TU Bergakademie Freiberg

Winter 12/13

Computer Algebra System Mathematica

1 Introduction

1.1 About Mathematica

Mathematica (`www.wolfram.com/mathematica/`) is one of the highest developed computer algebra systems (CAS) available today. It is a versatile and complex tool to solve mathematical problems.

- **Numeric calculations**
  Several thousand built-in numerical functions are available. The degree of precision is adjustable by the user.
- **Symbolic calculations**
  The rules of integral and differential calculus are implemented. The program can use variables in these operations, for example to determine the antiderivative of a function.
- **Visualization tool**
  Mathematica can display 2D or 3D plots of analytic functions or list data.

1.2 The program

The two important components of Mathematica are

- **Frontend**
  The Frontend provides user-friendly input and processes the output from the Kernel. Its form and functionality may depend on the OS of your computer.
- **Kernel**
  The Kernel does the actual calculations independently in the background. This part is run just after the first calculation is requested from the user.

In the Frontend you can create and save documents called **Notebooks**, which are usually given the prefix `.nb`. A Notebook can be used like a paper notebook, or like a sequence of commands for the Kernel. The contents of a Notebook file is organized into cells. In the Notebook window, cell boundaries are indicated by brackets.
In addition to data cells (input and output), a notebook can contain text cells, for example comments or headings. Cell behaviour can be controlled with the **Cell** submenu.
When entering text into a new Notebook, a new **input**-cell is automatically created. A

cell can extend over several input lines (line break with **Return**), and can contain several mathematical expressions.

To let Mathematica calculate math expressions in a cell, mark the cell or place the cursor must be inside. The calculation is then started by pressing the combination <**Shift**> + <**Return**>. Alternatively, the **Enter** key on the numeric keypad can be used. A simple <**Return**> just inserts a new line into the cell. Cells can be run regardless of their position within the Notebook. All variables, results or packages are updated by the last cell that was run. All Cells in a Notebook can be run in sequence using the submenu **Evaluation -> Evaluate Notebook**.

The input cells are marked by a tag `In[j]:=` (j=running index) after processing. If the evaluation of the cell is successful, Mathematica produces an **output** cell marked by `Out[j]=`, directly below the Input cell.

1.3 Syntax

All expressions in Mathematica obey a standard syntax. You have two alternatives to enter into input cells:

- Using the classical syntax, which is universal to all Mathematica frontends. The result looks like a computer program, and may not be easy to read.
- Input using items from predefined palettes. Try the palette **Basic Math Assistant**, which can be found in the menu **Palettes**.

1.4 Packages

Several more extensive commands and subprograms are contained in seperate **packages**, which can be loaded on demand. This can be done by the command `Needs` or via the operator `<<`. After the name of the package, an acute ` character must follow. When using `Needs`, a string argument is required.

```
In[1]:= Needs["VectorAnalysis`"]

In[2]:= Coordinates[]
Out[2]= {Xx,Yy,Zz}

In[3]:= Grad[9x^2+4y^2+z^2, Cartesian[x,y,z]]
Out[3]= {18x,8y,2z}
```

A package should be loaded before any command from the package is processed.

1.5 Finding commands

The comprehensive online help system is very useful in working with Mathematica. It can be accessed directly via menu point **Help > Documentation Center** or by pressing **F1**. Using **F1** when the cursor is placed on a command shows the help entry for the command. Another opportunity is to display a short synopsis for a command and its options within the Notebook window by entering `?function` into an input cell.

```
In[4]:= ?Log
        Log[z] gives the natural logarithm of z (logarithm to base e).
        Log[b,z] gives the logarithm to base b.  >>
```

The help ends with a cross reference symbol >>, which brings up the extensive help page when activated.

Lg	**sin**	**Falsch**
Log	**Sin**	**Richtig**

2 Using Mathematica

2.1 A Calculator

All standard arithmetic functions are available:

Operation	Input
addition	`x+y`
negation	`-x`
subtraction	`x-y`
multiplication	`x y` oder `x*y`
division	`x/y`
potentiate	`x^y`

example:

```
In[1]:= 2(12-2)+(2+1)^2
Out[1]= 29
```

The sequence of calculation adheres to the mathematic conventions (association, commutation, ..)and can be influenced by **round** parentheses. An example for exponential notation:

```
In[2]:= 5.43 10^-45
Out[2]= 5.43 × 10^-45
```

Mathematica delivers, when possible, the exact result in decimal notation:

```
In[3]:= 3^68
Out[3]= 278 128 389 443 693 511 257 285 776 231 761
```

To obtain this in exponential notation, you can apply the function `N[`*expression*`]` (for: **Numerical value**) to the expression. It will be rounded:

```
In[4]:= N[3^68]
Out[4]= 2.78128 × 10^32
```

The rounding precision can be given as an optional number:

```
In[5]:= N[Pi,30]
Out[5]= 3.14159265358979323846264338328
```

Some constants are implemented and can be addressed by names and special characters, for example π as `Pi` or Euler's number e as `E`. For the integration of functions ∞: `Infinity` is useful. All built-in functions or constants begin with capital letters.

If an Integer is entered without dot, Mathematica treats this as an exact number. In expressions containing only exact numbers, the results are given in exact form as well:

```
In[6]:= 1/2 + 3/7
```
$$\texttt{Out[6]=}\ \frac{13}{14}$$

If you enter the number with a dot, Mathematica interprets it as rounded. If such a number is contained in an expression, the result is determined numerically:

```
In[7]:= 1/2 + 3.0/7
Out[7]= 0.928571
```

2.2 Functions

2.2.1 Built-in functions

Mathematica features both standard and packaged built-in functions. Arguments passed to a function are enclosed in **brackets**. The function names are **very** case-sensitive. They always begin with a capital letter. In trigonometric functions, the argument is always in **radians**.

```
In[8]:= Sin[45.]
Out[8]= 0.850904

In[9]:= Sin[Pi/4.]
Out[9]= 0.707107
```

As in arithmetic operations, exact function arguments result in exact computation and output:

```
In[10]:= Log[1/5]
Out[10]= -Log[5]
```

A numerical, rounded result will be shown when the function `N[`*`expression`*`]` is applied:

```
In[11]:= N[%]
Out[11]= -1.60944
```

In practical work, it may be comfortable to address a former output cell from the current input cell. This works like:

Input	means
`%`	last output
`%%`	next-to-last output
`%n`	output index n `Out[n]`

This type of addressing is frequently used throughout this tutorial. To understand the following example, check the output `out[1]` and `out[6]` above:

```
In[12]:= %1/14 + N[%6]
Out[12]= 3.
```

Complex numbers are noted using the imaginary unit `I`:

```
In[13]:= Sqrt[-4]
Out[13]= 2 i

In[14]:= Sin[I]
Out[14]= i Sinh[1]
```

The common operations with complex arguments are implemented. In particular, the functions `Re[z]` und `Im[z]` give the real and imaginary parts of the argument z:

```
In[15]:= Re[%%]
Out[15]= 0

In[16]:= Im[%14]
Out[16]= Sinh[1]
```

2.2.2 User-defined functions

You are free to define your own functions (see chapter 6). On the **left** hand side of such a definition, the variables, that are evaluated by the function, have to be included, marked with an **underscore**. The names of your own functions should be different from Mathematicas built-in functions. This can be achieved easily by giving them lower case first letters.

```
In[17]:= f[x_,y_]:=x^2 + 5y
```

After the definition has been processed, Mathematica will not generate an output cell. Self-defined functions may in all cases process numerical as well as symbolic arguments, as in the example:

```
In[18]:= f[3,7]
Out[18]= 44

In[19]:= f[a,b]^3
Out[19]= (a^2 + 5b)^3
```

2.3 Algebraic Manipulations of expressions

We have seen how Mathematica can process numerical data exactly and in approximation. It has also the ability to transform algebraic resp. arithmetic expressions with **symbols**:

```
In[20]:= Expand[%]
Out[20]= a^6 + 15a^4b + 75a^2b^2 + 125b^3
```

`Simplify[expression]` and `FullSimplify[expression]` are self-explaining:

```
In[21]:= Simplify[x^2 + 23x - 9x - x^3 + 18 + 2x/3 + 3x^2]
Out[21]= 18 + 44x/3 + 4x^2 - x^3

In[22]:= Simplify[x^2 + 2x^3/(3x) + 2x/(3x^2 + 5)]
Out[22]= 5x^2/3 + 2x/(5+3x^2)
```

2.4 Differentiation and Integration

Mathematica can determine the derivative of all elementary mathematical functions. The operator (ok, function) `D[f,x]` gives the partial derivative of the function f w.r.t. the variable x. It is implied here that all other symbols/variables in the expressions for f do not depend on x, unless such a dependence was declared beforehand to Mathematica.

```
In[23]:= D[Cos[3*x^2],x]
Out[23]= -6 x Sin[3 x^2]
```

It is possibe to use unknown functions as symbols in your expressions. They are differentiated as well:

```
In[24]:= D[f[x]^2,x]
Out[24]= 2 f[x] f'[x]
```

The operator (function) `Integrate[f,x]` finds the antiderivative F to a function f:

```
In[25]:= Integrate[Cos[3*x],x]
Out[25]= 1/3 Sin[3 x]
```

Mathematica seeks a symbolic expression in case a definite integral with symbolic interval borders is calculated:

```
In[26]:= Integrate[Exp[x]x,{x,0,a}]
Out[26]= 1 + (-1 + a) e^a
```

3 Vectors and matrices

3.1 Vectors and matrices as formatted lists

In Mathematica, vectors and matrices are represented by lists. Lists are sets of elements, seperated by commas, which are enclosed in curly braces **{}**. A matrix then is a "list of lists", where the sublists correspond to the rows of the matrix. You can alternatively input a matrix via menus **Insert** > **Table/Matrix** > **New**.

```
In[1]:= vektor1:={x,y,z}
        vektor2:={u,v,w}
        vektor3:={s,t}
        matrix1:={{a,b},{c,d}}
```

To display a matrix in the usual form, the function `MatrixForm[]` can be used:

```
In[5]:= MatrixForm[vektor1]
Out[5]//MatrixForm=
        ( x )
        ( y )
        ( z )
In[6]:= MatrixForm[matrix1]
Out[6]//MatrixForm=
        ( a  b )
        ( c  d )
```

A single element of a list can be addressed by giving its index in double brackets. For 2-dim lists like matrices, a single index gives a sublist (a row) and a single element has to be addressed by two indices.

```
In[7]:= matrix1[[2]]
Out[7]= {c,d}

In[8]:= matrix1[[2,2]]
Out[8]= d
```

3.2 Calculating with matrices

The "matrix product" of two matrices is represented by a dot ".", or alternatively by the function `Dot[a,b]`. The normal multiplication operator (*), multiplies only the elements with identical index with each other, thereby creating a new matrix. The vector product of two vectors is implemented in the function `cross[a,b]`.

```
In[9]:=  vektor1.vektor2
Out[9]=  {ux + vy + wz}

In[10]:= Dot[matrix1,matrix1]
Out[10]= {{a^2 + bc, ab + bd}, {ac + cd, cb + d^2}}

In[11]:= MatrixForm[Cross[vektor1,vektor2]]
Out[11]//MatrixForm=
```

$$\begin{pmatrix} \text{wy - vz} \\ \text{-wx + uz} \\ \text{vx - uy} \end{pmatrix}$$

When calculating the dot product of a vector with a matrix, Mathematica interprets the vector depending on the sequence of the arguments as a column or a row (second example, check this!):

```
In[12]:= MatrixForm[matrix1.vektor3]
Out[12]//MatrixForm=
```

$$\begin{pmatrix} \text{as + bt} \\ \text{cs + dt} \end{pmatrix}$$

```
In[13]:= MatrixForm[vektor3.matrix1]
Out[13]//MatrixForm=
```

$$\begin{pmatrix} \text{as + ct} \\ \text{bs + dt} \end{pmatrix}$$

Other matrix functions include `Transpose[]`, `Det[]` and `Inverse[]`.

Eigenvalues and Eigenvectors of a matrix can be calculated with the respective functions. The compound function `Eigensystem[]` creates an **object** containing the complete information.

```
In[13]:= Eigenvalues[matrix1]
Out[13]= {1/2 (a + d - Sqrt[a^2 + 4bc - 2ad + d^2]),
```

$$\frac{1}{2}\left(\texttt{a + d + }\sqrt{\texttt{a}^2\texttt{ + 4bc - 2ad + d}^2}\right)\Bigg\}$$

```
In[14]:= Eigenvectors[matrix1]
```

$$\texttt{Out[14]= }\left\{\left\{-\frac{\texttt{-a + d + }\sqrt{\texttt{a}^2\texttt{ + 4bc - 2ad + d}^2}}{\texttt{2c}},\ 1\right\},\ \left\{-\frac{\texttt{-a + d - }\sqrt{\texttt{a}^2\texttt{ + 4bc - 2ad + d}^2}}{\texttt{2c}},\ 1\right\}\right\}$$

4 Solving equations

4.1 Polynomial equations with one variable

The function `Solve` can be used to solve polynomial equations. The double equality sign designates an identity, which is the desired result of the equation. The single equality on the left side is used to assign the whole equation to the variable *gl1* (more information on value assignment in section 6).

```
In[1]:= gl1 = a*x^2 + b*x + c == 0
```

$$\texttt{Out[1]= c + bx + ax}^2\texttt{ == 0}$$

The result of the function `Solve` is a list of so-called **substitution rules** for the values of x which solve the equation. In the example, the variable `sol1` contains two substitution rules (indicated by arrows), for the two possible solutions of the quadratic equation. To get the actual results from the substitution rule, we have to apply (operator `/.`) it to the simple expression `x`. After that, `x` contains a list.

```
In[2]:= sol1 = Solve[gl1,x]
```

$$\texttt{Out[2]= }\left\{\left\{\texttt{x}\ \rightarrow\ \frac{\texttt{-b - }\sqrt{\texttt{b}^2\texttt{ - 4ac}}}{\texttt{2a}}\right\},\ \left\{\texttt{x}\ \rightarrow\ \frac{\texttt{-b + }\sqrt{\texttt{b}^2\texttt{ - 4ac}}}{\texttt{2a}}\right\}\right\}$$

```
In[3]:= x/.sol1
```

$$\texttt{Out[3]= }\left\{\frac{\texttt{-b - }\sqrt{\texttt{b}^2\texttt{ - 4ac}}}{\texttt{2a}},\ \frac{\texttt{-b + }\sqrt{\texttt{b}^2\texttt{ - 4ac}}}{\texttt{2a}}\right\}$$

For numerical solutions, the function NSolve is used.

4.2 Trancendental equations with one variable

Mathematica is able to solve some transcendental equations. It will, in general, show a warning that some (of many) solutions might be missing, and it will not be possible to display the solutions in a symbolic form. In this case, it is sometimes useful to get a single solution (root) in the vicinity of a given starting value. This is done by the function `FindRoot`, resulting in an approximate value.

```
In[4]:= FindRoot[Cos[x]==x, {x,1}]
Out[4]= {x → 0.739085}
```

The starting value in this example is 1.

4.3 Systems of equations with many variables

The function `Solve` can solve all linear equation systems and numerous systems of polynomial equations.

4.3.1 Two equations for one variable

The equations must be passed to the function in a list variable (within curly braces):

```
In[5]:= Solve[{x^2==1, x^3==1},x]
Out[5]= {{x → 1}}
```

4.3.2 Two equations for two variables

The system may be solved for both variables x and y. A solution for one variable, for example x, would include the other variable as a free parameter.

```
In[6]:= Solve[{x + y == 5, 5*x - a*y == 0},{x,y}]
```

$$\text{Out[6]= } \left\{\left\{x \to \frac{5a}{5 + a}, y \to \frac{25}{5 + a}\right\}\right\}$$

The function `LinearSolve` produces a vector solution x for the linear matrix equation system of the form:

$$m.x == b$$

In the next example, this function is applied to solve an inhomogenous linear equation system with the coefficient matrix km.

```
In[7]:= km = {{1,1},{5,-a}}
        b = {5,0}
        LinearSolve[km,b]
Out[7]= {{1,1},{5,-a}}
Out[8]= {5,0}
```

$$\text{Out[9]= } \left\{\frac{5a}{5 + a}, \frac{25}{5 + a}\right\}$$

4.3.3 Eliminating variables

Lets study two equations with the three variables x,y and a. Solving them for x and y, the solution will still contain the parameter a:

```
In[10]:= Solve[{x==y-3a, y==2x-a},{x,y}]
Out[10]= {{x → 4 a , y → 7 a}}
```

In this situation, we could eliminate for example a, and get a relation between x and y. The following options to `Solve` will instantly solve the system for x and eliminate a:

```
In[11]:= Solve[{x==y-3a, y==2x-a},x,a]
```

$$\text{Out[11]= } \left\{\left\{x \to \frac{4\ y}{7}\right\}\right\}$$

4.4 Differential equations

The functions `DSolve` can solve linear and nonlinear ordinary differential equations (ODE) as well as systems of differential equations. The results are returned as substitution rules, like in the `Solve` function.

```
In[12]:= DSolve[y''[x] == 3*y[x],y[x],x]
Out[12]= {{y[x] → e^(Sqrt[3] x) C[1] + e^(-Sqrt[3] x) C[2]}}
```

The general solution here contains undetermined constants, which can be fixed via additional starting or boundary values:

```
In[13]:= DSolve[{x''[t] + ω^2*x[t]==0, x'[0]==0, x[0]==1},x[t],t]
Out[13]= {{x[t] → Cos[tω]}}
```

Some solutions to differential equations can not be expressed in algebraic terms. In these cases, a numerical solution can be obtained by `NDSolve`. Such a solution can also be displayed graphically.

```
In[14]:= NDSolve[{x''[t] + Sin[x[t]]==0, x[0]==1, x'[0]==0},x,{t,0,10}]
Out[14]= {{x → InterpolatingFunction[{{0., 10.}},<>]}}
```

The numerical result in this example can be evaluated further:

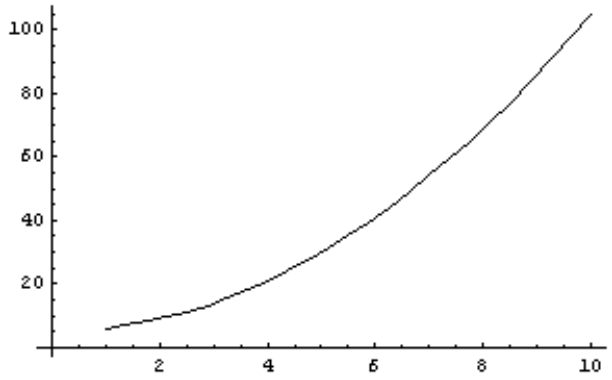
```
In[15]:= {x[1.5],x'[3.0]}/.%14[[1]]
Out[15]= {0.166939, -0.29119}
```

5 Graphics

A real-valued function can be displayed with the command `Plot`:

```
In[1]:= Plot[x^2 + 5, {x,1,10}]
```

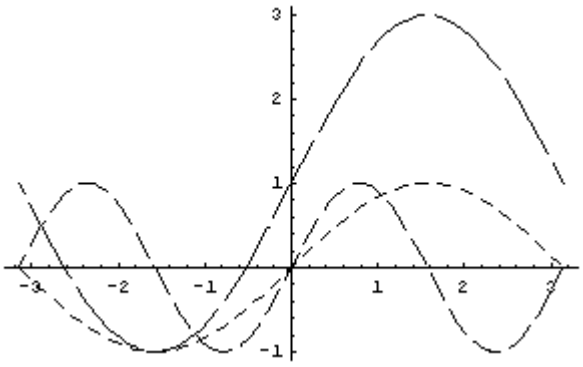
Out[1]=



Simultaneous display of many functions in one plot is possible. You can utilize different line styles or colours to mark your functions:

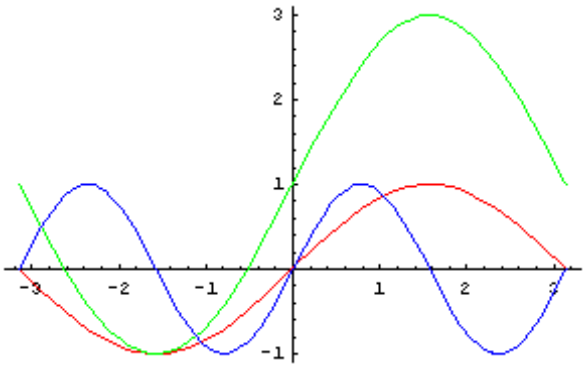
```
In[2]:= Plot[{Sin[x], 2*Sin[x] + 1, Sin[2x]}, {x,-Pi,Pi},
        PlotStyle→{Dashing[{.02,.02}],Dashing[{.15,.02}],
        Dashing[{.05,.02}]}]
```

Out[2]=




```
In[3]:= Plot[{Sin[x], 2*Sin[x] + 1, Sin[2x]}, {x,-Pi,Pi},
        PlotStyle→{RGBColor[1,0,0],RGBColor[0,1,0],
        RGBColor[0,0,1]}]
```

Out[3]=



As we have seen in the above example, many properties of plots can be influenced by **options**. Mathematica expects options generally as last parameter in a function call in form of **rules** with the syntax

$$\textit{Option} \rightarrow \textit{Value}.$$

In case of the `Plot`-function, the full function call including options thus looks like:

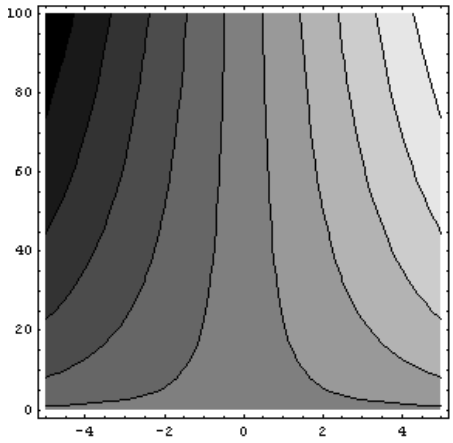
```
Plot[{f1,f2, ...},{x,xmin,xmax},Option1->Value1, Option2->Value2, ...]
```

Some experimenting may be necessary to find proper format options for a certain plot. The options and their syntax are documented in the help system, which can be activated over the menu **Help** > **Documentation Center** or by pressing **F1**.

Contour plots for instance are useful to display scalar fields. The field values are divided into intervals, which are marked via contour lines and greyscale or coulour gradients (here: light grey -> high values)

```
In[4]:= ContourPlot[x*Sqrt[y], {x,-5,5},{y,0,100}]
```

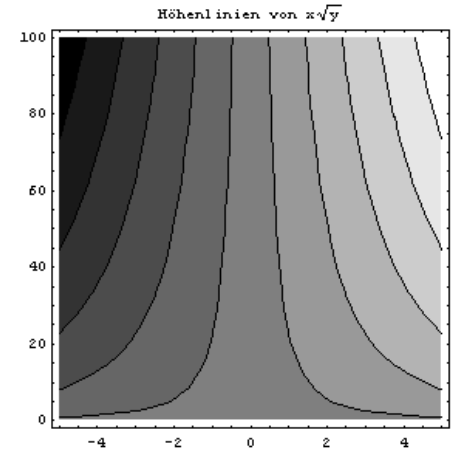
Out[4]=



Like `Plot`, also `ContourPlot` (notice the upper case letters) has numerous graphics design options. In the following example, we add a (german) headline:

```
In[5]:= ContourPlot[x*Sqrt[y], {x,-5,5},{y,0,100},
        PlotLabel→"Höhenlinien von x√y"]
```

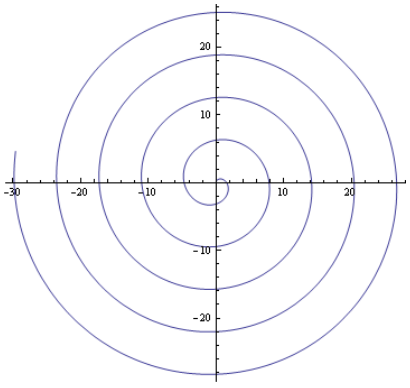
Out[5]=



The command `ParametricPlot` enables parametrized variables in plots. In the next example, a 2D parametric plot for a planar curve is described by two functions (in a list) for x- and y-coordinate depending on the same parameter t:

```
In[6]:= ParametricPlot[{t*Sin[t], t*Cos[t]}, {t,0,30}]
```

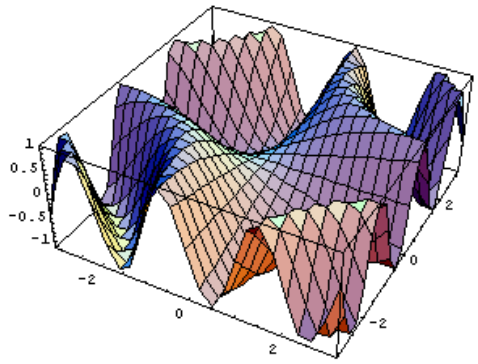
Out[6]=



Mathematica gives you the opportunity to create 3D plots. For the most 2D plot commands, a 3D equivalent exists. The next example is a 3D surface plot of a function *z=f(x,y)* using `Plot3D`:

```
In[7]:= Plot3D[Sin[x*y], {x,-Pi,Pi}, {y,-Pi,Pi}]
```

Out[7]=



6 Value assignment

6.1 Assignment with and without evaluation

In Mathematica, you have 2 alternatives to assign values to identifiers. An identifier is a string, which can be used as name for variables, symbols, functions, constants etc.

> The command `y=A` evaluates the expression A on the right side before assignment. The command `x:=A` assigns x to A without evaluating A.

If the values are constant, there is no difference between these two opportunities.

```
In[1]:= x:=7; y=7; {x,y}
Out[3]= {7,7}
```

(Here, the semicolon is used at the end of the first two terms to supress the output.) The variable assignments in the following example are identical only at the first glance. But indeed, they are different: the variable y is assigned to the value of f, which is 4. The variable x, in contrast, is assigned to the symbol f. If we change f later (in line `In[8]`), x will also change its value.

```
In[4]:= f=4;
In[5]:= x:=f; y=f; {x,y}
Out[7]= {4,4}

In[8]:= f=9; {x,y}
Out[9]= {9,4}
```

> For value assignments, `=` is the regular choice. The assignment `:=` is more useful for variables referring to functions.

6.2 Local value assignment

A local assignment means that the variable is assigned to a value locally, for example only for one statement. The corresponding operator is called a substitution. For example, the expression `In[10]` below is evaluated with `z=2`, but this assignment is not permanent. Consequently z is undefined again in line `In[10]`.

```
In[10]:= 5z-3/.z→2
Out[10]= 7
In[11]:= z
Out[11]= z
```

The application of **substitution rules**, which are generated by Solve, DSolve and other functions, is also a form of a local value assignment. The general form for applying a substitution rule is:

expression/.subsititution rule

If you want to put values from a substitution rule into a variable, you need to assign the above construction to the variable.