Some remarks on *Mathematica*

Dr. Andreas Stahel

10th May 1999

Contents

Chapter 1

Introduction

Mathematica is programming language designed for mathematical problems. The first version was introduced in 1988. The current version (as of September 1997) is 3.0. Its possible applications span all areas of science, engineering, business and education. You can use *Mathematica* as:

- **visualisation** tool for functions and data
- numerical and **symbolic calculator**
- solver for equations, exact and numerical methods
- **programming language**
- a modelling and data analysis environment

The *Mathematica* system is divided into two parts: the **kernel**, which actually performs computations, and the **front end**, which handles interaction with the user. The kernel works the same on all computers that run *Mathematica*. The front end on the other hand is adapted to the operating system and the graphical user interface. For many type of problems, what is built into the standard version of *Mathematica* will be sufficient. However if you have special needs, you may find that additional **packages** are useful. These packages can be loaded on demand with the command `Needs`. Below you find a listing of the directories with the standard packages.

```
Algebra       Geometry       Miscellaneous  StartUp
Calculus      Graphics       NumberTheory   Statistics
DiscreteMath  LinearAlgebra  NumericalMath  Utilities
```

1.1 Basic facts about *Mathematica* on a UNIX system

When using *Mathematica* typically two programs are running concurrently

- the **frontend**: this is the user interface to be seen on the screen. It is based on **notekbook** files which are typically saved at the end of a session and should contain all the usefull information about the calculations done during the session. The main task of the forntend is to pass information from the user to the kernel and back to the users. It may be called ears, eyes and mouth of *Mathematica*. The filename of a notebook should end with `.nb` .
- the **kernel**: this is the comutational engine of *Mathematica*. All comutations and transformations are done by the kernel and all results are stored in the kernel. The are given to the frontend for display purposes only. The kernel may be called the brain of *Mathematica*.
- it is possible to run frontend and kernel on different computers, but we will not make use of this feature.

1.1.1 Starting the program *Mathematica*

To start *Mathematica* on a typical UNIX system there are different possibilities:

- search for it in the menue of all possible applications on the system.
- type `mathematica &` on a command line.
- *Mathematica* may be used without the help of a notebook but with a very simple command line inteface. We will not use this feature.

Notebooks are hierarchically structured in a manner similar to a table of contents in a book. However, if one of the topics catches your eye, you do not thumb to the page corresponding to that topic; rather, you double click on the cell grouping line that corresponds to that heading. This will open the group and reveal individual cells that contain the text, formulae, and graphics associated with that heading. A closed group of cells is indicated by a hook on the bottom of the cell grouping line. An open grouping will not have a hook. There is a cell containing the title and the cell containing the text. The grouping line is on the far right, and it does not have a hook on the bottom. To close a group, double-click on the grouping line to the right. You should notice how the pointer changes shape when in position to click on the cell bracket. To open a group, double-click again on the hooked line.

1.1.2 Starting and interrupting a command

- by pressing the keys `Shift` and `Enter` you can pass the content of the current cell to the kernel which will try to evaluate the expressions. The cursor can be anywhere in the current cell.
- by marking a section between the double lines at the right edge of the notebook and then pressing `Shit` and `Enter` the complete section will be evaluated.
- If you want to interrupt an ongoing calculation the you can go to the menue `Kernel -> Interrupt Evaluation` or `Kernel -> Abort Evaluation`. There are also key shortcuts for these operations.
- With the menue `Kernel -> Evaluation -> Evaluate Notebook` all coands in the current notebook will be evaluated.
- It can be quite handy to restart the kernel, without leaving the notebook. This will assure the all prevoiusly defined variables, commands and functions are cleared. This can be achieved with the menue `Kernel -> Quit Kernel`.

1.1.3 Error messages and typographical errors

If *Mathematica* is asked to use a command or variable with a name very similar to an already known name, then it will indicate that there might be a typographical error. The warning will only be shonw once an can be ignored, but often the warning is justified. See the example below.

Mathematica

```
Plod[Sin[x],{x,0,10}]
.
General::spell1:
   Possible spelling error: new symbol name "Plod"
     is similar to existing symbol "Plot".
.
Plod[Sin[x], {x, 0, 10}]
```

If desired the warning messages can be turned off

Mathematica

```
Off[General::spell1]
Remove[werta,wertb,wertc]
{werta=1,wertb=3,wertc=2}
.
{1, 3, 2}
```

Mathematica

```
Remove[werta,wertb,wertc]
On[General::spell1]
{werta=1,wertb=3,wertc=2}
.
General::spell1:
   Possible spelling error: new symbol name "wertb"
     is similar to existing symbol "werta".
General::spell:
   Possible spelling error: new symbol name "wertc"
     is similar to existing symbols {werta, wertb}.
.
{1, 3, 2}
```

1.2 A few examples

All builtin symbols of *Mathematica* start with a capital letter. To avoid name conflicts it is a good idea to choose names that start with a non upper–case letter. Then the command `?@` will give a list of all user defined symbols in that session.

1.2.1 Symbolic and exact computations

Mathematica is one of a few packages that is able to symbolic computations. Other such packages are *Maple*, *MuPAD* and *MathCAD*. *Mathematica* will give exact results, if possible. It will not replace exakt expressions by numerical approxmations, unless you tell it to do so.

Mathematica

```
Sin[Pi/4]
.
   1
-------
Sqrt[2]
```

The function `N[]` will comvert exact expressions by numerical approximations. Thre command `N[ expr, n]` computes the numerical approximation of `expr` with n digits precision.

Mathematica

```
{Sin[E],N[Sin[E]]}
.
{Sin[E], 0.410781}
```

As a simple example *Mathematica* can find the first and third derivative of the function $\sin^2(3x)$.

Mathematica

```
D[Sin[3 x]^2,x]
D[Sin[3 x]^2,{x,3}]
.
6 Cos[3 x] Sin[3 x]
-216 Cos[3 x] Sin[3 x]
```

The indefinite integral of a given rational function can be computed

Mathematica

```
Integrate[x^4/(x^2-1),x]
.
     3
    x    Log[1 - x]   Log[1 + x]
x + -- + ---------- - ----------
    3        2            2
```

or the value of a definite integral as an exact value or the numerical approximation with as many digits as desired

Mathematica

```
Integrate[x^4/(x^2-1),{x,2,3}]//Simplify
N[%,30]
.
22   Log[2]   Log[3]   Log[4]
-- + ------ + ------ - ------
3      2        2        2

7.53606588738741552432233989107
```

Mathematica can perform algebraic operations for you

Mathematica

```
result=Expand[(a+b+c)^3]

 3      2          2     2                  2          2        2    3
a  + 3 a  b + 3 a b  + b  + 3 a  c + 6 a b c + 3 b  c + 3 a c  + 3 b c  + c
```

One can also try to find common factors in the above, lengthy expression.

Mathematica

```
Factor[result]
.
           3
(a + b + c)
```

*Mathematica*can also try to decide whether a given condition is satisfied or not.

Mathematica

```
2^3 < 9
Sin[x]^2+Cos[x]^2==1
Simplify[Sin[x]^2+Cos[x]^2==1]
x+1 > x+1+x^2
Simplify[x+1 > x+1+x^2]
(x+1 > x+1+x^2) /. x-> 3
Sin[E]>0
.
True
      2          2
Cos[x]  + Sin[x]  == 1
True
                 2
1 + x > 1 + x + x
                 2
1 + x > 1 + x + x
False
Sin[E] > 0
```

The (maybe) surprising results of the above comparisons ask to be commented.

- $2^3 = 8$ is strictly smaller than 9
- At first *Mathematica* will not apply the trigonometric identity and thus can not decide, whether the condition is satisfied or not. In this case it just returns the original question.
- Now $\cos^2 x + \sin^2 x$ is replaced by 1, then the equality is satisfied.
- Since x might also be a complex expression *Mathematica* can not assume that $x^2 > 0$.
- Simplify does not help in this situation.
- If x is replaced by 3, then the condition is not satisfied.
- The result of `Sin[E]` is the symbolic expression $\sin e$ and not a number. Thus it can not be compared with the number 0. If the numerical value would be used (`N[Sin[E]]`) then the condition is satisfied.

1.2.2 Graphics

The following *Mathematica* commands defines a new function `f` and then the graph of this function is plotted for arguments $-2 \leq x \leq 3\,\pi$.

Mathematica

```
f[x_] := Sin[3 x]
Plot[ f[t], {t,-2,3Pi}];
```

Now we determine the derivative of this function and plot the original function and the derivative in the same figure. Only the values $-2 \leq y \leq 5$ will be shown.

Mathematica

```
df[x_] = D[f[x],x]
Plot[{f[x],df[x]},{x,0,7},PlotRange ->{-2,5}];
.
3 Cos[3 x]
```

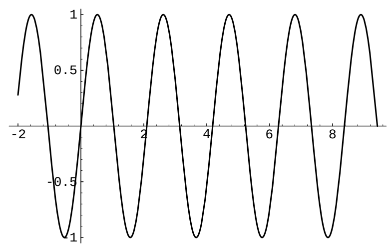


Figure 1.1: Graph of the function $f(x) = \sin(3x)$

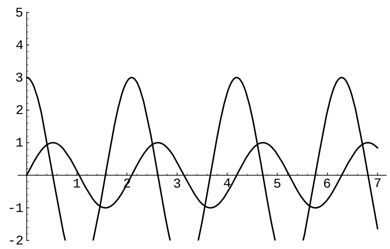


Figure 1.2: Graph of the functions $f(x)$ and $f'(x)$

The same result can also be generated with only one line of code.

Mathematica

```
Plot[Evaluate[{f[x],D[f[x],x]}],{x,0,7},PlotRange ->{-2,5}];
```

Functions of multiple variables can be plotted just as easily. Find the result in figure 1.3.

Mathematica

```
Plot3D[Sin[Sqrt[x^2+2y^2]],{x,-6,6},{y,-6,6}]
```

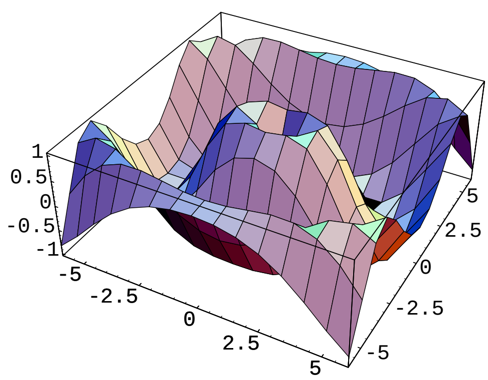


Figure 1.3: A function depending on two variables

Mathematica can also generate bar charts. Find the result in figure 1.4.

Mathematica

```
Needs["Graphics`Graphics`"]
g4=BarChart[{1,3,-2, 6,2}];
```

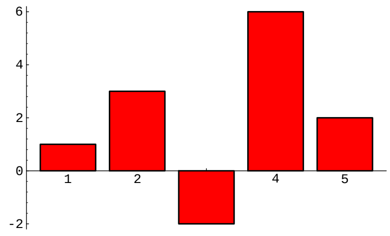


Figure 1.4: A simple bar chart

1.2.3 Solving of equations

With *Mathematica* many equations can be solved easily. The solutions for the quadratic equation

$$a\,x^2 + b\,x + c = 0$$

can be produced by

Mathematica

```
Solve[ a x^2 + b x + c==0 ,x]
.
                  2
       -b - Sqrt[b  - 4 a c]
{{x -> ---------------------},
                2 a

                  2
       -b + Sqrt[b  - 4 a c]
  {x -> ---------------------}}
                2 a
```

This simple, well know formula is certainly not worth the effort to learn more about *Mathematica*. But try to find exact solutions to the following equation.

Mathematica

```
Solve[ x^4+x^3+x^2+x+1 ==0,x]
.
                              5    Sqrt[5]
                       Sqrt[-(-) - -------]
         1    Sqrt[5]         2        2
{{x -> -(-) + ------- - --------------------},
         4       4                2

                              5    Sqrt[5]
                       Sqrt[-(-) - -------]
         1    Sqrt[5]         2        2
  {x -> -(-) + ------- + --------------------},
         4       4                2

                              5    Sqrt[5]
                       Sqrt[-(-) + -------]
         1    Sqrt[5]         2        2
  {x -> -(-) - ------- - --------------------},
         4       4                2

                              5    Sqrt[5]
                       Sqrt[-(-) + -------]
```

```
          1    Sqrt[5]          2       2
  {x -> -(-) - ------- + -------------------}}
          4       4              2
```

If one is searching for numerical approximations instead of **exact** solutions just use

Mathematica

```
sol = NSolve[ x^4+x^3+x^2+x+1 ==0,x]
.
{{x -> -0.809017 - 0.587785 I},
  {x -> -0.809017 + 0.587785 I},
  {x -> 0.309017 - 0.951057 I},
  {x -> 0.309017 + 0.951057 I}}
```

The result does not consist of the four solution, but of four replacement rules. From those we can get to the numbers by applying the rules to the expression x.

Mathematica

```
x /. sol
.
{-0.809017 - 0.587785 I, -0.809017 + 0.587785 I,
  0.309017 - 0.951057 I,  0.309017 + 0.951057 I}
```

One can find approximate solutions to equations where it is impossible to give exact solutions. Search for solutions of the transcendental equation $\tan x = 2x$. We are looking for one solution close to $x \approx 1.0$ and another solution at $x \approx 4.0$. See also figure 1.5.

Mathematica

```
FindRoot[2 x == Tan[x],{x,1.0}]
FindRoot[2 x == Tan[x],{x,4.0}]
.
{x -> 1.16556}
{x -> 4.60422}
```

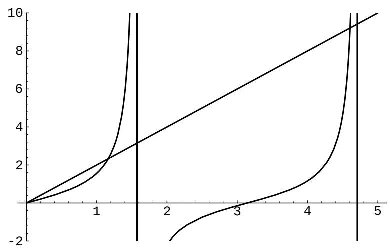


Figure 1.5: Solutions of the equation $2\,x - \tan x$

Mathematica is also capable of solving **differential equations**.

Mathematica

```
DSolve[ {y''[x]- 2 y'[x] + y[x] == 1 ,y[0]==0},y[x],x]
.
Solve::svars:
   Warning: Equations may not give solutions for all
```

```
"solve" variables.

               x     x
{{y[x] -> 1 - E  + E  x C[2]}}
```

1.2.4 The command Random[] and a simple application

Mathematica has many builtin functions as an example we look at the function `Random[]` The function `Random[]` can be used to generate random numbers.

Mathematica

```
Random[]
.
0.644135
```

Here is the on–line documentation (*Mathematica*-2.2) of this command.

Mathematica

```
?Random
.
Random[ ] gives a uniformly distributed pseudorandom Real in the range 0 to 1.
   Random[type, range] gives a pseudorandom number of the specified type,
   lying in the specified range. Possible types are: Integer, Real and Complex.
   The default range is 0 to 1. You can give the range {min, max} explicitly;
   a range specification of max is equivalent to {0, max}.
```

Here are two illustrative examples

Mathematica

```
Table[Random[Integer,{1,6}],{dummy,10}]
.
{4, 3, 1, 4, 1, 5, 2, 1, 2, 5}
```

Mathematica

```
Table[Random[Real,{1,6}],{dummy,10}]
.
{3.68619, 5.99674, 2.91196, 4.96586, 3.50209, 5.71108, 3.84267, 5.12475,  5.16159, 5.01069}
```

Now we will use this command to create a simple simulation. As result we should obtain an approximate value of π.

Mathematica

```
(* number of points to be sampled *)
n=1000;
(* generate a table of n uniformly distributed complex numbers
   in the unit square square     *)
tries=Table[ Random[Complex,{0,1+I}],  {counter,n}];

(* define the criterion to pick all numbers with
   absolut value smaller than 1 *)
criterion[z_] = (Abs[z]<1);
(* find the fraction of points inside the unit circle *)
frac=Length[Select[tries,criterion]]/n;
```

```
N[frac*4]
.
3.136
```

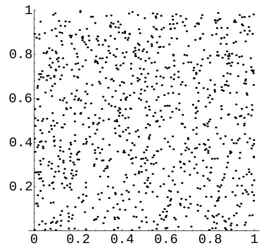


Figure 1.6: Uniformly distributed points on the unit square

The figure 1.6 show the result of this simulation for 100 points. It was generated with the code below.

Mathematica

```
Show[Graphics[
Map[Point,Transpose[{Re[tries],Im[tries]}]]],
 PlotRange -> {0,1},
 AspectRatio-> Automatic,
 Axes -> True];
```

1.3 Getting help

The most recent version of *Mathematica* has a extensive on-line documentation. With the question mark `?` one can list a short description of a known command.

Mathematica

```
?Solve
.
Solve[eqns, vars] attempts to solve an equation or set of equations for
   the variables vars. Any variable in eqns but not vars is regarded as a
   parameter. Solve[eqns] treats all variables encountered as vars
   above. Solve[eqns, vars, elims] attempts to solve the equations for
   vars, eliminating the variables elims.
```

One could even get more information by `??Solve`. Combined with other special characters you can list all symbols with a given combination of letters. To see all symbols with the combination `Solve` try the following.

Mathematica

```
?*Solve*
.
DSolve          MainSolve       Solve
DSolveConstants NDSolve         SolveAlways
LinearSolve     NSolve
```

Within the standard **Help Browser** you have access to the *Mathematica* book. In section 1.5.7 you find an introduction the the command `Solve` with a few examples. Further information is given in section 3.4 .

Often one wants to know about already defined symbols. If the names of all your own symbols start with small letters, there is a simple command to achieve this.

Mathematica

```
?@
```

1.4 A few general remarks

1.4.1 `Clear[]` and `Remove[]`

Mathematica does store copies of all previously defined variables. When defining new variables it is a good idea to first remove or clear previous definitions. This is done with the commands `Clear[]`, `ClearAll[]` and `Remove[]`.

- `Clear[]` clears values and definitions of the given symbol. It does not clear the attributes, messages and defaults.
- `ClearAll[]` clears values, definitions, attributes, messages and defaults of the given symbol.
- `Remove[]` removes a given symbol completely. The names will no longer be recognised by *Mathematica*. This is a more radical solution than `Clear[]` .
- All of the above commands can work with patterns. `Clear["my*"]` will clear all symbols that start with the letters `my` . `Remove["Global‘my*"]` will remove all symbols in the context `Global` that start with the two letters `my`.
- A **Context** is a construction used in packages to provide local name spaces for variables. Most user variables life in the default context `Global`. Most of the *Mathematica* packages create their own context, to avoid name conflicts.

While testing *Mathematica* code it is an **excellent idea** to clear or remove symbols with the same name before you try to apply new definitions. This will prevent many problems do to left over, old values of symbols.

The following sequence of operations will illustrate the difference between `Clear[]` and `Remove[]`.

Mathematica

```
f[x_] = Exp[x]
?f
.
 x
E
Global‘f

f[x_] = E^x
```

Mathematica

```
Clear[f]
?f
.
Global`f
```

Mathematica

```
Remove[f]
?f
.
Information::notfound:
   Symbol f not found.
```

The code below first creates the symbol `aa` with the value 3, then the different symbol `aa` with values 5 in the new context `mycontext`. Then they are multiplied. The first symbol is actually in the context `Global`.

Mathematica

```
mycontext`aa=3
aa=5
mycontext`aa * aa
.
3
5
15
```

A frequent error is to use a command in a package, without first loading the package. The result will most often be the input.

Mathematica

```
BarChart[{1,3,-2,6,2}]
.
BarChart[{1, 3, -2, 6, 2}]
```

To correct the problem, the package has to be loaded. But there is still no graph.

Mathematica

```
Needs["Graphics`Graphics`"]
BarChart[{1,3,-2,6,2}]
BarChart::shdw:
   Warning: Symbol BarChart  appears in multiple contexts {Graphics`Graphics`, <<1>>};
   definitions in context Graphics`Graphics` may shadow or be shadowed by other definitions.
.
BarChart[{1, 3, -2, 6, 2}]
```

The warning message gives you information about a possible problem. The reason for this should be clear after the following.

Mathematica

```
?BarChart
.
Global`BarChart
```

Mathematica

```
?Graphics`Graphics`BarChart
.
BarChart[list1, list2, ...] generates a bar chart of the data in the lists.
```

First the symbol `BarChart` in the context `Global`` has to be removed. Only now the *Mathematica* will see the graphics command `BarChart`. Thus the following code will finally produce the desired graphic.

Mathematica

```
Remove[BarChart]
BarChart[{1,3,-2,6,2}]
```

1.4.2 Parenthesis, brackets and all that

In *Mathematica* different types of brackets are used. Many problems are due to badly chosen types of brackets.

Symbol	English	Deutsch	Français	Usage
(...)	parenthesis	runde Klammer	parenthèse	gouping
[...]	square bracket	eckige Klammer	crochet	arguments of functions
{...}	brace	geschweifte Kl.	accolade	lists
[[...]]	double bracket	??	double crochet	access to parts of lists
(*...*)				remarks

Below you find a few examples.

Mathematica

```
2^3^2
.
512
```

Mathematica

```
2^(3^2)
.
512
```

Mathematica

```
(2^3)^2
.
64
```

1.4.3 Definition and evaluation of functions

Arguments of functions have to be indicated by square brackets.

Mathematica

```
Sin[Pi]
.
0
```

Mathematica

```
Plot[Sin[2x],{x,0,1.2}]
```

A difference has to be made between definition of functions and evaluation of functions. In the definition an the arguments in the square brackets have to be followed by underscores, e.g. $[x\_]$ instead of $[x]$ only. This is the first, and simplest case of **pattern matching**.

Mathematica

```
(* Definition of a new function *)
f[x_] = x^2 Sin[2x]
(* Evaluation of this function for two arguments *)
{f[a],f[Pi]}
.
 2
x  Sin[2 x]

  2
{a  Sin[2 a], 0}
```

The above point is the source of many problems, as *Mathematica* will not give any warning about missing underscores, but not deliver the expected result either. This problem is due to the enormous flexibility of *Mathematica*, thus it can not do an type checking. The following example should illustrate this behaviour.

Mathematica

```
fbad[x] = x^3 Cos[x];
{fbad[x],fbad[a]}
.
  3
{x  Cos[x], fbad[a]}
```

In the first evaluation the argument is exactly x, thus the above definition does apply. But the second argument is not called x, thus the function is not defined and *Mathematica* simply returns the name of the function with the given argument. This call leads to unexpected results.

Mathematica

```
Plot[fbad[x],{x,0,3}]
.
Plot::plnr: CompiledFunction[{x}, <<1>>, -CompiledCode-][x]
     is not a machine-size real number at x = 0.
Plot::plnr: CompiledFunction[{x}, <<1>>, -CompiledCode-][x]
     is not a machine-size real number at x = 0.125.
Plot::plnr: CompiledFunction[{x}, <<1>>, -CompiledCode-][x]
     is not a machine-size real number at x = 0.25.
General::stop:
   Further output of Plot::plnr
     will be suppressed during this calculation.
```

Mathematica will do the following

1. create a new variable, called x
2. let `x=0`
3. try to compute `fbad[0]`
4. let `x=0.125`
5. try to compute `fbad[0.125]`
6. proceed up to $x = 3$
7. now try to draw the graph through the above points.

In this example x will be set to 0 and only the the command `Plot` tries to evaluate the function. Since `f[0]` is not defined it does not return a number and `Plot` will complain about the wrong type of result. This could be fixed by first evaluating the expression `fbad[x]` (leads to $x^3 Cos[x]$) and only then start to plot.

Mathematica

```
Plot[Evaluate[fbad[x]],{x,0,3}]
```

Mathematica will do the following

1. evaluate the expression `fbad[x]`
2. create a new variable, called x
3. let `x=0`
4. try to compute the above expression for $x = 0$
5. let `x=0.125`
6. try to compute the above expression for $x = 0.125$
7. proceed up to $x = 3$
8. now try to draw the graph trough the above points.

This will lead to the desired result. But the minor modification

Mathematica

```
Plot[Evaluate[fbad[a]],{a,0,3}]
```

will fail again. All of the above problems can be avoided if the definition of the function is done with the underscore `fbad[x_]`.

The above method also applies to functions of more than one variable, e.g.

Mathematica

```
profit[price_,CostOfMatrerial_,costOfLabour_] = price-CostOfMatrerial-costOfLabour
```

In this section we only introduced **immediate evaluation** of definitions of functions. **Delayed evaluation** and its consequences will be discussed later.

1.4.4 Creation and access to lists, vectors, matrices

A **list** is a fundamental object in *Mathematica*. Must of the data structures in *Mathematica* are bases on lists, or on nested lists. A list is surrounded by braces and the different entries in the list are separated by commas. An entry in a list can be a list itself. The data types of entries in a list do not have to be the same. This flexibility is a strong point of *Mathematica*. Her are a few simple examples.

Mathematica

```
mylist={2,3, Sin[Pi/2]}
newlist= {hi, there, {"what's up", doc}}
.
{2, 3, 1}
{hi, there, {"what's up", doc}}
```

With double brackets `[[ ]]` we can get to individual entries in a list. For nested lists the different components are separated by commas.

Mathematica

```
mylist[[2]]
{newlist[[2]],newlist[[3,2]]}
.
3
{there, doc}
```

The command `FullForm` does show the internal representation of a data object.

Mathematica

```
FullForm[newlist]
.
list[hi, there, list["what's up", doc]]
```

This does apply to other types in *Mathematica* too, as the following examples show.

Mathematica

```
FullForm[{a,{b,c},d}]
FullForm[a+b*c]
FullForm[Sqrt[x^2]]
FullForm[Sin[-x]]
.
List[a, List[b, c], d]
Plus[a, Times[b, c]]
Power[Power[x, 2], Rational[1, 2]]
Times[-1, Sin[x]]
```

In the last example one should observe that *Mathematica* replaced $\sin(-x)$ by the equivalent expression $-\sin x$. This happens frequently, e.g.

Mathematica

```
FullForm[x+y+z]
FullForm[z+y+x]
.
Plus[x, y, z]
Plus[x, y, z]
```

The command `Table[]` will generate lists in *Mathematica*. The effect of this command is easy to grasp with the help of a few examples.

Mathematica

```
Table[n^2,{n,4}]
Table[Sin[x],{x,-2,5}]
Table[1/k,{k,1/2,4,1/2}]
.
{1, 4, 9, 16}

{-Sin[2], -Sin[1], 0, Sin[1], Sin[2], Sin[3], Sin[4], Sin[5]}

       2  1  2  1  2  1
{2, 1, -, -, -, -, -, -}
       3  2  5  3  7  4
```

Mathematica

```
Table[1/{n+k},{n,1,3},{k,1,5}]//MatrixForm
.
1   1   1   1   1
-   -   -   -   -
2   3   4   5   6

1   1   1   1   1
-   -   -   -   -
3   4   5   6   7

1   1   1   1   1
-   -   -   -   -
4   5   6   7   8
```

Vectors are non-nested list and a **matrix** is represented by a list of lists. The command `MatrixForm` displays a list of lists as a matrix. Operations on vectors and matrices will be discussed later.

1.5 Exercises

• **Exercise 1–1:**
Generate a list of all commands to contain the combination of the letters `Solve`.

• **Exercise 1–2:**
Find and read the online documentation on the command `FindRoot`.

• **Exercise 1–3:**
Find all information about the function sinh with the Help Browser.

• **Exercise 1–4:**
Find the first 100 digits of the number π.

• **Exercise 1–5:**
Create a list of the inverse of all the square numbers from 1 to 13, i.e.

$$\frac{1}{1}\;,\;\frac{1}{4}\;,\;\frac{1}{9}\;,\;\ldots\frac{1}{169}$$

1.5.1 Answers to exercises

Solution to Exercise 1–1 :

Mathematica

```
?*Solve*
```

Solution to Exercise 1–2 : Use the Help–Browser or

Mathematica

```
??FindRoot
```

Solution to Exercise 1–4 :

Mathematica

```
N[Pi,1000]
```

Solution to Exercise 1–5 :

Mathematica

```
Table[1/n^2,{n,1,13}]
```

Chapter 2

Definition of functions and elementary graphics

2.1 Definition of functions

In the previous chapter we already considered very simple definition of functions. Thus we can now have a closer look at a few details. As a reminder an elementary example.

Mathematica

```
f[x_] := Sin[x] Exp[- 0.2 x]
f[a_,b_] := (1-a^2) Cos[3b]
Plot3D[f[x,y],{x,-3,3},{y,-2,1}];
```

With `??f` you can now look up the information stored by *Mathematica*.

Mathematica

```
??f
.
Global`f
f[x_] := Sin[x]*Exp[-0.2*x]
f[a_, b_] := (1 - a^2)*Cos[3*b]
```

Thus *Mathematica* does know about two functions with the name **f**, one function with one argument and another function with two arguments. Depending on how the function is called, one or the other definition is used. This is illustrated by the code below.

Mathematica

```
{f[1],f[3,0],f[1,1,3]}
.
{0.818731 Sin[1], -8, f[1, 1, 3]}
```

2.1.1 Conditional definitions

Sometimes two (or more) formulas have to be used to compute the value of a function, depending on the value of the argument. An example is given below.

$$f(x) = \begin{cases} 1 & \text{if} \quad -1 \leq x \leq 1 \\ x^2 & \text{if} \quad 1 < x \end{cases}$$

This function is defined by the following *Mathematica* code.

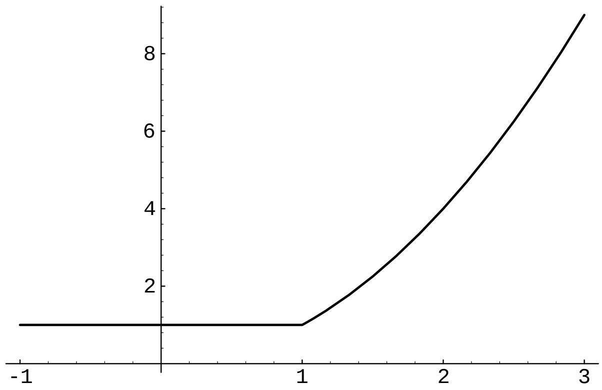


Figure 2.1: A function, defined by a conditional definition

Mathematica

```
Clear[f]
f[x_] := 1 /; (-1<= x && x<=1)
f[x_] := x^2 /; x>1
```

To get the graph in figure 2.1 use

Mathematica

```
Plot[f[s],{s,-1,3}]
```

Now we tag on another definition for the function `f`. Then we have an ambiguous definition for the domain $x < -0$. A closer look at the information stored by *Mathematica*. The different definitions are sorted by the rule of thumb: from the special to the more general definition. When evaluation `f`, the definition are checked from top to bottom, the first definition to apply is used. If none of the definitions apply, the the input is returned as result.

Mathematica

```
f[x_] := -1 /; x<0
??f
.
Global`f
f[x_] := 1 /; -1 <= x && x <= 1
f[x_] := x^2 /; x > 1
f[x_] := -1 /; x < 0
```

A simple test confirms this observation.

Mathematica

```
f[-0.5]
.
1
```

With the help of the command `Which[]` the above function can be defined with one line of code.

Mathematica

```
Clear[f]
f[x_] := Which[-1<=x && x<=1, 1,x>1, x^2]
```

The syntax for `Which[]` is

Which[condition1,code1, condition2,code2, ..., conditionN,codeN]

This definition is usually easier to use for further computations. E.g. one can find the derivative of this function with

Mathematica

```
D[f[r],r]
.
Which[r <= 1, 0, r > 1, 2 r]
```

The solution with the command `Which[]` should be prefered.

2.1.2 Replacements rules

Mathematica has a very powerful mechanism to replace symbols (or pattern) in an expression by other expressions. The syntax should be easy to understand.

`expr /. x -> value`	replace `x` by `value` in the expression `expr`
`expr /. {x->value1 , y->value2}`	perform several replacements
`expr //. rules`	apply `rules` until the result no longer changes

A simple example shall illustrate those operations. Let us consider an expression. If we were to equate it to zero we would get the equation of a circle of radius R.

Mathematica

```
expr = x^2 + y^2 - R^2
.
  2    2    2
-R  + x  + y
```

Now reconsider the same expression, but with $R = 2$.

Mathematica

```
expr /. R -> 2
.
      2    2
-4 + x  + y
```

Try to solve the equation for a circle with radius $R = 3$ for the variable y.

Mathematica

```
sol=Solve[expr==0 /. R ->3,y]
.
                    2                     2
{{y -> -Sqrt[9 - x ]}, {y -> Sqrt[9 - x ]}}
```

We arrive at two possible solution, an upper semi circle and a lower semi circle. The solutions are actually given as replacement rules. To find the possible values of y for $x = 1$ we use

Mathematica

```
sol /. x-> 1
.
{{y -> -2 Sqrt[2]}, {y -> 2 Sqrt[2]}}
```

It is a good exercise to examine the result of `FullForm[sol]`. One can find out how to access only the upper semi circle. Try to figure out the structure of this multiply nested list.

Mathematica

```
FullForm[sol]
.
List[List[Rule[y, Times[-1, Power[Plus[9, Times[-1, Power[x, 2]]], Rational[1, 2]]]]],
     List[Rule[y, Power[Plus[9, Times[-1, Power[x, 2]]], Rational[1, 2]]]]
    ]
```

Mathematica

```
circle[x_] = sol[[2,1,2]]
.
           2
Sqrt[9 - x ]
```

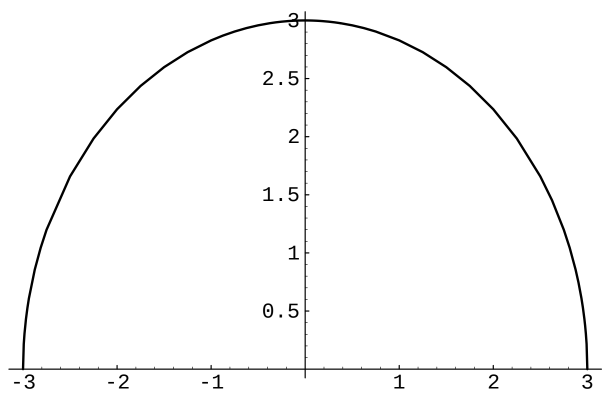


Figure 2.2: A semi circle

2.1.3 Immediate versus delayed evaluation

There are two ways to define a function.

Mathematica

```
Clear[g]
g[x_] := Sin[3 x]
g[x_]  = Sin[3 x]
```

With an **immediate definition** by `lhs = rhs` *Mathematica* computes the given expression `rhs` immediately. The result is then used by *Mathematica* whenever the expression `lhs` shows up. `rhs` is evaluated only once.
With a **delayed evaluation** by `lhs := rhs` nothing is computed. *Mathematica* only stores the information, that it knows how to compute `lhs`. If later on *Mathematica* finds the expression `lhs`, then it evaluate `rhs`. Thus `rhs` may be evaluated more than once.

Very similar arguments apply also to replacement rules. There are **immediate replacements rules ->** and **delayed replacements rules :>**.

The following examples should illustrate the sometimes delicate difference between immediate and delayed definition.

Mathematica

```
Clear[a,b,c,d]
a=5;
b := a^2;
c = a^2;
{b,c}
.
{25, 25}
```

Now b and c are equals to 25. But now we change the value of $a = 3$ and recompute b and c.

Mathematica

```
a=3;
{b,c}
.
{9, 25}
```

For b we obtain the new values 3^2 since the expression a^2 is recomputed. But the value of c remains 25 since *Mathematica* only recalls the result of the above calculation.

The following example shows that the same arguments also apply to algebraic operations.

Mathematica

```
fDelayed[x_] := Expand[x]
fImmediate[x_] = Expand[x]
fDelayed[(x+b)^2]
fImmediate[(a+b)^2]
.
 2            2
b  + 2 b x + x
      2
(a + b)
```

The difference is also shows up in the information stored by *Mathematica*.

Mathematica

```
?fDelayed
.
Global`fDelayed
fDelayed[x_] := Expand[x]
```

Mathematica

```
?fImmediate
.
Global`fImmediate
fImmediate[x_] = x
```

The command `Random[]` provides us with a good illustration of the differences between immediate and delayed evaluation.

- immediate evaluation:

  Mathematica

  ```
  twoDice = Random[Integer,{1,6}]+Random[Integer,{1,6}];
  Table[twoDice,{20}]
  .
  {8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8}
  ```

  The two dice are tossed only once, with a sum of 8. All subsequent call of the symbol `twoDice` refer to this one toss.

- delayed evaluation

  Mathematica

  ```
  twoDice := Random[Integer,{1,6}]+Random[Integer,{1,6}];
  Table[twoDice,{20}]
  .
  {10, 5, 9, 5, 12, 5, 3, 5, 4, 8, 9, 8, 10, 3, 5, 9, 7, 8, 8, 8}
  ```

  The command `twoDice` corresponds to tossing two dice and adding up the number of points. The `Table[]` command the generates a list of 20 such experiments.

2.2 Elementary graphics

Mathematica is extremely flexible in generating graphics and it can be an adventure to produce an elaborate graphics. But for simple problems there are easy to use commands. We will only examine those.

2.2.1 `Plot[]`

In the first section we already saw how to generate the graph given in figure 2.3. Observe that we used two **options** to force the y-domain to be $[-1, 2]$ and to label the axes.

Mathematica

```
Clear[f]
f[x_] := Cos[x] Exp[-x^2/50]
Plot[f[x],{x,-5,10},
     PlotRange -> {-1,2},
     AxesLabel->{"x","y"}];
```

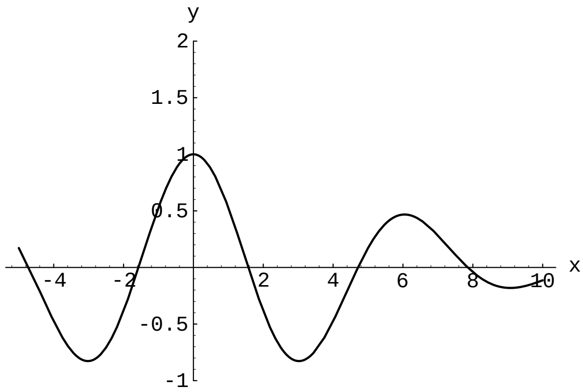


Figure 2.3: A first plot

It can be very important to choose the right values for the options of the command `Plot[]`. With `??Plot` you get a brief description of `Plot[]` and the default values for all the possible options. The Help Browser will also show some examples.

Mathematica

```
??Plot
.
Plot[f, {x, xmin, xmax}] generates a plot of f as a function of x from xmin to xmax.
   Plot[{f1, f2, ...}, {x, xmin, xmax}] plots several functions fi.

Attributes[Plot] = {HoldAll, Protected}

Options[Plot] = {AspectRatio -> GoldenRatio^(-1), Axes -> Automatic,
    AxesLabel -> None, AxesOrigin -> Automatic, AxesStyle -> Automatic,
    Background -> Automatic, ColorOutput -> Automatic, Compiled -> True,
    DefaultColor -> Automatic, Epilog -> {}, Frame -> False,
    FrameLabel -> None, FrameStyle -> Automatic, FrameTicks -> Automatic,
    GridLines -> None, MaxBend -> 10., PlotDivision -> 20., PlotLabel -> None,
    PlotPoints -> 25, PlotRange -> Automatic, PlotRegion -> Automatic,
    PlotStyle -> Automatic, Prolog -> {}, RotateLabel -> True, Ticks -> Automatic,
    DefaultFont :> $DefaultFont, DisplayFunction :> $DisplayFunction}
```

With `Options[Plot]` you would get a list of all the options to `Plot[]`. With the function browser or with `?PlotPoints` one can extract a bit information about the option `PlotPoints`.

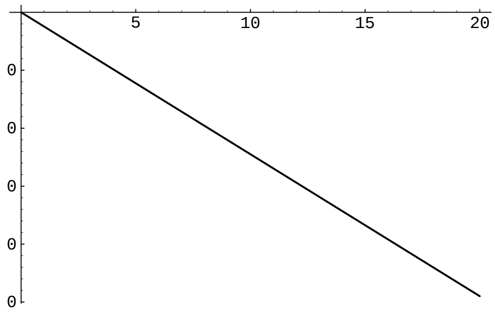
Mathematica

```
?PlotPoints
.
PlotPoints is an option for plotting functions. With a single variable, PlotPoints -> n
   uses a total of n sample points. With two variables, PlotPoints -> n specifies that n
   points should be used in both the x and y directions, and PlotPoints -> {nx, ny}
   specifies different numbers of sample points for the two directions.
```

The default value of `PlotPoints` is 25. An unlucky choice of this option can have surprising results. Just study the code below and the resulting figure 2.4.

Mathematica

```
Plot[ Sin[Pi x] ,{x,0,20},PlotPoints -> 11];
Plot[ Sin[Pi x] ,{x,0,20},PlotPoints -> 12];
```



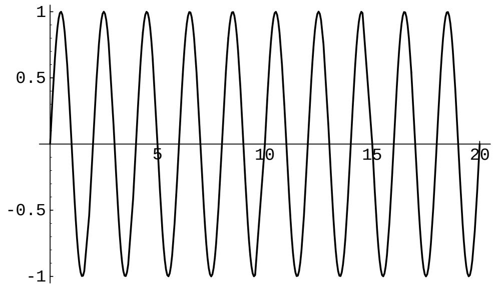


Figure 2.4: The option PlotPoints to Plot[]

2.2.2 How to put multiple graphs in one figure

In figure 2.5 you see the graphs of two functions in one graphic. In addition the type of line to be used was modified through the option PlotStyle .

Mathematica

```
Plot[{E^x, x^E}, {x, 0, 5},
PlotStyle -> {
  {Thickness[0.02], Dashing[{0.05, 0.03}]},
  {Thickness[0.01], RGBColor[1, 0, 1]}
  }
];
```

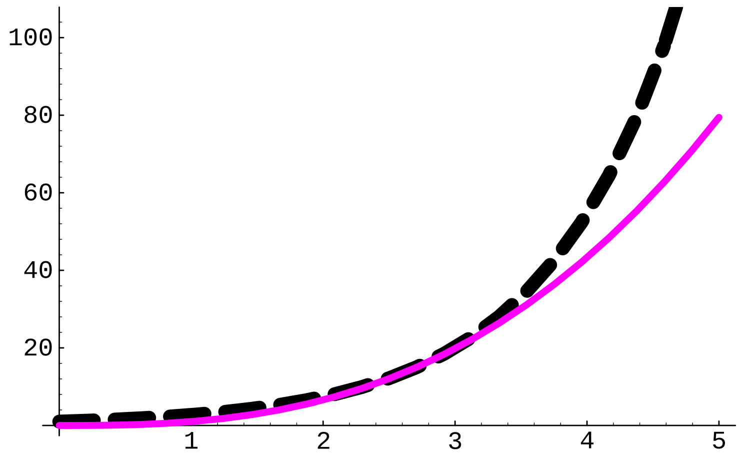


Figure 2.5: Multiple graphs with different plot styles

2.2.3 Plot3D[]

The subsequent commands plot the real part of the complex function $\cos z$ as functions of the real part x and the imaginary part y ($z = x + iy$). The result is seen in figure 2.6. For a 3-D graphics of this type it is important to choose a good point in space from which to look at the graph. This is done by the option `ViewPoint`. You find a description of this option in section 1.9.7 of the *Mathematica*-book (Version 3.0). You can use the **ViewPoint Selector**, to be found in the menu `Input -> 3D ViewPoint Selector`.

Mathematica

```
Plot3D[Re[Cos[x+I y]], {x,-14,5},{y,-1.5,1.5},
        BoxRatios -> {3,2,2},
        PlotPoints -> 30,
        ViewPoint ->{1,-2.5,2}];
```

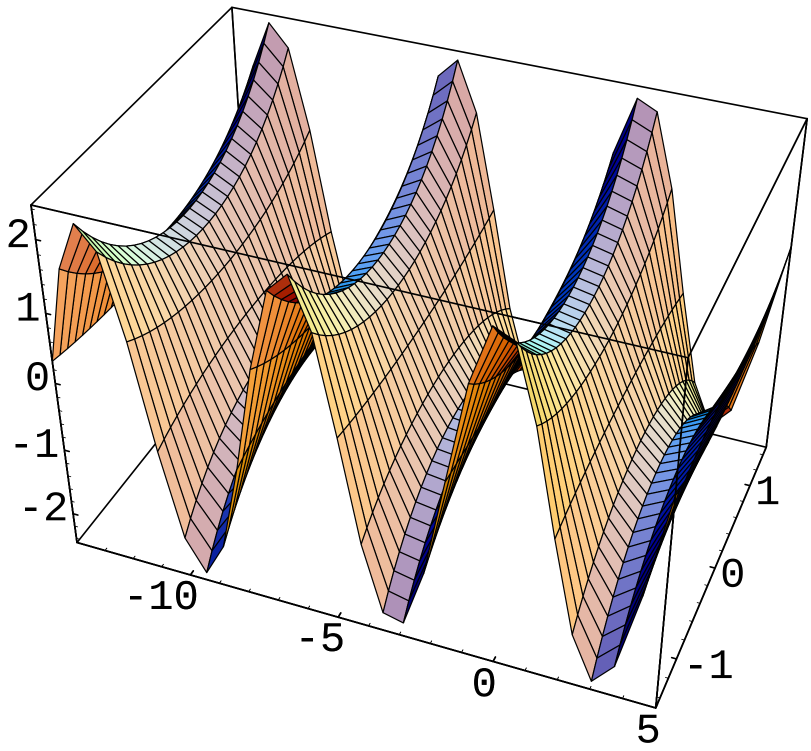


Figure 2.6: The real part of the complex function $\cos z$

2.2.4 ContourPlot[]

With the command `ContourPlot[]` you can plot level curves of a given function. This can be useful to visualise solution sets of functions of two variables. The solutions of an equation $f(x, y) = 0$ corresponds to all points (x, y) on the zero level. As an example consider the code below and figure 2.7.

Mathematica

```
ContourPlot[Sin[x^2-y^2], {x,-1,1},{y,-1,1},
        Contours -> 20];
```

```
ContourPlot[Sin[x^2-y^2], {x,-1,1},{y,-1,1},
        Contours -> 20,
        ContourShading -> False];
```

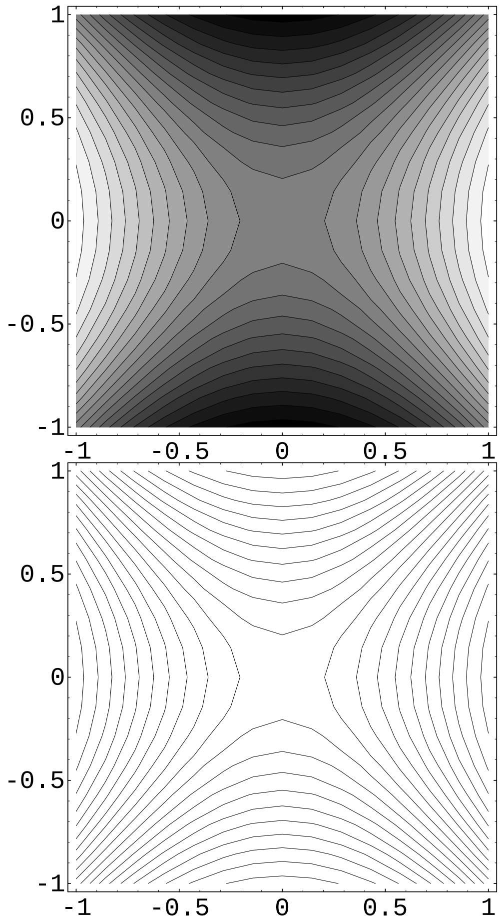


Figure 2.7: Level curves

2.2.5 `ParametricPlot[]` and `ParametricPlot3D[]`

If x and y are both given as functions of a parameter t,then the command `ParametricPlot[]` will visualise the corresponding curve. As an example we consider the ellipse

$$\left(\begin{array}{c} x\,(t) \\ y\,(t) \end{array}\right) = \left(\begin{array}{c} 2+\cos\omega \\ 1+3\,\sin\,\omega \end{array}\right) \quad \text{for} \quad 0 \leq \omega \leq 2\,\pi$$

To obtain figure 2.8 use the following code.

Mathematica

```
ParametricPlot[ {2 + Cos[omega], 1 + 3 Sin[omega]},{omega ,0,2 Pi}]
```

?? Considerably more complicated surfaces can be generated by `ParametricPlot3D[]`. The *Mathematica* book contains many impressively elaborate graphics. If in need of a special type of graphic browse through sections 1.9 and 2.9 oth the *Mathematica* book and the add-on packages.

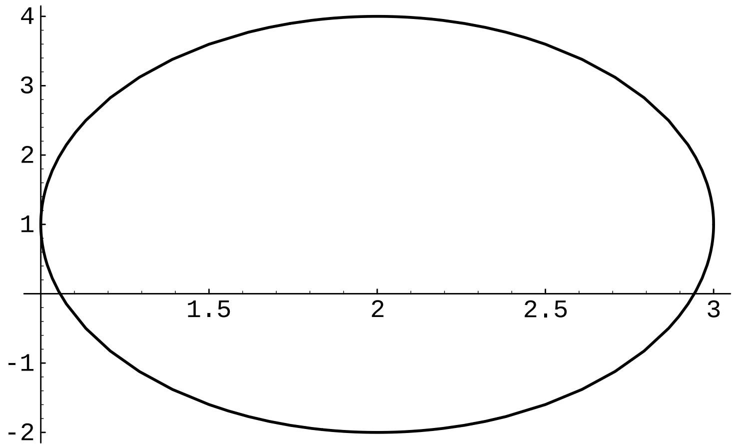


Figure 2.8: An ellipse by `ParametricPlot[]`

Mathematica

```
R= 3;
r= 1;
h= 2;
ParametricPlot3D[
    {(R +r Cos[phi]) Cos[omega],
     (R +r Cos[phi]) Sin[omega],
       h + r Sin[phi] },
    {omega ,0,2 Pi},{phi,0, 2Pi}];
```

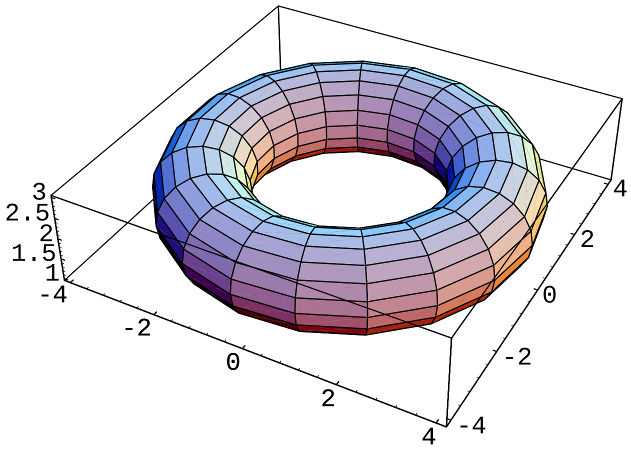


Figure 2.9: A torus generated by `ParametricPlot3D[]`

2.2.6 How to convert a graphic into a standard graphics format

With *Mathematica* 3.0

The command `Display[]` will allow you to convert graphic objects into many different graphics formats, Among them Encapsulated PostScript. This file can then be used by other applications, e.g. a typesetting program.

Mathematica

```
gg=Plot[Sin[x],{x,0,7},DisplayFunction -> Identity];
Display["blabla.eps",gg,"EPS"];
```

The above code generates a graphics object `gg`, without displaying it on the screen. The a PostScript file `blabla.eps` is created in the current directory with the generated graphics.

Some of the possible standard formats are `EPS`, `PBM`, `XBitmap`, `GIF`, `TIFF`, `PICT` and a few more.

2.3 Exercises

• Exercise 2–1:
Write a function to count the factors of 2 in the factorisation of a natural number.
Tip: `FactorInteger`

• Exercise 2–2:
Define a function

$$f(x,y) = \begin{cases} 1 & \text{if} \quad x^2+y^2 \leq 0.5 \\ 2-x^2-y^2 & \text{if} \quad 0.5 < x^2+y^2 \leq 2 \\ 0 & \text{if} \quad x^2+y^2 > 2 \end{cases}$$

Show the graph of this function.

• Exercise 2–3:
Explain the differences in comutation time for the following three commands.

Mathematica

```
Clear[f]
f[a_] = Integrate[ Exp[ a x] Sin[ 3x],{x,0,3}];
Table[f[a],{a,0,20}]; //Timing
.
{0.1 Second, Null}
```

Mathematica

```
Clear[f]
f[a_] := Integrate[ Exp[ a x] Sin[ 3x],{x,0,3}];
Table[f[a],{a,0,20}]; //Timing
.
{89.5 Second, Null}
```

Mathematica

```
Clear[f]
f[a_] := Integrate[ Exp[ a x] Sin[ 3x],{x,0,3}];
Table[Evaluate[f[a]],{a,0,20}]; //Timing
.
{3.33333 Second, Null}
```

• Exercise 2–4:
First define a new function

$$f(x,y) = \cos(2x - y)$$

(a) Show the graph of this function for the domain $-2 \leq x \leq 2$ and $-3 \leq x \leq 3$. The scaling on the three axis has to be identical.

(b) Show the graph of this function for the domain $-2 \leq x \leq 2$ and $-3 \leq x \leq 3$. The scaling on the three axis has to be identical and the box around the surface shoul not show.

(c) Choose different values for `ViewPoint`.

• Exercise 2–5:
Use `ContourPlot[]` and `Show[]` to locate common zeros of the two functions.

$$f(x,y) = x^2 + 2y^2 + xy + 4x + 4y - 10 \quad \text{and} \quad g(x,y) = x^2 + y^2 - 4$$

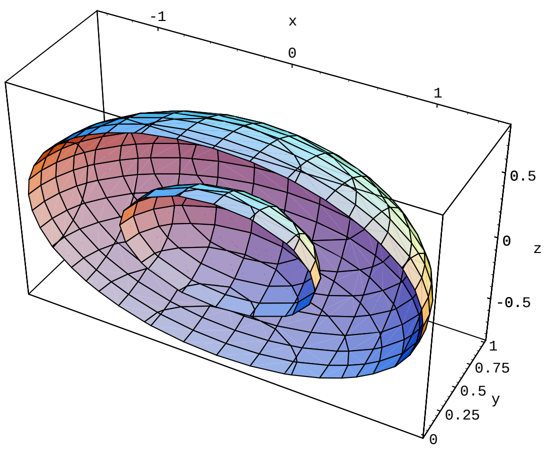
for $-6 \leq x \leq 3$, $-5 \leq y \leq 5$. The circle has be a circle on the screen (not an ellipse).

• Exercise 2–6:
Load the package `Graphics‘ContourPlot3D‘` and use one of the given commands to generate the level surfaces at levels $f = 0.5$ and $f = 2.0$, where

Mathematica

```
f[x_,y_,z_] = x^2 + 2 y^2 + 3 z^2 ;
```

Use the options `Axes -> True` and `PlotPoints->{5,5}` , among others. Use the domains of x, y and z from the graphics below. The resulting graphics should be similar to the result below.



2.3.1 Answers to exercises

Solution to Exercise 2–1 :

Mathematica

```
Clear[f,factorsOffTwo]
f[n_Integer] := FactorInteger[n]
factorsOffTwo[n_] :=
  Which[f[n][[1,1]]==2,(f[n][[1,2]]), f[n][[1,1]]!=2, 0]
```

Solution to Exercise 2–2 :

Mathematica

```
h[r_] := Which[ r^2<=0.5,1, r<=Sqrt[2] , 2-r^2, r^2>2 , 0]
(* Plot[h[r],{r,0,4}] *)
f[x_,y_] := h[Sqrt[x^2+y^2]]
Plot3D[f[x,y],{x,-2,2},{y,-2,2},PlotPoints -> 40];
```

or

Mathematica

```
f[x_,y_] := 1 /; (x^2+y^2)<=0.5 ;
f[x_,y_] := 2-x^2-y^2 /; ((x^2+y^2)>0.5 && (x^2+y^2) <=2 );
f[x_,y_] := 0 /; (x^2+y^2)>2 ;
```

Solution to Exercise 2–3 :

- Immediate evaluation. The integral is computed once, with a as a parameter. Then the result is computed 20 times, for different values of a. Only the `Table[]` command is timed.
- Delayed evaluation. The integral is not computed in the definition of f, but computed 20 times within `Table[]`. Thus the computation time is rather large.
- Delayed evaluation, with `Evaluate[]`. The integral is not computed in the definition of f. The command `Evaluate` in the `Table[]` forces *Mathematica* to first compute the integral and only then choose different values of a. Thus the timing measures one evaluation of the integral.

Solution to Exercise 2–4 :

Mathematica

```
Clear[test]
test[x_,y_] := Cos[2 x - y]
Plot3D[test[x,y],{x,-2,2},{y,-3,3},
   BoxRatios -> Automatic];
```

Mathematica

```
Plot3D[test[y,x],{x,-2,2},{y,-3,3},
   BoxRatios -> Automatic,
   Boxed -> False];
```

Solution to Exercise 2–5 :

Mathematica

```
f[x_,y_] := x^2 +2 y^2 + x y + 4x+ 4y -10
g[x_,y_] := x^2 + y^2 - 4
fig1=ContourPlot[f[x,y], {x,-6,3},{y,-5,5},
      Contours -> {0},
      ContourShading -> False,
      DisplayFunction -> Identity];
fig2=ContourPlot[g[x,y], {x,-6,3},{y,-5,5},
      Contours -> {0},
      ContourShading -> False,
      DisplayFunction -> Identity];
Show[{fig1,fig2},
      DisplayFunction -> $DisplayFunction,
      AspectRatio -> Automatic]
```

Solution to Exercise 2–6 :

Mathematica

```
  Needs["Graphics`ContourPlot3D`"]
f[x_,y_,z_] = x^2 + 2 y^2 + 3 z^2 ;
fig=ContourPlot3D[f[x,y,z], {x, -2,2}, {y,0,2}, {z, -1, 1},
  Contours ->{0.5,2},
  AxesLabel ->{"x","y","z"},
  Axes -> True,
  PlotPoints->{5,5},
  AspectRatio -> Automatic];
```

Chapter 3

Operations with lists

Lists are the basic data structure in *Mathematica*. one might even claim that **everything is a list**. Vectors and matrices are special cases of lists. Thus it is clear that one has to master basic operations on list. Some of the presentation in this section is taken from [Bahd95].

3.1 Creating lists

The basic command to create lists is `Table[]`. Depending on the type of arguments the command will react differently, see Table 3.1. In each of the `Table` commands the expression `expr` is evaluated with the given index as argument.

`Table[ expr ,{n}]`	create a list with `expr` the unnamed index takes the values 1, 2, ..., n
`Table[ expr ,{k,n}]`	create a list with `expr` the index `k` takes the values 1, 2, ..., n
`Table[ expr ,{k,n,m}]`	create a list with `expr` the index `k` takes the values n, n+1, n+2, ..., m
`Table[ expr ,{k,n,m,step}]`	create a list with `expr` the index `k` values from n to m with stepsize `step`
`Table[ expr ,{i,n},{j,m}]`	create a nested list with `expr` `i` will be the outer index, `j` the inner index
`Range[n]`	create a list with the numbers 1, 2, ..., n
`Range[n,m]`	create a list with the numbers n, n+1, n+2, ..., m
`Range[n,m,step]`	create a list of numbers from n to m with stepsize `step`
`Partition[list,n]`	partitions list into non-overlapping sublists of length n

Table 3.1: The command `Table[]`

Below find a few elementary examples with this command. The commands and cerresponding results should speak for themselves.

Mathematica

```
Table[Random[],{5}]
.
{0.137292, 0.446294, 0.601422, 0.539872, 0.424024}
```

Mathematica

```
Table[n^2,{n,9}]
.
{1, 4, 9, 16, 25, 36, 49, 64, 81}
```

Mathematica

```
Table[n^2,{n,3,9}]
.
{9, 16, 25, 36, 49, 64, 81}
```

Mathematica

```
Table[n^2,{n,3,9,4}]
.
{9, 49}
```

If the iterator is given correctly, the `Table[]` can also count backwards.

Mathematica

```
Table[n^2,{n,10,-3,-1}]
{100, 81, 64, 49, 36, 25, 16, 9, 4, 1, 0, 1, 4, 9}
```

When creating nested lists (matrices) with `Table[]` you should verify that the correct index is iterated first. In the example below the outer index `i` varies from 1 to 3. Then j varies from 1 to 4, for each of the values of i .

Mathematica

```
Table[ i Sin[j],{i,3},{j,4}]
.
{{  Sin[1],   Sin[2],   Sin[3],   Sin[4]},
 {2 Sin[1], 2 Sin[2], 2 Sin[3], 2 Sin[4]},
 {3 Sin[1], 3 Sin[2], 3 Sin[3], 3 Sin[4]}}
```

The you can get the same result with a nested call of the operator `Table[]`.

Mathematica

```
Table[ Table[ i Sin[j],{j,4}] ,{i,3}]
```

There are other commands to create lists, find some of them in Table 3.2.

The command `Array[]` is often called with a **pure function** as the first argument. The examples below will show you how to use it in a simple case.

Mathematica

```
Array[Sin,{5}]
.
{Sin[1], Sin[2], Sin[3], Sin[4], Sin[5]}
```

`Array[f, n]`	create the list {`f[1]`,`f[2]`, ... ,`f[n]`}
`Append[ oldlist, elem]`	returns `oldlist` with `elem` appended
`AppendTo[ oldlist, elem]`	replace `oldlist` by `oldlist` with `elem` appended
`Prepend[ oldlist, elem]`	returns `oldlist` with `elem` prepended
`PrependTo[ oldlist, elem]`	replace `oldlist` by `oldlist` with `elem` prepended
`Join[ listA, ListB]`	create a new list by joining two lists
`Flatten[ myList ]`	transform a nested list into a flat list
`Union[ list ]`	gives a sorted list of all the elements in `list`, in which all duplicate elements have been dropped

Table 3.2: Commands to create lists

Observe the rather cryptic notation in the second call of of `Array[]`. If your aim is to produce unreadable, but short code, use many of those shortcuts!

Mathematica

```
myFunc=Function[n,n^2]
Array[myFunc,{5}]
Array[#^2&,{5}]
.
              2
Function[n, n ]
{1, 4, 9, 16, 25}
{1, 4, 9, 16, 25}
```

If you already have some lists and you want to combine them in a given fashion *Mathematica* offers some commands for this. Observe the difference between `Append[]` and `Join[]`.

Mathematica

```
Append[{1,2,3}, {5,4}]
.
{1, 2, 3, {5, 4}}
```

Mathematica

```
Join[{1,2,3}, {5,4}]
.
{1, 2, 3, 5, 4}
```

Some problems can be solve by creating an empty list and the append (or prepend) elements to it.

Mathematica

```
Clear[myList]
myList={};
Table[ myList = Append[myList,k],{k,5}]
.
{{1}, {1, 2}, {1, 2, 3}, {1, 2, 3, 4}, {1, 2, 3, 4, 5}}
```

On occasion the difference between `Append` and `AppendTo` does create problems. Make sure that you use the desired command. Consider the following two slightly different solutions to very similar problems.

Mathematica

```
Clear[myList]
myList={1};
Do[ myList = Append[myList,2*Last[myList]];,{10}]
myList
.
{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024}
```

Mathematica

```
Clear[myList]
myList={1};
Do[ PrependTo[myList,2*First[myList]];,{10}]
myList
.
{1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1}
```

The command `Flatten[]` does remove „extra“ brackets in a nested list. The command can be applied to a given level of the list, see the online documentation for details.

Mathematica

```
Clear[myList]
myList=Table[Table[i+10j,{i,0,j}],{j,0,6}]
Flatten[myList]
.
{{0}, {10, 11}, {20, 21, 22}, {30, 31, 32, 33},
  {40, 41, 42, 43, 44}, {50, 51, 52, 53, 54, 55},
  {60, 61, 62, 63, 64, 65, 66}}

{0, 10, 11, 20, 21, 22, 30, 31, 32, 33, 40, 41, 42, 43,
  44, 50, 51, 52, 53, 54, 55, 60, 61, 62, 63, 64, 65, 66}
```

The command `Union[]` can be useful to eliminate duplicate elements.

Mathematica

```
Union[{a,b,1,A,b,a}]
.
{1, a, A, b}
```

3.2 Accessing parts of lists

There are a number of *Mathematica* commands to access parts of given lists. In Table 3.3 you find the most important ones of them.

The example below should be enough to get the idea on how to use those commands.

Mathematica

```
testList = Table[n, {n,10}]
Take[testList,3]
Take[testList,-3]
Take[testList,{3,7}]
```

`Take[list, n]`	creates a new list with the first `n` elements of `list`
`Take[list, -n]`	creates a new list with the last `n` elements of `list`
`Take[list, {n,m}]`	creates a new list with elements `n` trough `m` of `list`
`Drop[list, n]`	creates a new list by dropping the first `n` elements of `list`
`Drop[list, -n]`	creates a new list by dropping the last `n` elements of `list`
`First[list]`	give the first element of `list`
`Last[list]`	give the last element of `list`
`list[[n]]`	give the `n`-th element of `list`
`list[[n]]` = expr	set the `n`-th element of `list` to `expr`
`mat[[n,m]]`	give the (n, m) element of the matrix `mat`
`mat[[Range[n1,n2], Range[m1,m2]]]`	give the block matrix with row indeces n1, ... , n2 and column indeces m1, ... ,m2 of the matrix `m`

Table 3.3: Extracting parts of a list

```
.
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
{1, 2, 3}
{8, 9, 10}
{3, 4, 5, 6, 7}
```

Mathematica

```
Drop[testList,3]
Drop[testList,-3]
Drop[testList,{3,7}]
.
{4, 5, 6, 7, 8, 9, 10}
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 8, 9, 10}
```

Mathematica

```
(testMatrix = Table[ 10 i + j ,{i,6},{j,7}])//MatrixForm
testMatrix[[3,4]]
testMatrix[[Range[3,5],Range[2,3]]] //MatrixForm
.
11   12   13   14   15   16   17
21   22   23   24   25   26   27
31   32   33   34   35   36   37
41   42   43   44   45   46   47
51   52   53   54   55   56   57
61   62   63   64   65   66   67

34

32   33
42   43
52   53
```

3.3 Map, Apply, Head and all that

There are a few commands for lists that are specific to *Mathematica*. Find a brief description in Table 3.4. Masters of those commands can achieve amazing results with them, but they are not always easy to get used to.

`Head[ expr ]`	gives the head of an expression
`Apply[ f, expr]`	replaces the head of `expr` by `f`
`Map[f, expr]`	apply `f` to `expr` at level 1
`MapAt[f, expr, partspec]`	apply `f` to `expr` at position `partspec`
`Outer[f, list1, list2]`	gives the generalised outer product of the lists
`Select[ list, test]`	give the elements in `list` for which `test` gives `True`

Table 3.4: Special commands on lists

Each variable in *Mathematica* consists of a head and a tail. Examine this structure with the command `FullForm[]`.

Mathematica

```
Remove[myList]
myList={2,3,4};
FullForm[myList]
Head[myList]
Apply[Plus,myList]
.
List[2, 3, 4]
List
9
```

Another method to get to the head of an object is to access its 0-th component. It should not be used, but keep this in mind. Problems could occur if you try the access all elements of a vector and you start with index 0. This will give you the `Head` of your list and not the first element.

Mathematica

```
myList[[0]]
.
List
```

If you have a known function ant want to apply it to each element in a given list then you can do this with the help of `Map`.

Mathematica

```
myList ={0,a,Pi/2,5};
Map[Sin, myList]
.
{0, Sin[a], 1, Sin[5]}
```

If you function happens to be slightly more complicated than the above example, then you can first construct your function with the help of `Function`. This should also be used in combination with `Select[]` to choose only specific parts of a given list.

Mathematica

```
?Function
Function[body] or body& is a pure function. The formal parameters are # (or #1), #2, etc.
   Function[x, body] is a pure function with a single formal parameter x.
   Function[{x1, x2, ...}, body] is a pure function with a list of formal parameters.
   Function[{x1, x2, ...}, body, {attributes}] has the given attributes during evaluation.
```

Suppose you want to generate integer random numbers with different ranges. The four random numbers should be between 1 and 5, 1 and 3, 1 and 30, 1 and 200. Thus we first write a function `randomWithDomain` which will take one argument n and return an integer random number between 1 and n.

Mathematica

```
randomWithDomain = Function[Random[Integer,{1,#}]];
Map[randomWithDomain,{5,3,30,200}]
Map[randomWithDomain,{5,3,30,200}]
.
{5, 1, 21, 11}
{1, 2, 27, 150}
```

With the attribute `Listable` a function can be applied to each element of a list automatically. The mathematical standard function (e.g. `Sin[]`) are of this type.

Mathematica

```
Clear[myFunction]
myFunction = Function[x, x^2 ,{Listable}];
myFunction[{a,b,3}]
.
  2   2
{a , b , 9}
```

The command `Outer[]` will compute generalised outer products. It is best illustrated by two examples.

Mathematica

```
Outer[f,{a,b}]
.
{f[a], f[b]}
```

Mathematica

```
Outer[f,{a,b},{c1,c2,c3}]
.
{{f[a, c1], f[a, c2], f[a, c3]},
 {f[b, c1], f[b, c2], f[b, c3]}}
```

In previous chapters we already use the commands `Select[]` and `Length[]`. Consider the following simulation of 1000 random numbers, distributed with a normal distribution with mean 20 and standard deviation 5. Then we compute the percentage of random numbers that deviate less than one standard deviation from the mean.

Mathematica

```
Needs["Statistics`ContinuousDistributions`"]
Clear[myList,n]
n=1000;
myList=Table[Random[NormalDistribution[20,5]],{n}];
test=Function[#1>15 && #1<25];
Length[Select[myList,test]]/n //N
.
0.641
```

3.4 Exercises

• Exercise 3–1:
Create a list of the first 36 square numbers. Then use the command `Partition[]` to show it as a 6×6 or a 4×6 matrix.

• Exercise 3–2:
Consider the 10×4 matrix generated by

Mathematica

```
simul=Table[Random[],{10},{4}];egin{mathematica}
```

- Use `Map[]` and `Sort[]` to sort each column of `simul`.
- Extract the fourth column of the resulting matrix.
- Compute the average value, using `Apply[]` and `Plus`.

Why is the resultin aveage above 0.5 (usually)?

3.4.1 Solutions to exercises

Solution to Exercise 3–1 :

Mathematica

```
a=Table[k^2,{k,36}];
Partition[a,6]//MatrixForm
Partition[a,4]//MatrixForm
```

Solution to Exercise 3–2 :

Mathematica

```
simul=Table[Random[],{10},{4}];
(simul=Map[Sort,simul])//MatrixForm
fourthCol=Transpose[simul][[4]];
Apply[Plus,fourthCol]/Length[thirdCol]
```

Chapter 4

Files, data and a first program

Some of the commands in this chapter depend on the operation system of the computer. The examples were tested on a version of UNIX (Linux). You may have to make small adjustments.

4.1 Manipulating files and directories

Mathematica has a few commands to work with the file system on the given computer. Find a list of all those commands with `?*Directory*` and `?*File*`.

4.2 DOS and UNIX

In this chapter we will work with files and directories. The following commands show you how to change to a directory and then produce a new file to contain a listing of this directory. For this we use `Run[]` to execute an external program.

Mathematica

```
SetDirectory["~/tex/ndit/files"]
Run["ls > t.tmp"];
.
/home/sha/tex/ndit/files
```

The command `Run["ls>>t.tmp"]` generates a list with all the file names and is putting them in a new file `t.tmp`. With `!!t.tmp` you can look at the content of this file. It seems to be necessary to put the command `!!t.tmp` in a separate cell.

Mathematica

```
!!t.tmp
.

chap.aux
chap.dvi
chap.log
chap.tex
chap.toc
files.ans
files.aux
files.tex
t.tmp
```

Unfortunately DOS and UNIX do **not** use the same characters to indicate the end of a line. DOS uses the two characters CarrigeReturn and LineFeed (`\r\n`), while UNIX only use a single LineFeed (`\n`). Thus files from a DOS computer can not be read immediately on a UNIX computer. On many systems you will find programs to con vert from one format to the other. In the usual place you find the C-source of `dos2unix` and `unix2dos`. Within *Mathematica* you can then convert all files with the extension `.dat` in the current directory with the command

Mathematica

```
Run["dos2unix *.dat"]
```

4.3 Read and display data

In a subdirectory `.../daten` find a few files with data. When trying to read one of those you might get an error message of the following type

Mathematica

```
mydat=ReadList["daten1.dat",{Number,Number}]
.
Read::readn:
   Syntax error reading a real number from daten1.dat.
```

The reason might be that the end of lines in `daten1.dat` are not marked correctly yet. A call of the type `Run["dos2unix daten1.dat"]` might fix the problem.

Now we can read the data into a matrix `mydat` and the use `ListPlot[]` to generate an elementary graphics.

Mathematica

```
mydat=ReadList["daten1.dat",{Number,Number}];
ListPlot[mydat];
```

The file `daten2.dat` contains three columns of data. In the first column find the values of the independent variable, the second and third column contain the values of two dependent variables. We first read all the values in a matrix. This matrix is immediately transposed and then stored in `mydat`. Now we can access the second column of the original data as `mydat[[2]]`. In `dat1` and `dat2` we prepare the data to plot the two functions hidden in the data. The goal is to plot both data sets at once. These transformations are necessary to format the data for the command `ListPlot`.

Mathematica

```
Clear[mydat,dat1,dat2];
mydat=Transpose[ReadList["daten2.dat",{Number,Number,Number}]];
dat1=Transpose[{mydat[[1]],mydat[[2]]}];
dat2=Transpose[{mydat[[1]],mydat[[3]]}];
```

The package `Graphics‘MultipleListPlot‘` provides commands to combine multiple plots of lists. With the help of `RegularPolygon[]` generate regular triangles and squares as markers for the values to be plotted. Find the result in Figure 4.1 .

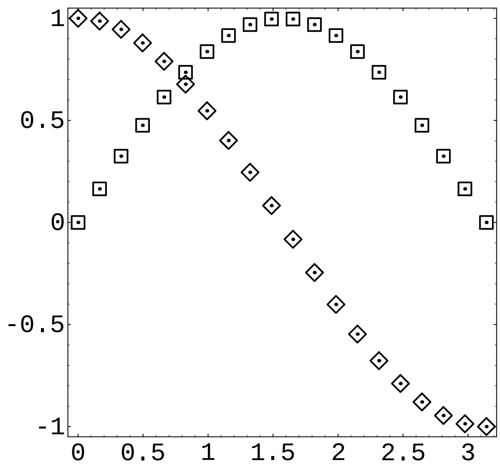


Figure 4.1: A result of `MultipleListPlot`

Mathematica

```
Needs["Graphics`MultipleListPlot`"];
triangle = RegularPolygon[3,0.020];
diamond  = RegularPolygon[4,0.020];
square   = Line[{{-0.015,-0.015},{ 0.015,-0.015},
              { 0.015, 0.015},{-0.015, 0.015},
              {-0.015,-0.015}}];
MultipleListPlot[dat1,dat2,
   DotShapes -> {MakeSymbol[square],MakeSymbol[diamond]},
   AspectRatio -> 1,
   Frame -> True,
   Axes -> False];
```

You might want to examine the result of the similar commands.

Mathematica

```
MultipleListPlot[dat1,dat2,
   DotShapes -> {MakeSymbol[triangle],MakeSymbol[diamond]},
   AxesLabel ->{"x","y"},
   PlotJoined -> True];
```

4.4 From data to 3D–graphics

4.4.1 Regular grid data

The file `daten3.dat` contains a 20×20–matrix of values of the real part of the function $\cos(x+i\,y)$ for $-\pi \leq x, y \leq 2\pi$. Thus the points will all be on this surface. The command `ListPlot3D[]` will generate a graphics using the data stored in `mydat`.

Mathematica

```
Clear[mydat];
mydat=ReadList["daten3.dat",Table[Number,{i,20}]];
ListPlot3D[mydat,MeshRange-> {{-Pi,2 Pi},{-Pi,2Pi}}];
```

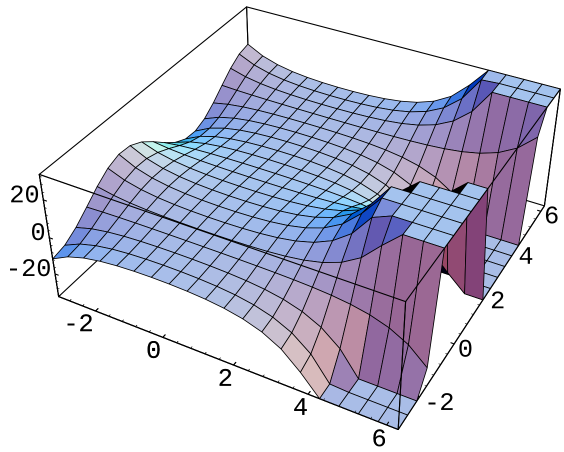


Figure 4.2: The result of `ListPlot3D`

With the above commands the domain for the variables x and y have to be typed in. It is preferable to store them in the file too and then extract them with an appropriate `Read[]` command. We will examine a new *Mathematica* function `PlotFile3D`. The code is taken from [Bahd95]. Find the code in the file `../PlotFile3D.m` in the usual place. The format of the data in the file is shown in Table 4.1. The file `daten4.dat` contains sample data in the correct form. `PlotFile3D` is the first complete function we write „on our own“. A few features are show in this function.

- The definition of a function can be stored in a file and then be loaded with `<<filename`. While testing a function you can also put code and documentation in an input cell in *Mathematica* and only move it to a file when the code is bug free.
- On–line documentation for that function can be put in the same file as the command. With `f::usage = "text"` define the usage message for the function `f`.
- With `fname_String` we force *Mathematica* to verify that the first argument of the function is a `String`, the file name in double quotes. This is a simple version of type checking.
- The variable `opts___` (three underscores) indicates that zero, one ore more optional arguments can be given. In this example they will be passed on to the plot command whose name is given as `plotType`.
- Functions with local variables can be created with `Module[]`. The first argument of `Module` is the list of local variables. The second argument is a sequence of commands, separated by semicolons. The result of the function is given by the result of the last command in that sequence. In this example it is the plot command. This might not be the case if the command `Return[]` is reached.
- It should not be too hard to find out how a file is opened for reading, different types of information are read and then the file is closed.

If the file `PlotFile3D.m` is in a directory known to *Mathematica* (see the variable `$Path`) the you can load the command with `<<PlotFile3D.m`. The on–line documentation of the command can then be accessed by `?PlotFile3D`

Mathematica

```
<<PlotFile3D.m
?PlotFile3D
```

Mathematica

```
PlotFile3D::usage = "PlotFile3D[ fname, plotType] reads data from the file
     fname, which must be given as a string,and creates a plot of type
     plotType, which must be either ListPlot3D, ListContourPlot, or
     ListDensityPlot. PlotFile3D[ fname, plotType, opts]
     passes options opts directly to plotType. The file fname must be in
     a specific format."

PlotFile3D[ fname_String, plotType_, opts___] :=
   Module[ {file, xmin, xmax, ymin, ymax, ny, nx, datamatrix},
             file = OpenRead[fname];
             If[ file === $Failed, Return[$Failed] ];
             {xmin, xmax, ymin, ymax} =
                 Read[ file,{Number,Number,Number,Number}];
             {ny, nx} = Read[ file, {Number, Number}];
             datamatrix = Partition[ ReadList[file, Number], nx];
             Close[file];
             plotType[datamatrix,
                 MeshRange -> {{xmin,xmax}, {ymin,ymax}}, opts]
   ]
```

$$
\begin{array}{llll}
x_{min} & x_{max} & y_{min} & y_{max} \\
n_x & n_y & & \\
z_{11} & & & \\
z_{21} & & & \\
z_{31} & & & \\
\vdots & & & \\
z_{n_x1} & & & \\
z_{12} & & & \\
z_{22} & & & \\
\vdots & & & \\
z_{n_x2} & & & \\
\vdots & & & \\
z_{1n_y} & & & \\
z_{2n_y} & & & \\
\vdots & & & \\
z_{n_xn_y} & & &
\end{array}
$$

Table 4.1: Code and format of the data for `PlotFile3D`

Read the data in file **daten4.dat** and generate Figure 4.3.

Mathematica

```
PlotFile3D["daten4.dat",ListPlot3D];
```

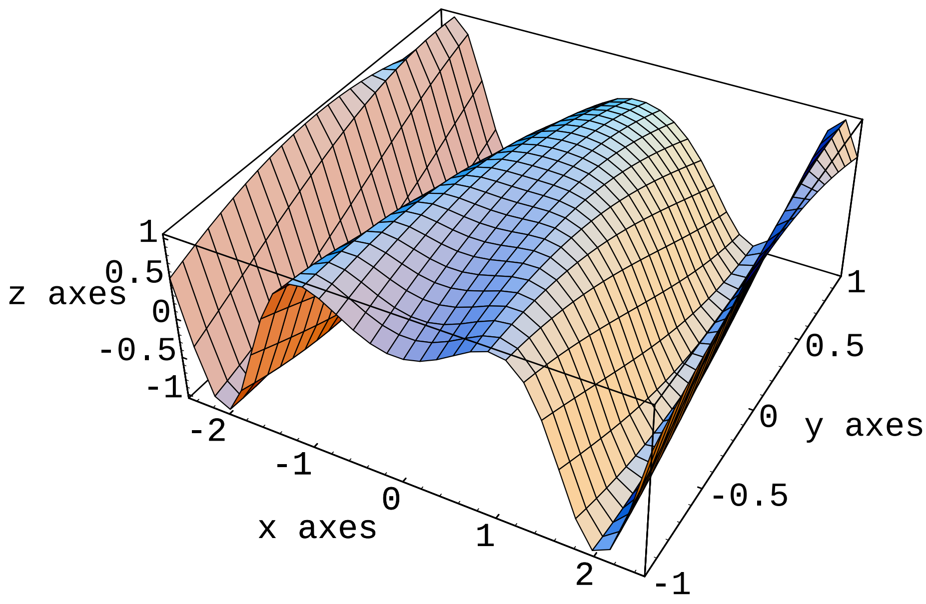


Figure 4.3: A result of `PlotFile3D`

In [Bahd95] you can also find information on how to read and display date give on a non uniform grid.

4.5 Basic input and output

Mathematica has all the basic input and output commands of a programming language. Find some to the commands to read from and write to files in Table 4.2.

`OpenRead[file]`	opens the file `file` for reading; returns an `InputStream` object; returns `$Failed` if file is not opened
`Read[stream]`	read an expression from `stream`
`Read[stream,type]`	read an object of given `type` from `stream`
`Read[stream,{type1,type2,...}]`	read objects of given types from `stream`
`ReadList[stream]`	reads all the remaining expressions in a file, and returns a list of them
`ReadList[stream, type]`	reads objects of the specified type from a file, until the end of the file is reached
`ReadList[stream, type1,type2, ...]`	reads objects with a sequence of types, until the end of the file is reached
`ReadList[stream,type,n]`	read `n` objects of given `type` from `stream`
`OpenWrite[file]`	opens the file `file` for writing; returns an `OutputStream` object
`OpenAppend[file]`	opens the file `file` for writing; returns an `OutputStream` object
`Write[stream,expr1,expr2,...]`	write the expressions to `stream` followed by a newline character
`WriteString[stream,expr1,expr2,...]`	converts the expressions strings, then write them to `stream` followed by a newline character
`Close[stream]`	close `stream`
`Streams[]`	list all currently open streams
`SetStreamPosition[stream,0]`	set the point to begin of `stream`
`SetStreamPosition[stream,Infinity]`	set the point to end of `stream`
`Skip[stream,type,n]`	skip over the next `n` objects of the given `type` in `stream`

Table 4.2: Input and output commands

4.5.1 `Read` and `ReadList`

Those two commands form the basis for all reading operations. You can read data of a given type from a given `stream`. In Table 4.3 find the basic type of data to be read. In Table 4.4 find options and their default values for the commands `Read` and `ReadList`. With a bit of programming practice it should be clear that most input and output problems can be solved with the commands at disposition. But the solutions are not always obvious. Thus we carefully walk through one example.

`Number`	exact or approximate number, possibly with `..e..` or `..E..` in exponential form
`Real`	approximate real number, possibly with `..e..` or `..E..` in exponential form
`String`	string terminated by a newline character
`Character`	single character, returned as one–character string
`Word`	string of characters delimited by `WordSeparators`
`Record`	string of characters delimited by `RecordSeparators`
`Expression`	a complete *Mathematica* expression, a newline character after a valid expression separates expressions
`Byte`	single byte returned as integer code

Table 4.3: Types for input and output

Option	Default	description
`WordSeparators`	`{" ","\t"}`	separators for words
`RecordSeparators`	`{"\n"}`	separators for words
`NullWords`	`False`	keep zero length words
`NullRecords`	`False`	keep zero length records
`RecordLists`	`False`	make separate list for each record

Table 4.4: Options for `Read[]`

First consider a file `daten6.dat` with the content shown below.

Mathematica

```
!!daten6.dat
.
Here is a polynomial:
1 + 2x + 3 x^2

Here is a matrix:
1 2 3
4 5 6
```

Now we do the following

1. Open the stream with `OpenStream[]`.
2. Read the first line as a string.
3. The definition of this polynomial is to be read as an `Expression`.
4. When reading the `expression` the end of line marker is not read, thus read it separately
5. Read the empty line.
6. Read the second line of text as `String`
7. Read the coefficients of the matrix.
8. Close the stream.

Mathematica

```
mystream=OpenRead["daten6.dat"];
line1=Read[mystream,String];
p[x_] = Read[mystream,Expression];
Read[mystream,Character];
Read[mystream,Character];
line2=Read[mystream,String];
mymatrix=ReadList[mystream,Number,RecordLists ->True];
Close[mystream];
```

4.5.2 Elementary output to a file

In the very elementary example below a few expressions are written to a file `tt.dat`.

Mathematica

```
file="tt.dat";
expr1= 2.2345;
expr2=Sin[2];
NL =  "\n";
stream=OpenWrite[file];
WriteString[stream,expr1,NL,expr2," = ",N[expr2,3]];
Close[stream];
!!tt.dat
.
2.2345
Sin[2] = 0.909
```

Mathematica has also a few commands to format numbers for output. The following example speaks for it self. But observe that most of the result can not be read by other programming languages since the exponents are on a separate line. It would be a rather tedious task to read those numbers correctly.

Mathematica

```
file="tt.dat";
expr=N[Sin[2]];
NL = "\n";
stream=OpenWrite[file];
WriteString[stream,
   "ScientificForm: \n",ScientificForm[expr],NL,NL,
   "ScientificForm: \n",ScientificForm[expr,4],NL,NL,
   "EngineeringForm: \n",EngineeringForm[expr],NL,NL,
   "NumberForm: \n",NumberForm[expr],NL,NL
   ]
Close[stream];
!!tt.dat
.
ScientificForm:
          -1
9.09297 10

ScientificForm:
        -1
9.093 10

EngineeringForm:
          -3
909.297 10

NumberForm:
0.909297
```

Since the above results are not suitable for reading by other programs [Bahd95] wrote two packages to resolve the problem. Find the code for them in the usual place in the files `FortranWrite.m` and `WriteArray.m`. If you plan to work with them, they have to be loaded first.

Mathematica

```
<<FortranWrite.m
<<WriteArray.m
```

`FortranWrite[]` will write numbers in a format readable by FORTRAN or C.

Mathematica

```
myMatrix={{1.0,2.0,3},{4.0,5,6}};
FortranWrite["testfile.dat",myMatrix];
!!testfile.dat
DeleteFile["testfile.dat"]
.
 1.000000e+00   2.000000e+00   3.000000e+00
 4.000000e+00   5.000000e+00   6.000000e+00
```

With `WriteArray[]` produce results that can be pasted into other programs and used there.

Mathematica

```
WriteArray[mymatrix,a,"testfile.dat"];
!!testfile.dat
DeleteFile["testfile.dat"]
.
a(1,1)=1
a(1,2)=2
a(1,3)=3
a(2,1)=4
a(2,2)=5
a(2,3)=6
```

Mathematica

```
f[x_] = Cos[2x];
WriteArray[Table[D[f[x],{x,n}],{n,0,3}],a,"testfile.dat"];
!!testfile.dat
DeleteFile["testfile.dat"]
.
a(1)=Cos(2*x)
a(2)=-2*Sin(2*x)
a(3)=-4*Cos(2*x)
a(4)=8*Sin(2*x)
```

The source code of `WriteArray` contains a few more examples hidden as comments in the file.

Chapter 5

Some mathematical operations

5.1 Derivatives and integrals

Mathematica is a extremely powerful tool to compute derivatives and integrals. Some of the basic commands are listed in Table 5.1.

`D[expr,x]`	partial derivative with respect to `x`
`D[expr,x,y]`	second order partial derivative respect to `x` and `y`
`D[expr,{x,n}]`	partial derivative of order `n` with respect to `x`
`Dt[expr,x]`	total derivative of `expr` with respect to `x`
`Integrate[expr,x]`	indefinite integral of `expr` with respect to `x`
`Integrate[expr,{x,a,b}]`	definite integral of `expr` with respect to `x`
	where $a \leq x \leq b$
`NIntegrate[expr,{x,a,b}]`	numerical evaluation of definite integral of `expr`
	with respect to `x` where $a \leq x \leq b$

Table 5.1: Derivatives and integrals

5.1.1 Elementary derivatives

Computing derivatives comes down to apply the known computational rules carefully, thus *Mathematica* is very powerful and reliable at it. Consider the following simple examples. Observe the difference between total and partial derivatives.

Mathematica

```
Clear[expr]
expr[x_,y_] := Sin[3 x + y] E^(2x)
```

Mathematica

```
D[expr[r,t],r]
.
   2 r                   2 r
3 E    Cos[3 r + t] + 2 E    Sin[3 r + t]
```

Mathematica

```
D[expr[r,t],t]
.
 2 r
E    Cos[3 r + t]
```

Mathematica

```
D[expr[r,t],{r,4}]
.
      2 r                     2 r
-120 E    Cos[3 r + t] - 119 E    Sin[3 r + t]
```

Mathematica

```
D[expr[x,y],{x,2},{y,20}]
.
    2 x                   2 x
12 E    Cos[3 x + y] - 5 E    Sin[3 x + y]
```

Now a few total derivatives.

Mathematica

```
D[expr[2t,t],t]
.
   4 t               4 t
7 E    Cos[7 t] + 4 E    Sin[7 t]
```

Mathematica

```
Dt[expr[x,t],t]
.
 2 x                                     2 x
E    Cos[t + 3 x] (1 + 3 Dt[x, t]) +   2 E    Dt[x, t] Sin[t + 3 x]
```

Mathematica

```
Dt[expr[x,t],t] /. x -> 2t
.
   4 t               4 t
7 E    Cos[7 t] + 4 E    Sin[7 t]
```

5.1.2 Derivatives of special functions

Mathematica does know about many special functions and their properties. Thus it may well serve as a table with such information and thus replace books like [AbraSteg]. Those that know how to read tables of special functions will also find the derivative of the Bessel function below in [AbraSteg].

Mathematica

```
D[BesselJ[1,x],x]
.
BesselJ[0, x] - BesselJ[2, x]
-----------------------------
              2
```

5.1.3 How to compute the gradient of a given function

To find the gradient of a function $f(x, y, z)$ we have to compute three partial derivatives.

$$\text{grad}\, f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial z}\right)$$

Thus it is easy to define a function `grad[]` which will do just this.

Mathematica

```
Clear[grad]
grad[f_] := {D[f,x],D[f,y],D[f,z]}
```

Applied to a simple function we get the desired result.

Mathematica

```
grad[2 x + y^2 + Cos[z]]
{2, 2 y, -Sin[z]}
```

Unfortunately this construction applies only to functions of three variables in cartesian coordinates. The variables have to be called x, y and z. This is not a flexible solution.

With the help of `Outer` we can give a construction of `grad` which will apply to a function with arbitrary many variables and we can choose the names of the variables. Her first an illustrative example of the command `Outer`.

Mathematica

```
Outer[f,{x,y,z},{a,b}]
.
{{f[x, a], f[x, b]},
 {f[y, a], f[y, b]},
 {f[z, a], f[z, b]}}
```

Now the improved definition of `grad`.

Mathematica

```
Clear[grad]
grad[func_,var_] := Flatten[Outer[D,{func},var]]
grad[Sin[x] + y,{x,y}]
.
{Cos[x], 1}
```

with an example

Mathematica

```
grad[2 x + y^2 + Cos[z],{x,y,z}]
.
{2, 2 y, -Sin[z]}
```

Mathematica has a package `Calculus‘VectorAnalysis‘` to compute gradients and related expressions. If you need of such operations often it is worth to learn how to use it.

Mathematica

```
Needs["Calculus‘VectorAnalysis‘"]
?Grad
.
Grad[f] gives the gradient of the scalar function f in
   the default coordinate system. Grad[f, coordsys]
   gives the gradient of f in the coordinate system
   coordsys.
```

5.1.4 Elementary integrals

Most scientists and engineers will be able to find the following result by hand too.

Mathematica

```
Integrate[x^3 -Cos[3 x],x]
.
   4
3 x  - 4 Sin[3 x]
-----------------
       12
```

The next example is either memorised or to be found in a table.

Mathematica

```
Integrate[1/Sqrt[1-x^2],x]
.
ArcSin[x]
```

The following would take a few minutes or a table.

Mathematica

```
Integrate[x^3 Sin[x],{x,0,Pi}]
.
          3
-6 Pi + Pi
```

The exact formula for the next integral is rather lengthy, but the numerical value is easy to compute with *Mathematica*.

Mathematica

```
NIntegrate[Sin[x]^2/(1+x^2),{x,0,Pi}]
.
0.530605
```

Mathematica will look for indefinite integrals, without bothering with domains of definition. The following result is only correct if $x > 0$. The correct result would be

$$\int \frac{1}{x}\, dx = \log |x| + C$$

Mathematica

```
Integrate[1/x,x]
.
Log[x]
```

5.2 Algebraic operations

With *Mathematica* you can perform algebraic operations on rather complex expressions.

`Expand[poly]`	multiply out `poly`
`ExpandAll[poly]`	multiply out any factors and integer powers `poly`
`Expand[poly,pattern]`	multiply out `poly`, avoid expanding which do not contain term matching `pattern`
`ExpandAll[poly,pattern]`	multiply out `poly`, avoid expanding `expr` which do not contain term matching `pattern`
`Factor[poly]`	factor `ploy` over the integers
`FactorTerms[poly]`	pull out any numerical factor in `poly`
`Collect[poly,x]`	collect terms in `poly` with like powers of `x`
`PolynomialQ[expr,x]`	test if `expr` is a polynomial with respect to `x`
`Variables[ploy]`	give a list of variables in `poly`
`Exponent[poly,x]`	highest order of `x` in `poly`
`Coefficient[poly,x]`	coefficient of `x` in `poly`
`PolynomialDivision[p, q, x]`	divide the polynomial `p` by the polynomial `q` with variable `x`
Note: in these commands `poly` must be in expanded form, using `Expand[poly]`	

Table 5.2: Algebraic operations on polynomials

The effects of the commands in Table 5.2 are best examined by running a few examples.

Mathematica

```
Expand[ (x + 2)^3 (x -y x + 2)]
.
                    2       3    4                2         3       4
16 + 32 x + 24 x   + 8 x   + x   - 8 x y - 12 x   y -  6 x   y - x   y
```

Mathematica

```
Expand[ (x + 2)^3 (x -y x + 2),y ]
.
           3             3             3
2 (2 + x)   + x (2 + x)   - x (2 + x)   y
```

Mathematica

```
Collect[(x + 2)^3 (x -y x + 2),x]
.
      2                              3              4
16 + x  (24 - 12 y) + x (32 - 8 y) + x  (8 - 6 y) +   x  (1 - y)
```

Mathematica

```
{PolynomialQ[ (x- Sin[ y])^3,x],
 PolynomialQ[ (x- Sin[ y])^3,y]}
.
{True, False}
```

Mathematica

```
Variables[(x- a Sin[ y])^3 (x+6)^100]
.
{a, x, Sin[y]}
```

Mathematica

```
Exponent[(x- a Sin[ y])^3 (x+6)^100,x]
.
103
```

Mathematica

```
Coefficient[Expand[(x- a Sin[ y])^3 (x+6)^2],Sin[y]]
.
        2         3        4
-108 a x  - 36 a x  - 3 a x
```

Some of the operations in Table 5.2 apply not only to polynomials, but to general expressions. *Mathematica* has many more commands to rearrange and transform expressions. It is impossible to give a complete survey with the given space and time. A few more are given in Table 5.3.

The most dangerous commands in Table 5.3 are by far `Simplify` and `FullSimplify`. It is by no means clear what the simplest form of an expression is. It depends on what should be done later on with the result. Usually it is a better idea to tell *Mathematica* what kind of transformation we are looking for, i.e. use one of the other commands. Nonetheless `Simplify` is very useful to find elementary simplifications in many cases.

5.3 Solving equations

Mathematica has many commands to solve a wide variety of types of equations. The most important point is to choose the command appropriate for the problem at hand. Table 5.4 might be useful.

All command in Table 5.4 have many `Options` to control the behaviour of the solution method.

`Expand[expr]`	multiply out `expr`
`ExpandAll[expr]`	multiply out any factors and integer powers `expr`
`Factor[expr]`	try to factor `expr`
`Together[expr]`	write `expr` as a single fraction
`Apart[expr]`	multiple fractions with simple denominators
`Apart[expr,vars]`	treat all variable other than `vars` as constants
`Expand[expr, Trig -> True]`	multiply out trigonometric expressions
	e.g. $\sin^2 x$ is rewritten in terms of $\cos 2x$
`Factor[expr, Trig -> True]`	factor trigonometric expressions
	e.g. $\sin 2x$ is rewritten as $2 \sin x \cos x$
`Factor[expr, Trig -> False]`	factor trigonometric expressions
	but the trigonometric are treated as indivisible objects
`ComplexExpand[expr]`	expand `expr`, assuming that all variables are real
`PowerExpand[expr]`	transform $(x\,y)^n$ in $x^n\,y^n$
`Simplify[expr]`	as the name says!
`FullSimplify[expr]`	try harder to simplify

Table 5.3: Transformation of terms

`Solve[]`	polynomial and and linear equations
`NSolve[]`	numerical solutions
`Eliminate[eq,vars]`	eliminate the variables `var` in the equation `eq`
`Roots[]`	polynomial equations
`NRoots[]`	numerical solutions
`LinearSolve[]`	systems of linear equations
`FindRoot[]`	numerical solution methods
	Newton's or secant method
`FindMinimum[]`	find the minimum of a function, numerical method
`DSolve[]`	exact solution of a differential equation
`NDSolve[]`	numerical solution of a differential equation

Table 5.4: Solution methods for equations

5.3.1 Algebraic equations, `Solve`, `Roots`, `Eliminate`

`Solve`, `Roots`

A polynomial equation in the variables x and y contains expressions of the form

$$x, x^2, x^3, \ldots \quad , \quad y, y^2, y^3, \ldots \quad , \quad x\,y, x^2y, x\,y^2, x^2y^2 \ldots$$

To rediscover the well known formula for the solutions of a quadratic equation

$$a\,x^2 + b\,x + c = 0$$

use the command `Solve[eqn,var]`.

Mathematica

```
Solve[a x^2 + b x + c==0,x]
.
                  2
       -b - Sqrt[b  - 4 a c]
{{x -> ---------------------},
                2 a

                   2
        -b + Sqrt[b  - 4 a c]
  {x -> ---------------------}}
                 2 a
```

or

Mathematica

```
Roots[a x^2 + b x + c==0,x]
.
                2                              2
     -b - Sqrt[b  - 4 a c]          -b + Sqrt[b  - 4 a c]
x == --------------------- || x == ---------------------
              2 a                            2 a
```

The only difference is the format of the solutions.

`Solve[]`	returns replacement rules
	may also be used for systems of equations
`Roots[]`	returns simpler equations
	can only be used for one polynomial equation
	uses `Factor[]` and `Decompose[]`

To find the points of intersection of the circle $x^2 + y^2 = 4$ with the parabola $y^2 = -x$ you can use `Solve[]`. First define the two equations as `eqn`, then solve them and store the solutions in `sol`.

Mathematica

```
Clear[eqn,sol]
eqn = {x^2+y^2==4, y^2-x==0};
sol = Solve[eqn,{x,y}]
.
       -1 - Sqrt[17]          Sqrt[-1 - Sqrt[17]]
{{x -> -------------, y -> -(-------------------)},
             2                      Sqrt[2]

        -1 - Sqrt[17]       Sqrt[-1 - Sqrt[17]]
  {x -> -------------, y -> -------------------},
              2                   Sqrt[2]

        -1 + Sqrt[17]         Sqrt[-1 + Sqrt[17]]
  {x -> -------------, y -> -(-------------------)},
              2                     Sqrt[2]

        -1 + Sqrt[17]       Sqrt[-1 + Sqrt[17]]
  {x -> -------------, y -> -------------------}}
              2                   Sqrt[2]
```

Surprisingly *Mathematica* returns four solutions, instead of the expected two. Verify that the given expression do actually solve the equations. Compute the numerical values of the solutions to discover that two of the solutions are complex.

Mathematica

```
Simplify[eqn /. sol]
.
{{True, True}, {True, True}, {True, True}, {True, True}}
```

Mathematica

```
N[sol]
.
{{x -> -2.56155, y -> 0. - 1.60049 I},

  {x -> -2.56155, y -> 0. + 1.60049 I},

  {x -> 1.56155, y -> -1.24962},

  {x -> 1.56155, y -> 1.24962}}
```

Both commands `Solve[]` and `Roots[]` try to find exact solution wherever possible. As a difficult example we construct a polynomial of degree 30 with known zeros.

Mathematica

```
badpoly = Expand[Product[(x- n), {n,1,30}]]
```

Now find the exact solutions

Mathematica

```
Solve[badpoly==0 ,x]
{{x -> 1}, {x -> 2}, {x -> 3}, {x -> 4}, {x -> 5},
  {x -> 6}, {x -> 7}, {x -> 8}, {x -> 9}, {x -> 10},
  {x -> 11}, {x -> 12}, {x -> 13}, {x -> 14},
  {x -> 15}, {x -> 16}, {x -> 17}, {x -> 18},
  {x -> 19}, {x -> 20}, {x -> 21}, {x -> 22},
  {x -> 23}, {x -> 24}, {x -> 25}, {x -> 26},
  {x -> 27}, {x -> 28}, {x -> 29}, {x -> 30}}
```

If we replace 0 by 0.0 then *Mathematica* is forced to use numerical methods. The deviations from the correct values are surprisingly large.

Mathematica

```
Solve[badpoly==0.0 ,x]
.
{{x -> 0.999999999999991}, {x -> 1.999999999992566},
  {x -> 3.000000001961303}, {x -> 3.999999817073554},
  {x -> 5.000009198182413}, {x -> 5.999708772009817},
  {x -> 7.0063595043429}, {x -> 7.922112440012595},
  {x -> 9.1054166961055},
  {x -> 9.53358535642172 - 0.991759096334617 I},
  {x -> 9.53358535642172 + 0.991759096334617 I},
  {x -> 11.34305333327745 - 2.706353487980411 I},
  {x -> 11.34305333327745 + 2.706353487980411 I},
  {x -> 12.13698443824903},
  {x -> 13.64965155436332 - 4.249309995444764 I},
  {x -> 13.64965155436332 + 4.249309995444764 I},
  {x -> 16.37397745006214 - 5.409178339167666 I},
  {x -> 16.37397745006214 + 5.409178339167666 I},
  {x -> 18.66833166334549},
  {x -> 19.39950754731527 - 6.14178287114558 I},
  {x -> 19.39950754731527 + 6.14178287114558 I},
  {x -> 22.60354525576519 - 6.372044889524747 I},
  {x -> 22.60354525576519 + 6.372044889524747 I},
  {x -> 25.85469884287421 - 5.915790673397 I},
  {x -> 25.85469884287421 + 5.915790673397 I},
  {x -> 28.83924464760501 - 4.610794814435022 I},
  {x -> 28.83924464760501 + 4.610794814435022 I},
  {x -> 31.04180471574195}, {x -> 31.04180471574195},
  {x -> 31.88294006187226}}
```

Those deviations are not due to a problem within *Mathematica*. All other software packages will run into the same problem. There are equations where it is impossible to find accurate solutions with numerical methods. The above problem is the most famous problem of this type.

With Solve[]or Eliminate[] eliminate one, or more variables from a system of equations.

Mathematica

```
Clear[eqn]
eqn = {x^2+y^2==r^2, y^2-x==0};
Solve[eqn,x,y]
.
                          2                            2
       -1 - Sqrt[1 + 4 r ]          -1 + Sqrt[1 + 4 r ]
{{x -> -------------------}, {x -> -------------------}}
                2                            2
```

Mathematica

```
Eliminate[eqn,y]
.
 2
r  == x (1 + x)
```

The command Solve[] can **not** find solutions to transcendental equations. In some cases it will return a result but it should be verified by independent computations. Most often a warning message is issued.

Mathematica

```
Solve[Tan[x] == 2x^2 - 2x,x ]
.
Solve::tdep:
   The equations appear to involve transcendental
     functions of the variables in an essentially
     non-algebraic way.
.
                            2
Solve[Tan[x] == -2 x + 2 x , x]
```

Mathematica

```
Solve[Sin[x] + y/(1+y)==0,x ]
.
Solve::ifun:
   Warning: Inverse functions are being used by Solve,
     so some solutions may not be found.
.
                 y
{{x -> -ArcSin[-----]}}
               1 + y
```

Roots[] can only solve polynomial equations. For other types of equations you might see surprising answers.

Mathematica

```
Roots[Tan[x] == 2x^2 - 2x,x ]
.
     2 - Sqrt[4 + 8 Tan[x]]
x == ---------------------- ||
               4

     2 + Sqrt[4 + 8 Tan[x]]
x == ----------------------
               4
```

Mathematica

```
Roots[Sin[x] == x/(1+x),x ]
.
Roots::neq:
       x
   -(-----) + Sin[x] is expected to be a polynomial
     1 + x
     equation in the variable x.
.
                  x
Roots[Sin[x] == -----, x]
                1 + x
```

5.3.2 Numerical solutions, NSolve, NRoots

Those two commands are comparable to the commands in the previous section, but they generate numerical answers. It is easy to give a problem for which exact solutions are not known. NSolve[] does apply to systems of polynomial equations too [1].

For the equation

$$x^5 - x^3 + 2x - 13 = 0$$

it is impossible to give exact solutions. Nonetheless *Mathematica* will give reliable approximative solutions.

Mathematica

```
Solve[ x^5 - x^3 + 2 x - 13==0,x ]
.
                       3    5
{ToRules[Roots[2 x - x  + x  == 13, x]]}
```

Mathematica

```
NSolve[ x^5 - x^3 + 2 x - 13==0,x ]
.
{{x -> -1.47254 - 0.981677 I},
 {x -> -1.47254 + 0.981677 I},
 {x -> 0.618106 - 1.43068 I},
 {x -> 0.618106 + 1.43068 I},
 {x -> 1.70886}}
```

[1]The commands NSolve[] and NRoots[] seems to use a mixture of exact and numerical methods. The command FindRoot[] uses numerical methods only.

Mathematica

```
Roots[ x^5 - x^3 + 2 x - 13==0,x ]
.
                    3    5
Roots[-13 + 2 x - x  + x  == 0, x]
```

Mathematica

```
NRoots[ x^5 - x^3 + 2 x - 13==0,x ]
.
x == -1.47254 - 0.981677 I ||
  x == -1.47254 + 0.981677 I ||
  x == 0.618106 - 1.43068 I ||
  x == 0.618106 + 1.43068 I || x == 1.70886
```

But there are problems for which one can not get reliable numerical solutions either. An example was given in the previous section.

Mathematica

```
NSolve[badpoly==0.0,x]
NRoots[badpoly==0.0,x]
```

5.3.3 Systems of linear equations, `LinearSolve`, `NullSpace`

There are specials algorithms to solve systems of linear equations. *Mathematica* does know how to apply them, if you tell it that the equations at hand are in fact linear. To solve the system

$$\begin{aligned} 3\,x + a\,y &= 3 \\ 2\,x - 7\,y &= 0 \end{aligned}$$

you should use the command `LinearSolve`. The command `Solve` will give you an answer.

Mathematica

```
Solve[{3 x + a y ==3,2 x - 7 y ==0},{x,y}]
.
              2 a              -6
{{x -> 1 + ---------, y -> ---------}}
           -21 - 2 a       -21 - 2 a
```

The same system of equations should be rewritten with the help of matrices and vectors

$$\begin{bmatrix} 3 & a \\ 2 & -7 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$$

and then be solved by `LinearSolve[]`.

Mathematica

```
Clear[mat,const];
mat={ {3,a} ,{2 , -7} };
const={3, 0};
LinearSolve[mat,const]
.
        2 a        -6
{1 + ---------, ---------}
     -21 - 2 a  -21 - 2 a
```

Mathematica can now make use of the additional information that the system is linear.

The system

$$\begin{array}{rrrcl} 3\,x & +a\,y & +2\,z & = & 3 \\ 2\,x & -7\,y & -3\,z & = & 0 \end{array}$$

has more unknowns than equations, thus it is impossible to find a unique solution. `LinearSolve[]` will give you one possible solution., but **no** indication that there might be more solutions.

Mathematica

```
Clear[mat,const]
mat={ {3,a,2} ,{2 , -7,-3}};
const={3, 0};
LinearSolve[mat,const]
.
        2 a         -6
{1 + ---------, ---------, 0}
     -21 - 2 a  -21 - 2 a
```

The command `NullSpace[mat]` will return a list of vectors which span the null space of the given matrix `mat`). By adding multiples of those vectors to the result of `LinearSolve[]` you will obtain all solutions. If there is a unique solution, the `NullSpace[mat]` will return no solution, e.i. an empty list.

Mathematica

```
NullSpace[mat]
.
    2         13 a           13
{{-(-) - -------------, ---------, 1}}
    3    3 (-21 - 2 a)  -21 - 2 a
```

5.3.4 Transcendental equations, `FindRoot`

To search for solutions to transcendental equations the **method of Newton** is known to be very efficient. *Mathematica* uses this method with the command `FindRoot`. As an example consider the function

$$f\,(x) = \cos x - x/10$$

whose graph you find in Figure 5.1. First define this function in *Mathematica*.

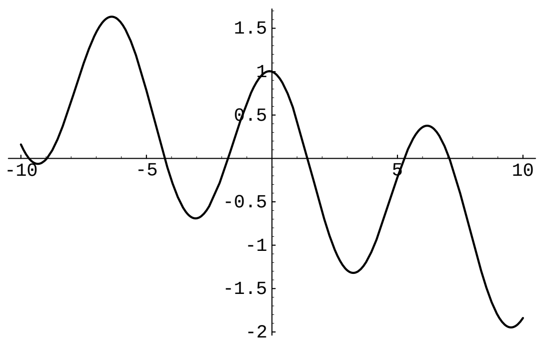


Figure 5.1: Graph of the function $\cos x - x/10$

Mathematica

```
func[x_] := Cos[x]-0.1 x
fig1=Plot[func[x],{x,-10,10}];
```

Then look for zeros close to the values $x = 1$ with the command `FindRoot`.

Mathematica

```
FindRoot[func[x]==0,{x,1}]
.
{x -> 1.42755}
```

The main drawback of the method of Newton is the fact, that the initial guess has to be close enough to the actual solution. Otherwise a number of undesired effect might happen. The example below does not produce the solution closest to 0, even with the initial values $x = 0$. This is illustrated by Figure 5.1.

Mathematica

```
FindRoot[func[x]==0,{x,0}]
.
{x -> 5.26712}
```

The method of Newton can not be used if the function in question can not be differentiated. Situations like this can occur if you function is constructed by a numerical method.

Different methods used by the command `FindRoot` are shown in Table 5.5.

`FindRoot[lhs==rhs, {x,x0}]`	one equation, Newton's method
`FindRoot[lhs==rhs, {x,x0,x1}]`	one equation, secant method
`FindRoot[lhs==rhs, {x,x0,x1,x2}]`	one equation, secant method
	terminate if outside of $[x1, x2]$
`FindRoot[{eqn1, eqn2}, {x,x0},{y,y0}]`	two equation, Newton's method
`FindRoot[{eqn1, eqn2}, {x,x0,x1},{y,y0,y1}]`	two equation, secant method

Table 5.5: Methods used by `FindRoot[]`

5.3.5 Extermal problems, `FindMinimum`

Mathematica does have a special command to find minima of functions of one ore more variables, `FindMinumum`.

Mathematica

```
FindMinimum[(x-2.5)^2+ (y+Pi)^2+1 ,{x,1},{y,1}]
.
{1., {x -> 2.5, y -> -3.14159}}
```

5.4 Solutions to differential equations

5.4.1 DSolve

With the help of `DSolve[]` you can search for solutions of ordinary differential equations. The result will be given in the form of replacement rules. The differential equation

$$\ddot{y}(t) = -4\,y(t) + t^2$$

is solved by

Mathematica

```
DSolve[D[y[t],{t,2}] == -4 y[t] + t^2 ,y[t],t]
                    2
            1     t
{{y[t] -> -(-) + -- + C[2] Cos[2 t] - C[1] Sin[2 t]}}
            8     4
```

Thus the general solution is

$$y(t) = \frac{-1}{8} + \frac{t^2}{4} + c_2 \cos(2t) - c_1 \sin(2t)$$

The constants c_1 and c_2 will have to be determined by initial conditions. If we add the initial conditions

$$y(0) = 0 \quad \text{and} \quad \dot{y}(0) = 1$$

the the equation has a unique solution whose graph you find in Figure 5.2.

Mathematica

```
sol=DSolve[{D[y[t],{t,2}] == -4 y[t] + t^2 ,
          y[0]==0,y'[0]==1},
          y[t],t]
fig3=Plot[y[t] /. sol ,{t,0,2}];
.
                    2
            1     t     Cos[2 t]    Sin[2 t]
{{y[t] -> -(-) + -- + -------- + --------}}
            8     4        8           2
```

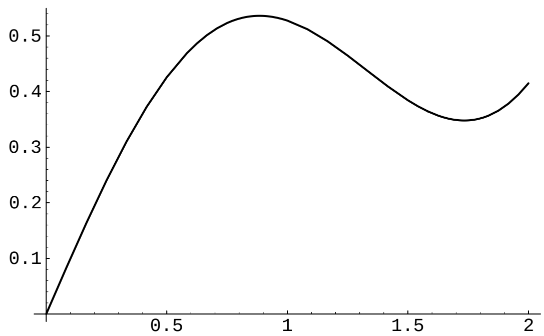


Figure 5.2: Solution of a differential equation

Many equations can not be solved by `DSolve[]` and one has to resort to numerical methods, i.e. `NDSolve[]`.

Mathematica

```
DSolve[ y''[t] == -Sin[y[t]] , y[t],t]

DSolve::dnim:
   Built-in procedures cannot solve this differential
     equation.
.
DSolve[y''[t] == -Sin[y[t]], y[t], t]
```

5.4.2 NDSolve

To get a numerical approximation of the differential equation

$$\ddot{y} = -\sin y \quad \text{with} \quad y(0) = 0 \quad \text{and} \quad \dot{y}(0) = 1$$

use

Mathematica

```
diffeq = {y''[t] == -Sin[y[t]]};
initval = {y[0]==0, y'[0]==1};
nsol= NDSolve[ Join[diffeq,initval] , y ,{t,0,5}]
.
{{y -> InterpolatingFunction[{0., 5.}, <>][t]}}
```

The Figure 5.3 is generated by

Mathematica

```
Plot[y[t] /. nsol ,{t,0,5}];
```

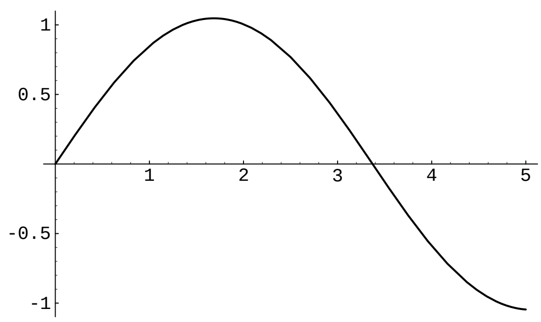


Figure 5.3: Numerical solution of a differential equation

You can do further calculations with this solution. Let us search a zero of the solution with `FindRoot[]`.

Mathematica

```
Clear[f,y];
f[t_] = (y[t] /. nsol[[1]])
FindRoot[f[t] == 0 ,{t,3}]
.
InterpolatingFunction[{0., 5.}, <>][t]
{t -> 3.3715}
```

Chapter 6

Simple programming

6.1 Loops and such

The classic loop constructions `Do` and `While` are also available in *Mathematica*. It is often considered bad (*Mathematica*) style to use them. The command `Table` is implemented more efficiently in *Mathematica* and should be preferred. There are also many other ways to operate on lists.

6.1.1 `While`

`While[test, body]` evaluates test, then body, repetitively, until test first fails to give `True`. If `Break[]` is generated in the evaluation of body, the `While` loop exits. Most often a `While` loop will not return a result, but with *Mathematica* it can return a result. Unless `Return[]` or `Throw[]` are generated, the final value returned by `While` is `Null`.

The loop below does attempt to print all natural numbers whose square is larger than 200. But it will quit if the square is exactly equals to 16.

Mathematica

```
j=1;
While[j^2<200, If[j^2==16,Break[]];Print[j];j++]
```

The `While` loop below will test all numbers of the form $j^2 + j + 41$ if they are prime. The first non-prime will be returned as a result, but the loop will stop at $j = 100$.

Mathematica

```
j=1;
res=While[j<100,If[PrimeQ[j^2+j+41]==False,Return[j]];j++]
```

6.1.2 The classic `Do[] loop`

It should be very easy to find examples for this command.

Mathematica

```
Do[Print[j],{j,0,7,2}]
Do[Print[10*i+j],{i,4},{j,2,5}]
```

You can use `Return`, `Break`, `Continue` and `Throw` inside a `Do` loop.

6.1.3 `For[]` is for C programmers

For those familiar with C there is an implementation of `For[]`.

`Do[expr, {imax}]`	evaluates expr imax times.
`Do[expr, {i, imax}]`	evaluates expr with the variable i successively taking on the values 1 through imax (in steps of 1).
`Do[expr, {i, imin, imax}]`	starts with i = imin.
`Do[expr, {i, imin, imax, di}]`	uses steps di.
`Do[expr, {i, imin, imax}, {j, jmin, jmax}, ... ]`	evaluates expr looping over different values of j, etc. for each i.

Table 6.1: The command `Do[]`

Mathematica

```
?For
.
"For[start, test, incr, body] executes start, then repeatedly evaluates body
     and incr until test fails to give True."
```

As an example consider

Mathematica

```
For[j=-3,j<=10,j=j+2,Print[j,"->",j^2]]
.
-3 -> 9
-1 -> 1
 1 -> 1
 3 -> 9
 5 ->25
 7 ->49
 9 ->81
```

6.1.4 `Table`

Used to generate lists, documented in another chapter.

6.1.5 `Apply`, `Map`, `Listable`

The command `Apply[]` can be used to change the header of a list. Consider the internal representation of the two expressions $\{1, 2, 3\}$ and $a + b + c$ in *Mathematica* with the command `FullForm[]`

Mathematica

```
FullForm[{1,2,3}]
FullForm[a+b+c]
.
List[1,2,3]
Plus[a,b,c]
```

The first expression has the head `List`, the second the head `Plus`. If we replace `list` by `Head` in the first expression, then the sum will be computed.

Mathematica

```
Apply[Plus,{1,2,3}]
.
6
```

A function can have the attribute `Listable`. When applied to a list, the function will be evaluated for each element in the list.

Mathematica

```
??Sin

"Sin[z] gives the sine of z."
Attributes[Sin] = {Listable, NumericFunction, Protected}
```

A self defined function is not listable by default, but the attribute can be set by `SetAttributes[]`. This is illustrated below.

Mathematica

```
Remove[g1,g2]
SetAttributes[g2,Listable];
g1[{1,2,3,4,5,6}]
g2[{1,2,3,4,5,6}]
.
g1[{1,2,3,4,5,6}]
{g2[1],g2[2],g2[3],g2[4],g2[5],g2[6]}
```

A similar effect can be achieved by `Map[]`.

Mathematica

```
Remove[g]
g[{x_,y_}] = x+y;
g[{{1,2},{3,4},{5,6}}]
Map[g,{{1,2},{3,4},{5,6}}]
.
g[{{1,2},{3,4},{5,6}}]
{3,7,11}
```

6.2 Simple subprograms, `Module`

Modules can be used to define simple functions. They allow to use local variables. As an example we define a function to exchange two rows of a given matrix. With a simple implementation we also define another function

Mathematica

```
Clear[SwapRows,Pivot];
SwapRows[A_,i_,j_] :=
     Module[{tmp},tmp=A;tmp[[i]]=A[[j]];tmp[[j]]=A[[i]];tmp]

Pivot[A_,i_,j_] :=
     Module[{tmp},
                tmp=A;
                For[k=Length[A[[1]]],k>=j,k=k-1,tmp[[i,k]]=tmp[[i,k]]/tmp[[i,j]]];
                For[l=i+1,l<=Length[A],l=l+1,
```

```
            For[k=Length[A[[1]]],k>=j,k=k-1,
                   tmp[[l,k]]=tmp[[l,k]]-tmp[[l,j]]*tmp[[i,k]] ] ];
        tmp]
```

The results of the code below will now show you how to use this code to solve a system of linear equations.

Mathematica

```
(mat={{0,-2,  7,12},{2,-10,12,28},{2, -5,-5,-1}})//MatrixForm
(mat=SwapRows[mat,1,2])//MatrixForm
(mat=Pivot[mat,1,1])//MatrixForm
(mat=Pivot[mat,2,2])//MatrixForm
(mat=Pivot[mat,3,3])//MatrixForm
```

Usually it is a good idea to give some documentation of what your code should be doing. The string `SwapRows::usage=` will be given as answer by *Mathematica* to the command `?SwapRows` or to `Information[SwapRows]`

Mathematica

```
Clear[SwapRows];
SwapRows::usage =
  "SwapRows[matrix,row1,row2] does interchange the rows row1 and row2 in the \
given matrix";

SwapRows[A_,i_,j_] :=
     Module[{tmp},tmp=A;tmp[[i]]=A[[j]];tmp[[j]]=A[[i]];tmp]
```

Mathematica

```
?SwapRows

"SwapRows[matrix,row1,row2] does interchange the rows row1 and row2 in the given matrix"
```

6.3 An example: fitting data

6.3.1 Reading Data from file

Set the directory, then read the data from the file. The first row will contain the time, the second row the temperature of the bath and the third row the temperature to be examined.

Mathematica

```
SetDirectory["~sha/f97"];
ursp:=Transpose[ReadList["gruppe2.prn", {Number,Number,Number}]]
{time,bath,temperature}=ursp ;
```

6.3.2 Visual test, by plotting the data

The first few point in the data set do not contain valid information and thus have to be removed. Remove the first 6 data points, then plot the data. This first visual test is important as you will catch blatant errors.

Mathematica

```
Needs["Graphics`MultipleListPlot`"]
drops=6;
temperature=Drop[temperature,drops];
time=Drop[time,drops];
bath=Drop[bath,drops];
data1=Transpose[{time,bath}];data2=Transpose[{time,temperature}];
g1=MultipleListPlot[data1,data2,Join->True,AxesLabel->{ s ,C },
    PlotStyle->PointSize[0.02]];
```

6.3.3 Use Fit[], show the results

Compute the average temperature of the bath. Then find the new values of the temperature on the new scale. Then plot the new data, they should be on a straight line. Use Fit[] to determine the parameters of the line. Then plot both, data and the fitted straight line.

Mathematica

```
bathAverage=Apply[Plus,bath]/Length[bath]
f[temp_]= Log[bathAverage-temp]
newTemperature=f[temperature];fitData=Transpose[{time,newTemperature}];
g2=ListPlot[fitData,AxesLabel->{s,Log(f(x))}];
fitFunc=Fit[fitData,{1,x},x]
g3=Plot[fitFunc,{x,0,125},DisplayFunction -> Identity];
Show[g2,g3];
```

Now compute the original exponential function and show the graph, together with the measured data.

Mathematica

```
fitExponential[x_]=bathAverage-Exp[fitFunc]//Simplify
g4=Plot[{fitExponential[x],bathAverage},{x,10,125},
    DisplayFunction -> Identity];
Show[g4,g1,DisplayFunction -> $DisplayFunction];
```

6.3.4 Use Regression[] to obtain more information

The *Mathematica*–package on linear regression will do a more careful analysis of the data that `Fit[]`.

Mathematica

```
Needs["Statistics`LinearRegression`"]
Regress[fitData,{1,x}, x]
```

Among other the standard error (SE) of the y-intercept and the slope of the straight line are estimated.

Chapter 7

Some additional topics

7.1 Pattern matching

Pattern matching is a very powerful tool in *Mathematica*. It is used for many internal procedure in *Mathematica*. We can only offer a very brief glimpse on this interesting topic. In Table 7.1 you find a list of the most commonly use pattern. Table 7.2 shows a few commands to check for patterns.

`_`	a pattern matching any expression
`x_`	a pattern named `x` matching any expression
`_h`	a pattern with head `h`
`x_h`	a pattern named `x` with head `h`
`x__`	a sequence of one or more expressions named `x`
`x___`	a sequence of zero or more expressions named `x`
`x_Integer`	an integer number
`x_Real`	a real number
`x_Complex`	a complex number
`x_List`	a pattern with head `List`
`x^n_`	the symbol `x` raised to any power (named `n`)
`x_^n_`	any number (named `x`) raised to any power (named `n`)

Table 7.1: Commonly used patterns

`Cases[list,form]`	give a list of all the elements at level 1 in `list` that math pattern `form`
`Position[list,form]`	give a list the positions at which object matching `form`appear in `list`
`Select[list,test]`	give the elements of `list` on which `test` gives `True`

Table 7.2: Test for pattern

This has to be illustrated by a few examples.

Mathematica

```
Cases[{x,x^3,Sqrt[x],Sin[x],x^(3+2)},x^_]
.
  3            5
{x , Sqrt[x], x }
```

Mathematica

```
Cases[{3,x,1,a^2,1+x^4,12},_Integer]
.
{3, 1, 12}
```

To find out about the internal working of pattern we look at the representation of a pattern. The pattern `Blank[]` matches anything.

Mathematica

```
FullForm[_+x]
FullForm[_+x_]
.
Plus[x, Blank[]]
Plus[Blank[], Pattern[x, Blank[]]]
```

This pattern will match any term, to which `x` is added, thus

Mathematica

```
Position[{a+x,x+a,a+b x^x,x-a,y+x},_+x]
.
{{1}, {2}, {4}}
```

Observe that *Mathematica* did rearrange some of the sums in alphabetical order. Thus the pattern`_+x` was matched by `x+a`, but not by `y+x`.

The pattern `f[x_,y_]` stands for a function with two arguments, while `f[{x_,y_}]` stands for a function with one argument which has to be a list of two arguments. There is a difference as the example below shows.

Mathematica

```
Clear[f,x,y]
f[x_,y_] = x+y;
f[{x_,y_}]= x*y;
{f[3,4],f[{3,4}]}
??f
{7, 12}

Global`f

f[x_, y_] = x + y
f[{x_, y_}] = x*y
```

We can create a function of two variables which

- returns the sum of the two arguments, if the first argument is an integer.
- returns the product of the two arguments, if the first argument is an real number.
- is not defined otherwise.

Mathematica

```
Clear[dummy]
dummy[x_Integer,y_]=x*y;
dummy[x_Real,y_]=x*y;
{dummy[2,3],dummy[2.5,3],dummy[I,3]}
.
{6, 7.5, dummy[I, 3]}
```

We can also write a transformation rule that will turn a sum of the expressions in a product of two expressions. If this rule is applied repeatedly (`//.`) then finally all sums will be turned into products.

Mathematica

```
Clear[a,b,c,d,x,y]
sumToProduct = (x_+y_ -> x*y);
a+b+c+d /. sumToProduct
a+b+c+d //. sumToProduct
.
a (b + c + d)
a b c d
```

As a simple simple, sensible example we create a function `mean` to compute the average value of the given arguments or of the numbers in the given list.

Mathematica

```
Clear[mean]
mean[x__]   := Plus[x]/Length[x]
mean[{x__}] := mean[x]
```

Now we apply it to different arguments and obtain the expected results.

Mathematica

```
mean[a,b,4]
mean[{a,f,x}]
.
4 + a + b
---------
    3

a + f + x
---------
    3
```

With the command `Trace[]` we can verify that the second call of `mean` indeed calls the first definition of `mean`.

7.2 Some more graphics

7.2.1 Combining multiple graphics

Show and DisplayFunction

With `Show[]` *Mathematica* can combine multiple graphics.

Mathematica

```
?Show
.
Show[graphics, options] displays two- and  three-dimensional graphics,
   using the options specified. Show[g1, g2, ...] shows several plots
   combined. Show can also be used to play Sound objects.
```

With the option `DisplayFunction` one indicates where to display the graphics, if at all. With the value `DisplayFunction -> Identity` the result is not shown at all. `DisplayFunction -> $DisplayFunction` uses the standard display of the system in use.

The code below generates the fit of a polynomial with `Fit[]`. In this example *Mathematica* tries to fit a polynomial of degree five through the given set of data points. Then the graphics object `g1` is generated with `Plot`, without displaying the result. Next the graphics object `g2` is created as a plot of all the data points. The the command `Show` is used to combine the two result and finally display them on the screen. Find the result in Figure 7.1.

Mathematica

```
data={{0,38.11728395},{10,38.22016461},{20,38.94032922},{30,39.24897119},
     {40,38.16872428},{50,35.95679012},{60,32.66460905},{70,29.26954733},
     {80,26.02880658},{90,24.79423868},{100,26.49176955},{110,29.47530864},
     {120,31.89300412},{130,32.76748971},{140,31.79012346},{150,28.24074074},
     {160,20.31893004},{170,11.67695473},{180,4.62962963}};
fit1 = Fit [data, {1, x, x^2, x^3,x^4,x^5}, x];
g1=Plot [fit1,{x, 0, 180},DisplayFunction -> Identity];
g2=ListPlot[data,DisplayFunction -> Identity];
Show[g1,g2,DisplayFunction -> $DisplayFunction];
```

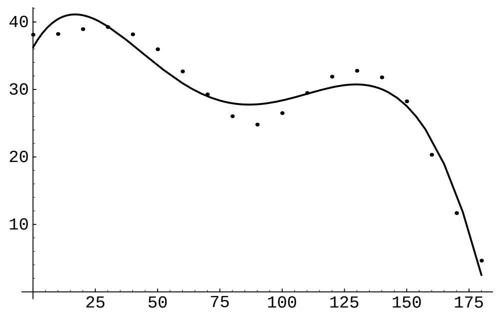


Figure 7.1: Plot of some data point and a fitted polynomial

7.2.2 GraphicsArray

Sofar we always generated just one graphics at the end. There are situations when multiple graphics of similar type should be shown as a group. To display an array of graphics *Mathematica* provides the command `GraphicsArray`. We first generate four graphics objects by

Mathematica

```
gj0 = Plot[BesselJ[0, x], {x, 0, 10},
        DisplayFunction -> Identity];
gj1 = Plot[BesselJ[1, x], {x, 0, 10},
        DisplayFunction -> Identity];
gj2 = Plot[BesselJ[2, x], {x, 0, 10},
        DisplayFunction -> Identity];
gj3 = Plot[BesselJ[3, x], {x, 0, 10},
        DisplayFunction -> Identity];
```

Then they are displayed with the help of `GraphicsArray` and we get the result in Figure 7.2.

Mathematica

```
ga = Show[GraphicsArray[{{gj0, gj1}, {gj2, gj3}}]];
```

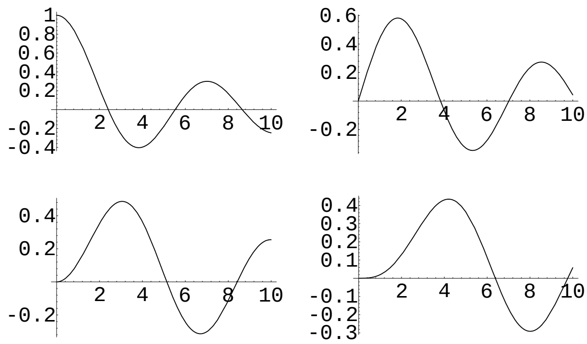


Figure 7.2: Fours graphics, shown as an array

Unfortunately the ranges differ from one graph to the other. If this is to be avoided, such that the results can be compared, then the option `PlotRange` can help. See the result in Figure 7.3.

Mathematica

```
Show[ga /. (PlotRange -> Automatic ) -> (PlotRange -> {-0.5,1})];
```

7.2.3 Animation

Mathematica can also do animations on screen. The used method is rather simple: generate a sequence of pictures and the display then frame by frame to generate the animation. The code below defines a command to generate the graphics of a handle with a wheel attached. One figure is generated as a simple test. Also use this as an example on how to use basic graphics objects like `Circle` and `Line` to compose more complicated objects. There are many more of those basic graphics objects, see the online help manual. These basic graphics command allow to generate very involved graphics sometimes with sizeable effort though.

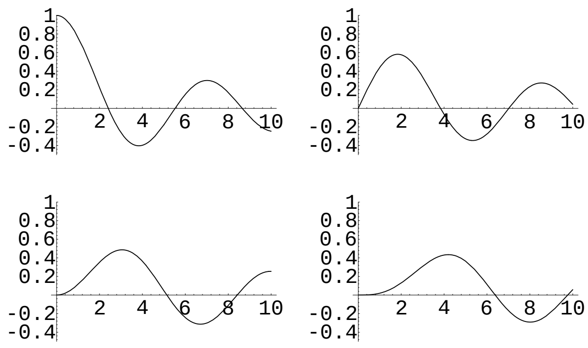


Figure 7.3: Four graphics, shown as an array, same domain

Mathematica

```
Clear[Rout,Rin,mycircle]
Rout=3;
Rin=1;
mycircle[angle_] := Show[Graphics[
    {Circle[Rout{Cos[angle], Sin[angle]}, Rin],
    Line[{{0,0},Rout{Cos[angle], Sin[angle]}}]},
    PlotRange -> {{-4,4},{-4,4}},
    AspectRatio -> Automatic]]
mycircle[2Pi/3];
```

Then a sequence of pictures with angles from 0 to 360 degree with stepsize 10 degree is generated. After all pictures are generated you can double click one of the figures to start the animation. A control panel will pop up on the border of the notebook. For large graphics it is advisable to delete the animation before saving the file, as the file might turn out to be huge. This applies in particular to 3D animations.

Mathematica

```
Do[mycircle[al],{al,0 ,360 Degree, 10 Degree}]
```

7.2.4 Text and fonts

To add some text to a graphic you can choose a few parameters, like size, location and font. On most systems the fonts „Courier“, „Helvatica“ and „Times“ are available; most often with the variations „*name*–Bold“, „*name*–Oblique“, „*name*–BoldOblique“. For the font „Times“ there is „Italic“ instead of „Oblique“. Examine the examples given below and the on line documentation to learn to master all the options, e.g. `FontForm`.

The following code leads to figure 7.4.

Mathematica

```
Clear[f,x,fig,fig1];
f[x_] := Expand[(x-4)^3 + 3 (x-4)^2]/3+2 - Sin[2 (x-1)];
a := 0.5;
b := 5.5;
xi := 2.0;
h  := 0.7 ;
fig1=Plot[f[x],{x,a,b},AxesLabel -> {"x","y"},Ticks -> None,
                      PlotRange ->{-1,6},DisplayFunction -> Identity];
text = Graphics[{Text["a",{a,-0.2},{0,1}],
                 Text["b",{b,-0.2},{0,1}],
                 Text["x",{xi,-0.3},{0,1}],
                 Text["x+h",{xi+h,-0.2},{0,1}]}];
fig  = Show[
           fig1,
           Graphics[ Line[{ {a,0},{a,f[a]}}] ],
           Graphics[ Line[{ {b,0},{b,f[b]}}] ],
           Graphics[ Line[{ {xi,0},{xi,f[xi]}}] ],
           Graphics[ Line[{ {xi+h,0},{xi+h,f[xi+h]}}] ],
           text,
           DisplayFunction -> $DisplayFunction
           ];
```

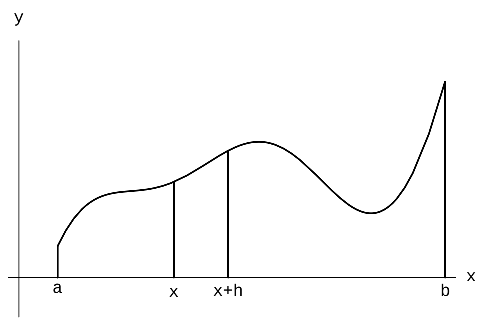


Figure 7.4: A figure with some simple text

To obtain figure 7.5 you may use

Mathematica

```
f[x_] := Expand[3*Sin[x] +1.5x];
a := 0.5;
b := 5.5;
ab := b-a;
kurve = Plot[f[x],{x,a,b+0.1 ab},
        PlotRange->{{a-0.1 ab,b+0.1 ab},{-2,8}},
        Ticks->None,
        DisplayFunction -> Identity];
n := 4;
deltax := (b - a)/n;
s = Table[a+i*deltax,{i,0,n}];
lines = Table[Line[{{s[[i]],0},{s[[i]],f[s[[i]]]}}],{i,1,n+1}];
texte = Graphics[{Text["a",{a,-0.17},{0,1}],
                  Text[FontForm["y = f(x)",
                {"Courier-Bold",15}],{5,6.8},{1,0}]}];
texte1 = Graphics[Flatten[Table[{Text["x",{s[[i]],-0.8},{0,1}],
                        Text[ToString[i-1],{s[[i]]+0.12,-0.85},{0,1}]},
                        {i,1,n+1}]]];
textaxes =Graphics[{
   Text[FontForm["y-Achse",{"Times-Italic",20}],
          {0.3,8},{0,1},{0,1}],
   Text[FontForm["x-Achse",{"Times-Italic",20}],
          {b+0.5,1},{1,0} ]
     }];
fig = Show[kurve,Graphics[lines],texte,texte1,textaxes,
                  DisplayFunction -> $DisplayFunction];
```

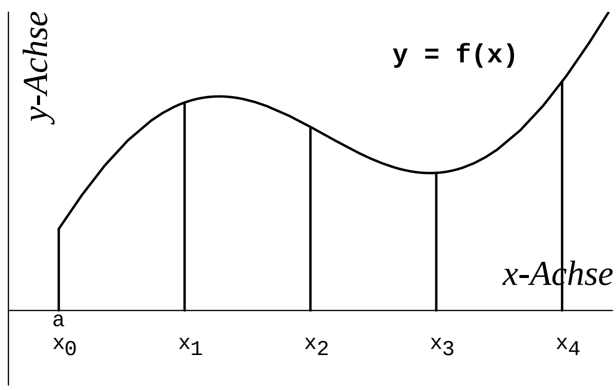


Figure 7.5: Graphics with some fancy text

The code below will lead to Figure 7.6. It is taken from lecture notes on integration. The rather long *Mathematica* code should illustrate the combination of some simple *Mathematica* code to generate all the lines in the figure and how to put it all together. All the essential parameters are listed at the top of the code section. If those are changed the graph will be adapted automatically.

Mathematica

```
f[x_] := Expand[(x-4)^3 + 3 (x-4)^2]/3+2 - Sin[2 (x-1)];
a := 0.5;
b := 5.5;
ab := b-a;
kurve = Plot[f[x],{x,a,b},
        PlotRange->{{a-0.1 ab,b+0.1 ab},{-2,5.5}},
        Ticks->None,
        DisplayFunction ->Identity];
n := 4;
deltax := (b - a)/(n+1);
s = Table[a+i deltax+(-1)^Random[Integer,{0,1}] Random[] deltax/3,{i,1,n}];
s = Prepend[s,a];
s = Append[s,b];
xi = Table[s[[i]] + (0.2 Random[] + 0.4) (s[[i+1]]-s[[i]]), {i,1,n+1}];
hlines = Table[Line[{{s[[i]],f[xi[[i]]]},{s[[i+1]],f[xi[[i]]]}}],
                        {i,1,n+1}];
vlinesleft = Table[Line[{{s[[i]],0},{s[[i]],f[xi[[i]]]}}],
                                {i,1,n+1}];
vlinesright = Table[Line[{{s[[i+1]],0},{s[[i+1]],f[xi[[i]]]}}],
                                {i,1,n+1}];
xilines = Table[Line[{{xi[[i]],0},{xi[[i]],f[xi[[i]]]}}],
                                {i,1,n+1}];
texte = Graphics[{Text["a",{a,-0.17},{0,1}],
                        Text["b",{b,-0.17},{0,1}],
                        Text[FontForm["y = f(x)",
                                {"Courier",14}],{6,4.8},{1,0}]}];
texte1 = Graphics[Flatten[Table[{Text["x",{s[[i]],-0.6},{0,1}],
                        Text[ToString[i-1],{s[[i]]+0.12,-0.65},{0,1}]},
                        {i,1,n+2}]]];
texte2 = Graphics[Flatten[Table[{
        Text[FontForm["x",{"Symbol",8}],{xi[[i]],-1.0},{0,1}],
        Text[FontForm[ToString[i],{"Symbol",6}],{xi[[i]]+0.15,-1.2},{0,1}]},
        {i,1,n+1}]]];
fig = Show[kurve,Graphics[{hlines,vlinesleft,vlinesright,
                {Dashing[{0.015,0.015}],xilines}}],texte,texte1,texte2,
                 AxesLabel->{x,y},
                 DisplayFunction -> $DisplayFunction];
```

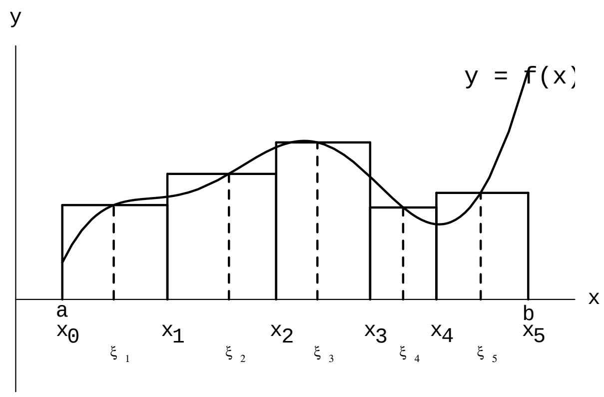


Figure 7.6: A graphics, automatically generated by *Mathematica*

`While[test, body]`	evaluates `test`, then `body`, repetitively, until `test` first fails to give `True`

`Do[expr, {imax}]`
`Do[expr, {i, imax}]`
 	
`Do[expr, {i, imin,imax}]`
`Do[expr, {i, imin, imax},`
` {j, jmin, jmax}, ...]`
	evaluates `expr imax` times
	evaluates `expr` with the variable `i`
	successively taking on the values 1 through `imax`
	starts with `i = imin`
	evaluates `expr` looping over different
	values of j, etc. for each i.

`For[start, test, incr, body]`	executes `start`, then repeatedly evaluates `body` and `incr` until `test` fails to give `True`.

Table 7.3: Classical iterators

`If[test,then,else]`
`Which[test1,expr1,test2,...]`
`Switch[expr,form1,val1,form2,...]`
 	
`expr1||expr2||expr3`
`expr1&&expr2&&expr3`
	evaluate `then` if `test` is `True`, else `else`
	evaluate `testi` until first is `True`, then `expri`
	compare `expr` with each of the `formi`,
	giving the first `val` it matches
	logical OR
	logical AND

Table 7.4: Tests

7.3 Procedural programming

7.3.1 Loops and Control structures

In Tables 7.3 and 7.4 you find some of the standard programming commands. In *Mathematica* they offer no particular surprise. Thus we will not go to great length to explain them.

7.3.2 Simulation of an M/M/1-system

Customers enter a post office with one counter according to a Poisson process with rate $\lambda = 2$ customers/minute. The service time is approximately exponentially distributed with mean $1/\mu = 1/3$ minute. We assume that the first customer enters the post office at t = 0. Simulate this system by generating 499 exponentially distributed random numbers with mean 1/2 (interarrival times) and 500 exponentially distributed random numbers with mean 1/3 (service times).

(a) Determine the list `arrival` of arrival times of all the customers.

(b) Determine the list `depart` of departure times of all the customers.

(c) Determine the average time a customers spends in the system.

(d) Determine how many costumers never wait in the queue.

Below find a non recursive solution to the simulation part of the above problem.

Mathematica

```
Needs["Statistics`DescriptiveStatistics`"]
Needs["Statistics`ContinuousDistributions`"]
n = 500;
lambda = 2;
mu = 3;
time= 0.0;

(* create tables filled with zeros *)
arrival = Table[time,{n}];
depart  = arrival;
service = Table[Random[ExponentialDistribution[mu]],{n}];
depart[[1]]=service[[1]];

(* now do a simple loop over the generated tables *)
Do[ arrival[[i]]= arrival[[i-1]] + Random[ExponentialDistribution[lambda]];
    depart[[i]] = service[[i]] + Max[arrival[[i]],depart[[i-1]]];
  ,{i,2,n}]
Mean[depart-arrival]
Length[Select[depart-service-arrival,#<=0.00001&]]
.
1.00825
173
```

Bibliography

[AbraSteg] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions.* Dover, 1972.

[Bahd95] T. B. Bahder. *Mathematica for Scientists and Engineers.* Addison Wesely, 1995.

[Wolf96] S. Wolfram. *Mathematica, A System for Doing Mathematics by Computer.* Cambridge University Press, New York, third edition, 1996.

List of Figures

List of Tables