Kurs ▮ Cours

2. Numerische Probleme
▮ Problèmes numériques

Die Gliederung dieses Kurses folgt in groben Zügen dem Buch von Nancy Blachman: A Practical Approach....
Hinweis: Kapitel 2 lesen!
Run mit WIN+*Mathematica* Version 5.2

▮ L'articulation de ce cours correspond à peu près à celle du livre de Nancy Blachman: A Practical Approach....
Indication: Lire le chapitre 2.
Testé avec *Mathematica* version 5.2+WIN

WIR94/98/99/2000/2007 // Copyright Rolf Wirz

Mathematica kennt ganze Zahlen (Integers), Fliesskommazahlen (mit Dezimalpunkt), rationale Zahlen, komplexe Zahlen sowie Zahlen, die durch Symbole gegeben sind (z.B. Pi).
▮ *Mathematica* connaît les nombres entiers (integers), les nombres avec virgule flottante (avec point décimal), les nombres rationaux, les nombres complexes ainsi que les nombres représentés par des symboles (p.ex. Pi).

2.1. Arithmetische Operationen
▮ Opérations arithmétiques

Probiere aus: ▮ Essaie:

```
In[1]:= 53 + 78
Out[1]= 131
```

Multiplikation: ▮ Multiplication:

```
In[2]:= 127*9721
Out[2]= 1234567

In[3]:= 127 9721
Out[3]= 1234567
```

Potenzieren: ■ Calculer les puissances:

```
In[4]:= 34^56

Out[4]= 57918773205287127842044254126179599852840968492056164062843692360166371779746690236416
```

```
In[5]:= %^(1/56)

Out[5]= 34
```

2.2. Rationale Zahlen
■ Nombres rationaux

Mathematica rechnet exakt, wenn nicht anders verlangt!
■ *Mathematica* calcule exactement, sauf si on le veut autrement!

Probiere aus: ■ Essaie:

```
In[6]:= 2/4 + 24/144
```

$$\text{Out[6]}= \frac{2}{3}$$

```
In[7]:= 2 + 2/5
```

$$\text{Out[7]}= \frac{12}{5}$$

2.3. Irrationale Zahlen
■ Nombres irrationaux

Mathematica rechnet exakt
■ *Mathematica* calcule exactement

Beispiel: Quadratwurzel ■ Exemple: racine carrée

```
In[8]:= Sqrt[17]
```

$$\text{Out[8]}= \sqrt{17}$$

```
In[9]:= Sqrt[17.0]

Out[9]= 4.12311
```

```
In[10]:= Sqrt[17]
```

$$\text{Out[10]}= \sqrt{17}$$

In[11]:= **Sqrt[17]//N**

Out[11]= 4.12311

2.4. Annäherung durch Dezimalbrüche
■ Approche par fractions décimales

Falls irgendwo in einem Ausdruck ein Dezimalpunkt vorkommt, gilt der ganze Ausdruck als Fliesskommazahl.
■ S'il y a quelque part dans une expression un point décimal, toute l'expression a la valeur d'un nombre à virgule flottante.

Probiere aus: ■ Essaie:

In[12]:= **1/2 + 2.4/144**

Out[12]= 0.516667

In[13]:= **.5/7 + Pi**

Out[13]= 3.21302

In[14]:= **.5/7 + Pi // N**

Out[14]= 3.21302

2.5. Komplexe Zahlen
■ Nombres complexes

"I" bedeutet die imaginäre Einheit. "Re" erzeugt den Realanteil, "Im" den Imaginäranteil, "Abs" den absoluten Betrag, "Conjugate" das konjugiert Komplexe, "Round" rundet. Schaue, was *Mathematica* tut und überlege wieso das Resultat so aussieht:
■ "I" signifie l'unité imaginaire. "Re" crée la partie réelle, "Im" la partie imaginaire, "Abs" la valeur absolue, "Conjugate" le complexe conjugé, "Round" arrondit. Observe ce que *Mathematica* fait et réfléchis pourquoi le résultat est ainsi:

In[15]:= **(2 + 4I)/(5 + 2I)**

Out[15]= $\frac{18}{29} + \frac{16\,i}{29}$

In[16]:= **Re[3 + 4I]**

Out[16]= 3

In[17]:= **Im[3 + 4I]**

Out[17]= 4

In[18]:= **Conjugate[3 + 4I]**

Out[18]= 3 − 4 i

```
In[19]:= Abs[3 + 4I]
Out[19]= 5

In[20]:= Round[2.7 - 8.6 I]
Out[20]= 3 - 9 i

In[21]:= Cos[3 + 4I] // N
Out[21]= -27.0349 - 3.85115 i

In[22]:= Sin[3 + 4I]
Out[22]= Sin[3 + 4 i]

In[23]:= Exp[3 + 4I] // N
Out[23]= -13.1288 - 15.2008 i

In[24]:= Log[3 + 4I]
Out[24]= Log[3 + 4 i]
```

2.6. Symbole und Zahlen, Listen etc.
■ Symboles et nombres, listes etc.

Für *Mathematica* ist ein Buchstabe oder eine Buchstabenfolge ein Symbol, z.B. eine Variable. Variablen können auch mit "$" beginnen. "7a" jedoch bedeutet "7 mal a". Probiere aus:

■ Pour *Mathematica* une lettre ou une suite de lettres sont un symbole, p.ex. une variable. Les variables peuvent aussi commencer par "$". "7a" par contre signifie "7 fois a". Essaie:

```
In[25]:= 7a
Out[25]= 7 a

In[26]:= a = 5
Out[26]= 5

In[27]:= 7a
Out[27]= 35
```

In "Miscellaneus/Units.nb" sind Einheiten und Funktionen zur Einheitenverwandlung gespeichert:

■ Dans "Miscellaneos/Units.nb" sont mémorisées les unités et les fonctions pour transformer les unités:

```
In[28]:= Needs["Miscellaneous`Units`"]

In[29]:= 120 Pound
Out[29]= 120 Pound

In[30]:= Convert[120 Pound, Kilogram]
Out[30]= 54.431 Kilogram
```

Wichtige transzendente Zahlen sind als Symbole gespeichert.
Probiere aus:
■ Les nombres transcendants importants sont mémorisés comme symboles.
Essaie:

```
In[31]:= E
Out[31]= e
In[32]:= N[E]
Out[32]= 2.71828
In[33]:= I
Out[33]= i
In[34]:= I^2
Out[34]= -1
In[35]:= Degree
Out[35]= °
In[36]:= N[Sin[Pi/4]]
Out[36]= 0.707107
In[37]:= N[Sin[45 Degree]]
Out[37]= 0.707107
```

Mathematica kann weiter mit Listen (Vektoren, Matrizen, Tabellen etc rechnen). Der Themenkreis wird später behandelt.
■ *Mathematica* sait aussi calculer avec des listes (vecteurs, matrices tableaux etc.) Cet ensemble de thèmes sera traité plus tard.

```
In[38]:= ??List
        {e1, e2, ... } is a list of elements. Mehr…
        Attributes[List] = {Locked, Protected}
```

2.7. Approximationen
■ Approximations

Rationale Zahlen sind exakt gespeichert. "N" bedeutet approximieren. Probiere aus:
■ Les nombres rationaux sont sauvés précisément. "N" signifie approximer. Essaie:

```
In[39]:= Sqrt[17]
Out[39]= √17
```

In[40]:= **N[%]**

Out[40]= 4.12311

In[41]:= **17^(1/2)**

Out[41]= $\sqrt{17}$

In[42]:= **% // N**

Out[42]= 4.12311

In[43]:= **17^(0.5)**

Out[43]= 4.12311

In[44]:= **N[Sqrt[17]]**

Out[44]= 4.12311

In[45]:= **Sqrt[17] // N**

Out[45]= 4.12311

In[46]:= **Precision[%]**

Out[46]= MachinePrecision

Wie gross ist also die Präzision? Die Präzision kann "beliebig" vergrðssert werden:
■ Combien la précision est-elle grande? La précision peut être agrandie "autant qu'on veut":

In[47]:= **N[Sqrt[17], 100]**

Out[47]=
```
4.12310562561766054982140985597407702514719922537362043439863357309495434633762159358786365081068429
7
```

Wie gross ist hier die Präzision? ■ Combien la précision est-elle grande?

In[48]:= **Pi**

Out[48]= π

In[49]:= **N[Pi]**

Out[49]= 3.14159

In[50]:= **N[Pi,500]**

Out[50]=
```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899
8628034825342117067982148086513282306647093844609550582231725359408128481117450
2841027019385211055596446229489549303819644288109756659334461284756482337867831
6
5271201909145648566923460348610454326648213393607260249141273724587006606315588
1
7488152092096282925409171536436789259036001133053054882046652138414695194151160
9
4330572703657595919530921861173819326117931051185480744623799627495673518857527
2
48912279381830119491
```

In[51]:= **N[E,100]**

Out[51]=
```
2.71828182845904523536028747135266249775724709369995957496696762772407663035354759
4571382178525166427
```

2.7. 1. Exakt versus approximiert:
▮ Exact versus approximé:

Probiere aus: ▮ Essaie:

```
In[52]:= 3/10 + 1/2
```

$$\text{Out[52]= } \frac{4}{5}$$

```
In[53]:= Precision[%]
```

$$\text{Out[53]= } \infty$$

```
In[54]:= .36 + 1/3
Out[54]= 0.693333

In[55]:= Precision[%]
Out[55]= MachinePrecision
```

2.7.2. Konvertierung approximativer Werte in exakte Werte:
▮ Transformation des valeurs approximatives en valeurs exactes:

Lasse Dir mit der Help-Funktion folgende Funktionen erklären:
▮ Fais-toi expliquer par la fonction Help les fonctions suivantes:

```
In[56]:= ?Rationalize

        Rationalize[x] takes Real numbers in x that are close to rationals, and
           converts them to exact Rational numbers. Rationalize[x, dx] performs the
           conversion whenever the error made is smaller in magnitude than dx. Mehr…

In[57]:= ?Round

        Round[x] gives the integer closest to x. Mehr…

In[58]:= ?Chop

        Chop[expr] replaces approximate real numbers
           in expr that are close to zero by the exact integer 0. Mehr…

In[59]:= ?Floor

        Floor[x] gives the greatest integer less than or equal to x. Mehr…

In[60]:= ?Ceiling

        Ceiling[x] gives the smallest integer greater than or equal to x. Mehr…
```

Probiere aus: ▮ Essaie:

```
In[61]:= Rationalize[3.1416]
```

$$\text{Out[61]= } \frac{3927}{1250}$$

In[62]:= **Rationalize[3.1415926536]**

Out[62]= 3.14159

In[63]:= **Rationalize[3.1415926536,0]**

Out[63]= $\frac{228350988}{72686377}$

In[64]:= **Round[2.57432]**

Out[64]= 3

In[65]:= **Round[2,45892]**

Round::argx : Round called with 2 arguments; 1 argument is expected. Mehr…

Out[65]= Round[2, 45892]

Was war los ? ■ Que se passe-t-il?

In[66]:= **Round[Sqrt[17]]**

Out[66]= 4

In[67]:= **Round[N[Sqrt[17]]]**

Out[67]= 4

In[68]:= **Chop[0.00000000002]**

Out[68]= 0

In[69]:= **Ceiling[3.4]**

Out[69]= 4

In[70]:= **Floor[3.4]**

Out[70]= 3

2.7.3. Arbeiten auf einem selber fixierten Präzisions-Niveau: ■ Travailler à un niveau de précision fixé par soi-même:

Man kann z.B. selbst eine Funktion definieren, die die auszugebenden Kommastellen festlegt:
■ On peut définir soi-même par exemple une fonction qui détermine le nombre de places dernière la virgule à émettre:

In[71]:= **n30[x_]:=N[x,30]**

Anwendung: ■ Application:

In[72]:= **n30[5/7]**

Out[72]= 0.714285714285714285714285714286

In[73]:= **N[5/7]**

Out[73]= 0.714286

Es gibt aber auch die globale Variable "$Post", in die sich eine Funktion eingeben lässt. Diese Variable wird auf jeden auszugebenen Ausdruck angewandt:
■ Mais il existe aussi la variable globale "$Post", dans laquelle on peut entrer une fonction. Cette variable est appliquée à chaque expression à émettre:

```
In[74]:= ?$Post

        $Post is a global variable whose value, if set, is applied to every output expression. Mehr...

In[75]:= $Post=n30

Out[75]= n30

In[76]:= Sqrt[3]

Out[76]= 1.732050807568877293527446341 51

In[77]:= Precision[%]

Out[77]= 30.
```

Der Inhalt von $Post soll auch wieder gelöscht werden können. Das geschieht durch Eingabe des Zeichens für "missing value", nämlich "." :
■ Il faut pouvoir aussi effacer le contenu de $Post. On peut faire cela en entrant le signe pour "missing value", c'est-à-dite "." :

```
In[78]:= $Post=.

In[79]:= Sqrt[3]

Out[79]= √3

In[80]:= N[Sqrt[3]]

Out[80]= 1.73205

In[81]:= N[Pi,25]

Out[81]= 3.1415926535897932384626 43

In[82]:= N[%,200]

Out[82]= 3.1415926535897932384626 43
```

Wie genau ist also die übliche Präzision?
■ Combien la précision courante est-elle exacte?

2.8. Formatierte Zahlenausgabe
■ Sortie formatée de nombres

Mathematica bietet Möglichkeiten, die Zahlenausgabe zu formatieren. So lässt sich die Leserlichkeit verbessern. Probiere aus:
Mathematica offre la possibilité de formater la sortie de nombres. $Ainsi on peut améliorer la lisibilité. Essaie:

In[83]:=
```
Options[NumberForm]
```

Out[83]=
```
{DigitBlock → ∞, ExponentFunction → Automatic, ExponentStep → 1,
 NumberFormat → Automatic, NumberMultiplier → ×, NumberPadding → {, },
 NumberPoint → ., NumberSeparator → ,, NumberSigns → {-, }, SignPadding → False}
```

In[84]:=
```
1.23456789 10^20
```

Out[84]=
$$1.23457 \times 10^{20}$$

In[85]:=
```
NumberForm[1.23456789 10^20,
        ExponentStep -> 19,
        NumberSigns  -> {"-","+"}
            ]
```

Out[85]//NumberForm=
$$+12.3457 \times 10^{19}$$

In[86]:=
```
NumberForm[1.23456789 10^20,
        ExponentStep -> 21,
        NumberSigns  -> {"-","+"}
            ]
```

```
NumberForm::sigz :
 In addition to the number of digits requested, one or more zeros will appear as placeholders. Mehr…
```

Out[86]//NumberForm=
```
+123456789000000000000.
```

In[87]:=
```
NumberForm[123456789 10^20,
        ExponentStep -> 1000,
        NumberSigns  -> {"-","+"}
            ]
```

Out[87]//NumberForm=
```
+12345678900000000000000000000
```

In[88]:=
```
BaseForm[1.4,2]
```

Out[88]//BaseForm=
$$1.0110011001100110011_{2}$$

In[89]:=
```
BaseForm[1.4,3]
```

Out[89]//BaseForm=
$$1.101210121012_{3}$$

In[90]:=
```
BaseForm[1.4,4]
```

Out[90]//BaseForm=
$$1.121212122_{4}$$

In[91]:=
```
BaseForm[1.4,5]
```

Out[91]//BaseForm=
$$1.2_{5}$$

In[92]:=
```
BaseForm[1.4,6]
```

Out[92]//BaseForm=
$$1.2222222_{6}$$

In[93]:= **BaseForm[1.4,7]**

Out[93]//BaseForm=
1.254126_7

In[94]:= **BaseForm[1.4,10]**

Out[94]//BaseForm=
1.4

In[95]:= **BaseForm[1.4,20]**

Out[95]//BaseForm=
1.8_{20}

Was war los ? ■ Que se passe-t-il?

2.9. Wichtige gespeicherte Konstanten ■ Constantes importantes mémorisées

Falls Dir eine der folgenden Konstanten unbekannt ist, so schlage in deinem Formelbuch nach. Probiere aus:
■ Si tu ne connais pas l'une des constantes suivantes, consulte ton manuel de formules. Essaie:

In[96]:= **??Catalan**

Catalan is Catalan's constant, with numerical value approximately equal to 0.915966. Mehr…

Attributes[Catalan] = {Constant, Protected}

In[97]:= **?Degree**

Degree gives the number of radians in one degree. It
has a numerical value of Pi/180. It is also a unit multiplier. Mehr…

In[98]:= **?E**

E is the exponential constant e (base of natural
logarithms), with numerical value approximately equal to 2.71828. Mehr…

In[99]:= **?EulerGamma**

EulerGamma is Euler's constant gamma, with numerical value approximately equal to 0.577216.
Mehr…

In[100]:=
?GoldenRatio

GoldenRatio is the golden ratio (1 + Sqrt[
5])/2, with numerical value approximately equal to 1.61803. Mehr…

In[101]:=
?I

I represents the imaginary unit Sqrt[-1]. Mehr…

In[102]:=
?Pi

Pi is the constant pi, with numerical value approximately equal to 3.14159. Mehr…

```
In[103]:=
Pi
```

```
Out[103]=
π
```

```
In[104]:=
NumberForm[N[Pi,45],
        NumberSeparator  -> " ",
        DigitBlock -> 5
            ]
```

```
Out[104]//NumberForm=
3.14159 26535 89793 23846 26433 83279 50288 41971 6940
```

```
In[105]:=
Log[E]
```

```
Out[105]=
1
```

```
In[106]:=
Cos[Pi/3]
```

```
Out[106]=
1/2
```

```
In[107]:=
N[Sin[45 Degree]]
```

```
Out[107]=
0.707107
```

```
In[108]:=
avogadro = 6.02250 10^23
```

```
Out[108]=
6.0225 × 10^23
```

```
In[109]:=
Log[avogadro]
```

```
Out[109]=
54.755
```

2.10. Zufallszahlen
∎ Nombres aléatoires

Mathematica besitzt die Funktion "Random", die gleichverteilte Pseudozufallszahlen generiert. Solche braucht man z.B. in der Statistik, um Testdaten zu erzeugen. Probiere aus:

∎ *Mathematica* possède la fonction "Random", qui génère des nombres pseudo-aléatoires en distribution homogène. On les emploie p.ex. dans la statistique, pour créer les données de test. Essaie:

In[110]:=
```
Random[]
```
Out[110]=
```
0.771439
```

In[111]:=
```
Random[Integer]
```
Out[111]=
```
1
```

In[112]:=
```
Random[Integer, {0, 100}]
```
Out[112]=
```
36
```

In[113]:=
```
Random[Complex, {1 + 3I, 4 + 7I}]
```
Out[113]=
```
1.93288 + 5.78143 i
```

Die Packages "ContinousDistributions.nb" und "DiscreteDistributions.nb" enthalten Definitionen, mit denen sich andere als gleichverteilte Pseudozufallszahlen erzeugen lassen. Probiere aus:

■ Les Packages "ContinousDistributions.nb" et "DiscreteDistributions.nb" contiennent des définitions par lesquelles on peut générer des nombres pseudo-aléatoires autres que distribués homogènement. Essaie:

In[114]:=
```
Needs["Statistics`ContinuousDistributions`"]
```

In[115]:=
```
Random[NormalDistribution[5]]
```
Out[115]=
```
4.77034
```

Der Pseudozufallszahlengenerator benutzt die Zeit seit dem Einschalten der Maschine als Variable zur Berechnung der Zahlen. Mit "SeedRandom" lässt sich der Generator wieder "zurücksetzen". Das kann benutzt werden, um mehrmals dieselben Zahlen zu generieren. Probiere aus:

■ Le générateur des nombres pseudo-aléatoires utilise le laps de temps dès le démarrage de la machine comme variable pour calculer les nombres. Par "SeedRandom" on peut "remettre en arrière" le générateur. On employe cela pour générer plusieurs fois les mêmes nombres. Essaie:

In[116]:=
```
??SeedRandom
```
```
SeedRandom[n] resets the pseudorandom number generator, using the integer n as a seed.
   SeedRandom[ ] resets the generator, using as a seed the time of day. Mehr...

Attributes[SeedRandom] = {Protected}
```

In[117]:=
```
SeedRandom[2]
```

In[118]:=
```
{Random[], Random[]}
```
Out[118]=
```
{0.238705, 0.844529}
```

In[119]:=
```
SeedRandom[2]
```

In[120]:=
```
{Random[], Random[]}
```
Out[120]=
```
{0.238705, 0.844529}
```

2.11. Iteratoren
▮ Itérateurs

Mathematica stellt iterative Funktionen (Iteratoren) zur Verfügung, um die Eingabe wiederholter Rechnungen abzukürzen. Einige sind hier gezeigt. Probiere aus:

▮ *Mathematica* met à disposition des fonctions itératives pour abréger l'entrée de calculs répétés. Voici quelques-uns. Essaie:

In[121]:=
```
??Table
```
```
Table[expr, {imax}] generates a list of imax copies of expr. Table[
   expr, {i, imax}] generates a list of the values of expr when i runs from
   1 to imax. Table[expr, {i, imin, imax}] starts with i = imin. Table[expr,
   {i, imin, imax, di}] uses steps di. Table[expr, {i, imin, imax}, {j, jmin,
   jmax}, ... ] gives a nested list. The list associated with i is outermost. Mehr...

Attributes[Table] = {HoldAll, Protected}
```

In[122]:=
```
Table[x, {5}]
```
Out[122]=
```
{x, x, x, x, x}
```

In[123]:=
```
??Product
```
```
Product[f, {i, imax}] evaluates the product of the expressions f as evaluated
   for each i from 1 to imax. Product[f, {i, imin, imax}] starts with i = imin.
   Product[f, {i, imin, imax, di}] uses steps di. Product[f, {i, imin, imax},
   {j, jmin, jmax}, ... ] evaluates a product over multiple indices. Mehr...

Attributes[Product] = {HoldAll, Protected, ReadProtected}
```

In[124]:=
```
Product[x + k y, {k, 4}]
```
Out[124]=
$$(x + y)\,(x + 2\,y)\,(x + 3\,y)\,(x + 4\,y)$$

In[125]:=

```
??Do
```

```
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. Mehr...

Attributes[Do] = {HoldAll, Protected}
```

In[126]:=

```
Do[Print["k: ", k], {k, 7, 9}]
```

```
k: 7

k: 8

k: 9
```

In[127]:=

```
??TableForm
```

```
TableForm[list] prints with the elements of list arranged in an array of rectangular cells.
  Mehr...

Attributes[TableForm] = {Protected, ReadProtected}

Options[TableForm] := {TableAlignments → Automatic, TableDepth → ∞,
  TableDirections → Column, TableHeadings → None, TableSpacing → Automatic}
```

In[128]:=

```
TableForm[Table[{k, 10^(-k), Chop[N[10^(-k)]]}, {k, 7, 14}]]
```

Out[128]//TableForm=

7	$\frac{1}{10000000}$	$1.\times 10^{-7}$
8	$\frac{1}{100000000}$	$1.\times 10^{-8}$
9	$\frac{1}{1000000000}$	$1.\times 10^{-9}$
10	$\frac{1}{10000000000}$	$1.\times 10^{-10}$
11	$\frac{1}{100000000000}$	0
12	$\frac{1}{1000000000000}$	0
13	$\frac{1}{10000000000000}$	0
14	$\frac{1}{100000000000000}$	0

In[129]:=

```
??Sum
```

```
Sum[f, {i, imax}] evaluates the sum of the expressions f as
   evaluated for each i from 1 to imax. Sum[f, {i, imin, imax}] starts with
   i = imin. Sum[f, {i, imin, imax, di}] uses steps di. Sum[f, {i, imin,
   imax}, {j, jmin, jmax}, ... ] evaluates a sum over multiple indices. Mehr...

Attributes[Sum] = {HoldAll, Protected, ReadProtected}
```

In[130]:=

```
Sum[i x^i, {i, 3, 23, 5}]
```

Out[130]=

$$3\,x^3 + 8\,x^8 + 13\,x^{13} + 18\,x^{18} + 23\,x^{23}$$

2.12. Matrizenrechnung
▮ Calcul des matrices

Probiere aus: ▮ Essaie:

In[131]:=
```
??MatrixForm
```

```
MatrixForm[list] prints with the elements of list arranged in a regular array. Mehr...

Attributes[MatrixForm] = {Protected, ReadProtected}

Options[MatrixForm] := {TableAlignments → Automatic, TableDepth → ∞,
  TableDirections → Column, TableHeadings → None, TableSpacing → Automatic}
```

In[132]:=
```
??Inverse
```

```
Inverse[m] gives the inverse of a square matrix m. Mehr...

Attributes[Inverse] = {Protected}

Options[Inverse] = {Method → Automatic, Modulus → 0, ZeroTest → (#1 == 0 &)}
```

In[133]:=
```
??Det
```

```
Det[m] gives the determinant of the square matrix m. Mehr...

Attributes[Det] = {Protected}

Options[Det] = {Modulus → 0}
```

In[134]:=
```
??Eigenvalues
```

```
Eigenvalues[m] gives a list of the eigenvalues of the square
   matrix m. Eigenvalues[{m, a}] gives the generalized eigenvalues of m with
   respect to a. Eigenvalues[m, k] gives the first k eigenvalues of m. Mehr...

Attributes[Eigenvalues] = {Protected}

Options[Eigenvalues] = {Cubics → False, Method → Automatic, Quartics → False}
```

In[135]:=
```
??Eigenvectors
```

```
Eigenvectors[m] gives a list of the eigenvectors of the square
   matrix m. Eigenvectors[{m, a}] gives the generalized eigenvectors of m with
   respect to a. Eigenvectors[m, k] gives the first k eigenvectors of m. Mehr...

Attributes[Eigenvectors] = {Protected}

Options[Eigenvectors] =
 {Cubics → False, Method → Automatic, Quartics → False, ZeroTest → Automatic}
```

In[136]:=
```
??LinearSolve
```

```
LinearSolve[m, b] finds an x which solves the matrix
   equation m.x==b. LinearSolve[m] generates a LinearSolveFunction[ ... ]
   which can be applied repeatedly to different b. Mehr...

Attributes[LinearSolve] = {Protected}

Options[LinearSolve] = {Method → Automatic, Modulus → 0, ZeroTest → Automatic}
```

In[137]:=
```
??NullSpace
```

```
NullSpace[m] gives a list of vectors that forms a basis for the null space of the matrix m.
  Mehr...

Attributes[NullSpace] = {Protected}

Options[NullSpace] =
  {Method → Automatic, Modulus → 0, Tolerance → Automatic, ZeroTest → Automatic}
```

In[138]:=
```
??RowReduce
```

```
RowReduce[m] gives the row-reduced form of the matrix m. Mehr...

Attributes[RowReduce] = {Protected}

Options[RowReduce] =
  {Method → Automatic, Modulus → 0, Tolerance → Automatic, ZeroTest → Automatic}
```

In[139]:=
```
??Transpose
```

```
Transpose[list] transposes the first two levels in list.
   Transpose[list, {n1, n2, ... }] transposes list so that the levels 1,
   2, ... in list correspond to levels n1, n2, ... in the result. Mehr...

Attributes[Transpose] = {Protected}
```

In[140]:=
```
??IdentityMatrix
```

```
IdentityMatrix[n] gives the n by n identity matrix. Mehr...

Attributes[IdentityMatrix] = {Protected}
```

In[141]:=
```
m1 = {{1, 2},{3, 4}}
```

Out[141]=
```
{{1, 2}, {3, 4}}
```

In[142]:=
```
Transpose[m1] // MatrixForm
```

Out[142]//MatrixForm=

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

In[143]:=
```
MatrixForm[m1]
```
Out[143]//MatrixForm=
$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

In[144]:=
```
MatrixForm[{{1, 2},{3, 4}}]
```
Out[144]//MatrixForm=
$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

In[145]:=
```
m2 = {{0, -1},{8, -6}}
```
Out[145]=
```
{{0, -1}, {8, -6}}
```

In[146]:=
```
m3 = {{0, -1},{8, -6}} // MatrixForm
```
Out[146]//MatrixForm=
$$\begin{pmatrix} 0 & -1 \\ 8 & -6 \end{pmatrix}$$

Das Matrixprodukt wird mit einem gewöhnlichen Punkt geschrieben. Probiere aus:

■ Le produit de matrices s'écrit avec un simple point. Essaie:

In[147]:=
```
m1.m2 // MatrixForm
```
Out[147]//MatrixForm=
$$\begin{pmatrix} 16 & -13 \\ 32 & -27 \end{pmatrix}$$

In[148]:=
```
m1.m3
```
Out[148]=
$$\{\{1, 2\}, \{3, 4\}\}.\begin{pmatrix} 0 & -1 \\ 8 & -6 \end{pmatrix}$$

Elementweise Multiplikation. Probiere aus:

■ Multiplication par élément. Essaie:

In[149]:=
```
m1 m2 // MatrixForm
```
Out[149]//MatrixForm=
$$\begin{pmatrix} 0 & -2 \\ 24 & -24 \end{pmatrix}$$

Probiere aus: ■ Essaie:

In[150]:=
```
?NullSpace
```
```
NullSpace[m] gives a list of vectors that forms a basis for the null space of the matrix m.
```
Mehr…

In[151]:=
```
m4 = {{1,2,3},{1,2,3},{0,0,1}}
```

Out[151]=
```
{{1, 2, 3}, {1, 2, 3}, {0, 0, 1}}
```

In[152]:=
```
NullSpace[m4]
```

Out[152]=
```
{{-2, 1, 0}}
```

In[153]:=
```
?RowReduce
```

```
RowReduce[m] gives the row-reduced form of the matrix m. Mehr...
```

In[154]:=
```
RowReduce[m4] // MatrixForm
```

Out[154]//MatrixForm=

$$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

In[155]:=
```
Eigenvectors[m1]
```

Out[155]=

$$\left\{\left\{-\frac{4}{3}+\frac{1}{6}\left(5+\sqrt{33}\right), 1\right\}, \left\{-\frac{4}{3}+\frac{1}{6}\left(5-\sqrt{33}\right), 1\right\}\right\}$$

In[156]:=
```
Transpose[m1] // MatrixForm
```

Out[156]//MatrixForm=

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

In[157]:=
```
m = Table[Random[], {3}, {3}]
```

Out[157]=
```
{{0.473928, 0.421515, 0.512692},
 {0.0228035, 0.529257, 0.0507012}, {0.74907, 0.632356, 0.0725195}}
```

In[158]:=
```
MatrixForm[m]
```

Out[158]//MatrixForm=

$$\begin{pmatrix} 0.473928 & 0.421515 & 0.512692 \\ 0.0228035 & 0.529257 & 0.0507012 \\ 0.74907 & 0.632356 & 0.0725195 \end{pmatrix}$$

In[159]:=
```
mInv = Inverse[m]
```

Out[159]=
```
{{-0.0355956, -1.65375, 1.40785},
 {-0.204582, 1.96936, 0.0694849}, {2.15159, -0.0904116, -1.35854}}
```

In[160]:=
```
MatrixForm[%]
```

Out[160]//MatrixForm=

$$\begin{pmatrix} -0.0355956 & -1.65375 & 1.40785 \\ -0.204582 & 1.96936 & 0.0694849 \\ 2.15159 & -0.0904116 & -1.35854 \end{pmatrix}$$

In[161]:=
```
m.mInv
```

Out[161]=

$$\{\{1., 6.93889 \times 10^{-17}, 0.\}, \{-1.38778 \times 10^{-17}, 1., 1.38778 \times 10^{-17}\}, \{-2.77556 \times 10^{-17}, 1.17961 \times 10^{-16}, 1.\}\}$$

In[162]:=
```
MatrixForm[%]
```

Out[162]//MatrixForm=

$$\begin{pmatrix} 1. & 6.93889 \times 10^{-17} & 0. \\ -1.38778 \times 10^{-17} & 1. & 1.38778 \times 10^{-17} \\ -2.77556 \times 10^{-17} & 1.17961 \times 10^{-16} & 1. \end{pmatrix}$$

In[163]:=
```
Chop[%]
```

Out[163]=

$$\{\{1., 0, 0\}, \{0, 1., 0\}, \{0, 0, 1.\}\}$$

In[164]:=
```
MatrixForm[%]
```

Out[164]//MatrixForm=

$$\begin{pmatrix} 1. & 0 & 0 \\ 0 & 1. & 0 \\ 0 & 0 & 1. \end{pmatrix}$$

In[165]:=
```
IdentityMatrix[7] // MatrixForm
```

Out[165]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

2.13. Lösen von Gleichungen
∎ Résoudre des équations

Probiere aus: ∎ Essaie:

In[166]:=
```
??Solve
```

```
Solve[eqns, vars] attempts to solve an equation or set of
   equations for the variables vars. Solve[eqns, vars, elims] attempts to
   solve the equations for vars, eliminating the variables elims. Mehr...

Attributes[Solve] = {Protected}

Options[Solve] = {InverseFunctions → Automatic, MakeRules → False, Method → 3,
  Mode → Generic, Sort → True, VerifySolutions → Automatic, WorkingPrecision → ∞}
```

In[167]:=
```
??NRoots
```

```
NRoots[lhs==rhs, var] gives a list of numerical approximations to the roots of a polynomial
  equation. NRoots[lhs==rhs, var, n] uses n-digit precision in the computations.

Attributes[NRoots] = {Protected}
```

In[168]:=
```
??FindRoot
```

```
FindRoot[lhs==rhs, {x, x0}] searches for a numerical solution to the equation lhs==
   rhs, starting with x=x0. FindRoot[{eqn1, eqn2, ... }, {{x, x0}, {y, y0}, ... }]
   searches for a numerical solution to the simultaneous equations eqni. Mehr...

Attributes[FindRoot] = {HoldAll, Protected}

Options[FindRoot] = {AccuracyGoal → Automatic, Compiled → True, DampingFactor → 1,
  EvaluationMonitor → None, Jacobian → Automatic, MaxIterations → 100, Method → Automatic,
  PrecisionGoal → Automatic, StepMonitor → None, WorkingPrecision → MachinePrecision}
```

In[169]:=
```
Solve[{2x - 3y ==8, 4x + y == 6},{x, y}]
```

Out[169]=
$$\left\{\left\{x \to \frac{13}{7},\ y \to -\frac{10}{7}\right\}\right\}$$

In[170]:=
```
Solve[m1.{{x},{y}}=={{2},{5}}, {x, y}]
```

Out[170]=
$$\left\{\left\{x \to 1,\ y \to \frac{1}{2}\right\}\right\}$$

In[171]:=
```
NRoots[x^4 +3 x^2 + 5x == 7, x]
```

Out[171]=
```
x == -1.63541 || x == 0.390304 - 2.2034 i || x == 0.390304 + 2.2034 i || x == 0.854804
```

FindRoot benutzt die Newton-Methode. Darin wird die Ableitung benutzt:

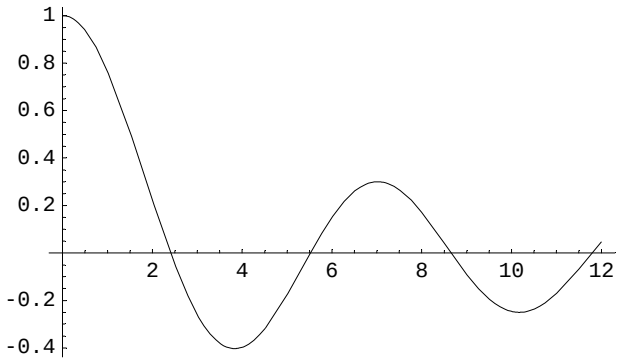
■ FindRoot emploie la méthode de Newton. On y employe la déduction:

In[172]:=
```
FindRoot[Sin[x]/x == 0, {x, 2}]
```

Out[172]=
```
{x → 3.14159}
```

Bessel-Funktionen: ■ Fonctions de Bessel

In[173]:=
```
Plot[BesselJ[0, x], {x, 0, 12}];
```



In[174]:=
```
FindRoot[BesselJ[0, x]/x == 0, {x, 12}]
```

Out[174]=
```
{x → 11.7915}
```

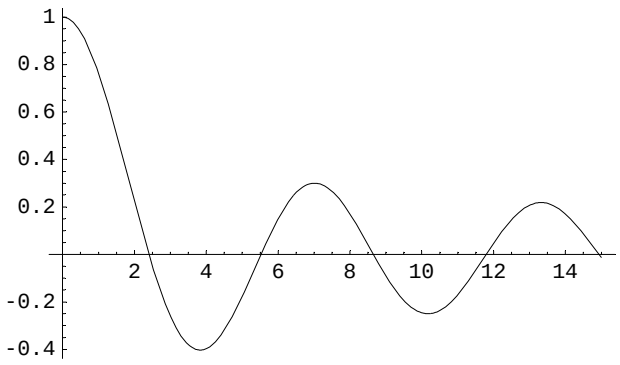
In[175]:=
```
FindRoot[BesselJ[0, x]/x, {x, 12}]
```

Out[175]=
```
{x → 11.7915}
```

In[176]:=
```
FindRoot[BesselJ[0, x]/x, {x, 7}]
```

Out[176]=
```
{x → 14.9309}
```

In[177]:=
```
Plot[BesselJ[0, x], {x, 0, 15}];
```

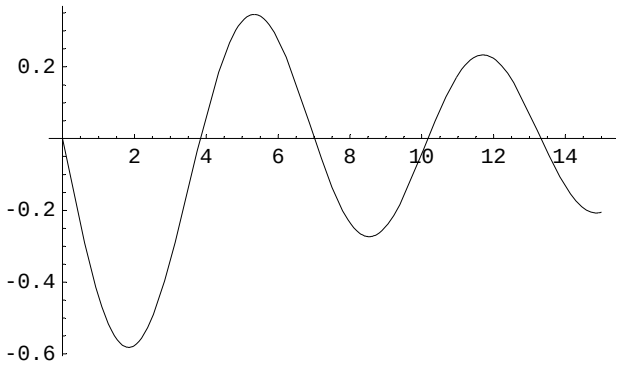


FindRoot benutzt die Newton-Methode. Darin wird die Ableitung benutzt. Hier ein Plot der Ableitung obiger Bessel-Funktion. (Beachte, dass die Ableitung bei x = 7 klein ist.)

FindRoot emploie la méthode de Newton. On y employe la déduction. Voici un plot de la déduction de la fonction de Bessel ci-dessus. (Remarque que la déduction est petite pour x = 7.)

In[178]:=

```
Plot[-BesselJ[1, x], {x, 0, 15}];
```



2.14. Numerische Integration etc.
■ Intégtation numérique etc.

Probiere aus: ■ Essaie:

In[179]:=

```
??NIntegrate
```

```
NIntegrate[f, {x, xmin, xmax}] gives a numerical approximation
   to the integral of f with respect to x from xmin to xmax. Mehr...

Attributes[NIntegrate] = {HoldAll, Protected}

Options[NIntegrate] =
 {AccuracyGoal → ∞, Compiled → True, EvaluationMonitor → None, GaussPoints → Automatic,
  MaxPoints → Automatic, MaxRecursion → 6, Method → Automatic, MinRecursion → 0,
  PrecisionGoal → Automatic, SingularityDepth → 4, WorkingPrecision → MachinePrecision}
```

In[180]:=

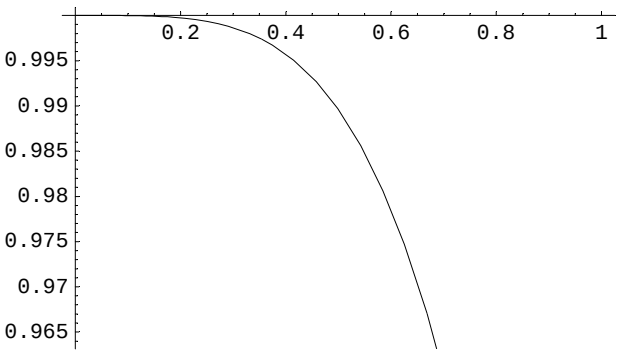
```
NIntegrate[Sin[x^2]/x^2, {x, 0, 1}]
```

Out[180]=

```
0.967577
```

In[181]:=

```
Plot[Sin[x^2]/x^2, {x, 0, 1}];
```



In[182]:=
```
NIntegrate[Sin[Sin[Cos[x^10]]], {x, 0, 1}]
```

Out[182]=
```
0.735169
```

In[183]:=
```
??NSum
```

```
NSum[f, {i, imin, imax}] gives a numerical approximation
   to the sum of the expressions f as evaluated for each i from imin to
   imax. NSum[f, {i, imin, imax, di}] uses a step di in the sum. Mehr...

Attributes[NSum] = {HoldAll, Protected}

Options[NSum] = {AccuracyGoal → ∞, Compiled → True, EvaluationMonitor → None,
  Method → Automatic, NSumExtraTerms → 12, NSumTerms → 15, PrecisionGoal → Automatic,
  VerifyConvergence → True, WorkingPrecision → MachinePrecision, WynnDegree → 1}
```

In[184]:=
```
NSum[(n^2 - n + 1) Log[n]/n^3, {n, 1, 100}]
```

Out[184]=
```
9.87012 + 0. i
```

In[185]:=
```
??NProduct
```

```
NProduct[f, {i, imin, imax}] gives a numerical approximation to the
   product of the expressions f as evaluated for each i from imin to imax.
   NProduct[f, {i, imin, imax, di}] uses a step di in the product. Mehr...

Attributes[NProduct] = {HoldAll, Protected}

Options[NProduct] = {AccuracyGoal → ∞, Compiled → True, EvaluationMonitor → None,
  Method → Automatic, NProductExtraFactors → 15, NProductFactors → 15, PrecisionGoal → Automatic,
  VerifyConvergence → True, WorkingPrecision → MachinePrecision, WynnDegree → 1}
```

In[186]:=
```
NProduct[(1 + 1/n)^n, {n, 1, 100}]
```

Out[186]=
```
2.89824 × 10^42 + 0. i
```

In[187]:=
```
Limit[(1 + 1/n)^n, n->Infinity]
```

Out[187]=
```
e
```

2.15. Numerische Lösung von Differentialgleichungen ▮ Solution numérique d'équations différentielles

Probiere aus: ▮ Essaie:

In[188]:=

```
??NDSolve
```

```
NDSolve[eqns, y, {x, xmin, xmax}] finds a numerical solution to the ordinary differential
   equations eqns for the function y with the independent variable x in the
   range xmin to xmax. NDSolve[eqns, y, {x, xmin, xmax}, {t, tmin, tmax}] finds a
   numerical solution to the partial differential equations eqns. NDSolve[eqns, {y1,
   y2, ... }, {x, xmin, xmax}] finds numerical solutions for the functions yi. Mehr...

Attributes[NDSolve] = {Protected}

Options[NDSolve] = {AccuracyGoal → Automatic, Compiled → True, DependentVariables → Automatic,
  EvaluationMonitor → None, MaxStepFraction → 1/10, MaxSteps → Automatic, MaxStepSize → Automatic,
  Method → Automatic, NormFunction → Automatic, PrecisionGoal → Automatic, SolveDelayed → False,
  StartingStepSize → Automatic, StepMonitor → None, WorkingPrecision → MachinePrecision}
```

In[189]:=

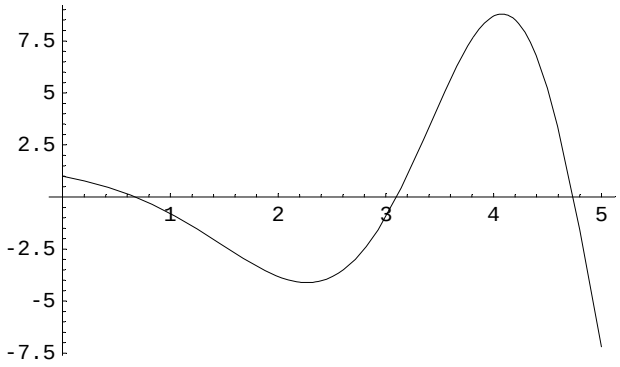
```
NDSolve[{y''[x] - y'[x] + x y[x] == 0, y'[0] == -1, y[0] == 1},
        y, {x, 0, 5}]
```

Out[189]=

```
{{y → InterpolatingFunction[{{0., 5.}}, <>]}}
```

In[190]:=

```
Plot[y[x] /. %, {x, 0, 5}];
```



"Putzmaschine" einsetzen
▮ Employer la "machine de nettoyage"

In[191]:=

```
(* Old Form: Remove["Global`@*"] *)
```

In[192]:=

```
Remove["Global`*"]
```