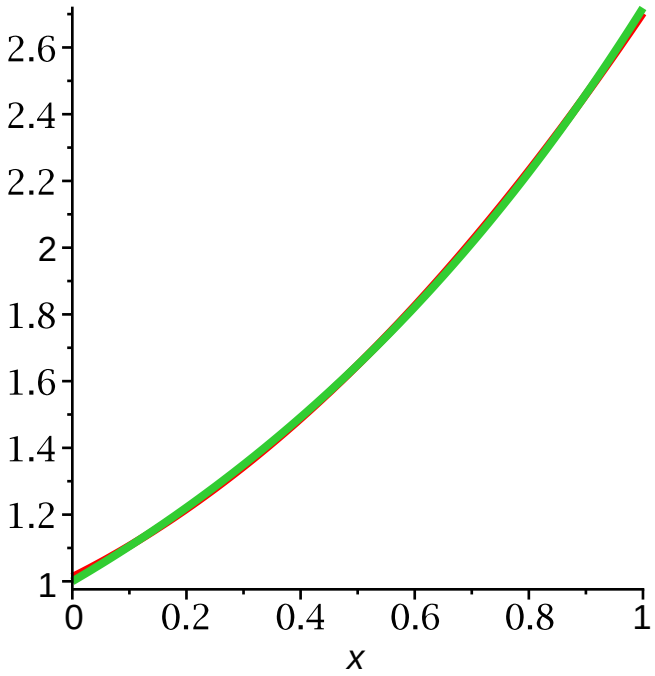
$$c := \begin{bmatrix} -105+39\,e \\ 588-216\,e \\ -570+210\,e \end{bmatrix} \quad \textbf{(4.4.4.3)}$$

```
> q:=x->c[1]*p1(x)+c[2]*p2(x)+c[3]*p3(x);
```

$$q := x \to c_1\, p1(x) + c_2\, p2(x) + c_3\, p3(x) \quad \textbf{(4.4.4.4)}$$

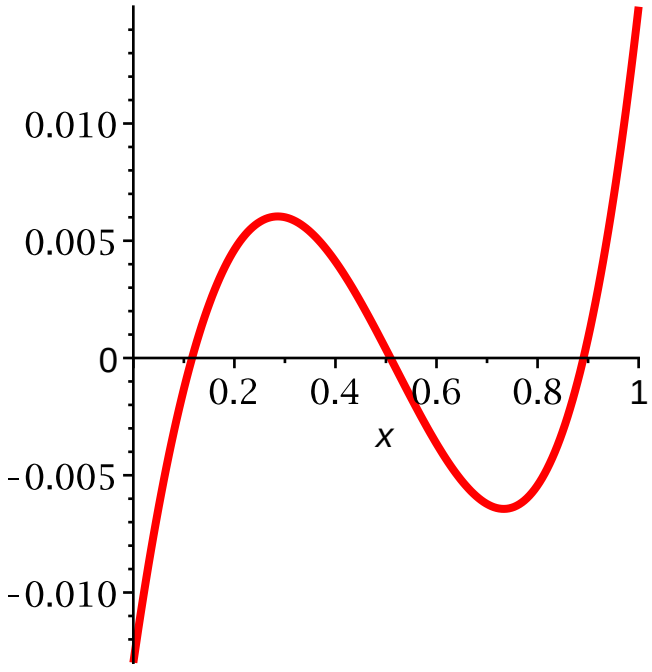
One way to judge the goodness of fit is with a plot:

```
> plot({exp(x),q(x)},x=0..1,thickness=3);
```



The fit is quite good, so it is more informative to graph the error:

```
> plot(exp(x)-q(x),x=0..1,thickness=3);
```



We can also judge the fit by the relative error in L^2:

```
> evalf(l2norm(exp-q)/l2norm(exp));
                          0.002907224229
```

(4.4.4.5)

The error is less than 0.3%.

Particularly if you wish to use a subspace with a larger dimension, typing in the formulas for the Gram matrix and the right-hand side of the normal equations can be quite monotonous. One can avoid the repetitive typing, but only by representing the basis functions differently. Here is an example:

```
> unassign('x');
> p[1]:=x->1:
> p[2]:=x->x:
> p[3]:=x->x^2:
```

Now, for each $i = 1, 2, 3$, p[i] is one of the basis functions. Here is the third, for example:

```
> p[3](x);
```

$$x^2 \tag{4.4.4.6}$$

I can now define the Gram matrix using the alternate version of the **Matrix** command:

```
> G:=Matrix(3,3,(i,j)->l2ip(p[j],p[i]));
```

$$G := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix} \tag{4.4.4.7}$$

I define the right-hand side similarly:

```
> b:=Vector(3,i->l2ip(p[i],exp));
```

$$b := \begin{bmatrix} -1+e \\ 1 \\ -2+e \end{bmatrix} \tag{4.4.4.8}$$

Just for fun, I will now repeat the above computation, but with sixth-degree polynomials. I will define the basis functions themselves in a vector:

```
> p:=Vector(7,i->(x->x^(i-1)));
```

$$p := \begin{bmatrix} x \to x^{1-1} \\ x \to x^{2-1} \\ x \to x^{3-1} \\ x \to x^{4-1} \\ x \to x^{5-1} \\ x \to x^{6-1} \\ x \to x^{7-1} \end{bmatrix} \qquad \textbf{(4.4.4.9)}$$

Now I will compute the Gram matrix and the right-hand side, and solve the normal equations:

```
> G:=Matrix(7,7,(i,j)->l2ip(p[j],p[i])):
> b:=Vector(7,i->l2ip(p[i],exp)):
> c:=LinearSolve(G,b):
```

Now, the best sixth-degree approximation is

$$q(x) = \sum_{i=1}^{7} c_i x^{i-1}.$$

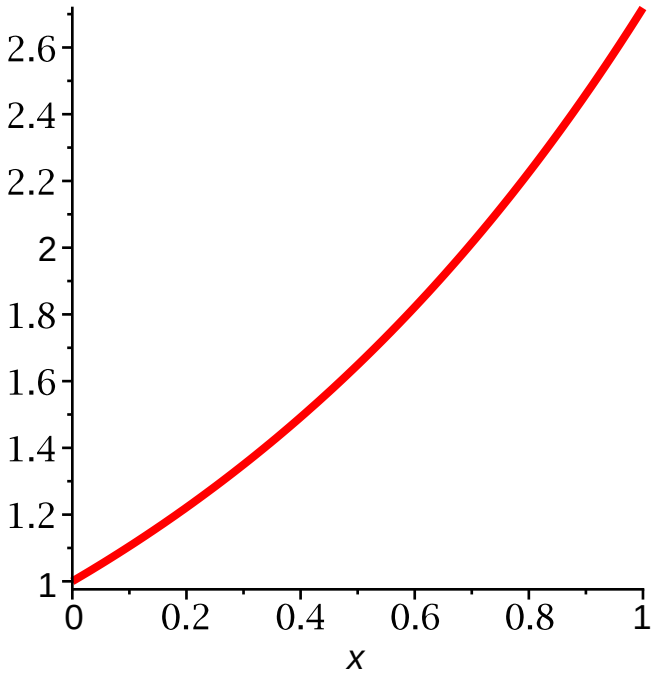
To form this expression without an undue amount of typing, I can use the **add** command:

```
> unassign('x');
> q:=x->add(c[i]*p[i](x),i=1..7);
```

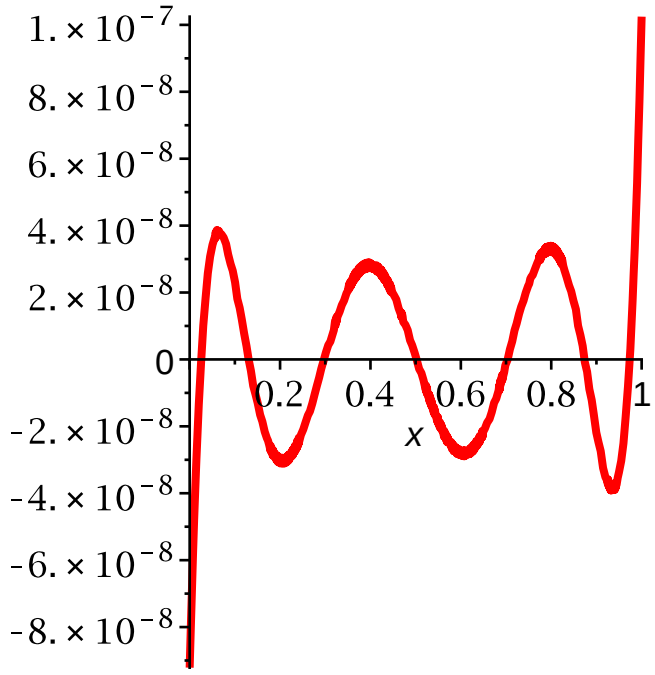
$$q := x \to add(c_i\, p_i(x),\ i = 1..7) \qquad \textbf{(4.4.4.10)}$$

Now I will plot the approximation and the error:

```
> plot(q(x),x=0..1,thickness=3);
```




```
> plot(exp(x)-q(x),x=0..1,thickness=3);
```



The fit is now very good.

Section 3.5: Eigenvalues and eigenvectors of a symmetric matrix

Maple can compute the eigenvalues and eigenvectors of a square matrix:

```
> A:=Matrix([[1,2,-1],[4,0,1],[-7,-2,3]]);
```

$$A := \begin{bmatrix} 1 & 2 & -1 \\ 4 & 0 & 1 \\ -7 & -2 & 3 \end{bmatrix} \tag{4.5.1}$$

```
> res:=Eigenvectors(A);
```

$$res := \begin{bmatrix} 2 \\ 1+\sqrt{15} \\ 1-\sqrt{15} \end{bmatrix}, \begin{bmatrix} 0 & -\frac{(-1+\sqrt{15})\sqrt{15}}{15+7\sqrt{15}} & \frac{(-1-\sqrt{15})\sqrt{15}}{15-7\sqrt{15}} \\ \frac{1}{2} & -\frac{-15+4\sqrt{15}}{15+7\sqrt{15}} & -\frac{-15-4\sqrt{15}}{15-7\sqrt{15}} \\ 1 & 1 & 1 \end{bmatrix} \tag{4.5.2}$$

The first element of the results is a vector containing the eigenvalues, and the second is a matrix whose columns are the corresponding eigenvectors.

```
> l:=res[1];
```

$$l := \begin{bmatrix} 2 \\ 1+\sqrt{15} \\ 1-\sqrt{15} \end{bmatrix} \tag{4.5.3}$$

```
> V:=res[2];
```

$$V := \begin{bmatrix} 0 & -\frac{(-1+\sqrt{15})\sqrt{15}}{15+7\sqrt{15}} & \frac{(-1-\sqrt{15})\sqrt{15}}{15-7\sqrt{15}} \\ \frac{1}{2} & -\frac{-15+4\sqrt{15}}{15+7\sqrt{15}} & -\frac{-15-4\sqrt{15}}{15-7\sqrt{15}} \\ 1 & 1 & 1 \end{bmatrix} \tag{4.5.4}$$

To access the individual eigenvectors, the following syntax can be used to extract a column from a matrix:

```
> V[1..3,1];
```

$$\begin{bmatrix} 0 \\ \frac{1}{2} \\ 1 \end{bmatrix} \tag{4.5.5}$$

Here I check the eigenvalue-eigenvector pairs:

```
> A.V[1..3,1]-l[1]*V[1..3,1];
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.5.6}$$

```
> A.V[1..3,2]-l[2]*V[1..3,2];
```

$$\left[\left[-\frac{\left(-1+\sqrt{15}\right)\sqrt{15}}{15+7\sqrt{15}} - \frac{2\left(-15+4\sqrt{15}\right)}{15+7\sqrt{15}} - 1 + \frac{\left(1+\sqrt{15}\right)\left(-1+\sqrt{15}\right)\sqrt{15}}{15+7\sqrt{15}} \right], \left[-\frac{4\left(-1+\sqrt{15}\right)\sqrt{15}}{15+7\sqrt{15}} + 1 + \frac{\left(1+\sqrt{15}\right)\left(-15+4\sqrt{15}\right)}{15+7\sqrt{15}} \right], \left[\frac{7\left(-1+\sqrt{15}\right)\sqrt{15}}{15+7\sqrt{15}} + \frac{2\left(-15+4\sqrt{15}\right)}{15+7\sqrt{15}} + 2 - \sqrt{15} \right]\right] \tag{4.5.7}$$

```
> simplify(%);
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.5.8}$$

```
> A.V[1..3,3]-l[3]*V[1..3,3];
```

$$\left[\left[\frac{\left(-1-\sqrt{15}\right)\sqrt{15}}{15-7\sqrt{15}} - \frac{2\left(-15-4\sqrt{15}\right)}{15-7\sqrt{15}} - 1 - \frac{\left(1-\sqrt{15}\right)\left(-1-\sqrt{15}\right)\sqrt{15}}{15-7\sqrt{15}} \right], \left[\frac{4\left(-1-\sqrt{15}\right)\sqrt{15}}{15-7\sqrt{15}} + 1 + \frac{\left(1-\sqrt{15}\right)\left(-15-4\sqrt{15}\right)}{15-7\sqrt{15}} \right], \left[-\frac{7\left(-1-\sqrt{15}\right)\sqrt{15}}{15-7\sqrt{15}} + \frac{2\left(-15-4\sqrt{15}\right)}{15-7\sqrt{15}} + 2 + \sqrt{15} \right]\right] \tag{4.5.9}$$

```
> simplify(%);
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.5.10}$$

You should notice again the value of the **simplify** command. When you obtain a surprising result, you should make sure that the result is simplified.

The problem of computing the eigenvalues of an n by n matrix is equivalent to the problem of finding the roots of an nth degree polynomial. One of the most famous results of 19th-century mathematics is that it is impossible to find a formula (analogous to the quadratic formula) expressing the roots of an arbitrary polynomial of degree 5 or more. Therefore, asking *Maple* for the (exact) eigenvalues of a large matrix is most likely an exercise in futility. Here is an example:

```
> B:=Matrix(5,5,(i,j)->1/(i+j-1));
```

$$B := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{bmatrix} \qquad \textbf{(4.5.11)}$$

```
> Eigenvectors(B):
```

The output of this command has been suppressed because it occupies many pages; if you look at it (by replacing the colon by a semicolon and executing the command), you will see that *Maple* has computed the eigenvalues and eigenvectors in terms of the (unknown) roots of the characteristic polynomial. *Maple* can manipulate the output in various ways, but I do not find it useful for my purposes in this tutorial.

In a case like this, it is possible to have *Maple* estimate the eigenvalues numerically. In fact, if the matrix given to **Eigenvectors** is numerical, then *Maple* automatically performs a numerical computation:

```
> res1:=Eigenvectors(evalf(B));
```

$$res1 := \begin{bmatrix} 1.56705069109486 + 0.\,I \\ 0.208534218668013 + 0.\,I \\ 0.0114074915726153 + 0.\,I \\ 0.000305898025240857 + 0.\,I \\ 0.0000328793927030469 + 0.\,I \end{bmatrix}, [[-0.767854735059365 + 0.\,I, -0.601871478348877 + 0.\,I, -0.214213624244780 + 0.\,I, \qquad \textbf{(4.5.12)}$$

```
-0.0471618059714938 + 0. I, 0.00617386770773715 + 0. I],
[-0.445791060459148 + 0. I, 0.275913417325606 + 0. I,
0.724102132235258 + 0. I, 0.432667323400602 + 0. I,
-0.116692784301902 + 0. I],
[-0.321578294476595 + 0. I, 0.424876622451948 + 0. I,
0.120453277762841 + 0. I, -0.667350401123385 + 0. I,
0.506163714831499 + 0. I],
[-0.253438943260472 + 0. I, 0.443903038708084 + 0. I,
-0.309573971066729 + 0. I, -0.233024579702172 + 0. I,
-0.767191173018389 + 0. I],
[-0.209822636581853 + 0. I, 0.429013353649275 + 0. I,
-0.565193409015386 + 0. I, 0.557599980894861 + 0. I,
0.376245498684095 + 0. I]]
> l1:=res1[1]:
> V1:=res1[2]:
```

Let us check the results of this numerical computation:

```
> B.V1[1..5,1]-l1[1]*V1[1..5,1];
```

$$\begin{bmatrix} -1.07198694365707\,10^{-11} + 0.\,I \\ -7.86570808486431\,10^{-12} + 0.\,I \\ -8.15492118277916\,10^{-12} + 0.\,I \\ 2.15810147530249\,10^{-11} + 0.\,I \\ 2.63102317710207\,10^{-11} + 0.\,I \end{bmatrix} \qquad \textbf{(4.5.13)}$$

The result is correct, up to roundoff error.

The above eigenpairs formally contain complex numbers, although the imaginary parts are all zero. In fact, since B is symmetric, we know *a priori* that the eigevalues are all real. We can tell *Maple* this when we define the matrix:

```
> B:=Matrix(5,5,(i,j)->1/(i+j-1),shape=symmetric);
```

$$B := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{bmatrix} \quad \textbf{(4.5.14)}$$

```
> res1:=Eigenvectors(evalf(B));
```

$$res1 := \begin{bmatrix} 0.00000328793927007522 \\ 0.000305898025240873 \\ 0.0114074915726152 \\ 0.208534218668013 \\ 1.56705069109486 \end{bmatrix}, [[-0.00617386770772625, \quad \textbf{(4.5.15)}$$

0.0471618059714956, 0.214213624244780, -0.601871478348877,
0.767854735059366],
[0.116692784301794, -0.432667323400628, -0.724102132235258,
0.275913417325605, 0.445791060459147],
[-0.506163714831327, 0.667350401123493, -0.120453277762843,
0.424876622451949, 0.321578294476595],
[0.767191173018448, 0.233024579702008, 0.309573971066733,
0.443903038708083, 0.253438943260472],
[-0.376245498684239, -0.557599980894780, 0.565193409015384,
0.429013353649275, 0.209822636581853]]

Since, in this tutorial, we mostly deal with symmetric matrices, the "shape= symmetric" option will be very useful.

Example 3.49

I will now use the spectral method to solve $Ax = b$, where A and b are given as follows:

```
> A:=Matrix([[11,-4,-1],[-4,14,-4],[-1,-4,11]],shape=
```

```
symmetric);
```

$$A := \begin{bmatrix} 11 & -4 & -1 \\ -4 & 14 & -4 \\ -1 & -4 & 11 \end{bmatrix} \quad \textbf{(4.5.1.1)}$$

```
> b:=<1,2,1>;
```

$$b := \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \textbf{(4.5.1.2)}$$

We need the eigenpairs of A; I will use symbolic computation for this example:

```
> res:=Eigenvectors(A);
```

$$res := \begin{bmatrix} 12 \\ 18 \\ 6 \end{bmatrix}, \begin{bmatrix} -1 & 1 & 1 \\ 0 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \textbf{(4.5.1.3)}$$

```
> l:=res[1];
```

$$l := \begin{bmatrix} 12 \\ 18 \\ 6 \end{bmatrix} \quad \textbf{(4.5.1.4)}$$

```
> V:=res[2];
```

$$V := \begin{bmatrix} -1 & 1 & 1 \\ 0 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \textbf{(4.5.1.5)}$$

Here is an important point: *Maple* does not automatically normalize symbolic eigenvectors. Of course, these eigenvectors are orthogonal since A is symmetric and the eigenvalues are distinct, but we must normalize the eigenvectors manually. I will call the normalized eigenvector u1, u2, u3:

```
> u1:=V[1..3,1]/sqrt(V[1..3,1].V[1..3,1]);
```

$$u1 := \begin{bmatrix} -\frac{1}{2}\sqrt{2} \\ 0 \\ \frac{1}{2}\sqrt{2} \end{bmatrix} \quad \textbf{(4.5.1.6)}$$

```
> u2:=V[1..3,2]/sqrt(V[1..3,2].V[1..3,2]);
```

$$u2 := \begin{bmatrix} \frac{1}{6}\sqrt{6} \\ -\frac{1}{3}\sqrt{6} \\ \frac{1}{6}\sqrt{6} \end{bmatrix} \quad \textbf{(4.5.1.7)}$$

```
> u3:=V[1..3,3]/sqrt(V[1..3,3].V[1..3,3]);
```

$$u3 := \begin{bmatrix} \frac{1}{3}\sqrt{3} \\ \frac{1}{3}\sqrt{3} \\ \frac{1}{3}\sqrt{3} \end{bmatrix} \quad \textbf{(4.5.1.8)}$$

The solution of $Ax = b$ is then

```
> x:=(u1.b/l[1])*u1+(u2.b/l[2])*u2+(u3.b/l[3])*u3;
```

$$x := \begin{bmatrix} \frac{11}{54} \\ \frac{7}{27} \\ \frac{11}{54} \end{bmatrix} \quad \textbf{(4.5.1.9)}$$

Check:

```
> A.x-b;
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \textbf{(4.5.1.10)}$$

Chapter 4: Essential ordinary differential equations

Section 4.2: Solutions to some simple ODEs

Second-order linear homogeneous ODEs with constant coefficients

Suppose we wish to solve the following IVP:

$$\frac{\partial^2}{\partial t^2} u + 4\left(\frac{\partial}{\partial t} u\right) - 3u = 0,\ 0 < t,$$
$$u(0) = 1,$$
$$\frac{\partial}{\partial t} u(0) = 0.$$

The characteristic polynomial is $r^2 + 4r - 3$, which has the following roots:

```
> r:=solve(r^2+4*r-3=0,r);
```

$$r := -2 + \sqrt{7},\ -2 - \sqrt{7} \qquad \textbf{(5.1.1.1)}$$

The general solution is

```
> unassign('c1','c2','t');
> u:=t->c1*exp(r[1]*t)+c2*exp(r[2]*t);
```

$$u := t \to c1\, e^{r_1 t} + c2\, e^{r_2 t} \qquad \textbf{(5.1.1.2)}$$

We can now solve for the unknowns c1 and c2:

```
> sols:=solve({u(0)=1,D(u)(0)=0},{c1,c2});
```

$$sols := \left\{c1 = \frac{1}{2} + \frac{1}{7}\sqrt{7},\ c2 = -\frac{1}{7}\sqrt{7} + \frac{1}{2}\right\} \qquad \textbf{(5.1.1.3)}$$

```
> assign(sols);
```

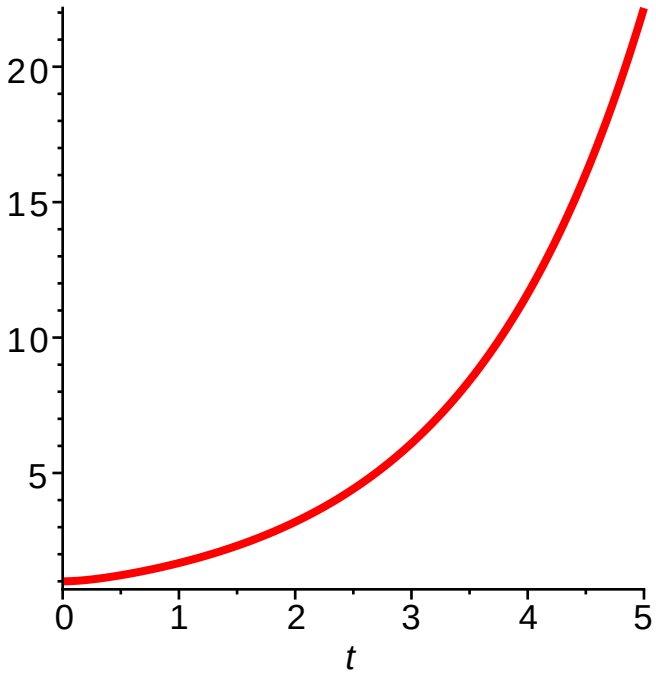
Here is the solution,

```
> u(t);
```

$$\left(\frac{1}{2} + \frac{1}{7}\sqrt{7}\right) e^{(-2+\sqrt{7})t} + \left(-\frac{1}{7}\sqrt{7} + \frac{1}{2}\right) e^{(-2-\sqrt{7})t} \qquad \textbf{(5.1.1.4)}$$

and here is its graph:

```
> plot(u(t),t=0..5,thickness=3);
```



A special inhomogeneous second-order linear ODE

Consider the IVP

$$\frac{\partial^2}{\partial t^2} u + 4\,u = \sin(\pi\,t),\ 0 < t,$$
$$u(0) = 0,$$
$$\frac{\partial}{\partial t} u(0) = 0.$$

The solution, as given in Section 4.2.3 of the text, is

```
> unassign('t','s');
```

```
> (1/2)*int(sin(2*(t-s))*sin(Pi*s),s=0..t);
```

$$\frac{\pi \sin(t) \cos(t) - \sin(t\,\pi)}{-4 + \pi^2} \qquad \textbf{(5.1.2.1)}$$

```
> u:=unapply(%,t);
```

$$u := t \to \frac{\pi \sin(t) \cos(t) - \sin(\pi\,t)}{-4 + \pi^2} \qquad \textbf{(5.1.2.2)}$$

Check:

```
> diff(u(t),t$2)+4*u(t);
```

(5.1.2.3)

$$\frac{-4\,\pi\sin(t)\cos(t)+\sin(t\,\pi)\,\pi^2}{-4+\pi^2}+\frac{4\,(\pi\sin(t)\cos(t)-\sin(t\,\pi))}{-4+\pi^2} \tag{5.1.2.3}$$

```
> simplify(%);
```

$$\sin(t\,\pi) \tag{5.1.2.4}$$

```
> u(0);
```

$$0 \tag{5.1.2.5}$$

```
> D(u)(0);
```

$$0 \tag{5.1.2.6}$$

First-order linear ODEs

Now consider the following IVP:

$$\frac{\partial}{\partial t}\,u-\frac{u}{2}=-t,\,0<t,$$
$$u(0)=1.$$

Section 4.2.4 contains an explicit formula for the solution:

```
> unassign('t','s');
> exp(t/2)+int(exp((t-s)/2)*(-s),s=0..t);
```

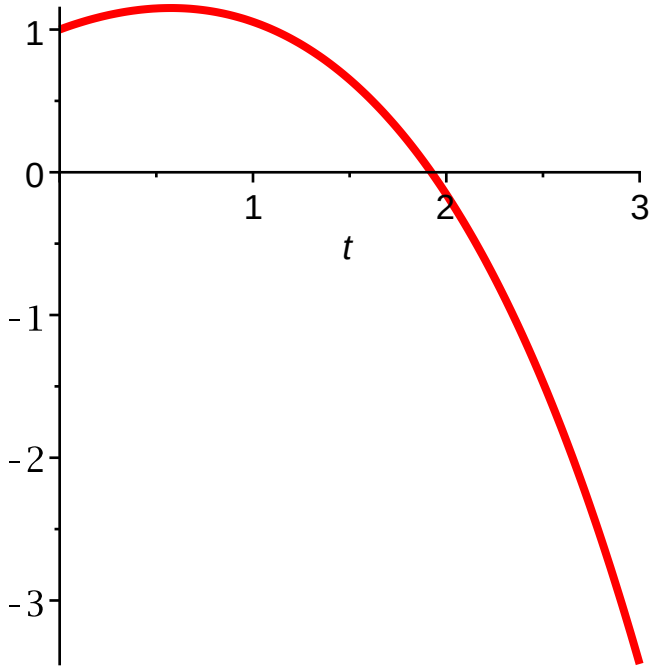
$$-3\,e^{\frac{1}{2}\,t}+4+2\,t \tag{5.1.3.1}$$

```
> u:=unapply(%,t);
```

$$u:=t\rightarrow -3\,e^{\frac{1}{2}\,t}+4+2\,t \tag{5.1.3.2}$$

Here is a graph of the solution:

```
> plot(u(t),t=0..3,thickness=3);
```



Just out of curiosity, let us determine the value of t at which $u(t)$ is zero:

```
> solve(u(t)=0,t);
```

$$-2\,\mathrm{LambertW}\left(-\frac{3}{4}\,e^{-1}\right)-2,\ -2\,\mathrm{LambertW}\left(-1,\ -\frac{3}{4}\,e^{-1}\right)-2 \qquad \textbf{(5.1.3.3)}$$

The **solve** command returns two solutions, expressed in terms of the **LambertW** function. We can evaluate the results numerically:

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

$$-1.160262798,\ 1.922557526 \qquad \textbf{(5.1.3.4)}$$

The second solution is the one we want.

We can also solve the equation numerically using **fsolve**, which searches for (an approximation to) a single root on a given interval:

```
> fsolve(u(t)=0,t=1..2);
```

$$1.922557526 \qquad \textbf{(5.1.3.5)}$$

It is often preferable to find the numerical solution directly using **fsolve**.

Section 4.3: Linear systems with constant coefficients

Since *Maple* can compute eigenvalues and eigenvectors (either numerically or,

when possible, symbolically), it can be used to solve linear systems with constant coefficients. I will begin with a simple example, solving the homogeneous IVP

$$\frac{\partial}{\partial t} x = Ax,\ 0 < t,$$
$$x(0) = x_0.$$

I will take the matrix A and the initial vector x_0 to be as follows:

```
> A:=Matrix([[1,2],[3,4]]);
```

$$A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \textbf{(5.2.1)}$$

```
> x0:=<4,1>;
```

$$x0 := \begin{bmatrix} 4 \\ 1 \end{bmatrix} \quad \textbf{(5.2.2)}$$

The first step is to find the eigenpairs of A:

```
> res:=Eigenvectors(A);
```

$$res := \begin{bmatrix} \frac{5}{2} + \frac{1}{2}\sqrt{33} \\ \frac{5}{2} - \frac{1}{2}\sqrt{33} \end{bmatrix}, \begin{bmatrix} \frac{2}{\frac{3}{2} + \frac{1}{2}\sqrt{33}} & \frac{2}{\frac{3}{2} - \frac{1}{2}\sqrt{33}} \\ 1 & 1 \end{bmatrix} \quad \textbf{(5.2.3)}$$

```
> l:=res[1];
```

$$l := \begin{bmatrix} \frac{5}{2} + \frac{1}{2}\sqrt{33} \\ \frac{5}{2} - \frac{1}{2}\sqrt{33} \end{bmatrix} \quad \textbf{(5.2.4)}$$

```
> V:=res[2];
```

$$V := \begin{bmatrix} \frac{2}{\frac{3}{2} + \frac{1}{2}\sqrt{33}} & \frac{2}{\frac{3}{2} - \frac{1}{2}\sqrt{33}} \\ 1 & 1 \end{bmatrix} \quad \textbf{(5.2.5)}$$

Next, the initial vector must be expressed in terms of the eigenvectors of A:

```
> c:=LinearSolve(V,x0);
```

$$c := \begin{bmatrix} \frac{1}{2} + \frac{9}{22}\sqrt{33} \\ -\frac{9}{22}\sqrt{33} + \frac{1}{2} \end{bmatrix} \tag{5.2.6}$$

I can now write down the desired solution:

```
> unassign('t');
> x:=t->c[1]*exp(l[1]*t)*V[1..2,1]+c[2]*exp(l[2]*t)*V[1..2,2];
```

$$x := t \to c_1 \, e^{l_1 t} \, V_{1..2,\,1} + c_2 \, e^{l_2 t} \, V_{1..2,\,2} \tag{5.2.7}$$

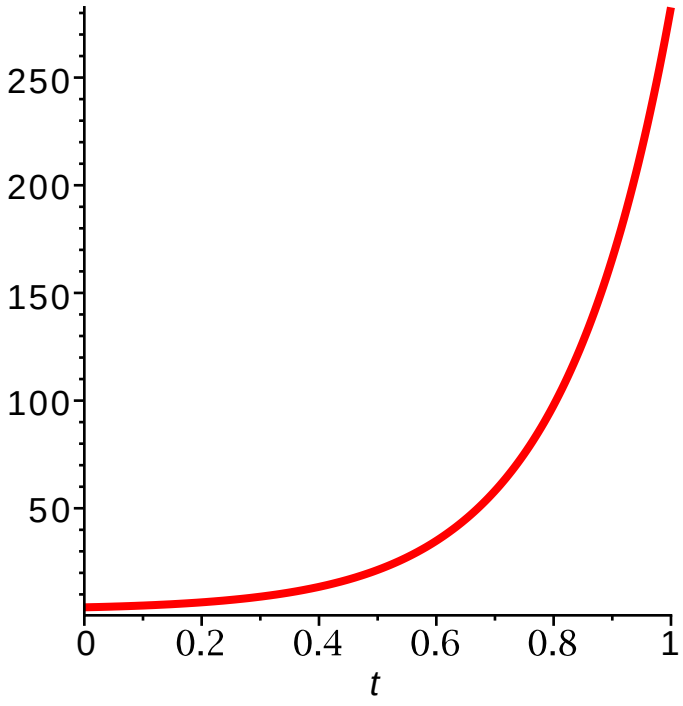
Notice that $x(t)$ is a vector (that is, the function x is vector-valued). For example, the first component is

```
> x(t)[1];
```

$$\frac{2\left(\frac{1}{2} + \frac{9}{22}\sqrt{33}\right) e^{\left(\frac{5}{2} + \frac{1}{2}\sqrt{33}\right)t}}{\frac{3}{2} + \frac{1}{2}\sqrt{33}} + \frac{2\left(-\frac{9}{22}\sqrt{33} + \frac{1}{2}\right) e^{\left(\frac{5}{2} - \frac{1}{2}\sqrt{33}\right)t}}{\frac{3}{2} - \frac{1}{2}\sqrt{33}} \tag{5.2.8}$$

Here is its graph:

```
> plot(x(t)[1],t=0..1,thickness=3);
```



(The solution is dominated by a rapidly growing exponential.)

Before I leave this example, I would like to check that the solution is correct. This requires differentiating the vector-valued function *x(t)*:

```
> diff(x(t), t);
```

Error, non-algebraic expressions cannot be differentiated

The **diff** command cannot be applied directly to a vector-valued function. There are two alternatives. First, **D** works fine:

```
> D(x)(t);
```

$$\left[\left[\frac{2\left(\frac{1}{2}+\frac{9}{22}\sqrt{33}\right)\left(\frac{5}{2}+\frac{1}{2}\sqrt{33}\right)e^{\left(\frac{5}{2}+\frac{1}{2}\sqrt{33}\right)t}}{\frac{3}{2}+\frac{1}{2}\sqrt{33}}\right.\right. \tag{5.2.9}$$

$$\left.\left.+\frac{2\left(-\frac{9}{22}\sqrt{33}+\frac{1}{2}\right)\left(\frac{5}{2}-\frac{1}{2}\sqrt{33}\right)e^{\left(\frac{5}{2}-\frac{1}{2}\sqrt{33}\right)t}}{\frac{3}{2}-\frac{1}{2}\sqrt{33}}\right],\right.$$

$$\left.\left[\left(\frac{1}{2}+\frac{9}{22}\sqrt{33}\right)\left(\frac{5}{2}+\frac{1}{2}\sqrt{33}\right)e^{\left(\frac{5}{2}+\frac{1}{2}\sqrt{33}\right)t}+\left(-\frac{9}{22}\sqrt{33}+\frac{1}{2}\right)\left(\frac{5}{2}\right.\right.\right.$$

$$\left.\left.\left.-\frac{1}{2}\sqrt{33}\right)e^{\left(\frac{5}{2}-\frac{1}{2}\sqrt{33}\right)t}\right]\right]$$

```
> simplify(D(x)(t) - A.x(t));
```

$$\begin{bmatrix}0\\0\end{bmatrix} \tag{5.2.10}$$

Alternatively, we can apply diff to each component of x(t) by using map:

```
> map(diff, x(t), t);
```

$$\left[\left[\frac{2\left(\frac{1}{2}+\frac{9}{22}\sqrt{33}\right)\left(\frac{5}{2}+\frac{1}{2}\sqrt{33}\right)e^{\left(\frac{5}{2}+\frac{1}{2}\sqrt{33}\right)t}}{\frac{3}{2}+\frac{1}{2}\sqrt{33}}\right.\right. \tag{5.2.11}$$

$$\left.\left.+\frac{2\left(-\frac{9}{22}\sqrt{33}+\frac{1}{2}\right)\left(\frac{5}{2}-\frac{1}{2}\sqrt{33}\right)e^{\left(\frac{5}{2}-\frac{1}{2}\sqrt{33}\right)t}}{\frac{3}{2}-\frac{1}{2}\sqrt{33}}\right],\right.$$

$$\left.\left[\left(\frac{1}{2}+\frac{9}{22}\sqrt{33}\right)\left(\frac{5}{2}+\frac{1}{2}\sqrt{33}\right)e^{\left(\frac{5}{2}+\frac{1}{2}\sqrt{33}\right)t}+\left(-\frac{9}{22}\sqrt{33}\right.\right.\right.$$

$$+\frac{1}{2}\left(\frac{5}{2}-\frac{1}{2}\sqrt{33}\right)e^{\left(\frac{5}{2}-\frac{1}{2}\sqrt{33}\right)t}\right]\Bigg]$$

```
> simplify(map(diff, x(t), t) - A.x(t));
```

$$\begin{bmatrix}0\\0\end{bmatrix} \quad (5.2.12)$$

```
> x(0)-x0;
```

$$\begin{bmatrix}\dfrac{2\left(\frac{1}{2}+\frac{9}{22}\sqrt{33}\right)}{\frac{3}{2}+\frac{1}{2}\sqrt{33}}+\dfrac{2\left(-\frac{9}{22}\sqrt{33}+\frac{1}{2}\right)}{\frac{3}{2}-\frac{1}{2}\sqrt{33}}-4\\0\end{bmatrix} \quad (5.2.13)$$

```
> simplify(%);
```

$$\begin{bmatrix}0\\0\end{bmatrix} \quad (5.2.14)$$

Inhomogeneous systems and variation of parameters

I will now show how to use *Maple* to solve inhomogeneous systems of the form

$$\frac{dx}{dt} = Ax + f(t), 0 < t,$$
$$x(0) = x_0.$$

Consider the matrix

```
> A:=Matrix([[1,2],[2,1]],shape=symmetric);
```

$$A := \begin{bmatrix}1 & 2\\2 & 1\end{bmatrix} \quad (5.2.1.1)$$

Let

```
> x0:=<0,1>;
```

$$x0 := \begin{bmatrix}0\\1\end{bmatrix} \quad (5.2.1.2)$$

```
> unassign('t');
```

```
> f:=t-><sin(t),0>;
```

$$f := t \to \langle \sin(t), 0 \rangle \tag{5.2.1.3}$$

I will solve this problem numerically. Here are the eigenpairs of A:

```
> res:=Eigenvectors(evalf(A));
```

$$res := \begin{bmatrix} -1. \\ 3. \end{bmatrix}, \begin{bmatrix} -0.707106781186548 & 0.707106781186548 \\ 0.707106781186548 & 0.707106781186548 \end{bmatrix} \tag{5.2.1.4}$$

```
> l:=res[1];
```

$$l := \begin{bmatrix} -1. \\ 3. \end{bmatrix} \tag{5.2.1.5}$$

```
> V:=res[2];
```

$$V := \begin{bmatrix} -0.707106781186548 & 0.707106781186548 \\ 0.707106781186548 & 0.707106781186548 \end{bmatrix} \tag{5.2.1.6}$$

Maple automatically normalized numerical eigenvectors, and the eigenvectors are necessarily orthogonal since A is symmetric.

```
> u1:=V[1..2,1];
```

$$u1 := \begin{bmatrix} -0.707106781186548 \\ 0.707106781186548 \end{bmatrix} \tag{5.2.1.7}$$

```
> u2:=V[1..2,2];
```

$$u2 := \begin{bmatrix} 0.707106781186548 \\ 0.707106781186548 \end{bmatrix} \tag{5.2.1.8}$$

Now, we have $f(t) = c_1(t)\, u_1 + c_2(t)\, u_2$ and $x_0 = b_1\, u_1 + b_2\, u_2$, where

```
> c1:=unapply(u1.f(t),t);
```

$$c1 := t \to -0.7071067811865476 \sin(t) \tag{5.2.1.9}$$

```
> c2:=unapply(u2.f(t),t);
```

$$c2 := t \to 0.7071067811865476 \sin(t) \tag{5.2.1.10}$$

```
> b1:=u1.x0;
```

$$b1 := 0.7071067811865475\,73 \tag{5.2.1.11}$$

```
> b2:=u2.x0;
```

$$b2 := 0.7071067811865475\,73 \tag{5.2.1.12}$$

We then solve the decoupled IVPs

$$\frac{\partial}{\partial t}\, a_1 = l_1\, a_1 + c_1(t),\ a_1(0) = b_1,$$
$$\frac{\partial}{\partial t}\, a_2 = l_2\, a_2 + c_2(t),\ a_2(0) = b_2.$$

The solutions, using the techniques of Section 4.3, are

```
> b1*exp(l[1]*t)+int(exp(l[1]*(t-s))*c1(s),s=0..t);
```
$$0.3535533906\, e^{-1.\,t} + 0.3535533906\cos(t) - 0.3535533906\sin(t) \qquad \textbf{(5.2.1.13)}$$

```
> a1 := unapply(%, t);
```
$$a1 := t \to 0.7071067811865475\, e^{-1.0\,t} - 0.3535533906\, e^{-1.\,t} + 0.3535533906\cos(t) - 0.3535533906\sin(t) \qquad \textbf{(5.2.1.14)}$$

```
> b2*exp(l[2]*t)+int(exp(l[2]*(t-s))*c2(s),s=0..t);
```
$$0.7778174593\, e^{3.\,t} - 0.07071067812\cos(t) - 0.2121320344\sin(t) \qquad \textbf{(5.2.1.15)}$$

```
> a2 := unapply(%, t);
```
$$a2 := t \to 0.7071067811865475\, e^{3.0\,t} + 0.07071067812\, e^{3.\,t} - 0.07071067812\cos(t) - 0.2121320344\sin(t) \qquad \textbf{(5.2.1.16)}$$

The solution to the original system is then

```
> x:=t->a1(t)*u1+a2(t)*u2;
```
$$x := t \to a1(t)\ u1 + a2(t)\ u2 \qquad \textbf{(5.2.1.17)}$$

Check:

```
> D(x)(t)-A.x(t)-f(t);
```
$$\begin{bmatrix} 1.90246707276742\ 10^{-11}\, e^{-1.\,t} + 5.70739011607202\ 10^{-11}\, e^{3.\,t} \\ -1.90246707276742\ 10^{-11}\, e^{-1.\,t} + 5.70739011607202\ 10^{-11}\, e^{3.\,t} \end{bmatrix} \qquad \textbf{(5.2.1.18)}$$

```
> x(0)-x0;
```
$$\begin{bmatrix} 0. \\ 2.22044604925031\ 10^{-16} \end{bmatrix} \qquad \textbf{(5.2.1.19)}$$

The answer is correct, up to roundoff error.

Section 4.4: Numerical methods for initial value problems

When we turn to numerical methods for IVPs in ODEs, we naturally wish to write programs to implement the methods. Since time-stepping methods involve repeated steps of the same form, it would be quite tedious to apply the method manually. Here is another strength of *Maple*: not only does it integrate symbolic and numerical computations with graphics, it also provides a programming environment. I will explain the basics of programming in *Maple*.

Interactive commands

Maple supports the usual programming constructs, such as loops and conditionals, which can be used both interactively and in programs. For example, suppose we wish to apply Euler's method to estimate the solution of

$$\frac{\partial}{\partial t} u = \frac{u}{1+t^2}, 0 < t,$$
$$u(0) = 1$$

on the interval $[0, 1]$. The exact solution is

```
> unassign('t');
> v:=t->exp(arctan(t));
```

$$v := t \to e^{\arctan(t)} \qquad \textbf{(5.3.1.1)}$$

I will use the exact solution to test my results below. In preparation for applying Euler's method, I define the right-hand side of the ODE as a function:

```
> unassign('u');
> f:=(t,u)->u/(1+t^2);
```

$$f := (t, u) \to \frac{u}{t^2+1} \qquad \textbf{(5.3.1.2)}$$

I now apply Euler's method with a step length of 0.1. The *Maple* command we need is the "repetition statement," a form of the familiar for-loop (in C programming) or do-loop (in Fortran programming). First, some initialization:

```
> dt:=0.1;
```

$$dt := 0.1 \qquad \textbf{(5.3.1.3)}$$

```
> n:=10;
```

$$n := 10 \qquad \textbf{(5.3.1.4)}$$

```
> u:=1.0;
```

$$u := 1.0 \qquad \textbf{(5.3.1.5)}$$

```
> tt:=0.0;
```

$$tt := 0. \qquad \textbf{(5.3.1.6)}$$

(Here is an important but subtle point. I definitely want to do numeric computations, not symbolic. Therefore, I assign the quantities *dt*, *u*, and *t* to have floating point values (0.1, 1.0, and 0.0) rather than exact values (1/10, 1, and 0). If I do not do this, then the symbolic computation becomes quite involved---all the quantities involved will be rational numbers, and the number of digits will explode. The results are not bad for 10 steps, but you would not want to do a symbolic computation for hundreds of steps.)

The following loop executes 10 steps of Euler's method (notice that I do not save the intermediate values of the computed solution):

```
> for i from 1 to n
> do
>    u:=u+dt*f(tt,u);
>    tt:=tt+dt;
> end do;
```

$$u := 1.100000000$$
$$tt := 0.1$$
$$u := 1.208910891$$
$$tt := 0.2$$
$$u := 1.325152323$$
$$tt := 0.3$$
$$u := 1.446725931$$
$$tt := 0.4$$
$$u := 1.571443684$$
$$tt := 0.5$$
$$u := 1.697159179$$
$$tt := 0.6$$
$$u := 1.821950295$$
$$tt := 0.7$$
$$u := 1.944228838$$
$$tt := 0.8$$
$$u := 2.062779377$$
$$tt := 0.9$$
$$u := 2.176745088$$
$$tt := 1.0 \qquad \textbf{(5.3.1.7)}$$

(If I do not wish to see the intermediate results, I can end the "end do" statement with a colon instead of a semicolon.)

Here is the relative error in the computed solution (at t=1.0):

```
> abs(v(1.0)-u)/abs(v(1.0));
```

$$0.007538920072 \qquad \textbf{(5.3.1.8)}$$

The result is not bad.

If I want to save the intermediate results (for plotting purposes, say), I can put them in a matrix. The following command creates an $n+1$ by 2 matrix filled with zeros.

```
> U:=Matrix(n+1,2);
```

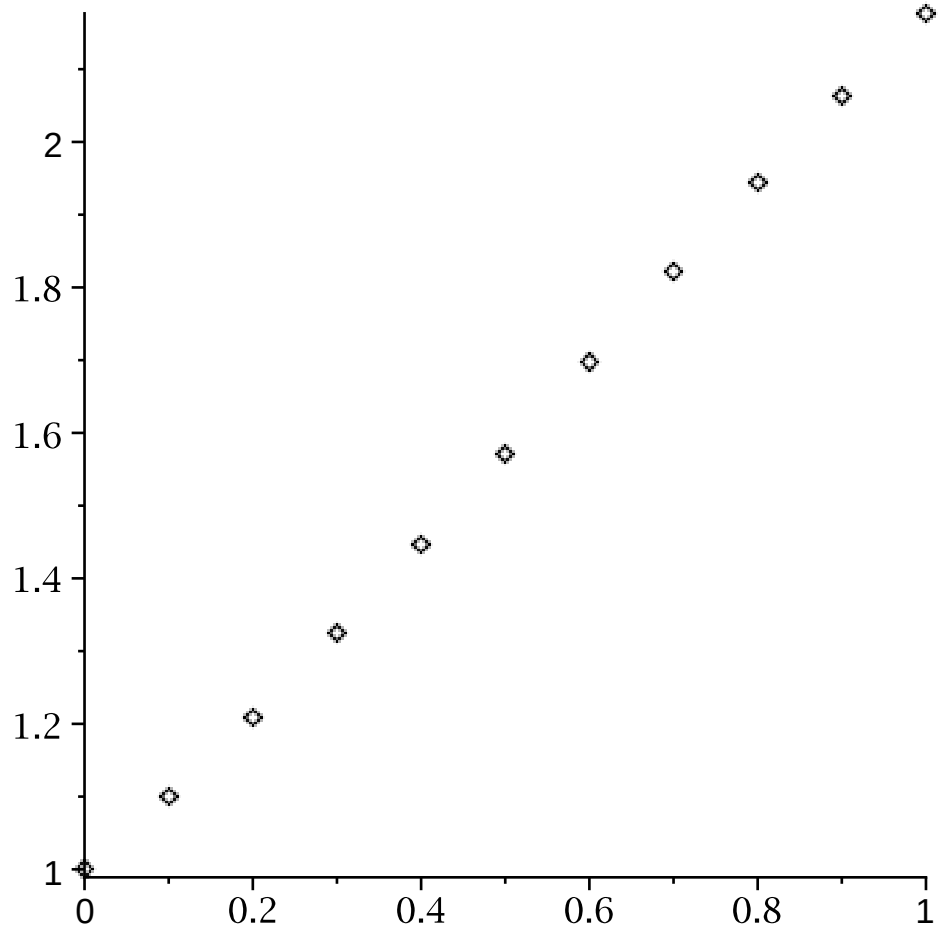
$$U := \begin{bmatrix} 11 \text{ x } 2 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix} \qquad \textbf{(5.3.1.9)}$$

Here is another version of the calculation:

```
> U[1,1]:=0.0:
> U[1,2]:=1.0:
> for i from 1 to n
> do
>    U[i+1,2]:=U[i,2]+dt*f(U[i,1],U[i,2]):
>    U[i+1,1]:=U[i,1]+dt:
> end do:
```

I can now look at the results using **pointplot:**

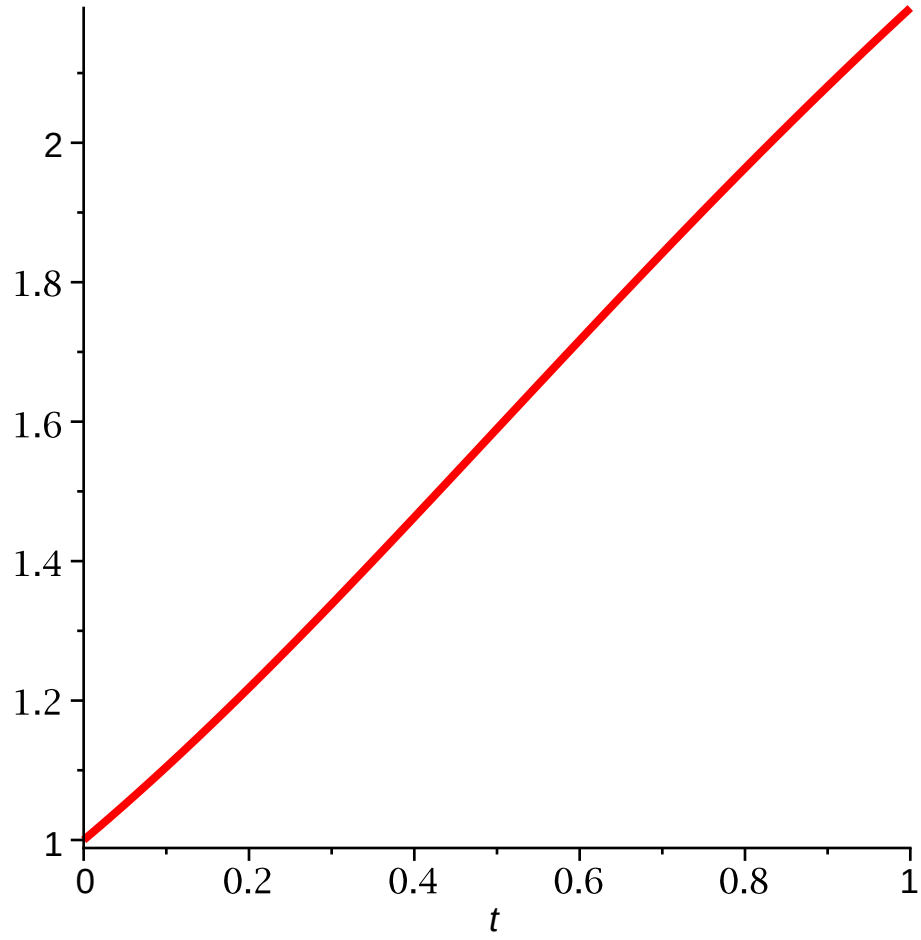
```
> with(plots,pointplot):
> pointplot(U,symbolsize=15);
```




```
> plot1:=%:
```

Here is the exact solution:

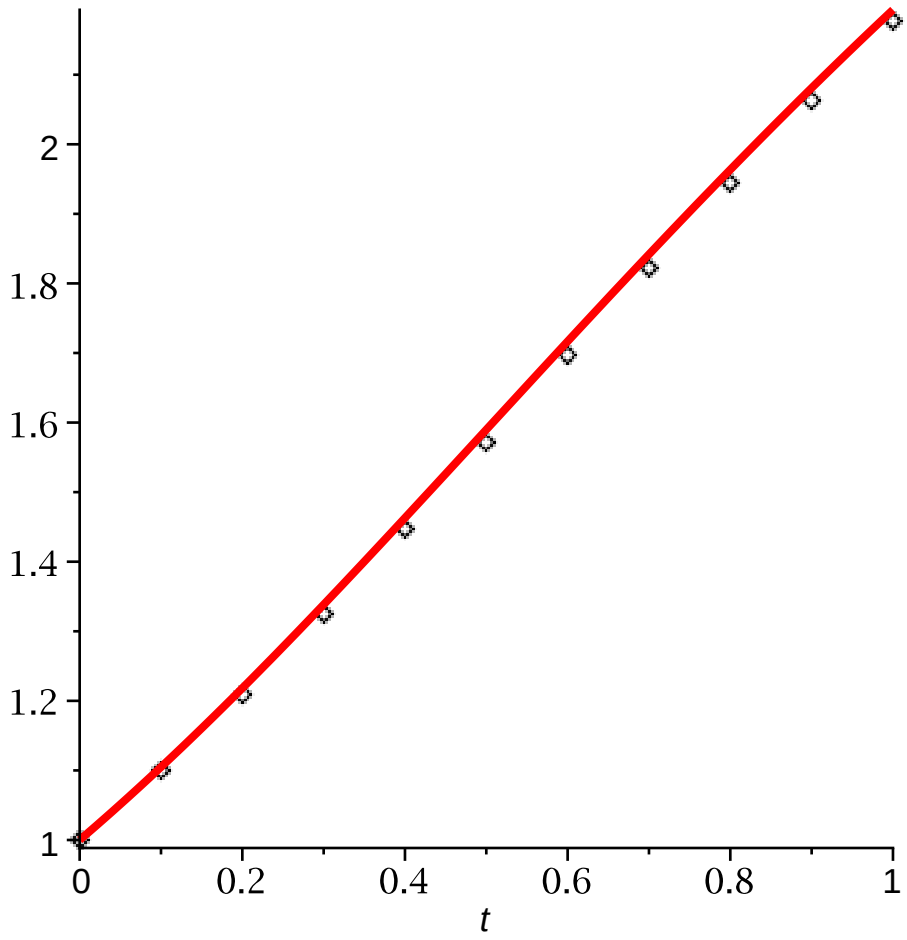
```
> plot(v(t),t=0..1,thickness=3);
```




```
> plot2:=%:
```

Here are the exacted and computed solution:

```
> with(plots,display):
> display(plot1,plot2);
```

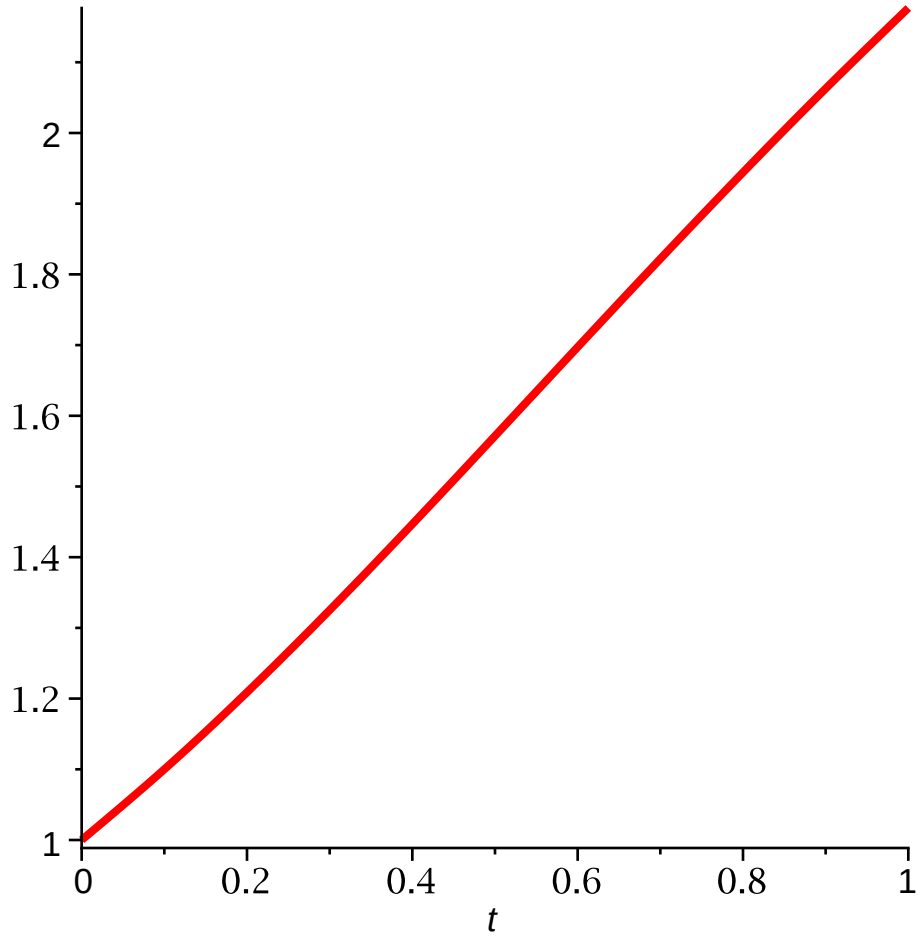


I can define a function representing the computed solution using interpolation. One way to do this is to use the **Spline** function (which is part of the **CurveFitting** package). **Spline** (by default) produces a piecewise cubic function agreeing with the data points, and it is simple to use, as the following example shows. (I recommend ending the command with a colon, as the output is quite complicated.)

```
> with(CurveFitting,Spline):
> Spline(U,t):
```

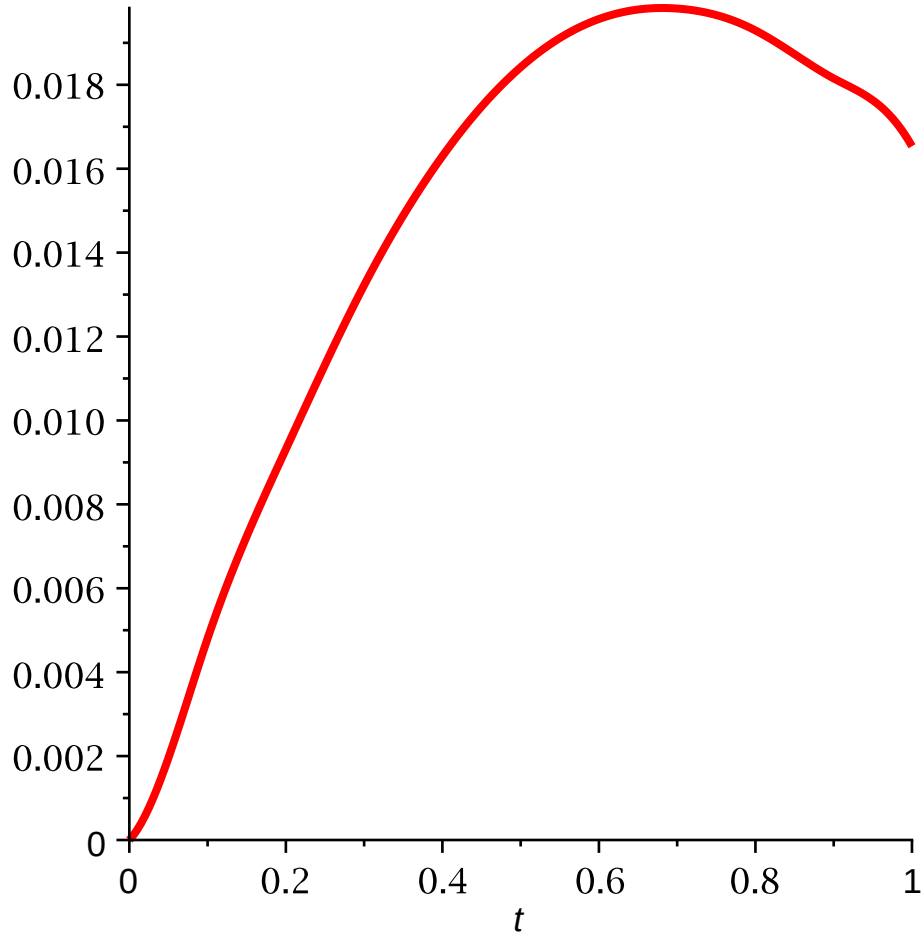
(The reason for the argument "*t*" is that Spline produces an expression in the given variable.)

```
> ui:=unapply(%,t):
> plot(ui(t),t=0..1,thickness=3);
```



Now we can compare the computed solution directly with the exact solution:

```
> plot(v(t)-ui(t),t=0..1,thickness=3);
```



Creating new Maple *commands*

For convenience, we can write a program to implement Euler's method. In *Maple*, writing a program really implies creating a new *Maple* command. A reasonable command for applying Euler's method to the IVP

$$\frac{\partial}{\partial t} u = f(t, u), 0 < t,$$
$$u(t_0) = u_0$$

would take as inputs f, t_0, u_0, as well as T (the final time) and n (the number of time steps), and would produce the table of t_i, u_i values (like the result U above). In *Maple*, a command is implemented as a procedure:

```
> myeuler:=proc(f,t0,u0,T,n)
> local i,U,dt;
> U:=Matrix(n+1,2);
> dt:=evalf(T/n);
  U[1,1]:=t0:
> U[1,2]:=u0:
```

```
> for i from 1 to n
> do
>    U[i+1,1]:=U[i,1]+dt:
>    U[i+1,2]:=U[i,2]+dt*f(U[i,1],U[i,2]):
> end do:
> U
> end proc;
```

$$myeuler := \mathbf{proc}(f, t0, u0, T, n) \qquad \textbf{(5.3.2.1)}$$

$$\mathbf{local}\ i, U, dt;$$

$$U := Matrix(n+1, 2);$$

$$dt := evalf(T/n);$$

$$U[1, 1] := t0;$$

$$U[1, 2] := u0;$$

$$\mathbf{for}\ i\ \mathbf{to}\ n\ \mathbf{do}$$

$$U[1+i, 1] := U[i, 1] + dt;\ U[1+i, 2] := U[i, 2] + dt * f(U[i, 1], U[i, 2])$$

$$\mathbf{end\ do};$$

$$U$$

$$\mathbf{end\ proc}$$

The last statement in the above procedure just echos the value of U, since the return value of the procedure is the last computed value.

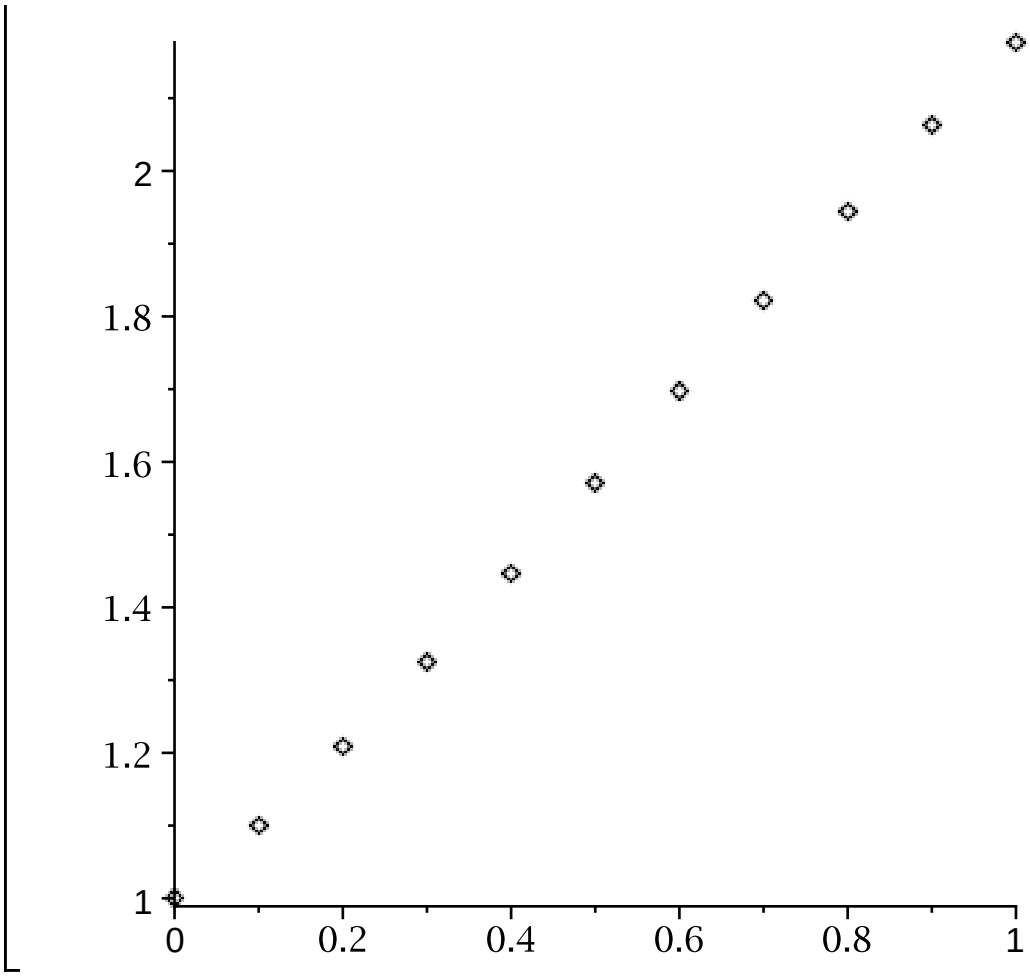
I can now use the **myeuler** procedure instead of typing out the loop each time I want to apply Euler's method. The following command repeats the above computation:

```
> U:=myeuler(f,0.0,1.0,1.0,10);
```

$$U := \begin{bmatrix} 11 \text{ x } 2 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix} \qquad \textbf{(5.3.2.2)}$$

Here is a graph of the computed solution:

```
> pointplot(U,symbolsize=15);
```



Notice that, as implemented above, the **myeuler** command uses local variables *i*, *U*, *dt*. When the command is invoked, the variables are known only during the execution of the command. Also notice that the return value of the procedure is the last result generated by it; thus, the desired output value U is echoed in the final statement.

As written, there is nothing indicating to the **myeuler** command that the IVP to be solved is scalar. Therefore, the same command will solve a system, such as

$$\frac{\partial}{\partial t} x_1 = -x_2,\ x_1(0) = 1,$$
$$\frac{\partial}{\partial t} x_2 = x_1,\ x_2(0) = 0,$$

whose solution is

```
> unassign('t');
> x:=t-><cos(t),sin(t)>;
```

$$x := t \rightarrow \langle \cos(t), \sin(t) \rangle \qquad \textbf{(5.3.2.3)}$$

Here is how I would apply **myeuler** to the system:

```
> unassign('u');
> f:=(t,u)-><-u[2],u[1]>;
```

$$f := (t, u) \rightarrow \langle -u_2, u_1 \rangle \qquad \textbf{(5.3.2.4)}$$

```
> n:=20;
```

$$n := 20 \qquad \textbf{(5.3.2.5)}$$

```
> U1:=myeuler(f,0.0,<1.0,0.0>,10.0,n);
```

$$U1 := \begin{bmatrix} 21\ x\ 2\ Matrix \\ Data\ Type: anything \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix} \qquad \textbf{(5.3.2.6)}$$

Now I have computed the solution, but it is not stored in a very convenient fashion. The second column of U1 consists of 2-vectors; for instance, here is the last computed value:

```
> U1[n+1,2];
```

$$\begin{bmatrix} -9.20609188079834 \\ 1.40856170654297 \end{bmatrix} \qquad \textbf{(5.3.2.7)}$$

I can display the components of the solution by first extracting them, along with the time grid:

```
> tt:=U1[1..n+1,1];
```

$$tt := \begin{bmatrix} 1\ ..\ 21\ Vector_{column} \\ Data\ Type: anything \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix} \qquad \textbf{(5.3.2.8)}$$

```
> U11:=Vector(n+1,i->U1[i,2][1]);
```

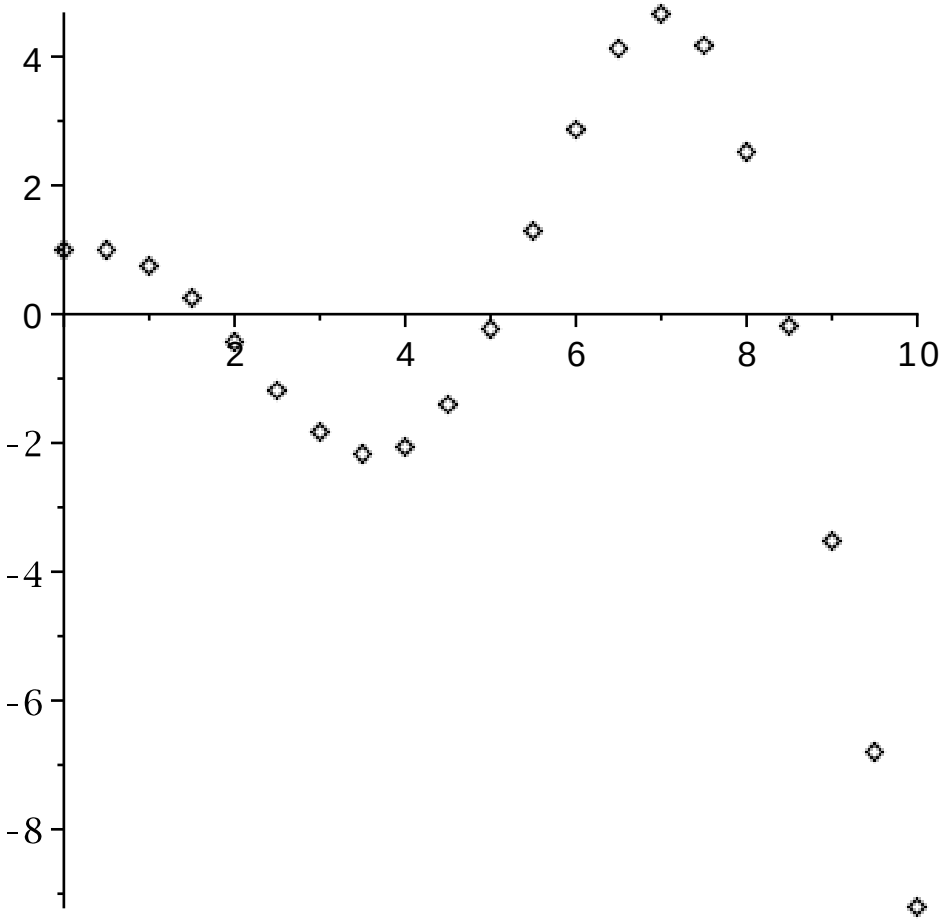
$$U11 := \begin{bmatrix} 1\ ..\ 21\ Vector_{column} \\ Data\ Type: anything \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix} \qquad \textbf{(5.3.2.9)}$$

```
> U12:=Vector(n+1,i->U1[i,2][2]);
```

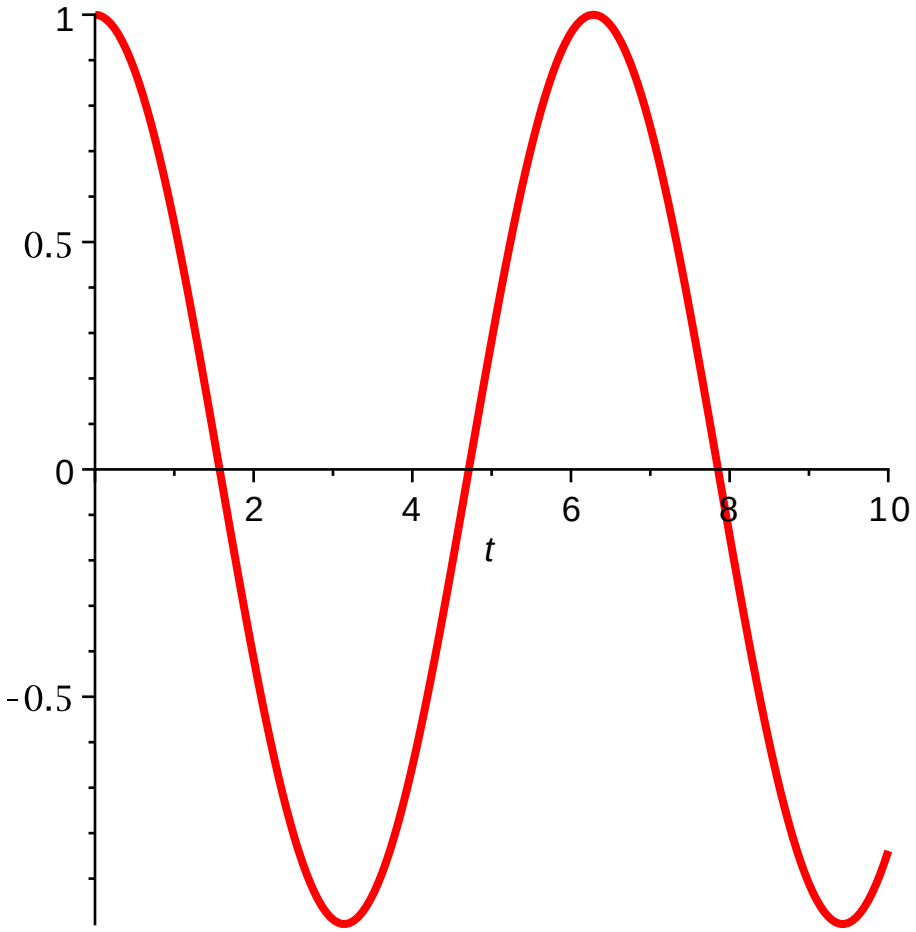
$$U12 := \begin{bmatrix} 1 \,..\, 21 \; Vector_{column} \\ Data\ Type: anything \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix} \qquad \textbf{(5.3.2.10)}$$

Now I can plot the first component of the solution using the **pointplot** command:

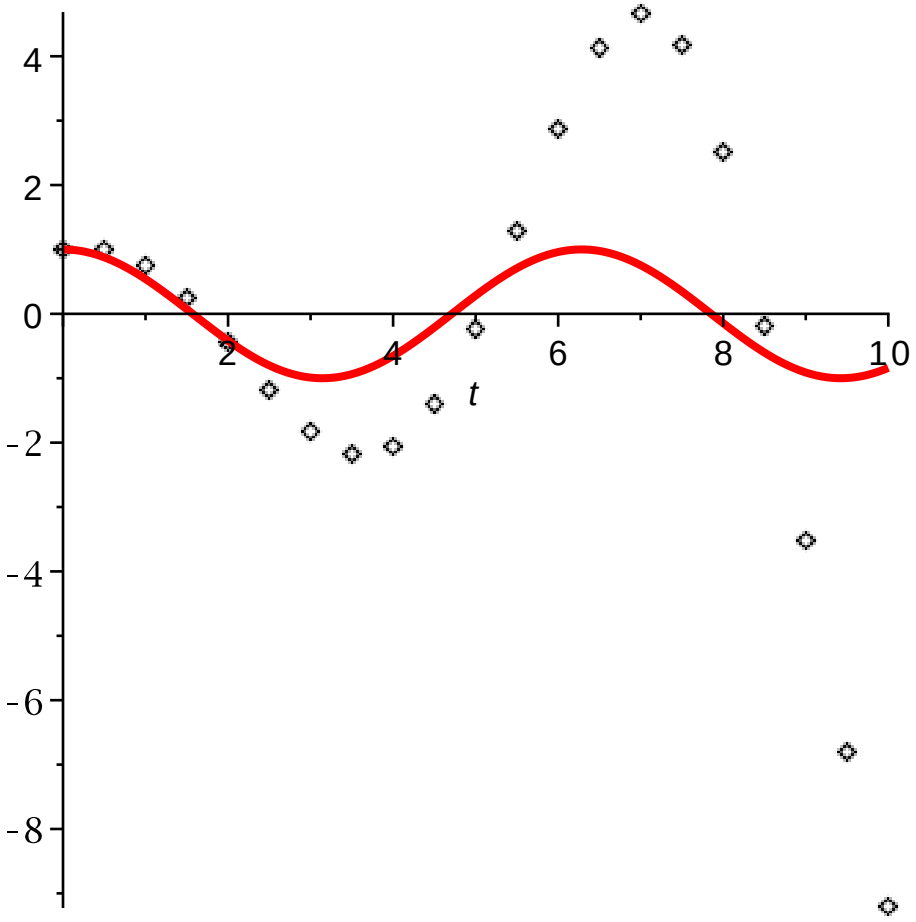
```
> pointplot(<tt|U11>,symbolsize=15);
```




```
> plot1:=%:
> plot(x(t)[1],t=0..10,thickness=3);
```




```
> plot2:=%:
> display(plot1,plot2);
```



Obviously the computed solution is not very accurate. I leave it to you to verify that, if the number of steps is increased, the error decreases in a satisfactory way.

Storing programs in files

Having decided that a certain algorithm is useful enough to be made into a *Maple* command, you will probably wish to save it in a file, so that it can be loaded (rather than type anew) whenever it is needed in a new *Maple* session. Doing this is simple enough: the *Maple* commands defining the function are typed into a plain text file and read into *Maple* using the **read** command. For example, I created a text file containing the **myeuler** command defined above (the file is also called "myeuler"). I read it in as follows:

```
> read myeuler;
myeuler := proc(f, t0, u0, T, n)                    (5.3.3.1)
    local i, U, dt;
    U := Matrix(n + 1, 2);
    dt := evalf(T / n);
```

```
    U[1, 1] := t0;
    U[1, 2] := u0;
    for i to n do
        U[1 + i, 1] := U[i, 1] + dt; U[1 + i, 2] := U[i, 2] + dt*f(U[i, 1],
        U[i, 2])
    end do;
    U
end proc
```

I can now use **myeuler** as before.

When working with user-defined files, it is important that *Maple* be able to find the files. By default, *Maple* looks in the current directory. You can determine the current directory with the **currentdir** command:

```
> currentdir();
        "/home/msgocken/books/pdebook2/tutorial/maple"          (5.3.3.2)
```

The most recent **read** command may have failed when you ran it; this would happen if the file "myeuler" is not stored (on your computer) in the current directory. You can change the current directory with the same **currentdir** command; the syntax is "currentdir(new_dir)", where "new_dir" is the name of the new directory (use the full path name). If the directory name contains special characters (like spaces), enclose it in quotes.

If necessary, you can give the full path name of the file. Here is an alternate version of the above command:

```
> read
  "/home/msgocken/books/pdebook2/tutorial/maple/myeuler";
myeuler := proc(f, t0, u0, T, n)                                  (5.3.3.3)
    local i, U, dt;
    U := Matrix(n + 1, 2);
    dt := evalf(T/n);
    U[1, 1] := t0;
    U[1, 2] := u0;
    for i to n do
        U[1 + i, 1] := U[i, 1] + dt; U[1 + i, 2] := U[i, 2] + dt*f(U[i, 1],
        U[i, 2])
```

```
    end do;
    U
end proc
```

(On a Windows-based machine, the directory separator (the backslash) must be repeated; also, the file name must be enclosed in quotes if it contains spaces.) The above command is likely to fail on your computer, unless you happen to have the same directory structure as I have.

Chapter 5: Boundary value problems in statics

Section 5.2: Introduction to the spectral method; eigenfunctions

I will begin this section by verifying that the eigenfunctions of the negative second derivative operator (under Dirichlet conditions),

$$\sin\left(\frac{n\pi x}{l}\right),\ n = 1,\ 2,\ 3,\ \ldots$$

are mutually orthogonal on the interval $[0,\ l]$. (This is not necessary, as we know from the symmetry of the operator that the eigenfunctions must be orthogonal. However, it is a chance to illustrate another capability of *Maple*.)

```
> unassign('x','m','n','l');
> int(sin(n*Pi*x/l)*sin(m*Pi*x/l),x=0..l);
```

$$\frac{l\,(n\cos(n\pi)\,\sin(m\pi) - m\sin(n\pi)\,\cos(m\pi))}{\pi\,(-n^2 + m^2)} \qquad \textbf{(6.1.1)}$$

```
> simplify(%);
```

$$\frac{l\,(n\cos(n\pi)\,\sin(m\pi) - m\sin(n\pi)\,\cos(m\pi))}{\pi\,(-n^2 + m^2)} \qquad \textbf{(6.1.2)}$$

At first glance, this result is surprising: Why did *Maple* not obtain the expected result, 0? However, a moment's thought reveals the reason: The integral is not necessarily zero unless m and n are integers, and *Maple* has no way of knowing that m and n are intended to represent integers. Fortunately, there is a way to give *Maple* this information:

```
> assume(n,integer);
> assume(m,integer);
> int(sin(n*Pi*x/l)*sin(m*Pi*x/l),x=0..l);
```

$$0 \qquad \textbf{(6.1.3)}$$

When you tell *Maple* to make assumptions about a symbol, it displays the symbol with a tilde (~) to indicate that assumptions are made about that symbol:

```
> m;
```

$$m\sim \tag{6.1.4}$$

```
> n;
```

$$n\sim \tag{6.1.5}$$

When performing Fourier series computations, the ability to assume that a symbol represents an integer is very useful.

Example 5.7

Let

```
> unassign('x');
> f:=x->x*(1-x);
```

$$f := x \to x\,(1-x) \tag{6.1.1.1}$$

I can easily compute the Fourier sine coefficients of f on the interval $[0, 1]$:

```
> 2*int(f(x)*sin(n*Pi*x),x=0..1);
```

$$-\frac{4\left(-1+(-1)^{n\sim}\right)}{n\sim^{3}\,\pi^{3}} \tag{6.1.1.2}$$

This result is fairly simple because n is known to be an integer.

Now I define the Fourier coefficient as a function, for convenience:

```
> a:=unapply(%,n);
```

$$a := n\sim \to -\frac{4\left(-1+(-1)^{n\sim}\right)}{n\sim^{3}\,\pi^{3}} \tag{6.1.1.3}$$

I can represent the (partial) Fourier sine series using the add command. I take the number of terms to be an input, along with the variable x, so that I can experiment with the convergence of the series.

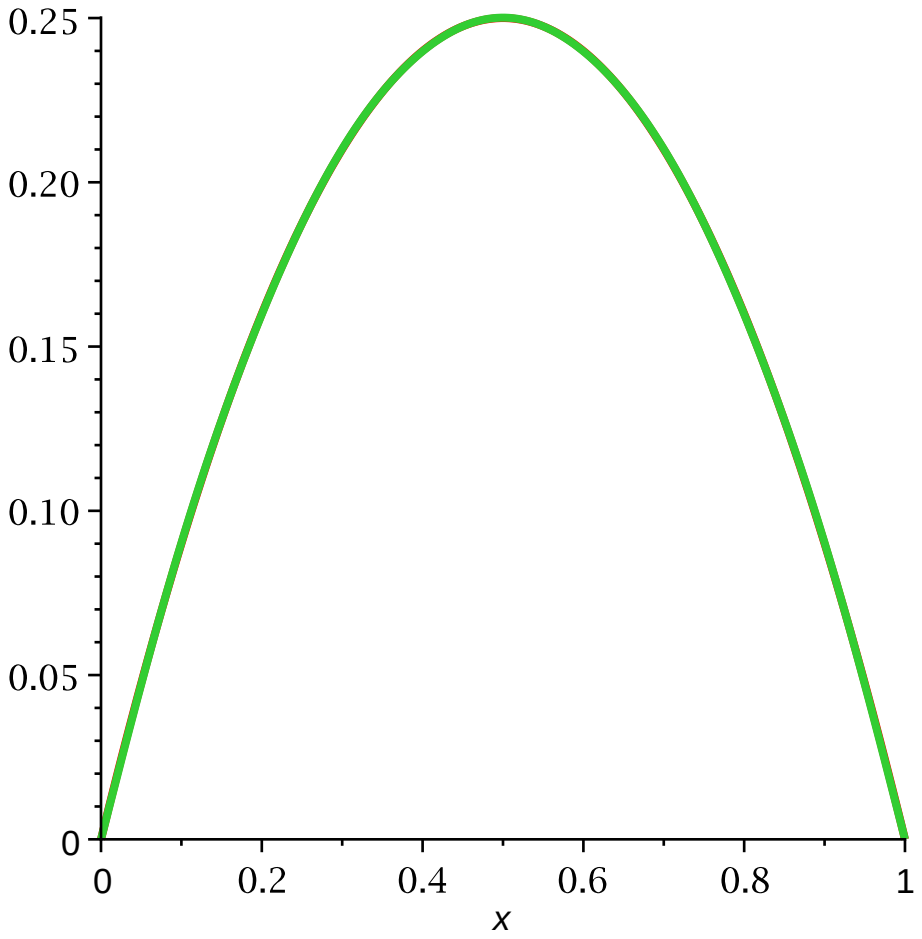
```
> unassign('x','N');
> S:=(x,N)->add(a(n)*sin(n*Pi*x),n=1..N);
```

$$S := (x, N) \to add(a(n)\,\sin(n\pi x),\ n = 1..N) \tag{6.1.1.4}$$

Here is a graph of the partial Fourier series with 10 terms. I include the

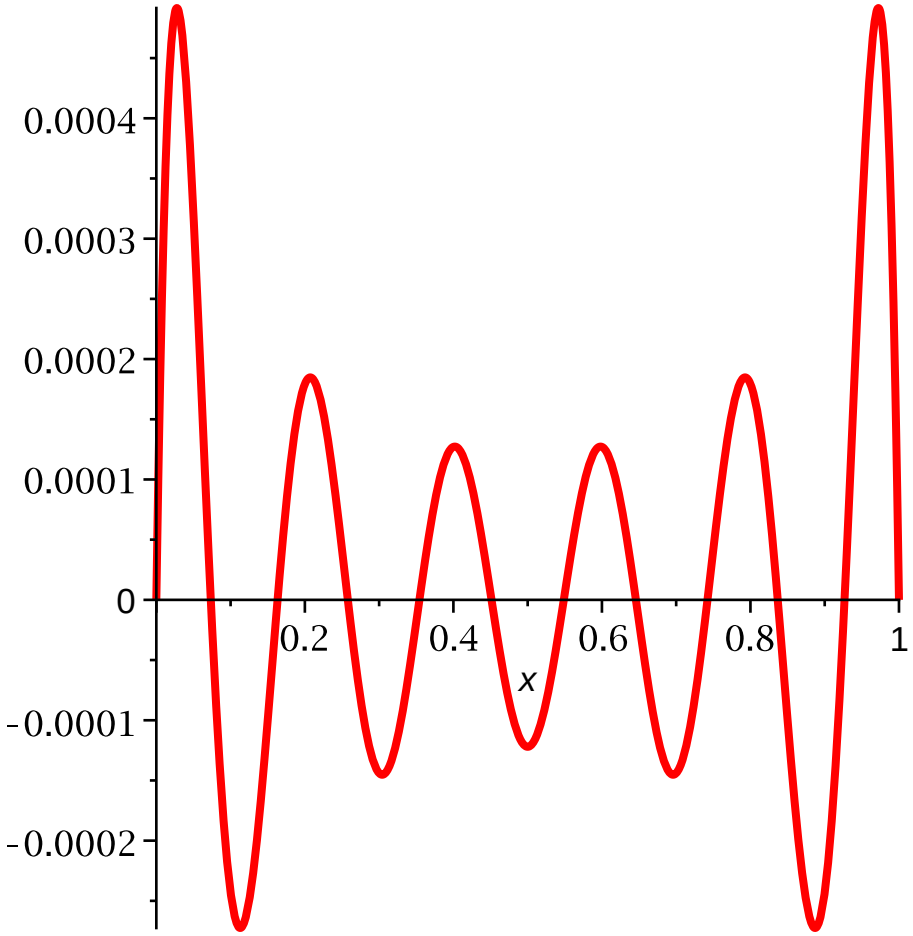
original function for comparison:

```
> plot({f(x),S(x,10)},x=0..1,thickness=3);
```



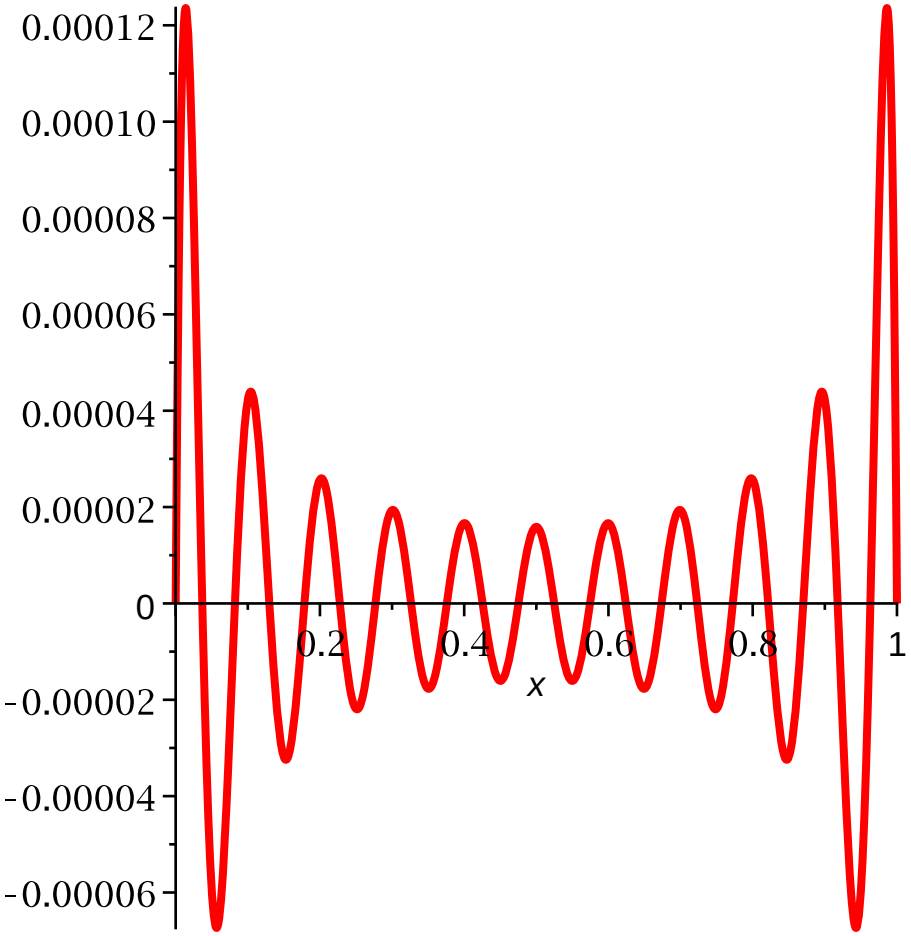
The two functions are so close that it is more informative to graph the error (that is, the difference):

```
> plot(f(x)-S(x,10),x=0..1,thickness=3);
```

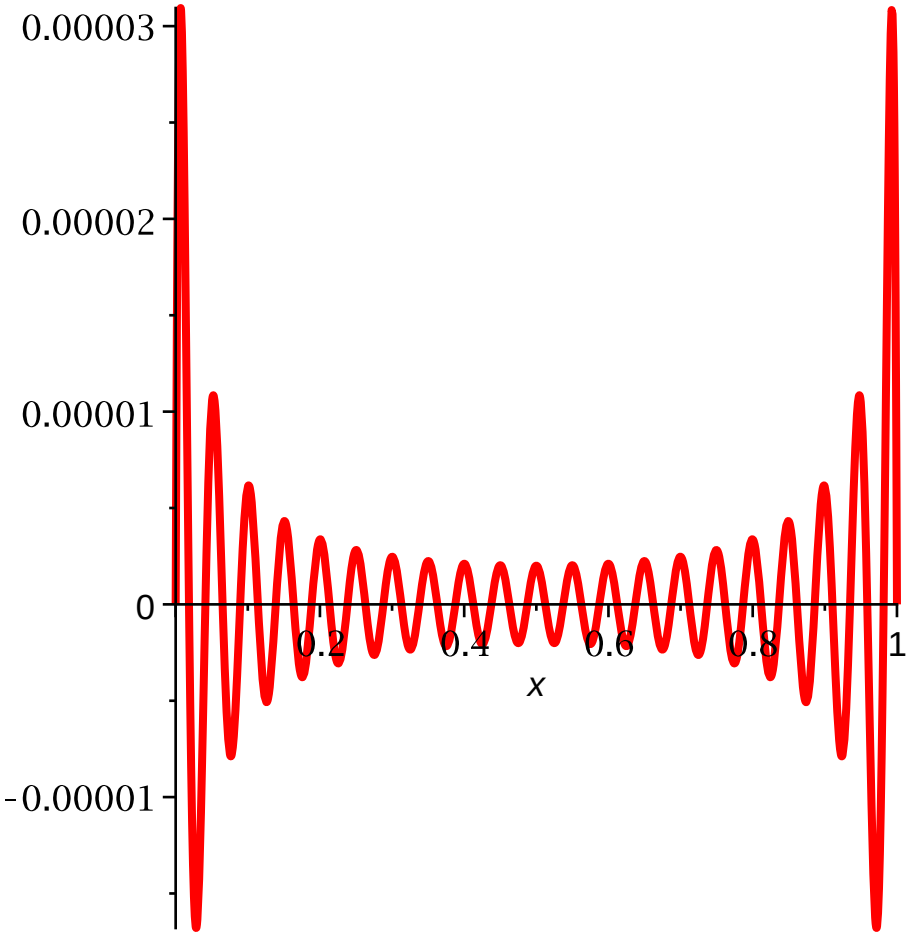


We can see how the approximation improves when we increase the number of terms:

```
> plot(f(x)-S(x,20),x=0..1,thickness=3);
```




```
> plot(f(x)-S(x,40),x=0..1,thickness=3);
```



Section 5.5: The Galerkin method

I will now show how to apply the Galerkin method with a polynomial basis. Suppose we wish to approximate the solution to the BVP

$$-\left(\frac{d}{dx}\left((1+x)\frac{du}{dx}\right)\right) = x^2, 0 < x, x < 1,$$
$$u(0) = 0, u(1) = 0$$

using the subspace spanned by the following four polynomials:

```
> unassign('x');
> p[1]:=x->x*(1-x):
> p[2]:=x->x*(1/2-x)*(1-x):
> p[3]:=x->x*(1/3-x)*(2/3-x)*(1-x):
> p[4]:=x->x*(1/4-x)*(1/2-x)*(3/4-x)*(1-x):
```

The energy inner product is defined as follows:

```
> unassign('u','v');
> a:=(u,v)->int((1+x)*diff(u(x),x)*diff(v(x),x),x=0..1);
```

$$a := (u, v) \to \int_0^1 (1+x) \left(\frac{\mathrm{d}}{\mathrm{d}x}\, u(x)\right) \left(\frac{\mathrm{d}}{\mathrm{d}x}\, v(x)\right) \mathrm{d}x \qquad \textbf{(6.2.1)}$$

The L^2 inner product is given by

```
> L:=(u,v)->int(u(x)*v(x),x=0..1);
```

$$L := (u, v) \to \int_0^1 u(x)\, v(x)\, \mathrm{d}x \qquad \textbf{(6.2.2)}$$

Now the calculation is simple (but it would be very tedious to carry out by hand): We just need to compute the stiffness matrix and the load vector, and solve the linear system. The stiffness matrix is

```
> K:=Matrix(4,4,(i,j)->a(p[j],p[i]),shape=symmetric);
```

$$K := \begin{bmatrix} \frac{1}{2} & -\frac{1}{30} & \frac{1}{90} & -\frac{1}{672} \\ -\frac{1}{30} & \frac{3}{40} & -\frac{19}{3780} & \frac{3}{896} \\ \frac{1}{90} & -\frac{19}{3780} & \frac{5}{567} & -\frac{41}{60480} \\ -\frac{1}{672} & \frac{3}{896} & -\frac{41}{60480} & \frac{43}{43008} \end{bmatrix} \qquad \textbf{(6.2.3)}$$

The load vector is

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

$$f := x \to x^2 \qquad \textbf{(6.2.4)}$$

```
> F:=Vector(4,i->L(p[i],f));
```

$$F := \begin{bmatrix} \frac{1}{20} \\ -\frac{1}{120} \\ \frac{1}{630} \\ -\frac{1}{2688} \end{bmatrix} \qquad \textbf{(6.2.5)}$$

Then the coefficients defining the (approximate) solution are

```
> c:=LinearSolve(K,F);
```

$$c := \begin{bmatrix} \frac{3325}{34997} \\ -\frac{9507}{139988} \\ \frac{1575}{69994} \\ \frac{420}{34997} \end{bmatrix} \qquad \textbf{(6.2.6)}$$

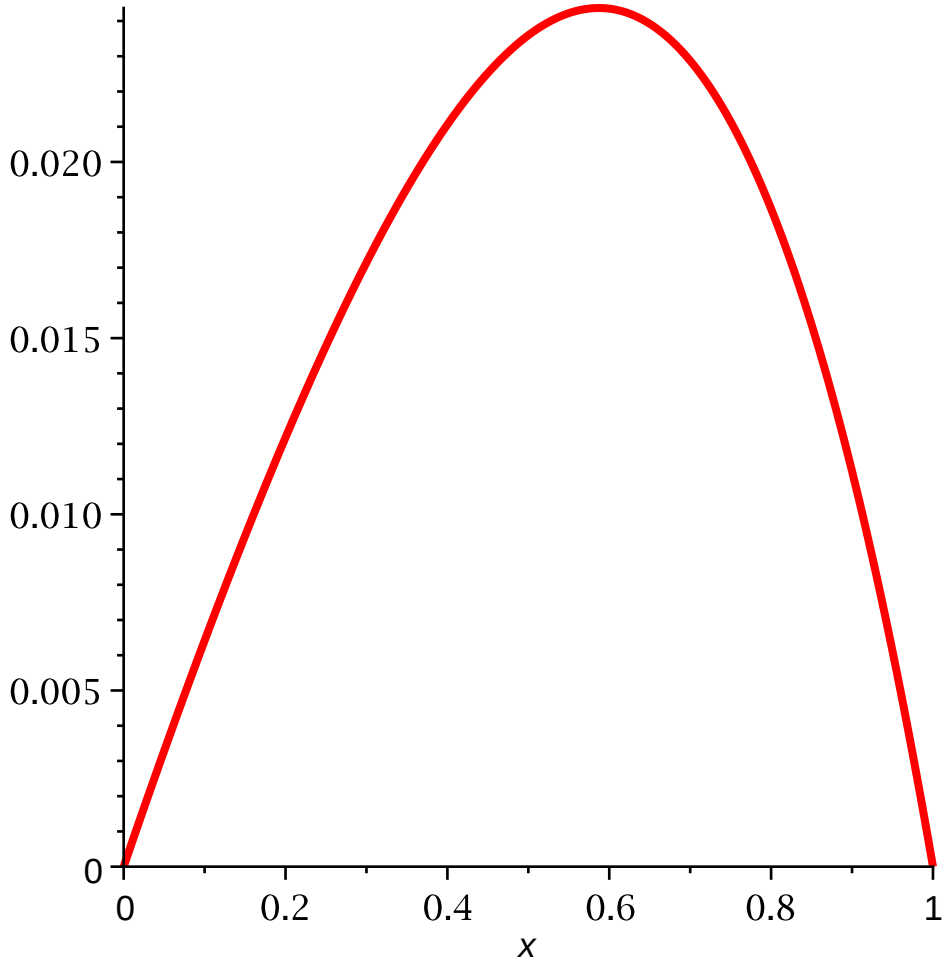
and the (approximate) solution is

```
> v:=unapply(add(c[i]*p[i](x),i=1..4),x);
```

$$v := x \rightarrow \frac{3325}{34997}\, x\,(1-x) - \frac{9507}{139988}\, x\left(\frac{1}{2}-x\right)(1-x) + \frac{1575}{69994}\, x\left(\frac{1}{3}-x\right)\left(\frac{2}{3}-x\right)(1-x) + \frac{420}{34997}\, x\left(\frac{1}{4}-x\right)\left(\frac{1}{2}-x\right)\left(\frac{3}{4}-x\right)(1-x) \qquad \textbf{(6.2.7)}$$

Here is a graph:

```
> plot(v(x),x=0..1,thickness=3);
```



The exact solution can be found by direct integration:

```
> unassign('x','s','c1','c2');
> int(-s^2,s=0..x)+c1;
```

$$-\frac{1}{3}x^3 + c1 \qquad \textbf{(6.2.8)}$$

```
> int((c1-s^3/3)/(1+s),s=0..x);
```

Warning, unable to determine if -1 is between 0 and x; try to use assumptions or use the AllSolutions option

$$\int_0^x \frac{c1 - \frac{1}{3}s^3}{1+s}\,ds \qquad \textbf{(6.2.9)}$$

Notice the error message; *Maple* could not compute the integral because it does not know if there is a singularity (a zero in the denominator) on the interval of integration. Since we are only interested in x≥0, we can impose this as an assumption:

```
> int((c1-s^3/3)/(1+s), s=0..x) assuming x≥0;
```

(6.2.10)

$$-\frac{1}{9}x^3-\frac{1}{3}x+\frac{1}{6}x^2+\frac{1}{3}\ln(1+x)+\ln(1+x)\,c1 \qquad \textbf{(6.2.10)}$$

```
> u:=unapply(%,x);
```

$$u:=x\rightarrow -\frac{1}{9}x^3-\frac{1}{3}x+\frac{1}{6}x^2+\frac{1}{3}\ln(x+1)+\ln(x+1)\,c1 \qquad \textbf{(6.2.11)}$$

```
> sols:=solve(u(1)=0,c1);
```

$$sols:=-\frac{1}{18}\,\frac{-5+6\ln(2)}{\ln(2)} \qquad \textbf{(6.2.12)}$$

```
> c1:=sols;
```

$$c1:=-\frac{1}{18}\,\frac{-5+6\ln(2)}{\ln(2)} \qquad \textbf{(6.2.13)}$$

Check:

```
> -diff((1+x)*diff(u(x),x),x);
```

$$\frac{1}{3}x^2+\frac{1}{3}-\frac{1}{3}x-\frac{1}{3(1+x)}+\frac{1}{18}\,\frac{-5+6\ln(2)}{(1+x)\ln(2)}-(1+x)\left(-\frac{2}{3}x+\frac{1}{3}-\frac{1}{3(1+x)^2}+\frac{1}{18}\,\frac{-5+6\ln(2)}{(1+x)^2\ln(2)}\right) \qquad \textbf{(6.2.14)}$$

```
> simplify(%);
```

$$x^2 \qquad \textbf{(6.2.15)}$$

```
> u(0);
```

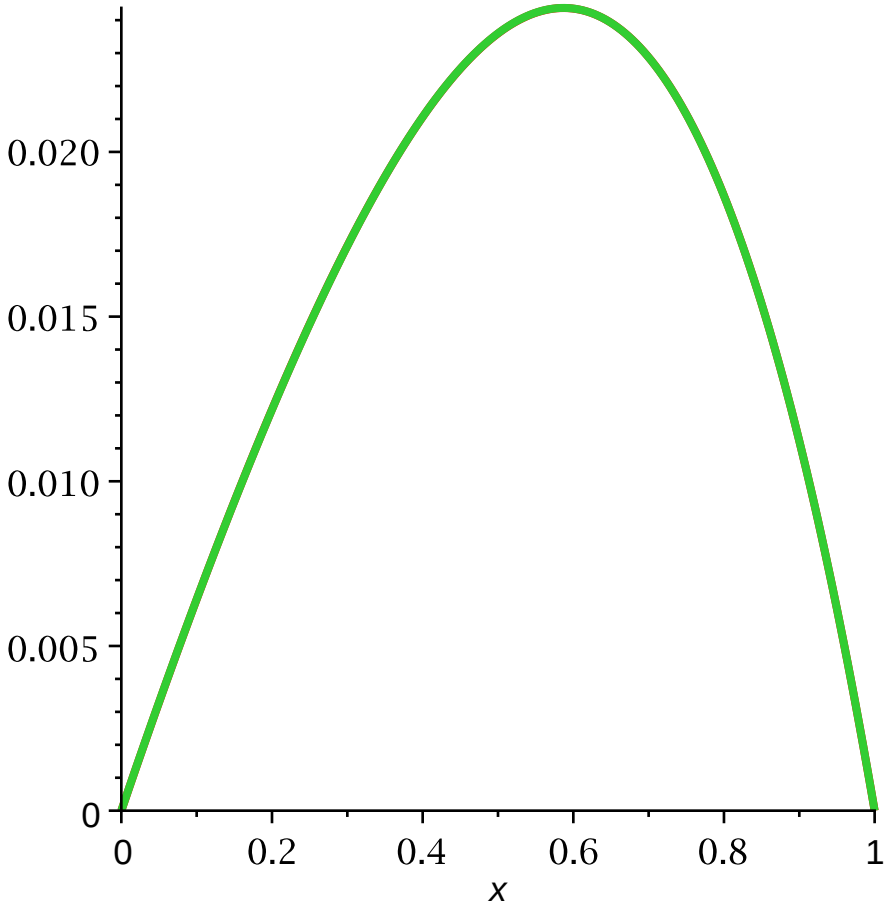
$$0 \qquad \textbf{(6.2.16)}$$

```
> u(1);
```

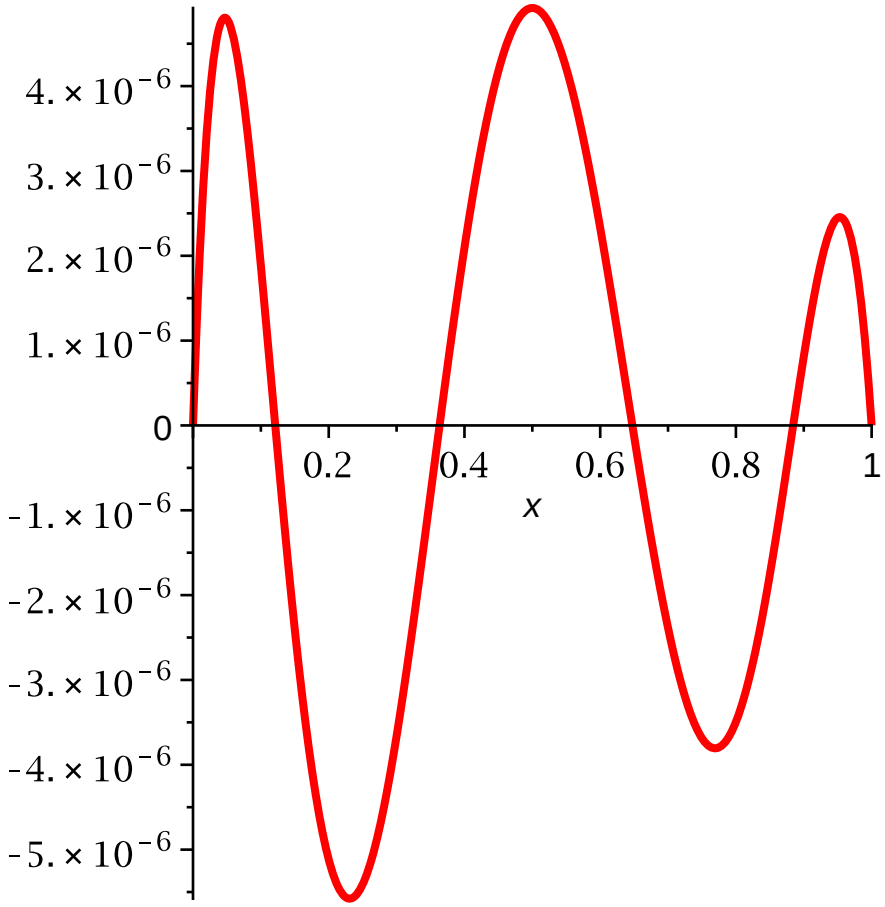
$$0 \qquad \textbf{(6.2.17)}$$

Now we can compare the exact and approximate solutions:

```
> plot({u(x),v(x)},x=0..1,thickness=3);
```




```
> plot(u(x)-v(x),x=0..1,thickness=3);
```



The computed solution is quite accurate.

Section 5.6: Piecewise polynomials and the finite element method

As in my textbook, I will only discuss piecewise linear finite element methods. You will recall that a piecewise linear function is determined by the mesh and the nodal values of the functions. For example, here is a regular mesh on the interval $[0, 1]$:

```
> with(LinearAlgebra):
> n:=5:
> X:=Vector(n+1,i->(i-1)/n);
```

(6.3.1)

$$X := \begin{bmatrix} 0 \\ \frac{1}{5} \\ \frac{2}{5} \\ \frac{3}{5} \\ \frac{4}{5} \\ 1 \end{bmatrix} \tag{6.3.1}$$

Here are the nodal values of the piecewise linear interpolant of the function $\sin(\pi x)$ relative to this mesh:

```
> Y:=Vector(n+1,i->evalf(sin(Pi*X[i])));
```

$$Y := \begin{bmatrix} 0. \\ 0.5877852524 \\ 0.9510565165 \\ 0.9510565165 \\ 0.5877852524 \\ 0. \end{bmatrix} \tag{6.3.2}$$

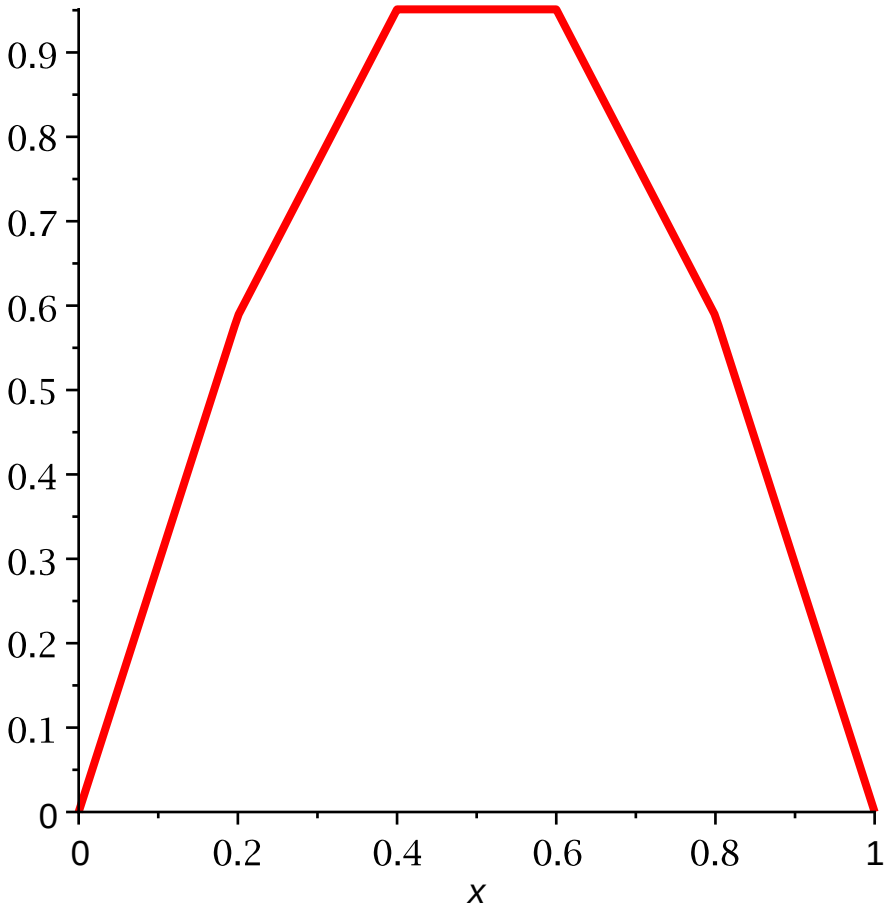
I can create the piecewise linear function defined by these nodal values using the **Spline** command. Since I want a piecewise *linear* interpolant, I must give the "degree=1" option:

```
> with(CurveFitting,Spline):
> unassign('x');
> p:=unapply(Spline(X,Y,x,degree=1),x);
```

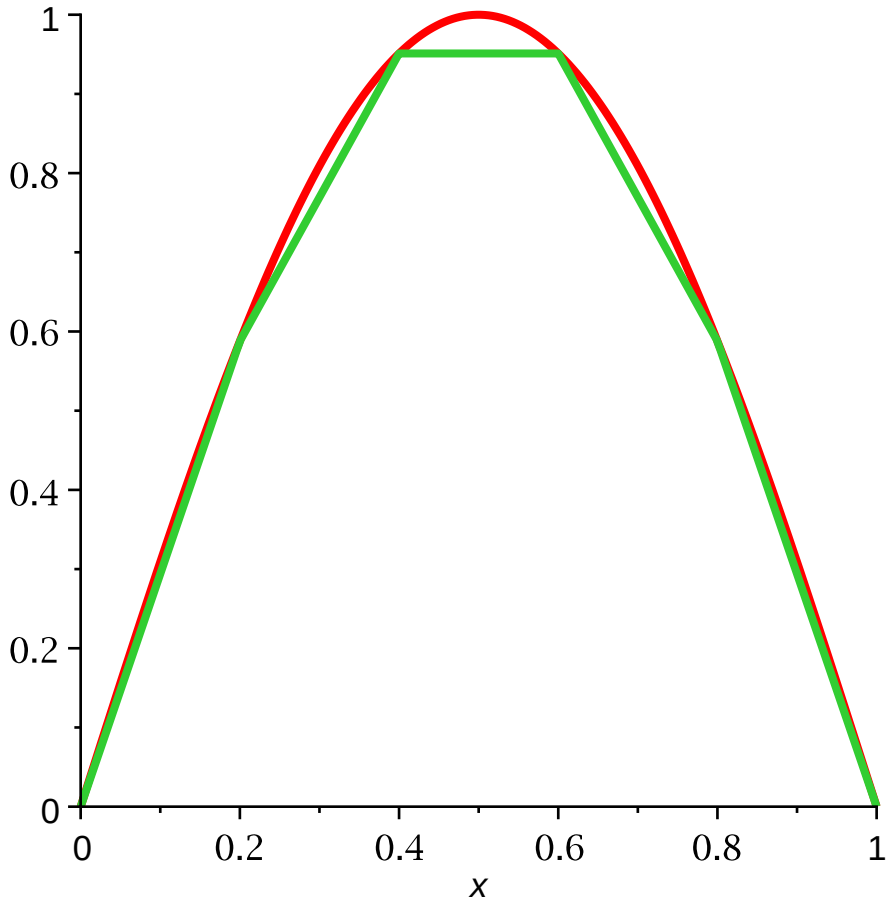
$$p := x \rightarrow piecewise\left(x < \frac{1}{5}, 2.938926262\, x, x < \frac{2}{5}, 0.2245139885 + 1.816356320\, x, x < \frac{3}{5}, 0.9510565165, x < \frac{4}{5}, 2.040870309 - 1.816356320\, x, 2.938926262 - 2.938926262\, x\right) \tag{6.3.3}$$

I can now plot the piecewise linear function:

```
> plot(p(x),x=0..1,thickness=3);
```




```
> plot({sin(Pi*x),p(x)},x=0..1,thickness=3);
```

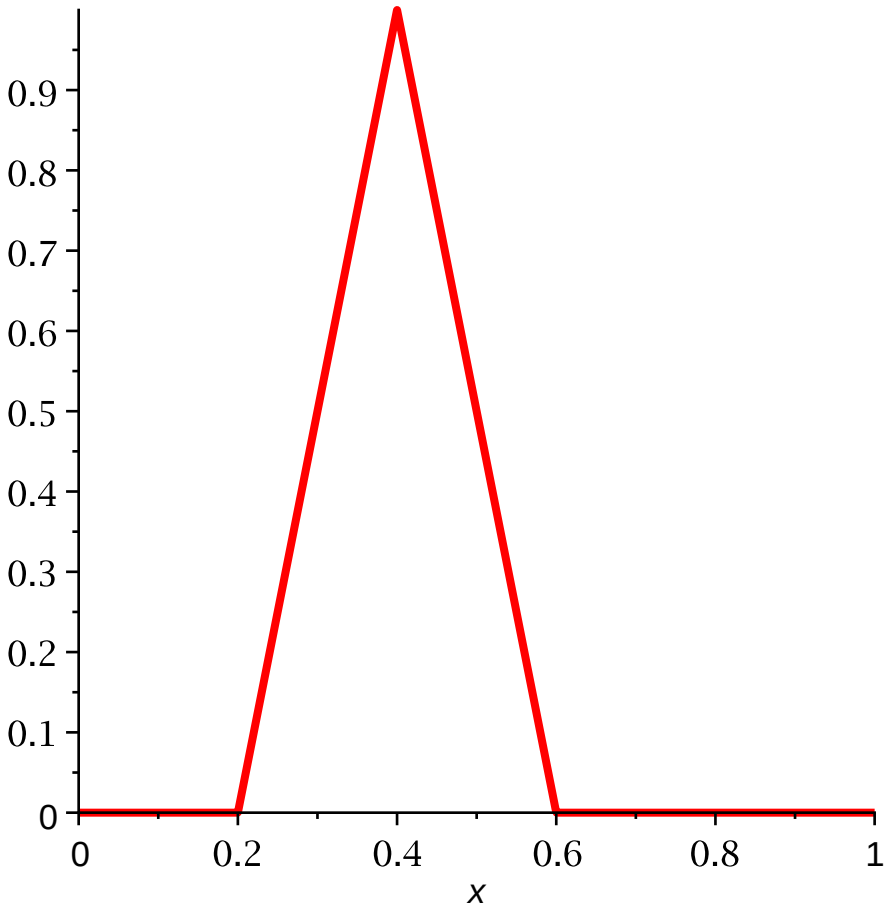


We can use the **Spline** command to represent the standard piecewise linear basis functions. Here is an example:

```
> Y3:=Vector(n+1,i->0):
> Y3[3]:=1:
> phi3:=unapply(Spline(X,Y3,x,degree=1),x);
```

$$\phi 3 := x \rightarrow piecewise\left(x < \frac{1}{5}, 0, x < \frac{2}{5}, -1 + 5\,x, x < \frac{3}{5}, 3 - 5\,x, x < \frac{4}{5}, 0, 0\right) \quad \textbf{(6.3.4)}$$

```
> plot(phi3(x),x=0..1,thickness=3);
```



I can now use the **Spline** command to create a basis and then apply the finite element method. For example, suppose I wish to apply the finite element method, with piecewise linear functions, to the following BVP:

$$-\left(\frac{\partial}{\partial x}\left(k(x)\left(\frac{\partial}{\partial x}u\right)\right)\right) = x, 0 < x, x < 1,$$
$$u(0) = 0, u(1) = 0.$$

I begin by defining a mesh and all of the basis functions:

```
> n:=10:
> X:=Vector(n+1,i->evalf((i-1)/n)):
> for i from 0 to n
> do
>    Y:=Vector(n+1,i->0.0):
>    Y[i+1]:=1.0:
>    phi[i]:=unapply(Spline(X,Y,x,degree=1),x):
> end do:
```

I define the energy and L^2 inner products for convenience. I will take $k(x) = 1 + x$ for this example.

```
> k:=x->1+x;
```

$$k := x \rightarrow x + 1 \qquad (6.3.5)$$

```
> a:=(u,v)->int(k(x)*D(u)(x)*D(v)(x),x=0..1);
```

$$a := (u, v) \rightarrow \int_0^1 k(x)\, \mathrm{D}(u)(x)\, \mathrm{D}(v)(x)\, \mathrm{d}x \qquad (6.3.6)$$

```
> L:=(u,v)->int(u(x)*v(x),x=0..1);
```

$$L := (u, v) \rightarrow \int_0^1 u(x)\, v(x)\, \mathrm{d}x \qquad (6.3.7)$$

I could assemble the stiffness matrix with a single command:

K:=Matrix(n-1,n-1,(i,j)->a(phi[j],phi[i]),shape=symmetric);

However, this command would not take advantage of the fact that K is sparse, and so would be very inefficient. Instead, I will create a sparse matrix with no nonzero entries and then fill in the nonzeros in two loops.

The following command creates an "empty" tridiagonal, symmetric matrix ("band [1,0]" means that only the first subdiagonal and the main diagonal of the matrix will be stored; this is sufficient, since the matrix is tridiagonal and symmetric):

```
> K:=Matrix(n-1,n-1,shape=symmetric,storage=band[1,0]);
```

$$K := \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (6.3.8)$$

Next, I fill in the main diagonal (as explained in the text, the stiffness matrix in this problem is tridiagonal---the only nonzeros lie on the main diagonal, and the first sub- and super-diagonals.). Notice that the main diagonal of K has $n-1$ entries.

```
> for i from 1 to n-1
> do
```

```
>    K[i,i]:=a(phi[i],phi[i]):
> end do:
```

```
> K;
```

$$\begin{bmatrix} 22. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 24. & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 26. & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 28. & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30. & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32. & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 34. & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 36. & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 38. \end{bmatrix} \quad \textbf{(6.3.9)}$$

Finally, I fill in the sub-diagonal (since I told *Maple* that K is symmetric, I do not need to fill in the super-diagonal).

```
> for i from 1 to n-2
> do
>    K[i+1,i]:=a(phi[i],phi[i+1]):
> end do:
```

(Notice that the loop executes $n-2$ times, since the sub- and super-diagonals each have $n-2$ entries.)

```
> K;
```

```
[[22., -11.50000000, 0, 0, 0, 0, 0, 0, 0],                    (6.3.10)
    [-11.50000000, 24., -12.50000000, 0, 0, 0, 0, 0, 0],
    [0, -12.50000000, 26., -13.50000000, 0, 0, 0, 0, 0],
    [0, 0, -13.50000000, 28., -14.50000000, 0, 0, 0, 0],
    [0, 0, 0, -14.50000000, 30., -15.50000000, 0, 0, 0],
    [0, 0, 0, 0, -15.50000000, 32., -16.50000000, 0, 0],
    [0, 0, 0, 0, 0, -16.50000000, 34., -17.50000000, 0],
    [0, 0, 0, 0, 0, 0, -17.50000000, 36., -18.50000000],
    [0, 0, 0, 0, 0, 0, 0, -18.50000000, 38.]]
```

Next, I compute the load vector:

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

$$f := x \rightarrow x^2 \quad \textbf{(6.3.11)}$$

```
> F:=Vector(n-1,i->L(phi[i],f));
```

$$F := \begin{bmatrix} 0.001166666667 \\ 0.004166666667 \\ 0.009166666667 \\ 0.01616666667 \\ 0.02516666667 \\ 0.03616666667 \\ 0.04916666667 \\ 0.06416666667 \\ 0.08116666667 \end{bmatrix} \quad \textbf{(6.3.12)}$$

Finally, I solve the linear system $KU = F$ to find the nodal values:

```
> U:=LinearSolve(K,F);
```

$$U := \begin{bmatrix} 0.00642674694414684 \\ 0.0121931970525418 \\ 0.0171649978189051 \\ 0.0210895047013156 \\ 0.0236284134076978 \\ 0.0243798656381844 \\ 0.0228938561171263 \\ 0.0186832376161287 \\ 0.0112317516465363 \end{bmatrix} \quad \textbf{(6.3.13)}$$

Now I have computed the nodal values of the approximate solution, and I can define the solution using the **Spline** command. I first need to add the boundary values to the vector *U*:

```
> U:=Vector(n+1,[0,U,0]);
```

$$U := \begin{bmatrix} 1 \text{ .. } 11 \ Vector_{column} \\ Data\ Type: anything \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix} \quad \textbf{(6.3.14)}$$

(The above command creates a vector with $n+1$ components, initialized from

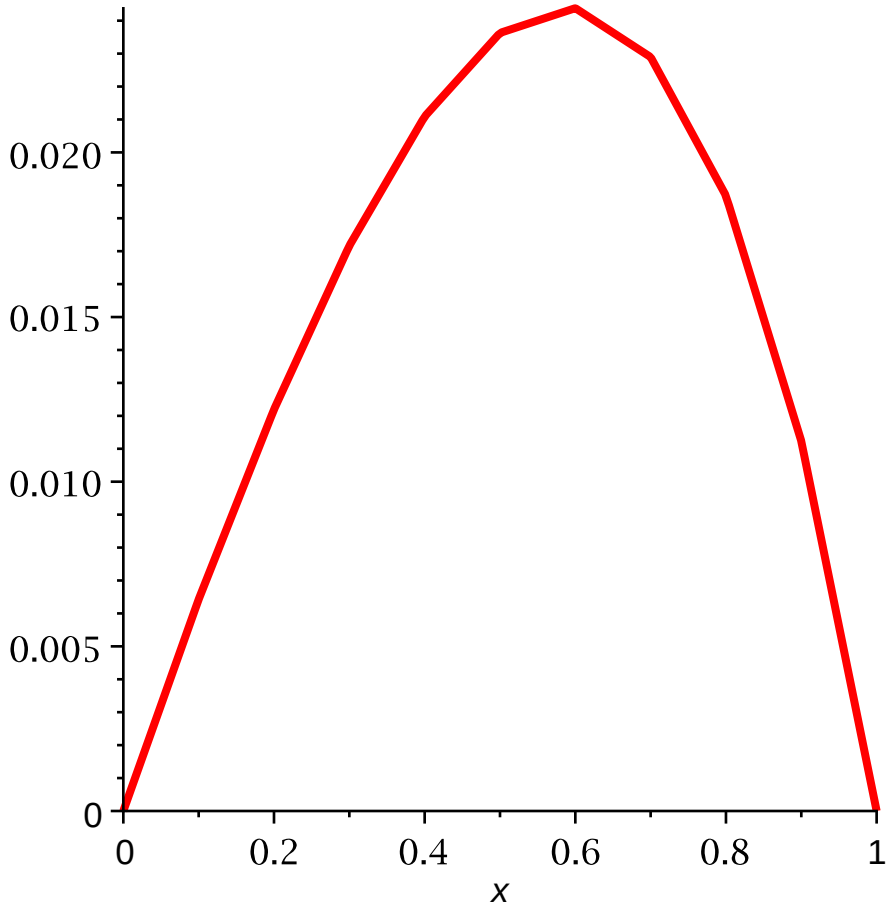
the list with elements 0, *U*, 0.)

```
> v:=Spline(X,U,x,degree=1);
```

$$v := \begin{cases} 0.0642674694414683\,x & x < 0.1000000000 \\ 0.000660296835751899 + 0.0576645010839494\,x & x < 0.2000000000 \\ 0.00224959551981509 + 0.0497180076636334\,x & x < 0.3000000000 \\ 0.00539147717167362 + 0.0392450688241050\,x & x < 0.4000000000 \\ 0.0109338698757869 + 0.0253890870638219\,x & x < 0.5000000000 \\ 0.0198711522552650 + 0.00751452230486564\,x & x < 0.6000000000 \\ 0.0332959227645328 - 0.0148600952105808\,x & x < 0.7000000000 \\ 0.0523681856241096 - 0.0421061850099762\,x & x < 0.8000000000 \\ 0.0782951253728674 - 0.0745148596959234\,x & x < 0.9000000000 \\ 0.112317516465363 - 0.112317516465363\,x & otherwise \end{cases} \quad \textbf{(6.3.15)}$$

Here is a graph of the approximate solution:

```
> plot(v(x),x=0..1,thickness=3);
```



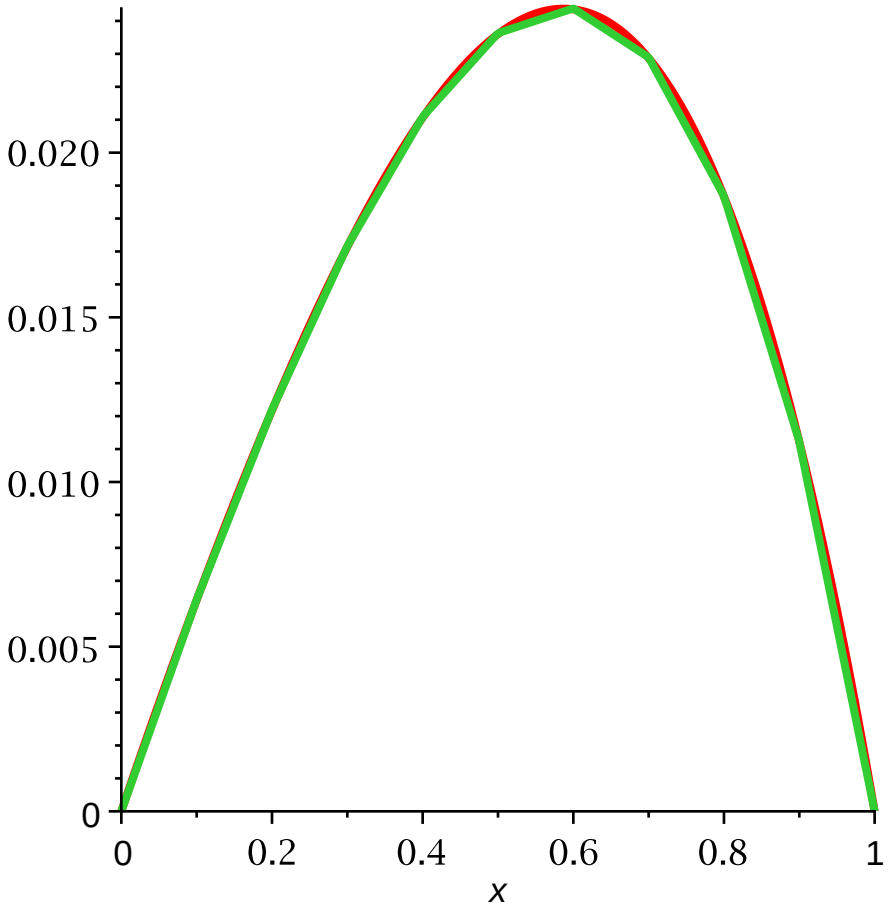
The exact solution was computed in the preceding section:

```
> u(x);
```

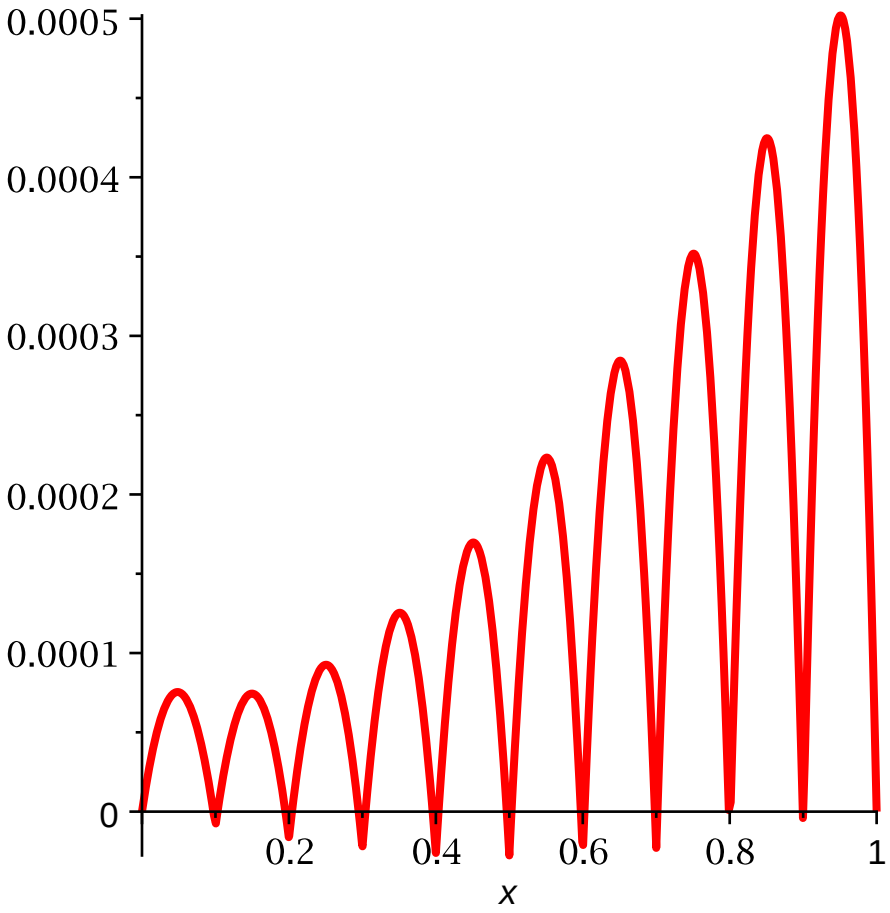
$$-\frac{1}{9}x^3-\frac{1}{3}x+\frac{1}{6}x^2+\frac{1}{3}\ln(1+x)-\frac{1}{18}\frac{\ln(1+x)\,(-5+6\ln(2))}{\ln(2)} \qquad \textbf{(6.3.16)}$$

Here is a comparison of the two solutions:

```
> plot({u(x),v(x)},x=0..1,thickness=3);
```




```
> plot(u(x)-v(x),x=0..1,thickness=3);
```



The above method for computing the stiffness matrix and load vector, while easy to implement, is still not particularly efficient, since *Maple* cannot manipulate the piecewise linear basis functions very quickly. It would be more efficient to define simple linear functions to represent the pieces of the basis functions, and then restrict the intervals of integration. Notice that the basis function ϕ_i satisfies

$$\phi_i(x) = \frac{x - X_i}{h}, X_i < x < X_{i+1},$$
$$\phi_i(x) = -\frac{x - X_{i+2}}{h}, X_{i+1} < x < X_{i+2},$$
$$\phi_i(x) = 0, \text{ otherwise,}$$

where h is the mesh size ($h = \frac{1}{n}$ in this case). (Notice that the nodes of the mesh are enumerated $X_1, X_2, ..., X_{n+1}$, rather than $X_0, X_1, ..., X_n$, as in the text. This is because *Maple* vectors are indexed beginning at 1.) Therefore I define

```
> unassign('x');
> phi1:=(x,i)->(x-X[i])*n;
```

(6.3.17)

$$\phi 1 := (x, i) \to (x - X_i)\ n \qquad \textbf{(6.3.17)}$$

```
> phi2:=(x,i)->-(x-X[i+2])*n;
```

$$\phi 2 := (x, i) \to -(x - X_{i+2})\ n \qquad \textbf{(6.3.18)}$$

(Notice that the mesh *X* and the integer *n* must be defined already.) Now I can compute the load vector more efficiently as follows:

```
> F:=Vector(n-1,i->evalf(int(phi1(x,i)*f(x),x=X[i]..X[i+1])+
>                        int(phi2(x,i)*f(x),x=X[i+1]..X[i+2])));
```

$$F := \begin{bmatrix} 0.001166666667 \\ 0.004166666667 \\ 0.009166666666 \\ 0.01616666667 \\ 0.02516666667 \\ 0.03616666666 \\ 0.04916666667 \\ 0.06416666667 \\ 0.08116666666 \end{bmatrix} \qquad \textbf{(6.3.19)}$$

You should be able to notice how much more quickly this computation executes, as opposed to the previous version.

I can apply the same technique to assemble the stiffness matrix, except that I have no need to differentiate the basis functions on the subintervals---I know that the slope of ϕ_i is either $\frac{1}{h} = n$ or $-\frac{1}{h} = -n$:

```
> K:=Matrix(n-1,n-1,shape=symmetric,storage=band[1,0]):
> for i from 1 to n-1
> do
>    K[i,i]:=evalf(n^2*int(k(x),x=X[i]..X[i+2])):
> end do:
> for i from 1 to n-2
> do
>    K[i+1,i]:=evalf(-n^2*int(k(x),x=X[i+1]..X[i+2])):
> end do:
> K;
```

$$\begin{bmatrix} 22.00000000 & -11.50000000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -11.50000000 & 24.00000000 & -12.50000000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -12.50000000 & 26.00000000 & -13.50000000 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \textbf{(6.3.20)}$$

```
[0, 0, -13.50000000, 28.00000000, -14.50000000, 0, 0, 0, 0],
[0, 0, 0, -14.50000000, 30.00000000, -15.50000000, 0, 0, 0],
[0, 0, 0, 0, -15.50000000, 32.00000000, -16.50000000, 0, 0],
[0, 0, 0, 0, 0, -16.50000000, 34.00000000, -17.50000000, 0],
[0, 0, 0, 0, 0, 0, -17.50000000, 36.00000000, -18.50000000],
[0, 0, 0, 0, 0, 0, 0, -18.50000000, 38.00000000]]
```

(Notice that, for a constant coefficient problem, there would be no need to perform any integrations in assembling K, since then the entries of the stiffness matrix would be constants. The diagonals entries would all be $\frac{2\,k}{h}$, and the offdiagonal nonzero entries would all be $-\frac{k}{h}$.)

This second method of assembling K and F is much more efficient than the first, although it is slightly more complicated because we must deal directly with the mesh, the indexing of the vectors, and so forth. Which method is preferable depends on how important speed is to you. (The second method is similar to how the method would be implemented in a finite element software package, since speed of execution is more important than programming convenience for such a package.)

Chapter 6: Heat flow and diffusion

Section 6.1: Fourier series methods for the heat equation

Example 6.2: An inhomogenous example

Consider the BVP

$$\frac{\partial u}{\partial t} - A\frac{\partial^2 u}{\partial x^2} = 10^{-7},\ 0< x <100,\ t>0,$$
$$u(x,\,0) = 0,\ 0< x <100,$$
$$u(0,\,t) = 0,\ t > 0,$$
$$u(100,\,t) = 0,\ t>0.$$

The constant A has the following value:

```
> A:=0.208;
```

$$A := 0.208 \qquad \textbf{(7.1.1.1)}$$

The solution can be written as

$$u(x,\,t) = \sum_{n=1}^{\infty} a_n(t)\,\sin\left(\frac{n\pi x}{100}\right),$$

where the coefficient $a_n(t)$ satisfies the IVP

$$\left[\frac{\partial}{\partial t}\,a_n\right] + \frac{A\,n^2\,\pi^2\,a_n}{100^2} = c_n,\ t>0,$$
$$a_n(0) = 0.$$

The values c_1, c_2, c_3, ... are the Fourier sine coefficients of the constant function 10^{-7}. Recall the use of the **assume** function for making *Maple* aware the n is an integer.

```
> unassign('n','x');
> assume(n,integer);
> 2/100*int(10^(-7)*sin(n*Pi*x/100),x=0..100);
```

$$-\frac{1}{5000000}\,\frac{-1+(-1)^{n\sim}}{n\sim\,\pi} \qquad \textbf{(7.1.1.2)}$$

```
> c:=unapply(%,n);
```

$$c := n\sim \to -\frac{1}{5000000}\,\frac{-1+(-1)^{n\sim}}{n\sim\,\pi} \qquad \textbf{(7.1.1.3)}$$

I can now compute $a_n(t)$ by the following formula (derived in Section 4.2.4 of the text):

```
> unassign('t','s');
> int(exp(-A*n^2*Pi^2*(t-s)/100^2)*c(n),s=0..t);
```

$$\frac{1}{n\sim^3}\left(0.0003101109080\left(-1.\,e^{-0.0002052877715\,t\,n\sim^2}\right.\right. \qquad \textbf{(7.1.1.4)}$$
$$\left.\left. + e^{-0.0002052877715\,t\,n\sim^2}\,(-1.)^{n\sim} + 1. - 1.\,(-1.)^{n\sim}\right)\right)$$

```
> a:=unapply(%,t,n);
```

$$a := (t,\,n\sim) \to \frac{1}{n\sim^3}\left(0.0003101109080\left(-1.\,e^{-0.0002052877715\,t\,n\sim^2}\right.\right. \qquad \textbf{(7.1.1.5)}$$
$$\left.\left. + e^{-0.0002052877715\,t\,n\sim^2}\,(-1.)^{n\sim} + 1. - 1.\,(-1.)^{n\sim}\right)\right)$$

I can now define the Fourier series solution:

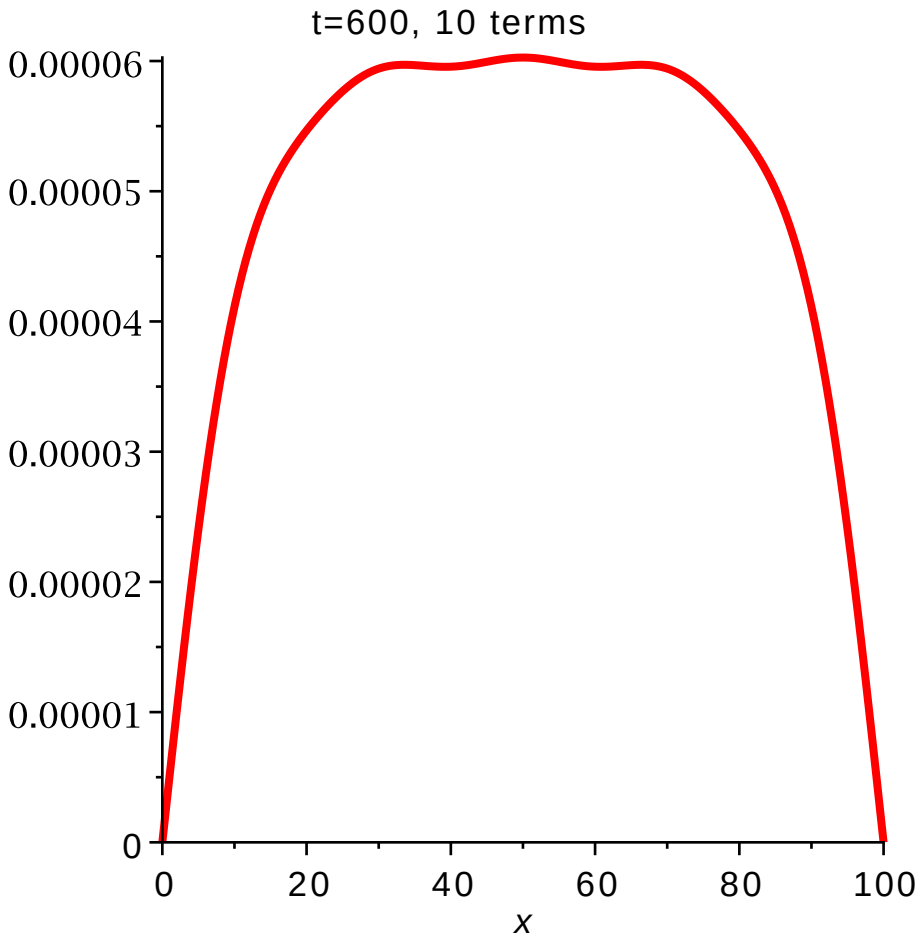
```
> unassign('N');
```

```
> u:=(x,t,N)->add(a(t,n)*sin(n*Pi*x/100),n=1..N);
```

$$u := (x, t, N) \rightarrow add\left(a(t, n)\sin\left(\frac{1}{100}n\pi x\right), n = 1..N\right) \qquad \textbf{(7.1.1.6)}$$

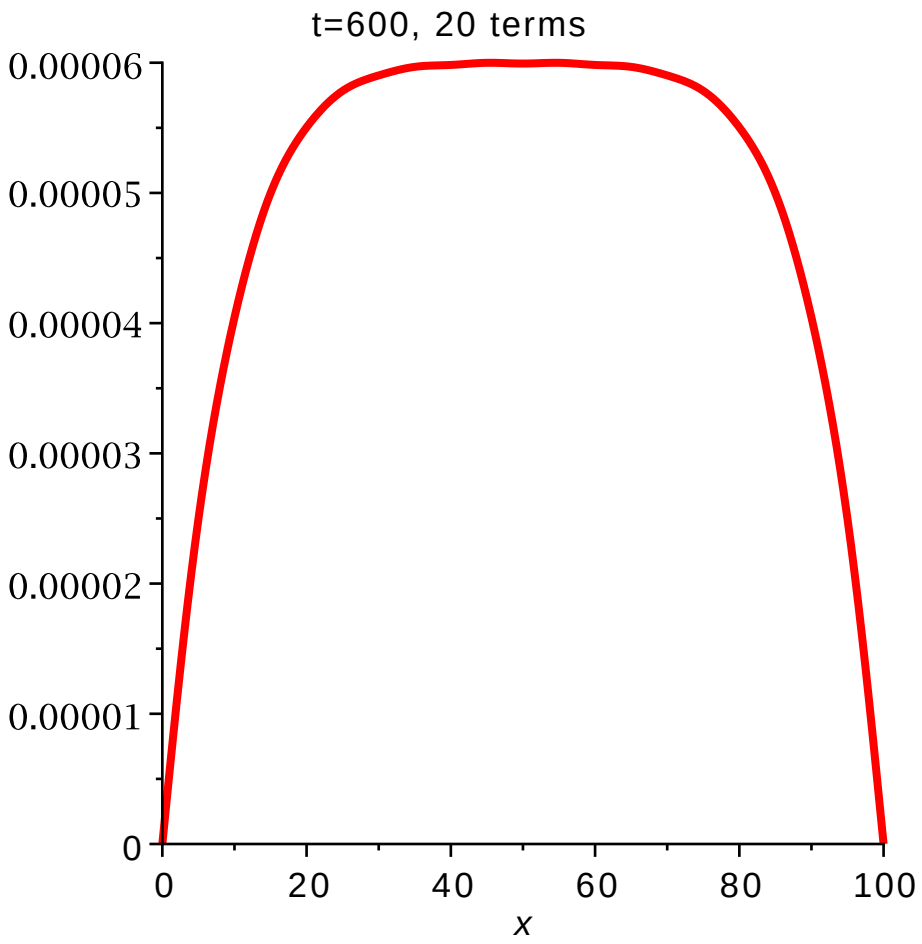
I can easily look at some "snapshots" of the solution. For example, I will show the concentration distribution after 10 minutes (600 seconds). Some trial and error may be necessary to determine how many terms in the Fourier series are required for a qualitatively correct plot. (As discussed in the text, this number decreases as t increases, due to the smoothing of the solution.)

```
> plot(u(x,600,10),x=0..100,title="t=600, 10 terms",
  thickness=3);
```

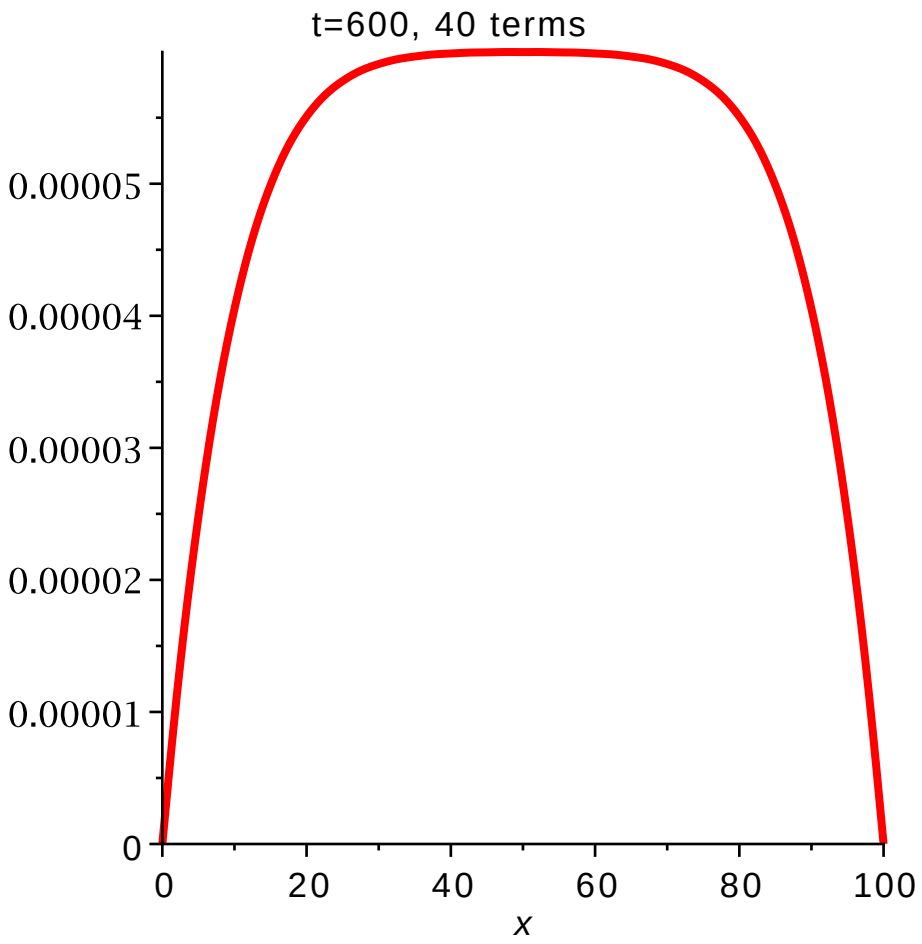


The wiggles at the top of the arch suggest that I did not use enough terms in the Fourier series, so I will try again with more terms. (Of course, perhaps these wiggles are really part of the solution. In that case, they will persist when I draw the graph with more terms.)

```
> plot(u(x,600,20),x=0..100,title="t=600, 20 terms",
  thickness=3);
```




```
> plot(u(x,600,40),x=0..100,title="t=600, 40 terms",
  thickness=3);
```



The above graphs suggest that 20 terms are probably sufficient for a qualitatively correct graph at $t = 600$.

Section 6.4: Finite element methods for the heat equation

Now I will show how to use the backward Euler method with the finite element method to (approximately) solve the heat equation. Since the backward Euler method is implicit, it is necessary to solve an equation at each step. This makes it difficult to write a general-purpose program implementing the backward Euler method, and I will not attempt to do so. Instead, the command **beuler** (defined below) applies the algorithm to the system of ODEs

$$M\frac{da}{dt} + K\,a = f(t),\ t>0,$$
$$a(0) = a_0,$$

which is the result of applying the finite element method to the heat equation.

```
> read(beuler);
```

```
beuler := proc(M, K, f, a0, n, dt)
    local i, L, U, rhs;
    U := Matrix(n + 1, 2, datatype = anything);
    for i from 0 to n do U[1 + i, 1] := i*dt end do;
    U[1, 2] := a0;
    L := M + dt*K;
    for i to n do
        rhs := `.`(M, U[i, 2]) + dt*f(U[1 + i, 1]);
        U[1 + i, 2] := LinearAlgebra:-LinearSolve(L, rhs)
    end do;
    U
end proc
```

(7.2.1)

To solve a specific problem, I have to compute the mass matrix M, the stiffness matrix K, the load vector $f(t)$, and the initial data a_0. The techniques should by now be familiar.

Example 6.9

A 100 cm iron bar, with $\rho = 7.88$ g/cm^3, $c = 0.437$ J/(g K), and $\kappa = 0.836$ W/(cm K), is chilled to an initial temperature of 0 degrees and then heated internally with both ends maintained at 0 degrees. The heat source is described by the function

```
> unassign('x','t');
> F:=(x,t)->10^(-8)*t*x*(100-x)^2;
```

$$F := (x, t) \to \frac{1}{100000000}\, t\, x\, (100 - x)^2 \qquad \textbf{(7.2.1.1)}$$

The temperature distribution is the solution of the IBVP

$$\rho c \frac{\partial u}{\partial t} - \kappa \frac{\partial^2 u}{\partial x^2} = F(x, t),\ 0<x<100,\ t>0,$$
$$u(x, 0) = 0,\ 0<x<100,$$
$$u(0, t) = 0,\ t>0,$$
$$u(100, t) = 0,\ t>0.$$

Using the standard piecewise linear basis functions and the techniques introduced in Section 5.6 of this tutorial, I compute the mass and stiffness matrices below:

```
> n:=100:
> h:=100.0/n:
> k:=0.836:
> p:=7.88:
> c:=0.437:
> K:=Matrix(n-1,n-1,shape=symmetric,storage=band[1,0],
  datatype=float):
> for i from 1 to n-1
> do
>    K[i,i]:=2*k/h:
> end do:
> for i from 1 to n-2
> do
>    K[i+1,i]:=-k/h:
> end do:
```

(Notice that, for this constant coefficient problem, there is no need to perform any integrations; we already know the entries in the stiffness matrix. The same is true for the mass matrix below.)

```
> M:=Matrix(n-1,n-1,shape=symmetric,storage=band[1,0],
  datatype=float):
> for i from 1 to n-1
> do
>    M[i,i]:=2*h*p*c/3:
> end do:
> for i from 1 to n-2
> do
>    M[i+1,i]:=h*p*c/6;
> end do:
```

I must perform some integrations to compute the load vector *F*:

```
> phi1:=(x,i)->(x-(i-1)*h)/h:
> phi2:=(x,i)->-(x-(i+1)*h)/h:
> Vector(n-1,i->int(F(x,t)*phi1(x,i),x=i*h-h..i*h)+int(F(x,
  t)*phi2(x,i),
>                    x=i*h..i*h+h)):
> f:=unapply(%,t):
```

Finally, before invoking **beuler**, I need the initial vector:

```
> a0:=Vector(n-1,i->0.0):
```

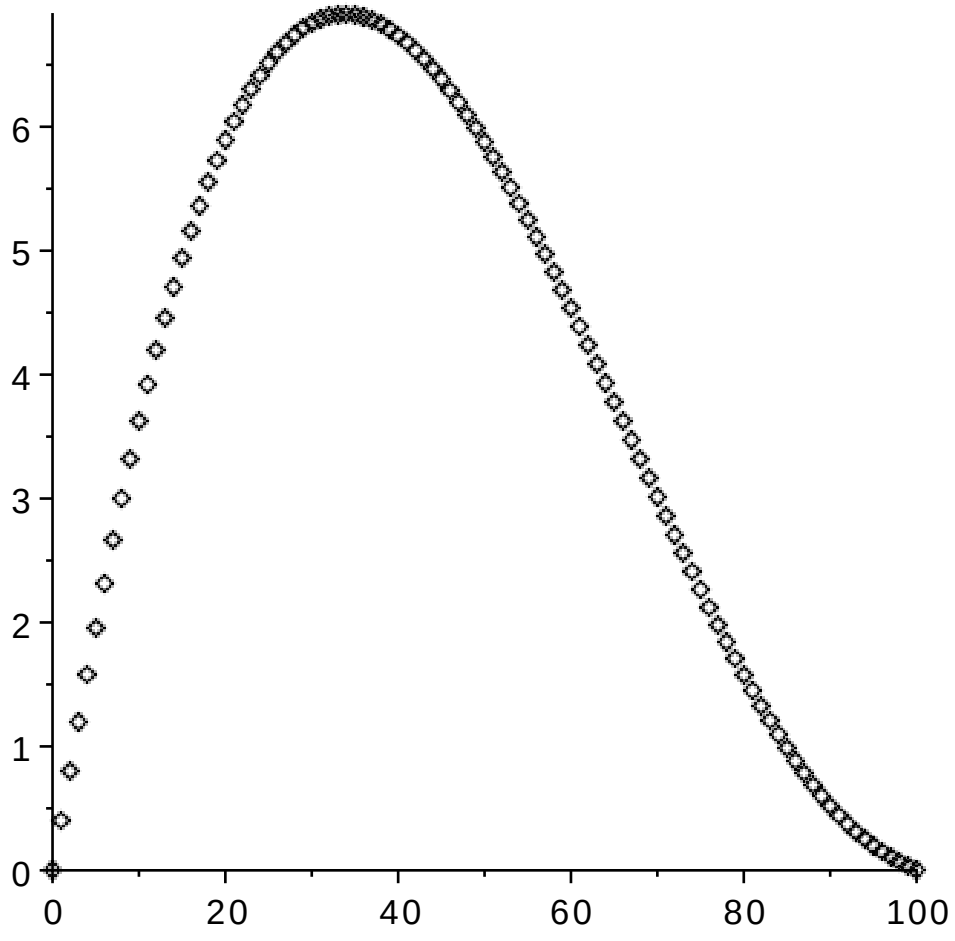
Now I choose the time step and the number of time steps, and invoke the backward Euler method:

```
> steps:=90:
> dt:=180.0/steps:
> U:=beuler(M,K,f,a0,steps,dt);
```

$$U := \begin{bmatrix} 91 \text{ x } 2 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix} \quad \textbf{(7.2.1.2)}$$

I now define the spatial grid and view the last time snapshot:

```
> X:=Vector(n+1,i->(i-1)*h):
> Y:=Vector(n+1,[0.0,U[steps+1,2],0.0]):
> with(plots,pointplot):
> pointplot(<X|Y>,symbolsize=15);
```



Chapter 8: First-order PDEs and the method of characteristics

Section 8.1: The simplest PDE and the method of characteristics

When solving PDEs in two variables, it is sometimes desirable to graph the solution as a function of two variables (that is, as a surface), rather than plotting snapshots of the solution. This is particularly appropriate when neither variable is time.

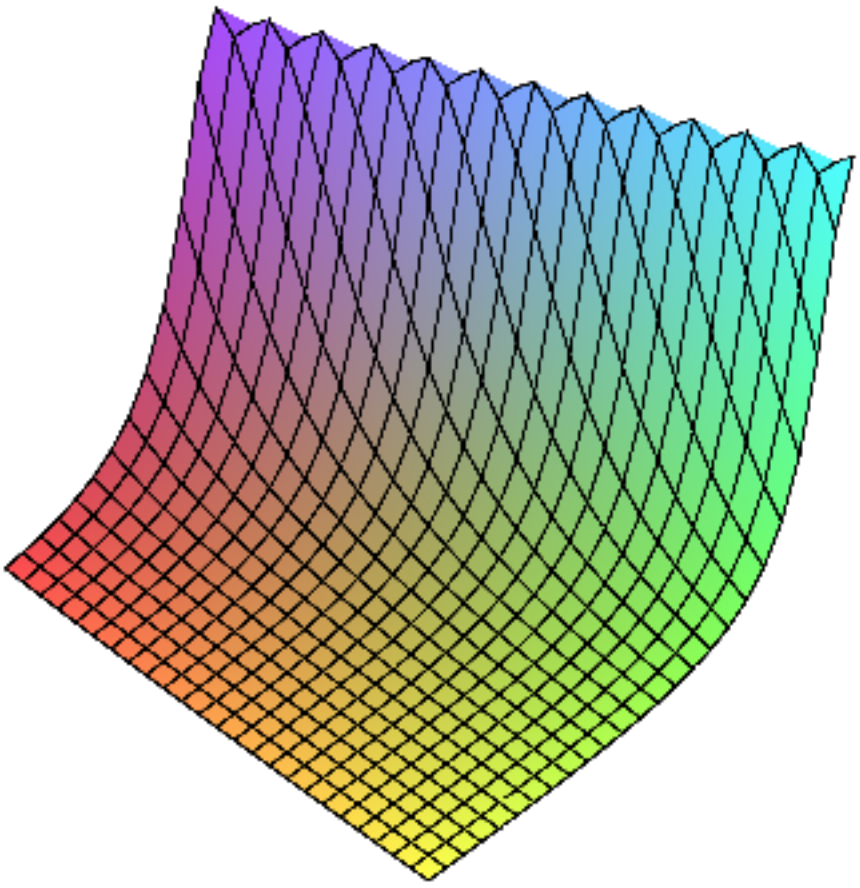
The plot3d function plots a function of two variables:

```
> unassign('u','x','y');
```

```
> u := (x, y) -> 1/(1 + (x + y/2)^2);
```

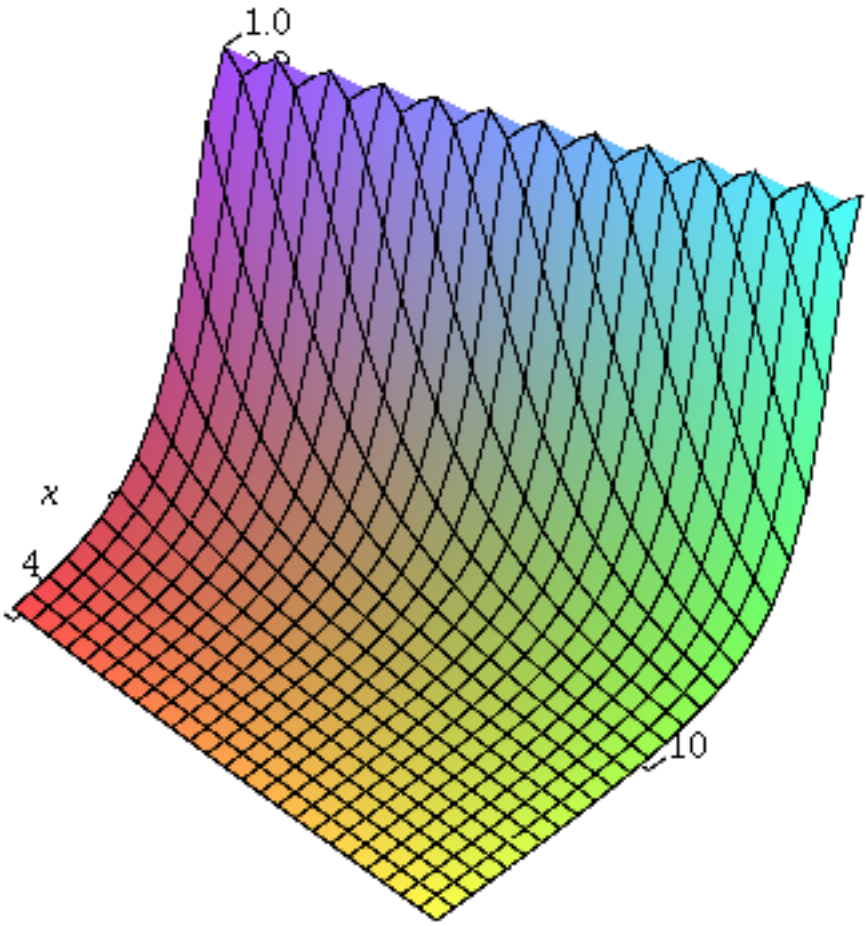
$$u := (x, y) \to \frac{1}{1 + \left(x + \frac{1}{2} y\right)^2} \qquad \textbf{(8.1.1)}$$

```
> plot3d(u(x, y), x = -5..5, y = 0..10);
```



Notice that, by default, the axes are not displayed in a three-dimensional plot. The following option causes *Maple* to display the axes:

```
> plot3d(u(x, y), x = -5..5, y = 0..10, axes = normal);
```



If you would like to see the surface from a different angle, you can click on the figure and rotate it by moving the mouse (i.e. put the pointer on the figure, hold down the left mouse button, and move the mouse).

Section 8.2: First-order quasi-linear PDEs

The purpose of the method of characteristics is to reduce a PDE to a family of ODEs. *Maple* has a function called **dsolve** that will solve many ordinary differential equations symbolically. I will illustrate the use of **dsolve** in the context of Example 8.6 from the text.

The method of characteristics reduces the PDE in Example 8.6 to the IVP

$$\frac{dv}{dt} = v^2,\ v(0) = \frac{1}{1+s^2}.$$

dsolve will solve this problem as follows:

```
> unassign('v','t','s');
```

```
> dsolve({D(v)(t) = v(t)^2, v(0) = 1/(1+s^2)}, v(t));
```

$$v(t) = \frac{1}{1+s^2-t} \qquad \textbf{(8.2.1)}$$

If you prefer, you can specifiy the ODE using **diff** rather than **D**:

```
> dsolve({diff(v(t), t) = v(t)^2, v(0) = 1/(1+s^2)}, v(t));
```

$$v(t) = \frac{1}{1+s^2-t} \qquad \textbf{(8.2.2)}$$

If no initial condition is given, **dsolve** will return the general solution of the ODE:

```
> sol := dsolve(D(v)(t) = v(t)^2, v(t));
```

$$sol := v(t) = \frac{1}{-t+\_C1} \qquad \textbf{(8.2.3)}$$

If you wish to assign the solution to a function, you can use **rhs** to extract the right-hand side (the formula for the solution) and then use **unapply**:

```
> v := unapply(rhs(sol), t);
```

$$v := t \to \frac{1}{-t+\_C1} \qquad \textbf{(8.2.4)}$$

```
> v(1);
```

$$\frac{1}{-1+\_C1} \qquad \textbf{(8.2.5)}$$

```
> _C2 := 4;
```

$$\_C2 := 4 \qquad \textbf{(8.2.6)}$$

```
> v(1);
```

$$\frac{1}{-1+\_C1} \qquad \textbf{(8.2.7)}$$

dsolve will also solve systems of ODEs (when possible). Here is the system from Example 8.7:

$$\frac{dx}{dt} = v,\ x(0) = s,$$
$$\frac{dy}{dt} = y,\ y(0) = 1,$$

$$\frac{dv}{dt} = x,\ v(0) = 2\ s.$$

When calling dsolve, the differential equations and initial conditions must be given as a list, as must the list of unknowns:

```
> unassign('x','y','v','s','t');
> sol := dsolve({D(x)(t) = v(t), D(y)(t) = y(t), D(v)(t) = x(t), x(0) = s, y(0)
      = 1, v(0) = 2·s}, {x(t), y(t), v(t)});
```

$$sol := \left\{v(t) = \frac{3}{2}\ s\,e^{t} + \frac{1}{2}\ s\,e^{-t},\ x(t) = \frac{3}{2}\ s\,e^{t} - \frac{1}{2}\ s\,e^{-t},\ y(t) = e^{t}\right\} \quad \textbf{(8.2.8)}$$

The solutions are given in a list, and can be accessed using indices:

```
> sol[1];
```

$$v(t) = \frac{3}{2}\ s\,e^{t} + \frac{1}{2}\ s\,e^{-t} \quad \textbf{(8.2.9)}$$

```
> v := unapply(rhs(sol[1]), t);
```

$$v := t \to \frac{3}{2}\ s\,e^{t} + \frac{1}{2}\ s\,e^{-t} \quad \textbf{(8.2.10)}$$

```
> x := unapply(rhs(sol[2]), t);
```

$$x := t \to \frac{3}{2}\ s\,e^{t} - \frac{1}{2}\ s\,e^{-t} \quad \textbf{(8.2.11)}$$

```
> y := unapply(rhs(sol[3]), t);
```

$$y := t \to e^{t} \quad \textbf{(8.2.12)}$$

Chapter 11: Problems in multiple spatial dimensions

Section 11.2: Fourier series on a rectangular domain

Fourier series calculations on a rectangular domain proceed in almost the same fashion as in one-dimensional problems. The key difference is that we must compute double integrals and double sums in place of single integrals and single sums. Fortunately, *Maple* makes this easy. We do not need to learn any new commands, since a double integral over a rectangle is just an iterated integral.

As an example, I will compute the Fourier double sine series of the following function f on the unit square:

```
> unassign('x','y');
> f:=(x,y)->x*(1-x)*y*(1-y);
```

$$f := (x, y) \to x\,(1 - x)\,y\,(1 - y) \quad \textbf{(9.1.1)}$$

The Fourier series has the form

$$\sum_{m=1}^{\infty}\left(\sum_{n=1}^{\infty} a_{mn}\sin(m\pi x)\,\sin(n\pi y)\right),$$

where

$$a_{mn}=4\left(\int_0^1\int_0^1 f(x,y)\,\sin(m\pi x)\,\sin(n\pi y)\,dy\,dx\right).$$

So we calculate as follows:

```
> unassign('m','n');
> assume(m,integer):
> assume(n,integer):
> 4*int(int(f(x,y)*sin(m*Pi*x)*sin(n*Pi*y),y=0..1),x=0..1);
```

$$\frac{16\left(1-(-1)^{n\sim}-(-1)^{m\sim}+(-1)^{m\sim+n\sim}\right)}{n\sim^3\,\pi^6\,m\sim^3} \qquad \textbf{(9.1.2)}$$

For convenience, I define a function representing the coefficient a_{mn}:

```
> a:=unapply(%,m,n);
```

$$a:=(m\sim,n\sim)\rightarrow\frac{16\left(1-(-1)^{n\sim}-(-1)^{m\sim}+(-1)^{m\sim+n\sim}\right)}{n\sim^3\,\pi^6\,m\sim^3} \qquad \textbf{(9.1.3)}$$

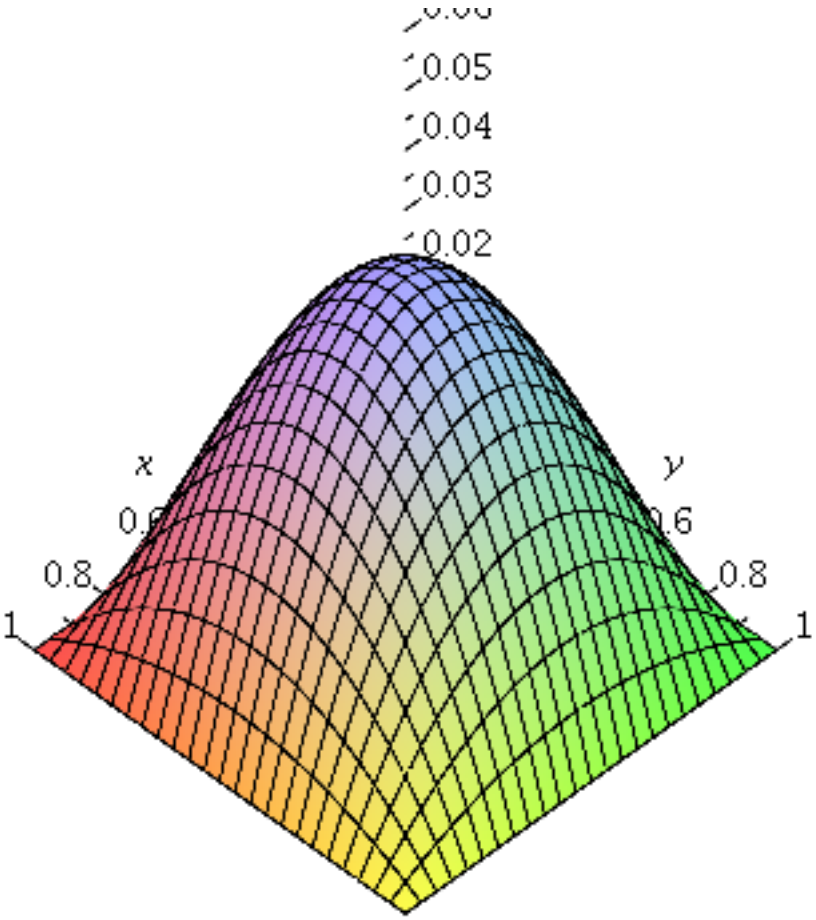
Finally, I define a function representing the (partial) Fourier series:

```
> unassign('M');
> S:=(x,y,M)->add(add(a(m,n)*sin(m*Pi*x)*sin(n*Pi*y),n=1..M),
  m=1..M);
```

$$S:=(x,y,M)\rightarrow add(add(a(m,n)\sin(m\pi x)\sin(n\pi y),n=1..M),m=1..M) \qquad \textbf{(9.1.4)}$$

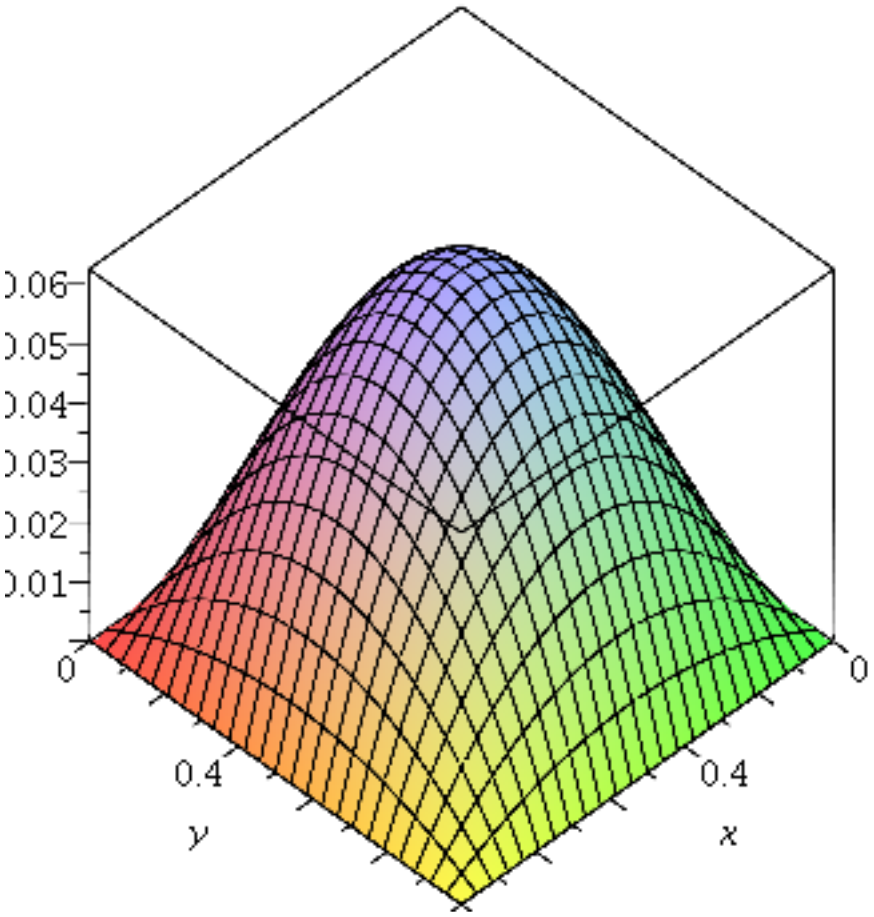
We can now plot *f* using **plot3d**:

```
> plot3d(f(x,y),x=0..1,y=0..1,axes=normal);
```



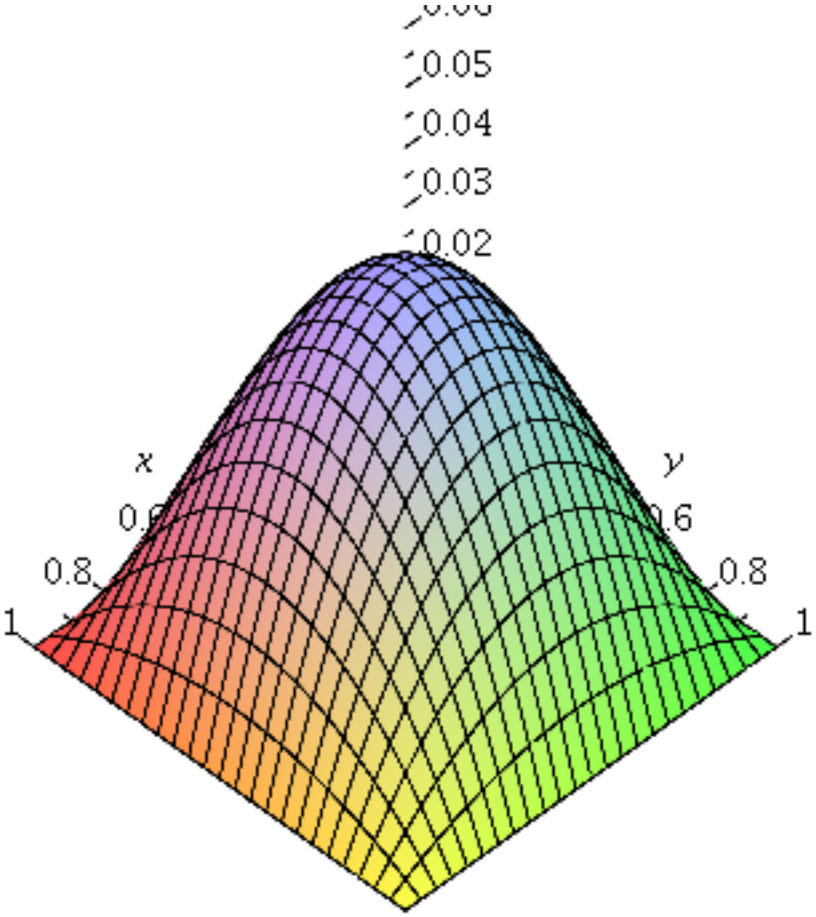
You may prefer "boxed" axes:

```
> plot3d(f(x,y),x=0..1,y=0..1,axes=boxed);
```



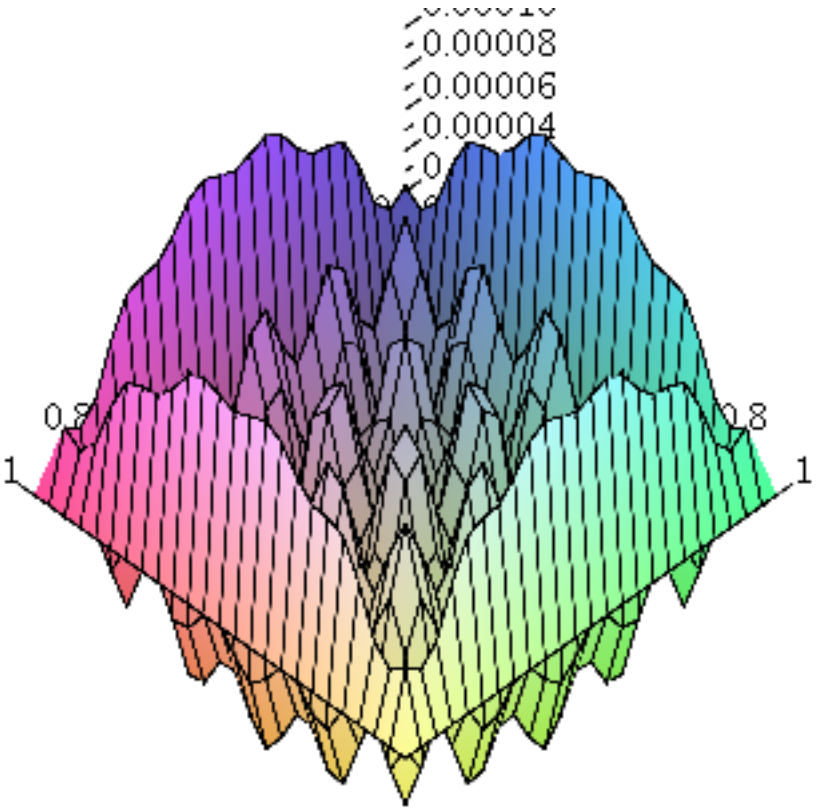
Here is the graph of the Fourier series with a total of 100 terms:

```
> plot3d(S(x,y,10),x=0..1,y=0..1,axes=normal);
```



The approximation looks pretty good. To confirm this, I will graph the error:

```
> plot3d(f(x,y)-S(x,y,10),x=0..1,y=0..1,axes=normal);
```

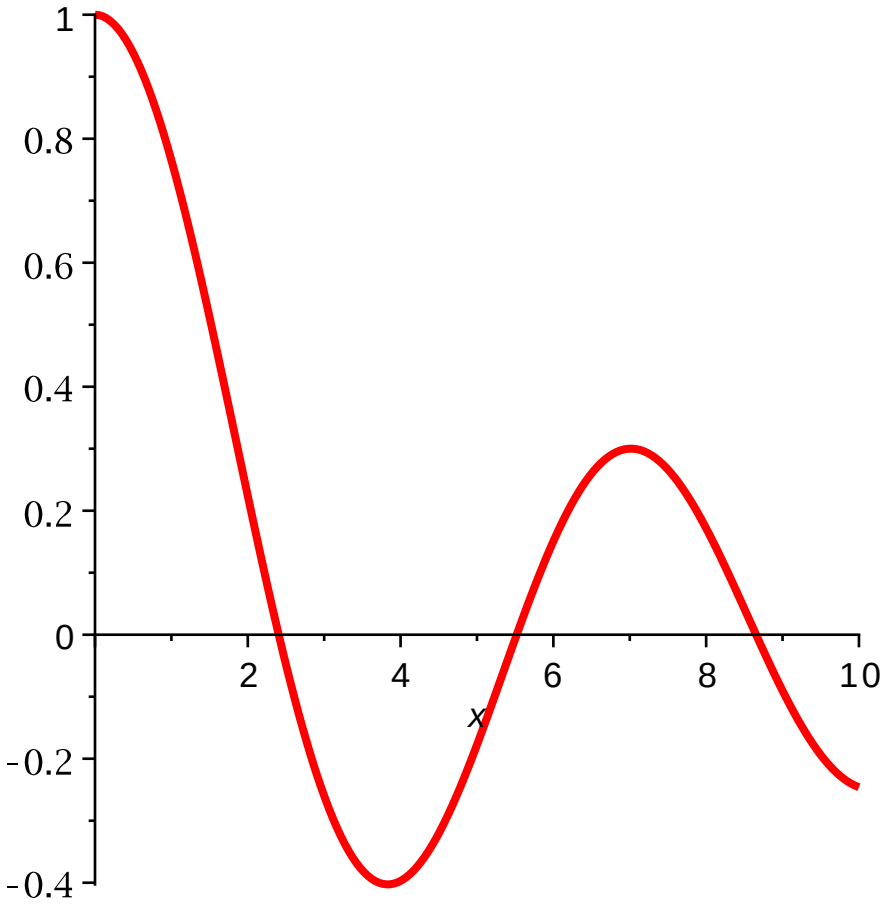


Looking at the vertical axes on the last two plots, we see that the relative difference between f and its partial Fourier series with 100 terms is only about one tenth of one percent.

Section 11.3: Fourier series on a disk

The Bessel functions $J_n(s)$ are built-in functions in *Maple*, just as are the more common elementary functions sine, cosine, exp, and so forth, and can be used just as conveniently. For example, here is the graph of J_0:

```
> plot(BesselJ(0,x),x=0..10,thickness=3);
```



(Notice that the first argument to the **BesselJ** function is the index n.)

As an example, I will compute the smallest roots $s_{0,1}$, $s_{0,2}$, $s_{0,3}$ of J_0. The above graph shows that these roots are approximately 2.5, 5.5, and 8.5, respectively.

```
> fsolve(BesselJ(0,x),x=2..3);
```

2.404825558 **(9.2.1)**

```
> fsolve(BesselJ(0,x),x=5..6);
```

5.520078110 **(9.2.2)**

```
> fsolve(BesselJ(0,x),x=8..9);
```

8.653727913 **(9.2.3)**

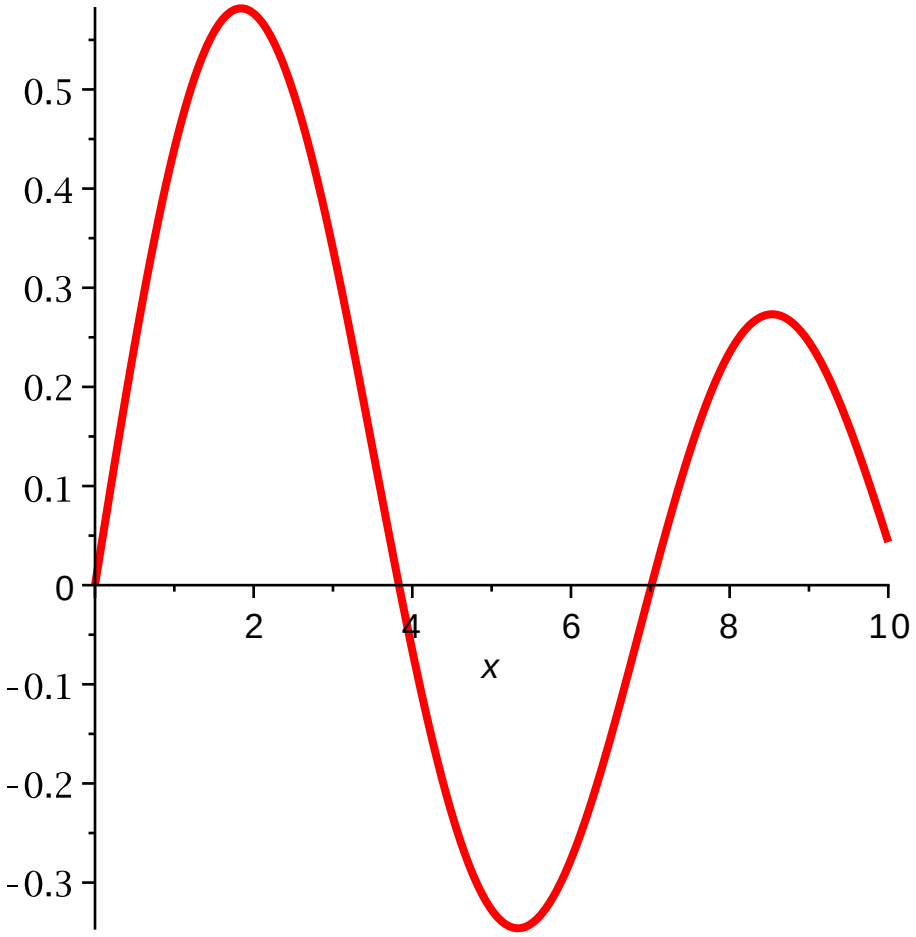
Graphics on the disk

Functions defined on a disk are naturally described by cylindrical coordinates, that is, as $z = f(r, \theta)$, where r, θ are polar coordinates. We can use a version of the **plot3d** command to plot a function given in polar coordinates; we just have to view the surface as given parametrically:

$$(x, y, z) = (r\cos(\theta), r\sin(\theta), f(r, \theta)).$$

For example, consider the eigenfunction $J_1(s_{1,1}\, r)\cos(\theta)$ of the Laplacian on the unit disk. First, I must compute the root $s_{1,1}$ of J_1:

```
> plot(BesselJ(1,x),x=0..10,thickness=3);
```




```
> fsolve(BesselJ(1,x),x=3..4);
```

$$3.831705970 \qquad \textbf{(9.2.1.1)}$$

```
> s11:=%;
```

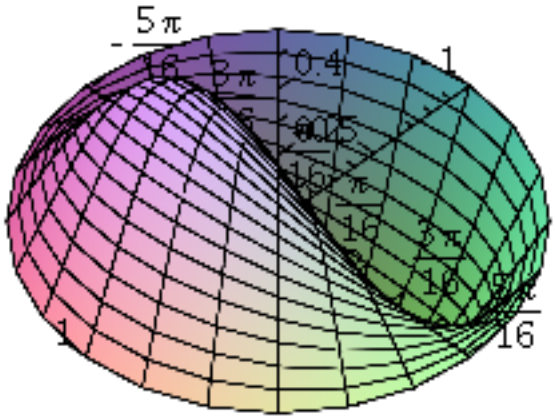
$$s11 := 3.831705970 \qquad \textbf{(9.2.1.2)}$$

```
> unassign('r','t');
> phi:=(r,t)->BesselJ(1,s11*r)*cos(t);
```

$$\phi := (r, t) \rightarrow \mathrm{BesselJ}(1, s11\, r)\cos(t) \qquad \textbf{(9.2.1.3)}$$

Here is a plot of the eigenfunction. Notice that I just give the parametric representation of the surface, followed by the range of the variables.

```
> plot3d([r*cos(t),r*sin(t),phi(r,t)],r=0..1,t=0..2*Pi,axes=
  normal);
```



Chapter 12: More about Fourier series

Section 12.1: The complex Fourier series

It is no more difficult to compute complex Fourier series than the real Fourier series discussed earlier. You should recall that the imaginary unit $\sqrt{-1}$ is represented by I in *Maple*. As an example, I will compute the complex Fourier series of the function $f(x) = x^2$ on the interval $[-1, 1]$.

```
> unassign('x');
> f:=x->x^2;
```

$$f := x \to x^2 \qquad \textbf{(10.1.1)}$$

```
> unassign('n');
> assume(n,integer):
> (1/2)*int(f(x)*exp(-I*Pi*n*x),x=-1..1);
```

(10.1.2)

$$\frac{2\,(-1)^{n\sim}}{\pi^2\,n\sim^2} \tag{10.1.2}$$

```
> simplify(%);
```

$$\frac{2\,(-1)^{n\sim}}{\pi^2\,n\sim^2} \tag{10.1.3}$$

```
> c:=unapply(%,n);
```

$$c := n\sim \to \frac{2\,(-1)^{n\sim}}{\pi^2\,n\sim^2} \tag{10.1.4}$$

The foregoing formula obviously does not hold for $n = 0$, and so I need to compute c_0 separately:

```
> (1/2)*int(f(x),x=-1..1);
```

$$\frac{1}{3} \tag{10.1.5}$$

```
> c0:=1/3;
```

$$c0 := \frac{1}{3} \tag{10.1.6}$$

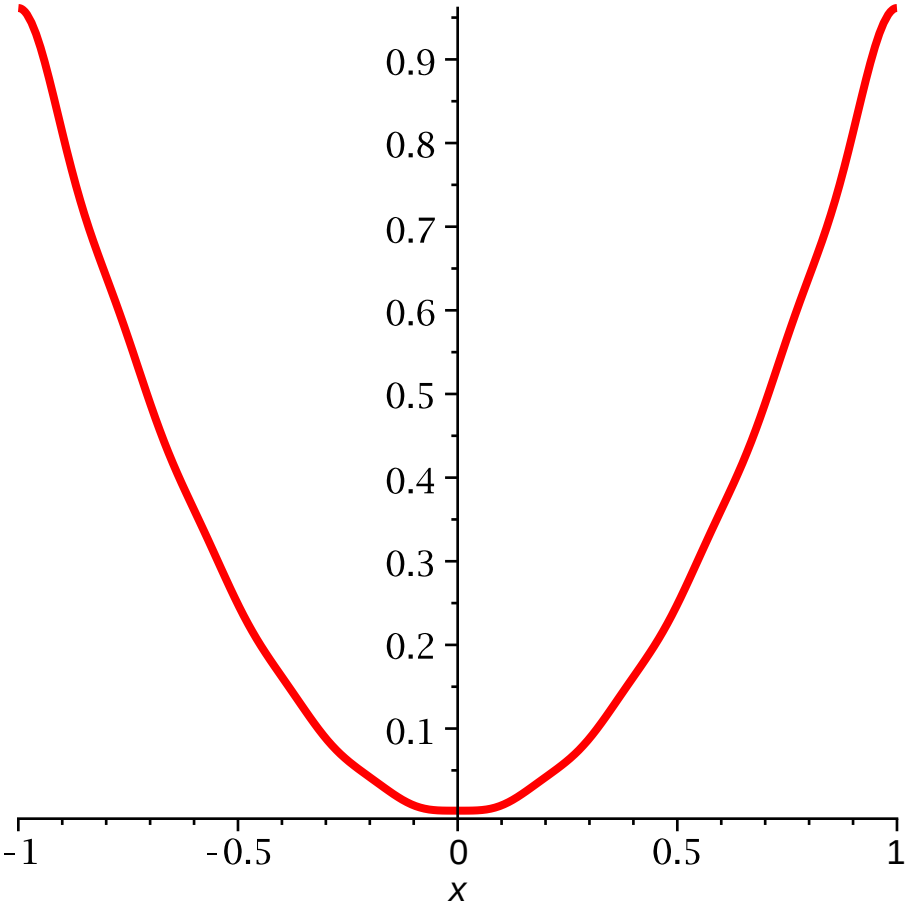
Now I can define the partial Fourier series. Recall from Section 12.1 of the text that, since f is real-valued, the following partial Fourier series is also real-valued:

```
> unassign('x','M');
> S:=(x,M)->c0+add(c(n)*exp(I*n*Pi*x),n=-M..-1)+
>              add(c(n)*exp(I*n*Pi*x),n=1..M);
```

$$S := (x, M) \to c0 + add\left(c(n)\,e^{I n \pi x}, n = -M..-1\right) + add\left(c(n)\,e^{I n \pi x}, n = 1..M\right) \tag{10.1.7}$$

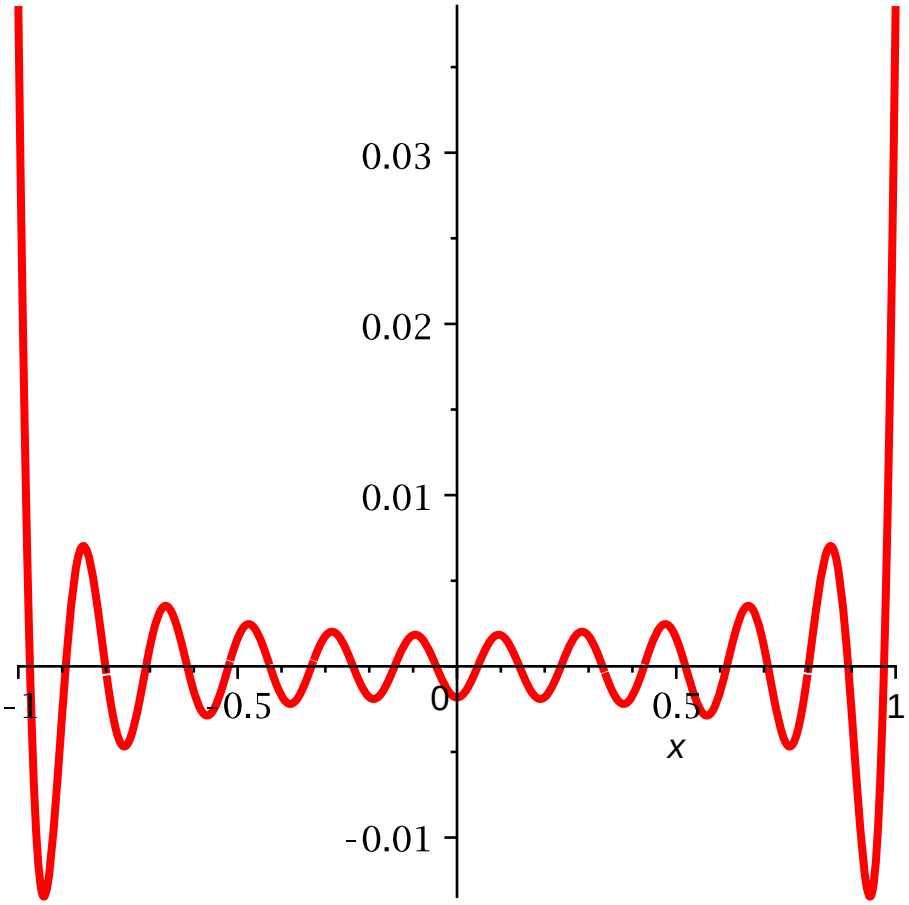
I will check the computation with a plot:

```
> plot(S(x,10),x=-1..1,thickness=3);
```



The approximation is not bad, as the following graph of the error shows:

```
> plot(f(x)-S(x,10),x=-1..1,thickness=3);
```



Section 12.2: Fourier series and the fast Fourier transform

Maple implements the fast Fourier transform in the command **FFT**. As mentioned in Section 12.2.3 in the text, there is more than one way to define the FFT (although all are essentially equivalent), and *Maple*'s definition differs slightly from that used in the text: Instead of associating the factor of $\frac{1}{M}$ with the DFT, *Maple* associates it with the inverse DFT (cf. the formulas for the DFT and inverse DFT in Section 12.2.2 of the text). Otherwise, the definition used by *Maple* is the same as that used in the text.

There are several unusual features to the **FFT** command in *Maple*. First of all, it operates only on sequences of length 2^m, where m is a positive integer. Secondly, it does the computation "in place"; that is, rather than accepting an input and producing an output, **FFT** replaces the input with the output, and just returns the number 2^m. Thirdly, the real and imaginary parts of the input sequence must be given in separate vectors.

Here is an example. Let f be the sequence of length 8, whose terms are 1,2,...,8. I define the real and imaginary parts of f:

```
> fre:=Vector(8,i->i):
> fim:=Vector(8,i->0.0):
```

I must pass the integer m (where the length of the sequence is 2^m), as well as these two arrays, to **FFT**
:

```
> FFT(3,fre,fim);
```

$$8 \qquad \textbf{(10.2.1)}$$

Recall that 8 is just the length of the input sequence, and that the input has now been replaced by the output:

```
> fre;
```

$$\begin{bmatrix} 36 \\ -4.000000002 \\ -4. \\ -3.999999998 \\ -4. \\ -3.999999998 \\ -4. \\ -4.000000002 \end{bmatrix} \qquad \textbf{(10.2.2)}$$

```
> fim;
```

$$\begin{bmatrix} 0. \\ 9.656854244 \\ 4. \\ 1.656854248 \\ 0. \\ -1.656854248 \\ -4. \\ -9.656854244 \end{bmatrix} \qquad \textbf{(10.2.3)}$$

As I explained in Section 12.2.2 of the text, a common operation when using the FFT is to swap the first and second halves of a finite sequence. Here is a command to do this; it takes as input a vector and returns a new vector with the two halves swapped. The **swap** command illustrates some more programming techniques, which you can learn by studying **swap** and using the on-line help.

```
> swap:=proc(x)
```

```
> local n,y;
  if not is(x,Vector) then
     error "Input must be of type Vector"
  end if;
  n:=op(1,x);
> if is(n,odd) then
>    error "Input vector must have even length"
> end if;
  y:=Vector(n,[x[(n/2+1)..n],x[1..(n/2)]]);
> y;
> end proc;
```

$$swap := \mathbf{proc}(x) \qquad \textbf{(10.2.4)}$$

$$\begin{aligned}
&\mathbf{local}\ n, y;\\
&\mathbf{if}\ \mathbf{not}\ is(x, Vector)\ \mathbf{then}\ \mathbf{error}\ \text{"Input must be of type Vector"}\\
&\mathbf{end\ if};\\
&n := op(1, x);\\
&\mathbf{if}\ is(n, odd)\ \mathbf{then}\ \mathbf{error}\ \text{"Input vector must have even length"}\\
&\mathbf{end\ if};\\
&y := Vector(n, [x[1/2*n+1..n], x[1..1/2*n]]);\\
&y\\
&\mathbf{end\ proc}
\end{aligned}$$

```
> x:=Vector(8,i->i):
> swap(x);
```

$$\begin{bmatrix} 5\\6\\7\\8\\1\\2\\3\\4 \end{bmatrix} \qquad \textbf{(10.2.5)}$$

Example

I will now repeat Example 12.3 from the text, except with 8 points instead of 6 (*N*=4 instead of *N*=3, in the notation of that example). First I define the initial sequence:

```
> unassign('x');
> f:=x->x^3;
```

$$f := x \to x^3 \qquad \textbf{(10.2.1.1)}$$

```
> x:=Vector(8,i->-1+(i-1)*2.0/8);
```

$$x := \begin{bmatrix} -1. \\ -0.7500000000 \\ -0.5000000000 \\ -0.2500000000 \\ 0. \\ 0.250000000 \\ 0.500000000 \\ 0.750000000 \end{bmatrix} \qquad \textbf{(10.2.1.2)}$$

```
> F:=Vector(8,i->f(x[i]));
```

$$F := \begin{bmatrix} -1. \\ -0.4218750000 \\ -0.1250000000 \\ -0.01562500000 \\ 0. \\ 0.01562500000 \\ 0.1250000000 \\ 0.4218750000 \end{bmatrix} \qquad \textbf{(10.2.1.3)}$$

```
> F[1]:=0;
```

$$F_1 := 0 \qquad \textbf{(10.2.1.4)}$$

Next, I shift the sequence using swap:

```
> F:=swap(F);
```

(10.2.1.5)

$$F := \begin{bmatrix} 0. \\ 0.01562500000 \\ 0.1250000000 \\ 0.4218750000 \\ 0 \\ -0.4218750000 \\ -0.1250000000 \\ -0.01562500000 \end{bmatrix} \qquad \textbf{(10.2.1.5)}$$

Now I can apply the FFT. However, I need to create the imaginary part of the sequence, and I also need to divide the sequence by its length (since *Maple* uses a different version of the FFT than I use in the text):

```
> F:=F/8:
> Fim:=Vector(8,i->0.0):
> FFT(3,F,Fim):
```

Now I must swap both the real and imaginary parts of the solution:

```
> F:=swap(F):
> Fim:=swap(Fim):
```

Here is the desired result:

```
> G:=F+I*Fim;
```

$$G := \begin{bmatrix} 0. + 0.\,\mathrm{I} \\ -2.18750018099456\,10^{-11} + 0.0460898041687500\,\mathrm{I} \\ 0. - 0.101562500000000\,\mathrm{I} \\ 2.18750018099456\,10^{-11} + 0.108589804131250\,\mathrm{I} \\ 0. + 0.\,\mathrm{I} \\ 2.18750018099456\,10^{-11} - 0.108589804131250\,\mathrm{I} \\ 0. + 0.101562500000000\,\mathrm{I} \\ -2.18750018099456\,10^{-11} - 0.0460898041687500\,\mathrm{I} \end{bmatrix} \qquad \textbf{(10.2.1.6)}$$

Creating a more convenient FFT command

I do not find *Maple*'s **FFT** command to be particularly convenient, for several

reasons:
1. I would rather not have to separate the sequence into real and imaginary parts.
2. I would prefer to use the definition in the text.
3. I prefer to not overwrite the input with the output.
As a further example of *Maple* programming, I will write a procedure that calls FFT, but has the features that I prefer. My new command will be called **myFFT**.

In order to implement **myFFT**, I will need the following capability: Given a vector with complex components, extract the real and imaginary parts. I can easily extract the real and imaginary parts of a complex number, since *Maple* has builtin commands to do this:

```
> Re(1+2*I);
```

$$1 \qquad \textbf{(10.2.2.1)}$$

```
> Im(1+2*I);
```

$$2 \qquad \textbf{(10.2.2.2)}$$

These commands can be applied to vectors:

```
> x:=<1+2*I,I,2-I,-0.5+I>;
```

$$x := \begin{bmatrix} 1+2\,I \\ I \\ 2-I \\ -0.5+1.\,I \end{bmatrix} \qquad \textbf{(10.2.2.3)}$$

```
> Re(x);
```

$$\begin{bmatrix} 1 \\ 0 \\ 2 \\ -0.5 \end{bmatrix} \qquad \textbf{(10.2.2.4)}$$

```
> Im(x);
```

$$\begin{bmatrix} 2 \\ 1 \\ -1 \\ 1. \end{bmatrix} \qquad \textbf{(10.2.2.5)}$$

Another way to accomplish the same thing is to map the commands to the

components of the vector:

```
> map(Re, x);
```

$$\begin{bmatrix} 1 \\ 0 \\ 2 \\ -0.5 \end{bmatrix} \quad \textbf{(10.2.2.6)}$$

```
> map(Im, x);
```

$$\begin{bmatrix} 2 \\ 1 \\ -1 \\ 1. \end{bmatrix} \quad \textbf{(10.2.2.7)}$$

I use the map command in implementing myFFT:

```
> read(myFFT);
myFFT := proc(x)
    local m, n, w, y, z;
    if not is(x, Vector) then
        error "Input must be of type Vector"
    end if;
    n := op(1, x);
    m := simplify(log[2](n));
    if not is(m, integer) then
        error "Input vector must have length n=2^m"
    end if;
    w := x / n;
    y := swap(map(Re, w));
    z := swap(map(Im, w));
    FFT(m, y, z);
    w := swap(y) + I*swap(z)
end proc
swap := proc(x)                                          (10.2.2.8)
    local n, y;
    if not is(x, Vector) then
```

```
        error "Input must be of type Vector"
    end if;
    n := op(1, x);
    if is(n, odd) then
        error "Input vector must have even length"
    end if;
    y := Vector(n, [x[1 / 2 * n + 1 ..n], x[1 ..1 / 2 * n]]);
    y
end proc
```

```
> x:=Vector(8,i->-1+(i-1)*2.0/8):
> f:=x->x^3:
> F:=Vector(8,i->f(x[i])):
> F[1]:=0:
> G:=myFFT(F);
```

$$G := \begin{bmatrix} 0. + 0.\,\mathrm{I} \\ -3.00000000000000\,10^{-11} + 0.0460898041700000\,\mathrm{I} \\ 0. - 0.101562500000000\,\mathrm{I} \\ 3.00000000000000\,10^{-11} + 0.108589804100000\,\mathrm{I} \\ 0. + 0.\,\mathrm{I} \\ 3.00000000000000\,10^{-11} - 0.108589804100000\,\mathrm{I} \\ 0. + 0.101562500000000\,\mathrm{I} \\ -3.00000000000000\,10^{-11} - 0.0460898041700000\,\mathrm{I} \end{bmatrix} \quad \textbf{(10.2.2.9)}$$

As an exercise, you can implement a similar function **myiFFT** as an interface to **iFFT.**

Chapter 13: More about finite element methods

Section 13.1: Implementation of finite element methods

In this section, I do more than just explain *Maple* commands, as has been my policy up to this point. I define a collection of *Maple* commands that implement piecewise linear finite elements on polygonal domains. The interested reader can experiment with and extend these commands to see how the finite element

method works in practice. The implementation of the finite element method follows closely the explanation I give in Section 13.1 of the text, although the data structure has been extended to allow the code to handle inhomogeneous boundary conditions. (However, the code itself has not been extended to handle inhomogeneous boundary conditions. This extension has been left as an exercise.)

The finite element code I provide consists of about a dozen commands, all of which are defined in the file "fempack". Therefore, before using these commands, you must read in the file using the **read** command (recall the earlier discussion of the current directory).

```
> with(LinearAlgebra):
> with(plots):
> read(fempack):
```

I will now describe the finite elements commands and illustrate their use.

The boundary value problem

The fempack finite element codes solves the following problem

$$-div(a(x, y)\ grad\, u) = f(x, y) \text{ in } \Omega,$$
$$u = 0 \text{ on } \Gamma_1,$$
$$\frac{\partial u}{\partial n} = 0 \text{ on } \Gamma_2,$$

where Γ_1 and Γ_2 partition the boundary of Ω. However, before any computation can be performed, the mesh must be described.

The mesh data structure

A triangulation of a planar, polygonal region is described by the following arrays:

T[NodeList]: An Mx2 array of real numbers, the coordinates of the nodes in the mesh.

T[NodePtrs]: An Mx1 array of integers; the ith entry equals the index of the ith node in T[FNodePtrs] if the node is free, and the negative of its index in T[CNodePtrs] if the node is constrained.

T[FNodePtrs]: An NNx1 array of integers, where NN is the number of free nodes. T[FNodePtrs][i] is the index of the ith free node in T[NodePtrs].

T[CNodePtrs]: An Kx1 array of integers, where K is the number of

constrained nodes. T[CNodePtrs][i] is the index of the ith constrained node in T[NodePtrs].

T[ElList]: An Lx3 array of integers, where L is the number of triangular elements. Each row corresponds to one element and contains pointers to the nodes of the triangle in T[NodeList].

T[ElEdgeList]: An Lx3 matrix. Each row contains flags indicating whether each edge of the corresponding element is on the boundary (flag is -1 if the edge is a constrained boundary edge, otherwise it equals the index of the edge in FBndyList) or not (flag is 0). The edges of the triangle are, in order, those joining vertices 1 and 2, 2 and 3, and 3 and 1.

T[FBndyList]: A Bx2 matrix, where B is the number of free boundary edges (i.e. those not constrained by Dirichlet conditions). Each row corresponds to one edge and contains pointers into T[NodeList], yielding the two vertices of the edge.

For more details, see Section 13.1.1 of 'Partial Differential Equations: Analytical and Numerical Methods' by Mark S. Gockenbach

Creating a mesh: rectangleMeshD, rectangleMeshN, and rectangleMeshTopD.

I provide three routines to create meshes:

T:=rectangleMeshD(nx,ny,lx,ly) creates a regular triangulation of the rectangle $[0, lx] \times [0, ly]$, with *nx* and *ny* subdivisions in the x and y directions, respectively. Dirichlet conditions are assumed on the boundary.

rectangleMeshN and rectangleMeshTopD work the same way, but assume Neumann conditions and mixed conditions (Dirichlet on the top edge, Neumann elsewhere), respectively.

Thus I provides the means to deal with a single domain shape, a rectangle, and under only three combinations of boundary conditions: Dirichlet and a certain combination of mixed Dirichlet and Neumann conditions. To use my code to solve BVPs on other domains, or under other boundary conditions, you will have to write code to generate the meshes.

Here I create a mesh:

```
> T:=rectangleMeshD(8,8,1.0,1.0):
```

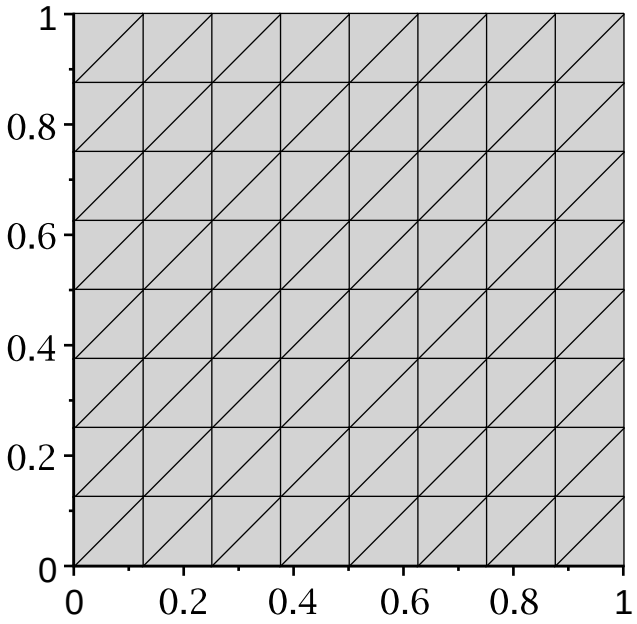
.

I have provided a command for viewing a mesh.

Viewing a mesh: showMesh

showMesh(T) can be used to display a triangular mesh. This command returns a plot structure, which can be displayed using the **display** command.

```
> display(showMesh(T));
```



Assembling the stiffness matrix and the load vector: stiffness and load

K=stiffness(T,a) assembles the stiffness matrix for the BVP

$$-div(a(x, y)\ grad\, u) = f(x, y) \text{ in } \Omega,$$
$$u = 0 \text{ on } \Gamma_1,$$
$$\frac{\partial}{\partial n} u = 0 \text{ on } \Gamma_2,$$

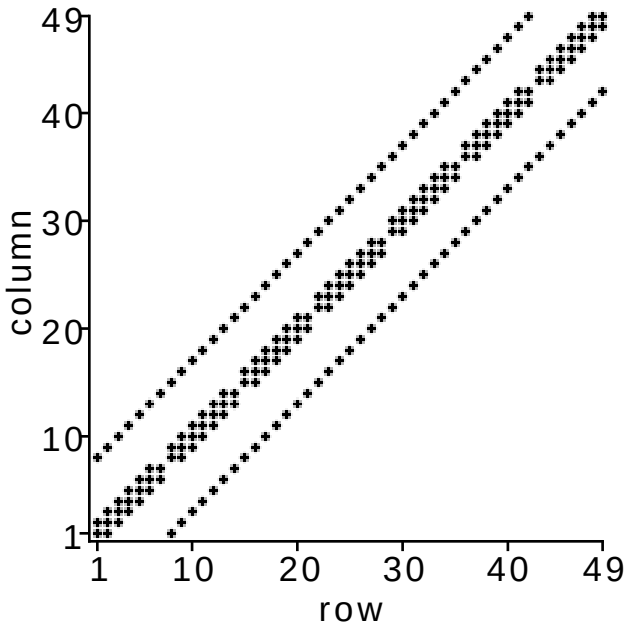
The domain Ω and the boundary conditions are defined by the mesh T, while the input a must be a function of two variables representing the coefficient $a(x, y)$ in the PDE.

Here is an example:

```
> unassign('x','y','a');
> a:=(x,y)->1+x^2:
> K:=stiffness(T,a):
```

Maple provides a method for viewing the sparsity pattern of a sparse matrix like K:

```
> sparsematrixplot(K);
```



The above plot is schematic; there is a box at position i, j if K_{ij} is nonzero.

F:=load(T,f) assembles the load vector for the BVP

$$-div(a(x, y)\ grad\, u) = f(x, y) \text{ in } \Omega,$$
$$u = 0 \text{ on } \Gamma_1,$$
$$\frac{\partial}{\partial n}\, u = 0 \text{ on } \Gamma_2,$$

The domain Ω and the boundary conditions are defined by the mesh T, while the input f must be a function of two variables representing the right hand side $f(x, y)$ in the PDE.

Here is an example:

```
> f:=(x,y)->1:
> F:=load(T,f):
```

Now that I have compute K and F for the problem

$$-div(\,(1 + x^2)\ grad\, u) = 1 \text{ in } \Omega,$$
$$u = 0 \text{ on bndy } \Omega,$$

I can solve for the nodal values of the approximate solution:

```
> u:=LinearSolve(K,F):
```

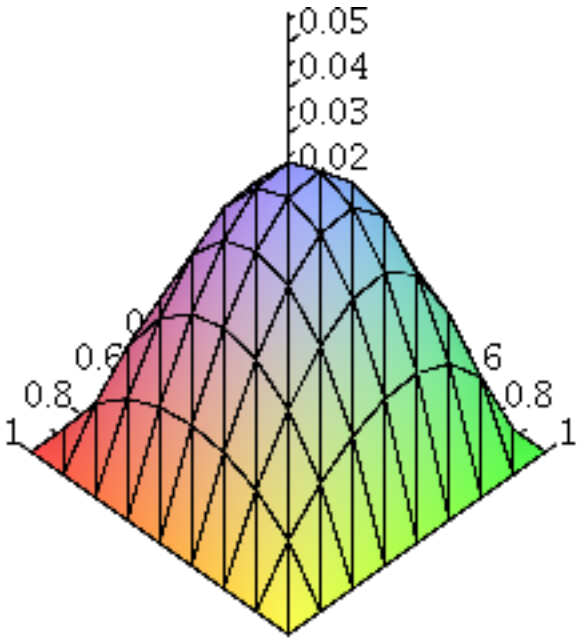
I provide a command for viewing a piecewise linear function.

Displaying a piecewise linear function: showPWLinFcn

s**howPWLinFcn(T,u)** creates a graph of the piecewise linear function defined by the mesh T and the nodal values u. The value at each constrained node is assumed to be zero. Like **showMesh**, **showPWLinFcn** returns the plot structure, which can be displayed using the **display** command.

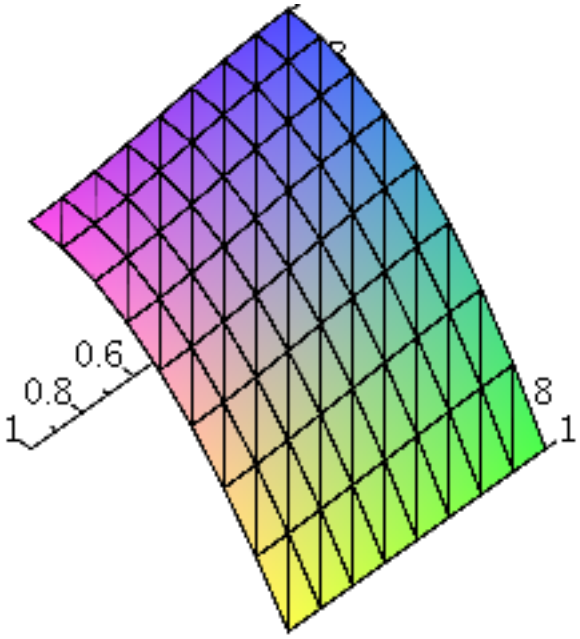
Here is the approximate solution just computed:

```
> display(showPWLinFcn(T,u));
```



To illustrate the effect of the boundary conditions, I will now solve the same PDE, but with the mixed boundary conditions described above:

```
> T:=rectangleMeshTopD(8,8,1.0,1.0):
> K:=stiffness(T,a):
> F:=load(T,f):
> u:=LinearSolve(K,F):
> display(showPWLinFcn(T,u));
```



If the angle at which you view the above surface is not satisfactory, you can select the figure and rotate it using the θ and ϕ angles on the toolbar (at the top of the *Maple* window), or simply by dragging the figure with the mouse.

Testing the code

To see how well the code is working, I can solve a problem whose solution is known, and compare the computed solution with the exact solution. I can easily create a problem whose solution is known; I just choose $a(x, y)$ and any $u(x, y)$ satisfying the boundary conditions, and then compute

$$-div(a(x, y)\ grad\, u(x, y))$$

to get the right hand side $f(x, y)$.

For example, suppose I take

```
> unassign('x','y','a','u');
> a:=(x,y)->1+x^2:
> u:=(x,y)->x*(1-x)*sin(Pi*y):
```

(Notice that u satisfies homogeneous Dirichlet conditions on the unit square.) Then I can compute f:

```
> simplify(-diff(a(x,y)*diff(u(x,y),x),x)-
>          diff(a(x,y)*diff(u(x,y),y),y));
```

$$-\sin(\pi y)\left(2x-6x^2-2-x^3\pi^2+x^4\pi^2-x\pi^2+x^2\pi^2\right) \quad \textbf{(11.1.7.1)}$$

```
> f:=unapply(%,x,y);
```

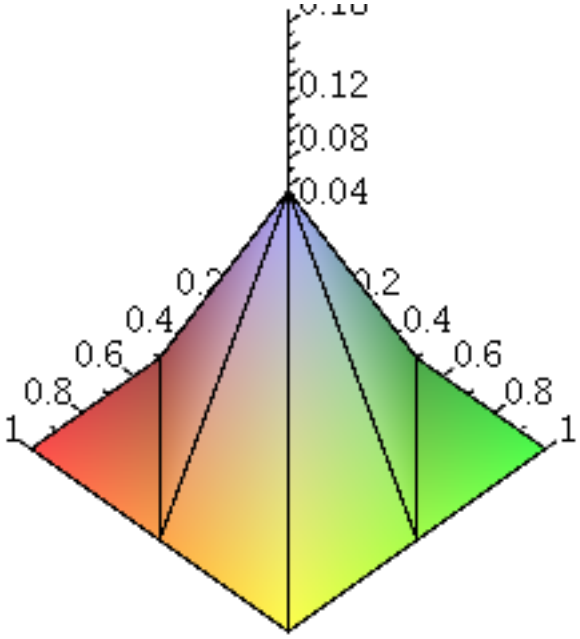
$$f:=(x, y)\rightarrow -\sin(\pi y)\left(2x-6x^2-2-x^3\pi^2+x^4\pi^2-x\pi^2+x^2\pi^2\right) \quad \textbf{(11.1.7.2)}$$

Now I will create a coarse grid and compute the finite element solution:

```
> T:=rectangleMeshD(2,2,1.0,1.0):
> K:=stiffness(T,a):
> F:=load(T,f):
> U:=LinearSolve(K,F):
```

Here is the computed solution:

```
> display(showPWLinFcn(T,U));
```



(On such a coarse grid, there is a single free node!)

I can determine how accurate the compute solution is (at least at the nodes) by comparing to the piecewise linear interpolant of $u(x, y)$. I provide a routine to make this convenient.

Computing a piecewise linear interpolant: nodalValues

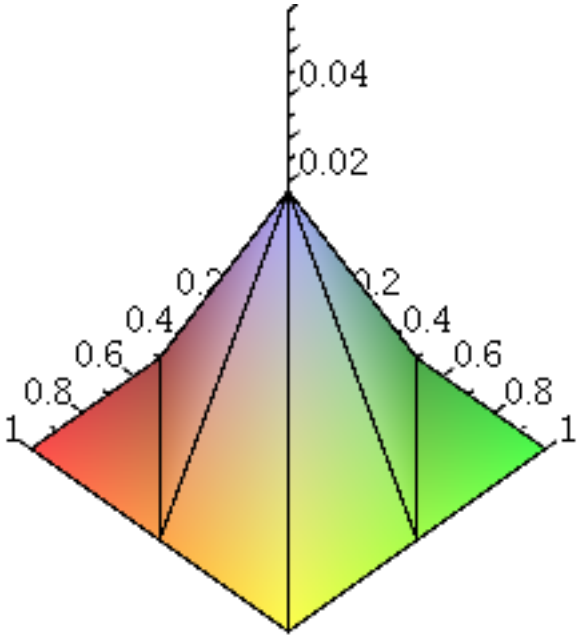
v:=nodalValues(T,u) computes the values of the input function $u(x, y)$ at the free nodes of the mesh *T*, and returns them in the vector *v*.

Here I use **nodalValues** to interpolate the exact solution of the previous example:

```
> V:=nodalValues(T,u):
```

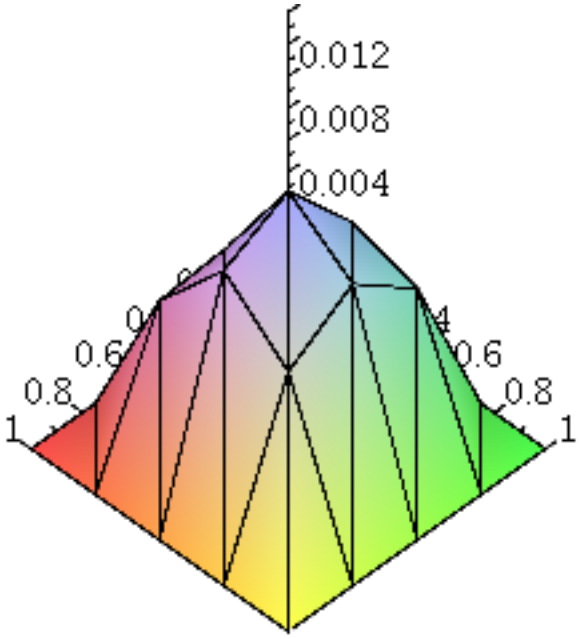
I then plot the difference between the exact and computed solution:

```
> display(showPWLinFcn(T,V-U));
```



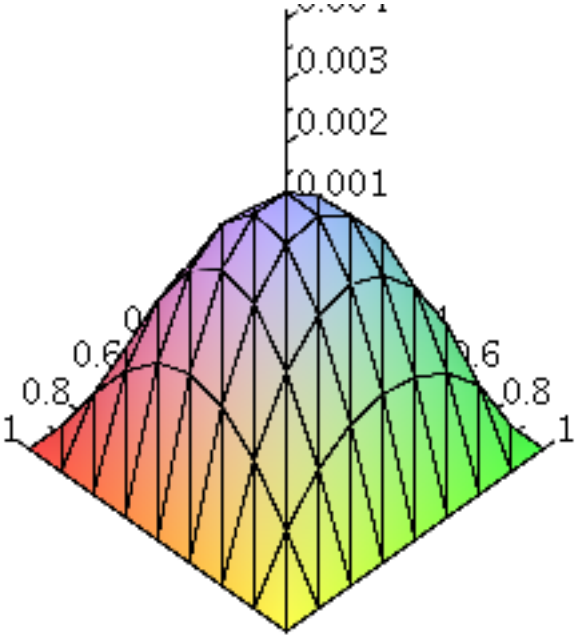
I will now repeat the previous calculations on finer and finer grids.

```
> T:=rectangleMeshD(4,4,1.0,1.0):
> K:=stiffness(T,a):
> F:=load(T,f):
> U:=LinearSolve(K,F):
> V:=nodalValues(T,u):
> display(showPWLinFcn(T,V-U));
```

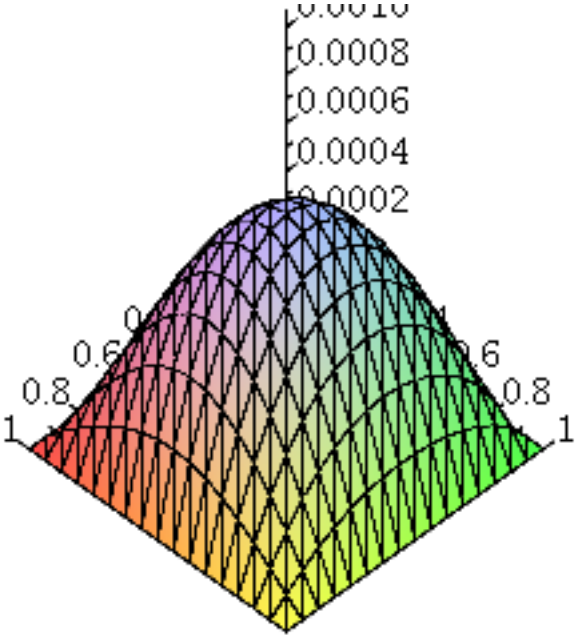



```
> T:=rectangleMeshD(8,8,1.0,1.0):
> K:=stiffness(T,a):
```

```
> F:=load(T,f):
> U:=LinearSolve(K,F):
> V:=nodalValues(T,u):
> display(showPWLinFcn(T,V-U));
```




```
> T:=rectangleMeshD(16,16,1.0,1.0):
> K:=stiffness(T,a):
> F:=load(T,f):
> U:=LinearSolve(K,F):
> V:=nodalValues(T,u):
> display(showPWLinFcn(T,V-U));
```



As these results show, the code produces increasingly accurate solutions as

the mesh is refined.

Using the code

My purpose for providing this code is so that you can see how finite element methods are implemented in practice. To really benefit from the code, you should study it and extend its capabilities. By writing some code yourself, you will learn how such programs are written. Here are some projects you might undertake, more or less in order of difficulty.

1. Write a command called m**ass** that computes the mass matrix. The calling sequence should be simply **M:=mass(T)**, where T is the triangulation.

2. Choose some other geometric shapes and/or combinations of boundary conditions and write mesh generation routines analogous to **rectangleMeshD.**

3. Extend the code to handle inhomogeneous Dirichlet conditions. Recall that such boundary conditions change the load vector, so load must be modified.

4. Extend the code to handle inhomogeneous Neumann conditions. As in the case of inhomogeneous Dirichlet conditions, the load vector is affected.

5. (**Hard**) Write a routine to refine a given mesh, according to the standard method suggested in Exercise 13.1.4 of the textbook.

As mentioned above, the mesh data structure includes the information necessary to solve exercises 3, 4, and 5.

Section 13.2: Solving sparse linear systems

In this section, I merely wish to point out that *Maple* allows you to define and use sparse matrices. In fact, I took advantage of this in the finite element code (specifically, **stiffness** produces a sparse matrix).

Here is the simplest way to define a sparse matrix:

```
> with(LinearAlgebra):
> n:=1000:
> A:=Matrix(n,n,storage=sparse,datatype=float):
```

The matrix A is an n by n zero matrix. I can now add nonzeros as desired. For example, I will make A a tridiagonal, nonsingular matrix:

```
> for i from 1 to n
```

```
> do
>    A[i,i]:=2.0:
> end do:
> for i from 1 to n-2
> do
>    A[i+1,i]:=-1.0:
>    A[i,i+1]:=-1.0:
> end do:
```

For comparison, I will create the same matrix, but using dense storage.

```
> A1:=Matrix(n,n,datatype=float):
> for i from 1 to n
> do
>    A1[i,i]:=2.0:
> end do:
> for i from 1 to n-2
> do
>    A1[i+1,i]:=-1.0:
>    A1[i,i+1]:=-1.0:
> end do:
```

I will now solve the linear system $Ax = b$, using both dense and sparse storage, and compare the time taken using *Maple*'s **time** command.

```
> b:=Vector(n,i->1.0,datatype=float):
> st:=time():
> x:=LinearSolve(A1,b):
> time()-st;
```

0.401 **(11.2.1)**

```
> st:=time():
> x:=LinearSolve(A,b):
> time()-st;
```

0.008 **(11.2.2)**

On my computer, the dense matrix-vector solve takes about one hundred times as long as the sparse matrix-vector solve. (Your results may be different.)

The other issue in using sparse storage is memory. Running on my laptop computer, *Maple* refuses to create a 10000 by 10000 dense matrix, but will allow me to create and use a 20000 by 20000 sparse matrix.

```
> A1 := Matrix(20000, 20000, datatype = float);
Error, (in Matrix) not enough memory to allocate rtable
> A := Matrix(20000, 20000, storage = sparse, datatype = float);
```

$$A := \begin{bmatrix} 20000 \text{ x } 20000 \text{ Matrix} \\ \text{Data Type: float}_8 \\ \text{Storage: sparse} \\ \text{Order: Fortran\_order} \end{bmatrix} \quad \textbf{(11.2.3)}$$