Mathematica Kompaktkurs

Inhaltsübersicht

1. Einführung

■ i. Allgemeines

Moderne Computeralgebra−Systeme (CAS) wie *Mathematica*, *Maple* oder *MuPad* sind mächtige Mathe-Programme, die neben dem numerischen Rechnen (mit beliebiger Genauigkeit) auch symbolische Mathematik und die Visualisierung von Ergebnissen beherrschen. Entgegen anderslautenden Bekundungen der Hersteller enthalten diese Pakete zwar nicht das mathematische Wissen der Welt und ersetzen auch keinen Mathematiker, aber sie können einem immerhin viele zeitaufwändige Arbeiten abnehmen.

■ ii. Starten des Programmes und Erste Hilfe

Unter Unix kann *Mathematica* von einem Terminal aus mit dem Kommando "`mathematica`" gestartet werden. Das sollte normalerweise das graphische Benutzerinterface aktivieren. Mathematica ist ein interaktives System, das Eingaben vom Benutzer liest, diese auswertet, und das Ergebnis unmittelbar ausspuckt:

```
In[1]:= 1 + 2
Out[1]= 3
```

> Eingaben werden in *Mathematica* immer durch gleichzeitiges Drücken von SHIFT und ↵ ausgewertet!

Die Ausgabe des Ergebnisses kann man unterdrücken, indem man ein Semikolon an die Eingabe anhängt. Vor allem bei umfangreichen Zwischenergebnissen hilft das, den Überblick zu bewahren.

Dieses Handbuch ist zu kurz, um auch nur die vorgestellten Funktionen von *Mathematica* im Detail zu behandeln. Für umfangreichere Informationen zu bestimmten Themen ist daher die Online−Hilfe unentbehrlich. Informationen zu speziellen Funktionen lassen sich zusätzlich mit "**?**" abrufen. (Abhängig von der verwendeten *Mathematica* Version, erscheinen Teile des Hilfe−Outputsals anklickbare Verknüpfungen zur Online−Hilfeund zum *Mathematica* Buch; wenn Sie Glück und die richtige Version haben wird die folgende Ausgabe mit einem More... abgeschlossen. Falls Sie draufklicken, werden Sie zu detaillierteren Informationen und Beispielen (!) weitergeführt) :

In[2]:=
```
? Log
```

```
Log[z] gives the natural logarithm of z (logarithm
   to base e). Log[b, z] gives the logarithm to base b. More...
```

Um eine Liste aller Funktionen zu erhalten, deren Namen auf ein bestimmtes Muster passen, kann man ein Sternchen in den gesuchten Namen einfügen:

In[3]:=
```
? Eq*
```

System‘

Equal EqualColumns EqualRows EquatedTo

In[4]:=
```
? F*List
```

System‘

FactorList FactorTermsList FixedPointList
FactorSquareFreeList FindList FoldList

■ iii. *Mathematica* als Taschenrechner

Die Notation von Ausdrücken in *Mathematica* orientiert sich an der gebräuchlichen mathematischen Notation. Eine gültige Eingabe ist beispielsweise

In[5]:=
```
(2^12 + 7.1) * (-1.5 / 2.5)
```

Out[5]=
```
-2461.86
```

Es gelten die gebräuchlichen Vorrangsregeln für Operatoren, Ausdrücke können aber auch mit *runden* Klammern explizit gruppiert werden. Funktionsargumente werden in *eckige* Klammern eingeschlossen, um sie von gewöhnlichen Ausdrücken zu unterscheiden:

In[6]:=
```
Sin[3.1] / Sqrt[2]
```

Out[6]=
```
0.029402
```

> Die Namen aller in Mathematica vordefinieren Funktionen, Prozeduren und Konstanten beginnen mit einem Großbuchstaben.

Das Multiplikations−Zeichen kann häufig weggelassen werden, etwa ist `2 Sin[3 x]` gleichbedeutend mit `2 * Sin[3 * x]`. Auch Leerzeichen zwischen zwei (abgeschlossenen) Ausdrücken erwirken die Multikplikation der beiden: `x y` ist also `x * y`, `xy` (ohne Leerzeichen) wird dagegen als *ein* Symbol "`xy`" gewertet.

Auf die letzen beiden Ausgaben kann mit % (letzte Ausgabe) und %% (vorletzte Ausgabe) Bezug genommen werden. Meist ist es aber sinnvoller, wichtige Zwischenergebnisse in Variablen zu speichern, um später auf sie zurückgreifen zu können.

2. Arithmetik

■ i. Exakte und Inexakte Arithmetik

Mathematica unterscheidet zwischen *exakten* und *inexakten*, also genäherten numerischen Ausdrücken. Exakte Rechnungen werden mit beliebiger Genauigkeit durchgeführt, bei inexakten kann es nach einigen Schritten zu Rundungsfehlern kommen. Kommazahlen und die `N`-Funktion erzwingen die numerische Auswertung eines Ausdrucks. `N` nimmt als optionalen Parameter die Anzahl der signifikanten Stellen des Ergebnisses:

```
In[7]:= Factorial[50]
Out[7]= 30414093201713378043612608166064768844377641568960512000000000000
```

```
In[8]:= N[%]
Out[8]= 3.04141 × 10^64
```

```
In[9]:= Sin[Pi / 3]
```

$$\text{Out[9]=}\ \frac{\sqrt{3}}{2}$$

```
In[10]:= N[%, 20]
Out[10]= 0.86602540378443864676
```

```
In[11]:= Sin[Pi / 3.0]
Out[11]= 0.866025
```

Vorsicht: Die Regeln, die *Mathematica* für numerische Rechnungen anwendet, sind kompliziert und können zu überraschenden Ergebnissen führen. Zudem ist umfangreiche Numerik *Mathematica* meist sehr langsam. In solchen Fällen kann es von Vorteil sein, die eigentlichen Berechnungen in C- oder Fortran-Programme auszulagern.

Ein Beispiel, warum man der Numerik von Mathematica (übrigens jeglicher Numerik) nichtimmer trauen sollte:

```
In[12]:= x = 1.11111111111111111111;
         Do[x = 2 * x - x, {60}];
         x == x + 1
Out[14]= True
```

```
In[15]:= Clear[x]
```

■ ii. Komplexe Zahlen

Neben dem Rechnen mit ganzen, rationalen und reellen Zahlen unterstützt Mathematica auch komplexe Arithmetik. Die imaginäre Einheit wird als "`I`" geschrieben.

Mathematica implementiert die für komplexe Arithmetik wichtigen Funktionen Abs[z], Re[z], Im[z], Conjugate[z] und Arg[z]. Achtung: falls z in symbolischer Form als z=x + y I gegeben ist, wird angenommen, dass sowohl x als auch y selbst komplex sind !

```
In[16]:= z = 2 + 3 I
Out[16]= 2 + 3 i
```

In[17]:= **Abs[z]**

Out[17]= $\sqrt{13}$

In[18]:= **Im[z]**

Out[18]= 3

In[19]:= **Conjugate[z]**

Out[19]= 2 − 3 i

In[20]:= **Arg[z]**

Out[20]= ArcTan[3/2]

In[21]:= **z = x + y I**

Out[21]= x + i y

In[22]:= **Re[z]**

Out[22]= −Im[y] + Re[x]

In[23]:= **Simplify[Exp[x] == Exp[x + 2 I Pi]]**

Out[23]= True

Wenn es nötig ist, zwischen exponentieller und trigonometrischer Schreibweise eines Ausdrucks zu wechseln, können `ExpToTrig` und `TrigToExp` verwendet werden:

In[24]:= **ExpToTrig[Exp[I x]]**

Out[24]= Cos[x] + i Sin[x]

In[25]:= **TrigToExp[%^2]**

Out[25]= $e^{2 i x}$

■ iii. Mathematische Funktionen und Konstanten

Alle gebräuchlichen mathematischen Konstanten...

`Pi, pi`	π
`E`	Eulersche Zahl
`I`	Imaginäre Einheit
`Infinity, inf`	Symbol für ∞
`Degree, °`	Umrechnungsfaktor Grad → Rad : $180/\pi$
`Integers, Rationals, Reals`	Zahlenmengen (etwa für Annahmen in `Simplify`)

... und Funktionen sind in Mathematica vordefiniert:

`Sin[x], Cos[x], Tan[x]`	Trigonometrische Funktionen
`Sinh[x], Cosh[x], Tanh[x]`	Hyperbolische Funktionen
`Log[x], Log[b, x], Exp[x]`	Logarithmus (natürlich / zur Basis b), Exponentialfunktion
`Sqrt[x]`	Quadratwurzel
`Random[]`	Zufallszahl zwischen 0 und 1

In[26]:= **?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}. More...
```

In[27]:= **?Arc***

System‘

ArcCos ArcCosh ArcCot ArcCoth ArcCsc ArcCsch ArcSec ArcSech ArcSin ArcSin

Neben diesen Standardfunktionen existieren viele spezielle Funktionen, wie die Gamma−Funktion `Gamma`, die Fehlerfunktion `Erf`, verschiedene Bessel-Funktionen (`BesselJ`, `BesselY`, `BesselI`, `BesselK`) oder die Kugelflächfenfunktionen `SphericalHarmonicY`. Mehr Informationen gibt es in der Online-Hilfe unter "special functions".

3. Fortgeschrittene Anwendungen

■ i. Listen

Mehrere Objekte können mithilfe von *Listen* zusammengefasst werden. Die Grundrechenarten und viele andere Funktionen sind für Listen *elementweise* definiert. Dadurch verhalten sich bspw. Listen von Zahlen wie Vektoren oder "Listen von Listen" wie Matrizen:

In[28]:= **v = {1, 2, 3, 4}**

Out[28]= {1, 2, 3, 4}

In[29]:= **w = -v * 2**

Out[29]= {−2, −4, −6, −8}

In[30]:= **w + v**

Out[30]= {−1, −2, −3, −4}

Zum Erstellen neuer Listen ist die Funktion `Table` häufig praktisch: Folgendes Beispiel liefert eine Liste mit fünf identischen Elementen

In[31]:= **Table[Sin[2], {5}]**

Out[31]= {Sin[2], Sin[2], Sin[2], Sin[2], Sin[2]}

Hier ist ein komplizierteres Beispiel. Das Ergebnis is eine 10×3-Matrix, in der die `n`-te Zeile die Zahlen n, n^2, n^3 enthält:

In[32]:= **potenzen = Table[{n, n^2, n^3}, {n, 10}]**

Out[32]=
```
{{1, 1, 1}, {2, 4, 8}, {3, 9, 27}, {4, 16, 64}, {5, 25, 125},
 {6, 36, 216}, {7, 49, 343}, {8, 64, 512}, {9, 81, 729}, {10, 100, 1000}}
```

Alle Befehle, die ganzzahlige Laufvariablen verwenden (vor allem `Table`, `Do`, `Sum`, `Product`), benutzen dafür dieselbe Syntax. Mögliche Eingabeformen sind

{nmax}	nmax Wiederholungen ohne Laufvariable (erstes Beispiel)
{n,nmax}	n geht von 1 bis nmax in Schritten von 1
{n,nmin,nmax}	n geht von nmin bis nmax in Schritten von 1 (zweites Beispiel)
{n,nmin,nmax,dn}	n geht von nmin bis nmax in Schritten von dn

`Part[x, n]` liefert den n-ten Eintrag einer Liste `x` zurück. Für eine *Matrix* `x` ist das eine Liste, nämlich die n-te *Zeile*. Um die n-te *Spalte* zu erhalten, muss `x` vorher transponiert werden:

```
In[33]:= qzahlen = Part[Transpose[potenzen], 2]
Out[33]= {1, 4, 9, 16, 25, 36, 49, 64, 81, 100}
```

Statt `Part` können auch doppelte eckige Klammern zum Indizieren von Listen verwendet werden. In beiden Fällen hat das erste Element den Index 1 (nicht etwa 0 wie in "C").

```
In[34]:= {potenzen[[3]], potenzen[[4, 3]]}
Out[34]= {{3, 9, 27}, 64}
```

Die Funktion `Apply` wendet eine Funktion auf alle Listenlemente *auf einmal* an

```
In[35]:= Apply[Plus, qzahlen]   (* Summiere alle Elemente *)
Out[35]= 385
```

`Map` dagegen wendet eine Funktion auf jedes Listenelement *einzeln* an

```
In[36]:= Map[Sin, qzahlen]
Out[36]= {Sin[1], Sin[4], Sin[9], Sin[16], Sin[25],
          Sin[36], Sin[49], Sin[64], Sin[81], Sin[100]}
```

■ ii. Selbstdefinierte Variablen und Funktionen

Symbole können verwendet werden, um Funktionen zu definieren oder beliebige Ausdrücke zu speichern. In allen späteren Rechnungen wird das Symbol automatisch durch den zugewiesenen Wert ersetzt:

```
In[37]:= rand = Random[]    (* setze rand auf eine feste Zufallszahl *)
Out[37]= 0.991911

In[38]:= rand * 2
Out[38]= 1.98382
```

Eigene *Funktionen* können auf ähnliche Weise definiert werden:

```
In[39]:= randint[min_, max_] := Random[Integer, {min, max}]

In[40]:= Table[randint[2, 10], {10}]
Out[40]= {4, 9, 10, 3, 7, 5, 6, 4, 4, 8}
```

Zwei Punkte sind in diesem Beispiel wichtig: Der Unterstrich "_" hinter `min` und `max` und das "`:=`" Symbol anstelle von "`=`" bei der Variablendefinition.

Variablen werden mit einem einfachen "`=`" definiert, Funktionen dagegen mit "`:=`". Auf der *linken* Seite der Zuweisung muss an die Argumentnamen einer Funktion ein Unterstrich angehängt werden.

Ungewünschte Definitionen können wieder gelöscht werden

```
In[41]:= Clear[v]          (* ein Symbol löschen *)

In[42]:= Clear[r1, r2]     (* mehrere Symbole löschen *)

In[43]:= Clear["Global‘*"] (* alle selbstdefinierten Symbole löschen *)
```

Eine spezielle Notation für Funktionen ist im Umgang mit Operationen wie `Map` oder `Nest` oft sehr praktisch. *Pure functions* sind Funktionen ohne Namen, die überall dort direkt eingefügt werden können, wo sonst ein Funktionsname erwartet würde. Statt

```
In[44]:= f[x_] = Sin[x]^2
Out[44]= Sin[x]^2

In[45]:= N[Map[f, Range[1, 10]]]
Out[45]= {0.708073, 0.826822, 0.0199149, 0.57275,
          0.919536, 0.078073, 0.431631, 0.97883, 0.169842, 0.295959}
```

kann man mit pure functions auch kürzer schreiben

```
In[46]:= N[Map[Sin[#]^2 &, Range[1, 10]]]
Out[46]= {0.708073, 0.826822, 0.0199149, 0.57275,
          0.919536, 0.078073, 0.431631, 0.97883, 0.169842, 0.295959}
```

Hierbei werden die Funktionsargumente mit `#`, `#2`, `#3`, … bezeichnet und die Definition der Funktion mit einem Ampersand "`&`" beendet.

■ iii. Symbolische Mathematik, Umformen von Ausdrücken

Symbole, denen noch kein Wert zugewiesen wurde, verhalten sich wie *Unbekannte*. Sie können verwendet werden, um allgemeine Berechnungen vorzunehmen

```
In[47]:= Sum[q^n, {n, 1, Infinity}]
Out[47]= -q/(-1 + q)

In[48]:= Integrate[Tan[c * y], y]
Out[48]= -Log[Cos[c y]]/c
```

Einige offensichtliche Vereinfachungen werden automatisch vorgenommen, in komplizierteren Fällen muss man `Simplify` verwenden

```
In[49]:= Simplify[Sin[x]^2 + Cos[x]^2]
Out[49]= 1

In[50]:= Simplify[(a^2 - b^2) / (a - b)]
Out[50]= a + b
```

Manche symbolischen Ausdrücke können nur unter bestimmten Annahmen vereinfacht werden. Zum Beispiel gilt `Sqrt[x^2] == x` nur für reelle `x`. In solchen Fällen müssen die Annahmen explizit an `Simplify` übergeben werden:

```
In[51]:= Simplify[Sqrt[x^2]]
Out[51]= Sqrt[x^2]

In[52]:= Simplify[Sqrt[x^2], x < 0]
Out[52]= -x
```

```
In[53]:= Simplify[Sin[x + 2 Pi n], n ∈ Integers]
Out[53]= Sin[x]
```

Manchmal findet `FullSimplify` zusätzliche Vereinfachungen, die `Simplify` nicht entdeckt

```
In[54]:= Simplify[(I / 4) / E^(2 * I * x) - I / 4 * E^(2 * I * x)]
Out[54]= -(1/4) ⅈ ℯ^(-2 ⅈ x) (-1 + ℯ^(4 ⅈ x))

In[55]:= FullSimplify[%]
Out[55]= Cos[x] Sin[x]
```

`Sqrt[x^2]` konnte in obigem Beispiel nicht vereinfacht werden, da `x` als komplexe Variable angenommen wurde, für die die gewöhnlichen Potenzregeln nur eingeschränkt gelten. Die Anwendung dieser Potenzregeln kann mit `PowerExpand` erzwungen werden

```
In[56]:= PowerExpand[{√(x^2), (a * b)^c, (a^b)^c}]
Out[56]= {x, a^c b^c, a^(b c)}
```

Symbole in Ausdrücken können mithilfe des `/.` Operators ersetzt werden (**unentbehrlich!**)

```
In[57]:= √a /. a → x^2 + y^2
Out[57]= √(x^2 + y^2)

In[58]:= a + b /. {a → b^2, b → c^3}
Out[58]= b^2 + c^3
```

■ iv. Schleifen und If−Anweisungen

Die wichtigsten aus imperativen Programmiersprachen bekannten Schleifentypen sind auch in *Mathematica* vorhanden. (Die von `Do` verwendeten Zählvariablen wurden im Abschnitt "Listen" erklärt.)

```
In[59]:= Do[Print["Hallo Welt!"], {5}]
         Hallo Welt!
         Hallo Welt!
         Hallo Welt!
         Hallo Welt!
         Hallo Welt!

In[60]:= liste = {};
         Do[liste = Append[liste, Prime[n]], {n, 1000, 1005}]
         liste
Out[62]= {7919, 7927, 7933, 7937, 7949, 7951}
```

Schleifen können auch mit `For` und `While` formuliert werden.

Häufig ist es einfacher und schneller anstelle dieser allgemeinen Schleifen spezialisierte Operationen wie `Table`, `Map`, `Nest` , oder `Apply` zu verwenden

```
In[63]:= Table[Prime[i], {i, 1000, 1005}]
Out[63]= {7919, 7927, 7933, 7937, 7949, 7951}
```

```
In[64]:= Map[Prime, Range[1000, 1005]]
Out[64]= {7919, 7927, 7933, 7937, 7949, 7951}

In[65]:= Nest[Sin, x, 3]
Out[65]= Sin[Sin[Sin[x]]]

In[66]:= Nest[Sin, 1.2, 3]
Out[66]= 0.71933

In[67]:= NestList[Sin, x, 3]
Out[67]= {x, Sin[x], Sin[Sin[x]], Sin[Sin[Sin[x]]]}
```

It−Anweisungfunktionieren ganz ähnlich wie in anderen Programmiersprachen auch

```
In[68]:= wuerfel[] := Random[Integer, {1, 6}]

In[69]:= If[wuerfel[] == wuerfel[],
          Print["Pasch"],
          Print["Kein Pasch"]]
         Kein Pasch
```

■ v. Eingabe von Formeln

Die Verwendung zweidimensionaler Formeln (etwa $\sqrt{2}$ anstelle von `Sqrt[2]`) und mathematischer Symbole kann die Notation manchmal merklich vereinfachen. Für die gebräuchlichsten Symbole stehen im File-Menü einige *Paletten* zur Verfügung (in den meisten Fällen reicht "BasicInput"). Zwischen den Platzhaltern ■ und □ kann mit der TAB−Taste hin−und hergewechselt werden.

Folgende Tabelle enthält einige nützliche Tastenkombinationen. Das Symbol "Ξ" steht dabei für die ESC−Taste, das "+" deutet an, dass die Tasten *gleichzeitig* zu drücken sind.

TAB	zum nächsten Platzhalten (■ oder □) springen
CTRL + 2	Wurzel – Zeichen einfügen $\sqrt{a}$
CTRL + SHIFT + 6	Bruch einfügen $\frac{a}{b}$
CTRL + ^	Exponenten einfügen a^b
CTRL + SPACE	zur nächsthöheren Schachtelungsebene gehen
Ξa Ξ, ..., Ξz Ξ	α, ..., ζ
Ξinf Ξ	∞
Ξelem Ξ	$\in$
−>	$\rightarrow$

Einen Ausdruck wie

```
In[70]:= 1 / (π Sqrt[x (1 - x)] + y);
```

erhält man also entweder mithilfe der Paletten und TAB, oder durch die Tastenfolge 1, CTRL+SHIFT+6, ESC, pi, ESC, CTRL+2, x(1−x), CTRL+SPACE, +y.

4. Analysis

■ i. Differentiation und Integration

Ableiten nach einer Unbekannten x

```
In[71]:= D[x Log[x], x]
Out[71]= 1 + Log[x]
```

Zweifache Ableitung nach x

```
In[72]:= D[Exp[x^2], {x, 2}]
Out[72]= 2 ℯ^x^2 + 4 ℯ^x^2 x^2
```

Gemischte Ableitung $\partial_y \partial_x$

```
In[73]:= D[Sin[x y], x, y]
Out[73]= Cos[x y] - x y Sin[x y]
```

Unbestimmtes Integral...

```
In[74]:= Integrate[1/(1 + x^2), x]
Out[74]= ArcTan[x]
```

und bestimmtes Integral werden mit `Integrate` berechnet. Im folgenden Fall ist das Ergebnis allerdings nicht geschlossen darstellbar:

```
In[75]:= Integrate[Sin[Sin[x]], {x, 0, 10}]
Out[75]= ∫_0^10 Sin[Sin[x]] dx
```

Numerische Integration löst das Problem

```
In[76]:= NIntegrate[Sin[Sin[x]], {x, 0, 10}]
Out[76]= 1.6296
```

Mehrfache Integrale lasssen sich einfach durch die Angabe mehrere Integrationsvariablen berechnen

```
In[77]:= Integrate[Exp[-(x^2 + y^2)], {x, 0, x1}, {y, 0, x2}]
Out[77]= 1/4 π Erf[x1] Erf[x2]
```

■ ii. Reihenentwicklungen

Analytische Funktionen können mit `Series` in eine Potenzreihe entwickelt werden. Neben dem Entwicklungspunkt muss außerdem noch angegeben werden, bis zu welcher Ordnung entwickelt werden soll (hier 6 bzw. 4)

```
In[78]:= Series[Sin[x], {x, 0, 6}]
```

$$\text{Out[78]}= x - \frac{x^3}{6} + \frac{x^5}{120} + O[x]^7$$

```
In[79]:= Series[Log[x + 1], {x, 0, 4}]
```

$$\text{Out[79]}= x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + O[x]^5$$

Mit Potenzreihen kann in gewissen Grenzen wie mit anderen Ausdrücken gerechnet werden

```
In[80]:= % + %%
```

$$\text{Out[80]}= 2\,x - \frac{x^2}{2} + \frac{x^3}{6} - \frac{x^4}{4} + O[x]^5$$

■ iii. Summen, Produkte und Grenzwerte

Zusätzlich können auch Grenzwerte, (unendliche) Reihen und (unendliche) Produkte ausgewertet werden

```
In[81]:= Limit[Sin[x] / x, x → 0]
```

$$\text{Out[81]}= 1$$

```
In[82]:= Sum[Binomial[n, m], {m, 0, n}]
```

$$\text{Out[82]}= 2^n$$

```
In[83]:= Sum[1 / n^2, {n, 1, ∞}]
```

$$\text{Out[83]}= \frac{\pi^2}{6}$$

```
In[84]:= Product[1 + 1 / k, {k, 1, n}]
```

$$\text{Out[84]}= 1 + n$$

```
In[85]:= Product[1 + 1 / k^2, {k, 1, ∞}]
```

$$\text{Out[85]}= \frac{\text{Sinh}[\pi]}{\pi}$$

5. Lösen von Gleichungen

■ i. Gewöhnliche Gleichungen

Um die Lösung einer beliebigen Gleichung zu finden, steht mit `FindRoot` ein numerischer Gleichungslöser zur Verfügung

```
In[86]:= FindRoot[Sin[x] == x^2, {x, 1}]
```

$$\text{Out[86]}= \{x \to 0.876726\}$$

Man kann den Suchbereich von `FindRoot` auf ein beliebiges Intervall einschränken, indem man zusätzlich zum Startwert die Intervallgrenzen übergibt (obwohl das meistens nicht nötig ist):

```
In[87]:= FindRoot[Exp[x] == 5 x^3, {x, 1, 0, 2}]
Out[87]= {x → 0.751204}
```

Im Gegensatz zu `FindRoot` versucht `Solve`, die Gleichung aufzulösen und eine allgemeine Lösung zu bestimmen. Wo das nicht geht (etwa im ersten Beispiel oben) gibt es eine Fehlermeldung, sonst eine Liste der Lösungen

```
In[88]:= Solve[x^2 + p x + q == 0, x]
Out[88]= {{x → 1/2 (-p - Sqrt[p^2 - 4 q])}, {x → 1/2 (-p + Sqrt[p^2 - 4 q])}}
```

Auch wenn `Solve` alle Lösungen findet, sind diese oft nur numerisch darstellbar

```
In[89]:= Solve[x^6 + x == 10, x]
Out[89]= {{x → Root[-10 + #1 + #1^6 &, 1]}, {x → Root[-10 + #1 + #1^6 &, 2]},
          {x → Root[-10 + #1 + #1^6 &, 3]}, {x → Root[-10 + #1 + #1^6 &, 4]},
          {x → Root[-10 + #1 + #1^6 &, 5]}, {x → Root[-10 + #1 + #1^6 &, 6]}}

In[90]:= N[%]
Out[90]= {{x → -1.50244}, {x → 1.43052},
          {x → -0.717234 - 1.3022 i}, {x → -0.717234 + 1.3022 i},
          {x → 0.753199 - 1.24011 i}, {x → 0.753199 + 1.24011 i}}
```

Stattdessen kann man auch gleich die numerische Variante von `Solve` benutzen

```
In[91]:= NSolve[x^6 + x == 10, x]
Out[91]= {{x → -1.50244}, {x → -0.717234 - 1.3022 i}, {x → -0.717234 + 1.3022 i},
          {x → 0.753199 - 1.24011 i}, {x → 0.753199 + 1.24011 i}, {x → 1.43052}}
```

Die vorgestellten Funktionen sind auch in der Lage, Gleichungssysteme in mehreren Variablesn zu lösen. Ein Beispiel

```
In[92]:= FindRoot[{Sin[x] == Cos[x], x^2 + y^2 == 1}, {x, .5}, {y, .2}]
Out[92]= {x → 0.785398, y → 0.618991}
```

■ ii. Differentialgleichungen

Schließlich können auch Differentialgleichungen und Systemen von Differentialgleichungen sowohl exakt als auch numerisch gelöst werden.

```
In[93]:= DSolve[y''[x] == ω^2 y[x], y[x ], x]
Out[93]= {{y[x] → E^(x ω) C[1] + E^(-x ω) C[2]}}

In[94]:= DSolve[{y'[0] == a, y''[x] == ω^2 y[x]}, y[x], x]
Out[94]= {{y[x] → (E^(-x ω) (a E^(2 x ω) + ω C[2] + E^(2 x ω) ω C[2]))/ω}}
```

In dieser Form wird eine Ersetzungsregel für `y[x]` zurückgeliefert. Ist man auch an Ableitungen von `y` interessiert, ist vorteilhaft nicht nach `y[x]`, sondern nach `y` aufzulösen:

```
In[95]:= {y[x], y'[x]} /. DSolve[{y[0] == 0, y'[0] == a, y''[x] == ω^2 y[x]}, y, x]
Out[95]= {{(a E^(-x ω) (-1 + E^(2 x ω)))/(2 ω), a E^(x ω) - 1/2 a E^(-x ω) (-1 + E^(2 x ω))}}
```

Auch für Differentialgleichungen können numerische Lösungen gesucht werden. `NDSolve` berechnet einige Lösungspunkte im angegebenen Intervall, die von der zurückgelieferten `InterpolatingFunction` interpoliert werden.

```
In[96]:= NDSolve[{y'[x] == Cos[x * y[x]], y[0] == 1}, y[x], {x, -5, 5}]
Out[96]= {{y[x] → InterpolatingFunction[{{-5., 5.}}, <>][x]}}
```

Mit folgender Ersetzung kann man dieses Ergebnis wie eine reguläre Funktion benutzen (etwa zum Plotten).

```
In[97]:= f[x_] := Evaluate[y[x] /. %]
```

In ähnlicher Weise lassen sich auch partielle Differentialgleichungen lösen. Mehr Informationen dazu gibt es (wieder einmal) in der Online−Hilfe.

6. Lineare Algebra

■ i. Matrizen und Vektoren

Wie bereits erläutert, verhalten sich Listen weitgehend wie Vektoren, da alle Grundrechenarten für sie elementweise definiert sind. Matrizen werden als Listen von Listen dargestellt

```
In[98]:= a = {a1, a2, a3}; b = {b1, b2, b3};

In[99]:= a.b    (* Skalarprodukt *)
Out[99]= a1 b1 + a2 b2 + a3 b3

In[100]:= Cross[a, b]    (* Kreuzprodukt *)
Out[100]= {-a3 b2 + a2 b3, a3 b1 - a1 b3, -a2 b1 + a1 b2}

In[101]:= m = {{2, 1, 0}, {0, 3, -1}, {0, 0, 3}}
Out[101]= {{2, 1, 0}, {0, 3, -1}, {0, 0, 3}}

In[102]:= % // MatrixForm    (* Äquivalent zu MatrixForm[%] *)
Out[102]//MatrixForm=
⎛2  1   0⎞
⎜0  3  -1⎟
⎝0  0   3⎠

In[103]:= m.a    (* Lineare Abbildung eines Vektors *)
Out[103]= {2 a1 + a2, 3 a2 - a3, 3 a3}
```

Um Matrizen zu erstellen gibt es natürlich `Table`, aber zusätzlich noch einige spezielle Funktionen. Eine 3×3 Matrix erstellen...

```
In[104]:= b = Table[i * j, {i, 3}, {j, 3}]
Out[104]= {{1, 2, 3}, {2, 4, 6}, {3, 6, 9}}
```

... oder eine 3×3 Diagonalmatrix...

```
In[105]:= d = DiagonalMatrix[{x1, x2, x3}]
Out[105]= {{x1, 0, 0}, {0, x2, 0}, {0, 0, x3}}
```

... oder eine 3×3 Einheitsmatrix

In[106]:=
```
IdentityMatrix[3]
```
Out[106]=
```
{{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}
```

b ist symmetrisch, stimmt als mit der eigenen Transponierten überein

In[107]:=
```
Transpose[b] == b
```
Out[107]=
```
True
```

Andere klassische Funktionen einer Matrix sind die Determinante und die Inverse

In[108]:=
```
Det[d]
```
Out[108]=
```
x1 x2 x3
```

In[109]:=
```
Inverse[d]
```
Out[109]=
$$\left\{\left\{\frac{1}{x1}, 0, 0\right\}, \left\{0, \frac{1}{x2}, 0\right\}, \left\{0, 0, \frac{1}{x3}\right\}\right\}$$

In[110]:=
```
Clear[a, b, d, m]
```

■ ii. Lineare Gleichungssysteme

Lineare Gleichungssysteme können natürlich mit den allgemeinen Lösungsverfahren von `Solve` aufgelöst werden. Das ist der praktischere Weg, wenn bereits die Gleichungen selbst vorliegen. Wenn dagegen nur die Matrix `m` und der Vektor `b` in der Gleichung `m.x == b` bekannt sind, ist es einfacher, die Funktion `LinearSolve` zu verwenden:

In[111]:=
```
LinearSolve[{{1, 2}, {3, 4}}, {a, b}]
```
Out[111]=
$$\left\{-2\,a + b, \frac{1}{2}\,(3\,a - b)\right\}$$

Für homogene LGSe kann man auch zu `NullSpace` greifen, um eine Basis des Kerns der Matrix zu ermitteln

In[112]:=
```
NullSpace[{{2, -2}, {4, -4}}]
```
Out[112]=
```
{{1, 1}}
```

Muss ein lineares Gleichungssystem mehrfach für verschiedene `b` aber konstantes `m` gelöst werden, kann es effizienter sein, zunächst eine LU−Dekompositionder Matrix durchzuführen:

In[113]:=
```
lu = LUDecomposition[{{1, 2}, {3, 4}}]
```
Out[113]=
```
{{{1, 2}, {3, -2}}, {1, 2}, 1}
```

In[114]:=
```
LUBackSubstitution[lu, {a, b}]
```
Out[114]=
$$\left\{-2\,a + b, \frac{1}{2}\,(3\,a - b)\right\}$$

■ iii. Eigenwerte und Eigenvektoren

Zur Berechnung der Eigenwerte und −vektoren einer Matrix stellt Mathematica die Funktionen `Eigenvalues` und `Eigenvectors` zur Verfügung. Für eine $n \times n$ Matrix liefert `Eigenvalues` immer `n` (nicht unbedingt verschiedene) Zahlen, während `Eigenvectors` bei weniger als `n` *unabhängigen* Eigenvektoren (wie im folgenden Beispiel) Nullvektoren anhängt.

In[115]:=
```
m = {{2, 1, 0}, {0, 3, -1}, {0, 0, 3}};
```

In[116]:=
```
Eigenvalues[m]
```
Out[116]=
```
{2, 3, 3}
```

```
In[117]:= Eigenvectors[m]
Out[117]= {{1, 0, 0}, {1, 1, 0}, {0, 0, 0}}
```

`Eigensystem` berechnet gleichzeitig die Eigenwerte und −vektoreneiner Matrix und gibt sie als Liste zurück.

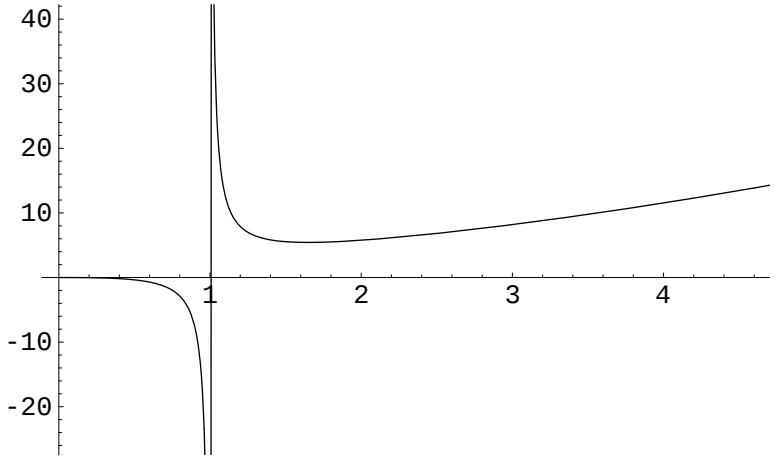
```
In[118]:= {ew, ev} = Eigensystem[m]
Out[118]= {{2, 3, 3}, {{1, 0, 0}, {1, 1, 0}, {0, 0, 0}}}
```

5. Grafik

■ i. Zweidimensional

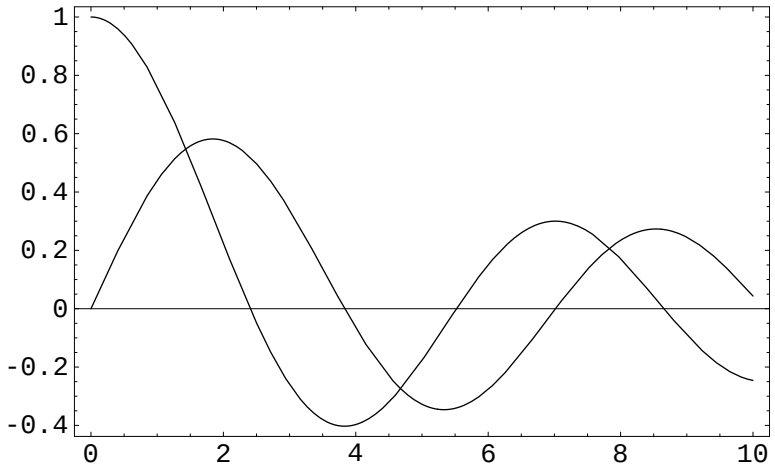
`Plot` stellt eine Funktion auf einem reellen Intervall dar…

```
In[119]:= Plot[x^2 / Log[x], {x, 0, 10}];
```



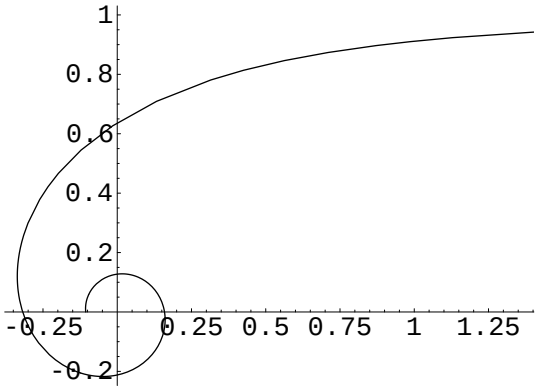
… oder auch mehrere Funktionen. Das folgende Beispiel zeigt auch, wie man eine *Option* an `Plot` übergibt (hier `Frame` → `True`).

In[120]:=
```
Plot[{BesselJ[0, x], BesselJ[1, x]}, {x, 0, 10}, Frame → True];
```



Parametrische Kurven in der Ebene werden von `ParametricPlot` dargestellt

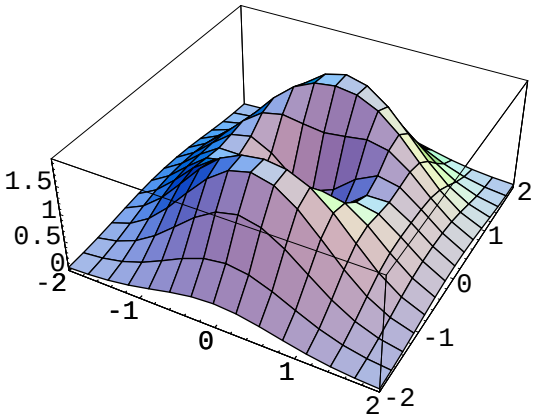
In[121]:=
```
ParametricPlot[{Cos[t] / t, Sin[t] / t},
  {t, 0, 3 Pi}, AspectRatio → Automatic, ImageSize → 200];
```



■ ii. Dreidimensional

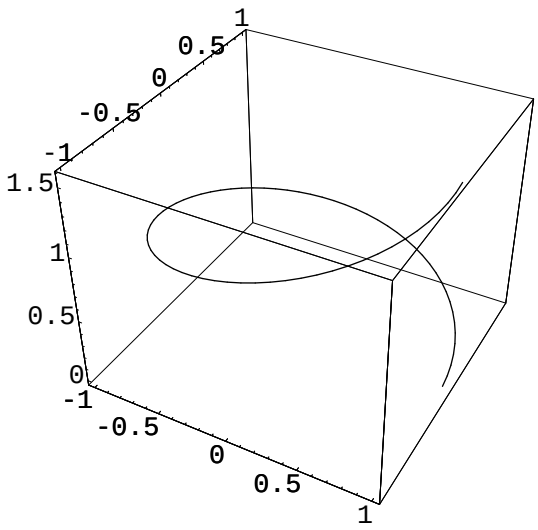
Eine Funktion $\mathbb{R}^2 \to \mathbb{R}$ kann mit `Plot3D` veranschaulicht werden

In[122]:=
```
Plot3D[(x^2 + 2 y^2) E^(1-x^2-y^2), {x, -2, 2}, {y, -2, 2}, ImageSize → 200];
```



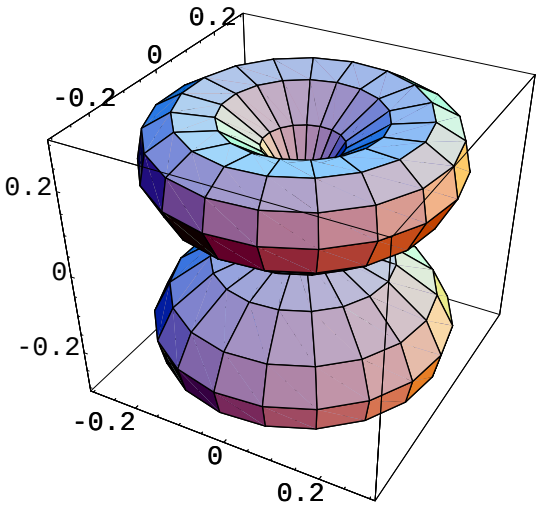
Parametrische Plots im Raum sind entweder Kurven, wenn nur ein Parameter verwendet wird, oder Oberflächen für zwei Parameter. Das zweite Beispiel zeigt den Betrag der Kugelflächenfunktion $Y_2{}^{-1}$

In[123]:=
```
ParametricPlot3D[{Cos[t], Sin[t], t / 4}, {t, 0, 2 Pi}, ImageSize → 200];
```



In[124]:=
```
r[t_, p_] := Abs[SphericalHarmonicY[2, -1, t, p]];
```

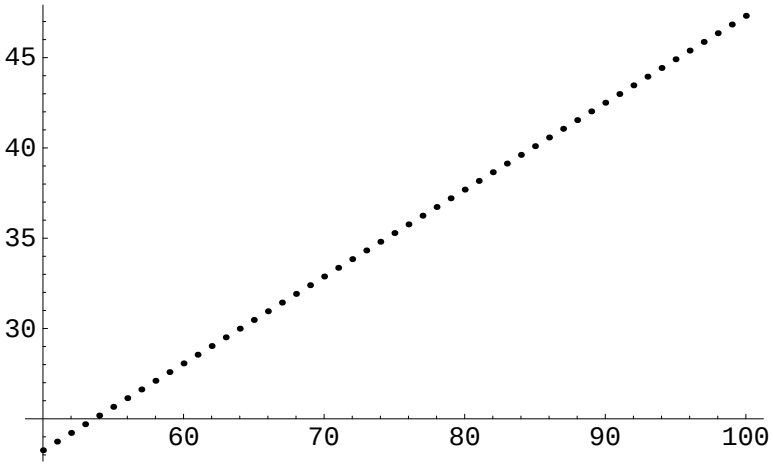
In[125]:=
```
ParametricPlot3D[r[θ, ϕ] {Sin[θ] Cos[ϕ], Sin[θ] Sin[ϕ], Cos[θ]},
  {θ, 0, Pi}, {ϕ, 0, 2 Pi}, AspectRatio → 1, ImageSize → 200];
```



■ iii. Andere Visualisierungsmöglichkeiten

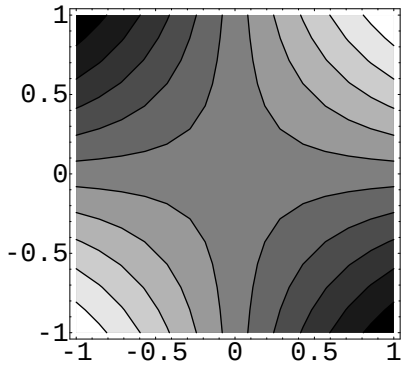
Diskrete Punktmengen werden mit `ListPlot` gezeichnet. Die Option `PlotJoined → True` verbindet die einzelnen Punkte.

In[126]:=
```
ListPlot[Table[{i, Log[Fibonacci[i]]}, {i, 50, 100}]];
```



`ContourPlot` bietet eine weitere Möglichkeit, eine Funktion $\mathbb{R}^2 \to \mathbb{R}$ darzustellen:

In[127]:=
```
ContourPlot[Sin[x y], {x, -1, 1}, {y, -1, 1}, ImageSize → 150];
```



7. Einige vollständige Beispiele

Die folgenden Abschnitte demonstrieren, wie man *Mathematica* in einigen einfachen konkreten Problemen einsetzt. Für die Grafiken verwenden wir dabei durchgehend folgende Optionen:

In[128]:=
```
SetOptions[{ListPlot, ParametricPlot, ContourPlot},
  ImageSize → 150, AspectRatio → Automatic];
```

■ i. Darstellung der n−tenEinheitswurzeln

Die Lösungen von Gleichungen liefert `Solve` in Form einer Liste von Ersetzungsregeln zurück:

In[129]:=
```
Solve[z^4 == 1, z]
```

Out[129]=
```
{{z → -1}, {z → -i}, {z → i}, {z → 1}}
```

Um eine Liste aller n−ten Einheitswurzeln (aller Lösungen der Gleichung $z^n == 1$) zu bestimmen, kann man also schreiben

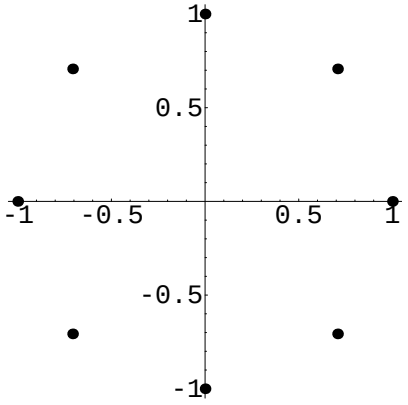
In[130]:=
```
einheitsw[n_] := Module[{z}, z /. Solve[z^n == 1, z]]
```

Separiert man Real−und Imaginärteil erhält man hieraus die entsprechende Punktmenge in der Gaußschen Zahlenebene, die man schließlich mit `ListPlot` darstellen kann

In[131]:=
```
ewxy = Map[{Re[#], Im[#]} &, einheitsw[8]]
```

Out[131]=
$$\left\{\{-1, 0\}, \{0, -1\}, \{0, 1\}, \{1, 0\}, \left\{-\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right\}, \left\{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right\}, \left\{\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right\}, \left\{-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right\}\right\}$$

In[132]:=
```
ListPlot[ewxy, PlotStyle → {AbsolutePointSize[4]}];
```



■ ii. Koch−Kurve(zum Knobeln)

Dieses Beispiel demonstriert die rekursive Konstruktion einer Koch−Kurve. Ausgangsfigur ist dabei das folgende gleichseitige Dreieck.

In[133]:=
```
dreieck = List[{{-1, 0}, {1, 0}}, {{1, 0}, {0, √3}}, {{0, √3}, {-1, 0}}];
```

Diese Funktion erstellt eine graphische Darstellung einer Koch-Kurve n-ter Ordnung. Die Kurve selbst wird wird von `verfeinerung` berechnet, die obiges Dreieck in n rekursiven Schritten "verfeinert" und eine Kantenliste zurückliefert.

In[134]:=
```
kochkurve[n_Integer] := Graphics[Map[Line, verfeinerung[dreieck, n]]]
```

Diese Verfeinerung läuft wie folgt ab:

In[135]:=
```
verfeinerung[kanten_, 0] := kanten;
verfeinerung[kanten_List, n_Integer] := verfeinerung[
  Flatten[Map[auswoelben, kanten], 1],
  n - 1]
```

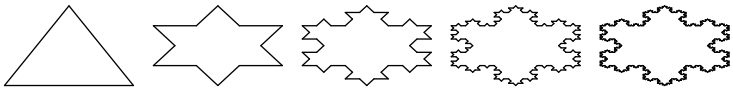
Schließlich das Auswölben einer Kante: Das mittlere Drittel jeder Kante wird entfernt und in die Lücke die Spitze eines kleineren gleichseitigen Dreiecks eingesetzt

In[137]:=
```
<< Geometry`Rotations`
rotmat = RotationMatrix2D[60 Degree]
```

Out[138]=
$$\left\{\left\{\frac{1}{2}, \frac{\sqrt{3}}{2}\right\}, \left\{-\frac{\sqrt{3}}{2}, \frac{1}{2}\right\}\right\}$$

In[139]:=
```
auswoelben[{a_, b_}] := Module[{x, y, s},
  x = a + (b - a) / 3;
  y = b - (b - a) / 3;
  s = x + rotmat.(y - x);
  List[{a, x}, {x, s}, {s, y}, {y, b}]]
```

In[140]:=
```
GraphicsArray[Table[kochkurve[i], {i, 0, 4}]] // Show
```



Out[140]= - GraphicsArray -

■ iii. Symmetrische Matrizen

Reelle symmetrische Matrizen treten in vielen Zusammenhängen auf, etwa als Kovarianzmatrix Σ in mehrdimensionalen Normalverteilungen

$$p(x) = A\, e^{\frac{-1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

oder als Trägheitstensor J bei der Definition der Rotationsenergie in der klassischen Mechanik

$$E_{rot}(\omega) = \frac{1}{2}\, \omega^T J\, \omega$$

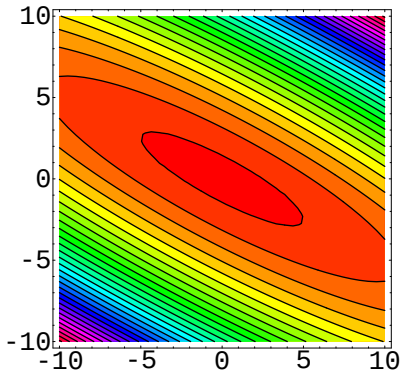
In beiden Fällen sind die mithilfe der Matrizen gebildeten Ausdrücke positiv definite *quadratische Formen.* Diese haben die besondere Eigenschaft, dass Punkte mit konstantem Funktionswert auf Hyperellipsoiden im jeweiligen Vektorraum liegen (etwa der Trägheitsellipsoid der ω−Vektoren mit konstanter Rotationsenergie). Das kann man im Zweidimensionalen folgendermaßen veranschaulichen.

In[141]:=
```
quadform[mtx_, vec_] := vec.mtx.vec
```

In[142]:=
```
m = {{2, 3}, {3, 6}};     (* symmetrisch und pos. def. *)
```

Einen Überblick über das Aussehen der zugehörigen quadratischen Form kann man sich mit ContourPlot verschaffen

In[143]:=
```
ContourPlot[quadform[m, {x, y}], {x, -10, 10}, {y, -10, 10},
  PlotPoints → 40, ColorFunction → Hue, Contours → 30];
```



Wir drücken die quadratische Form gleich in Polarkoordinaten aus, um eine der Ellipsen einfach parametrisieren zu können

In[144]:=
```
qform[r_, t_] = Simplify[quadform[m, {r Cos[t], r Sin[t]}]]
```

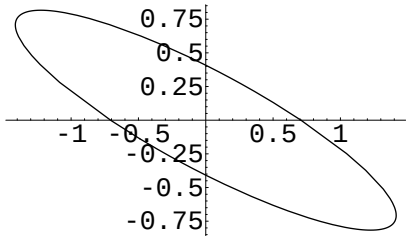
Out[144]= $-r^2\,(-4 + 2\,\text{Cos}[2\,t] - 3\,\text{Sin}[2\,t])$

Die Menge der Vektoren in der Ebene für die die Funktion konstant Eins ist, lässt sich dann wie folgt darstellen

In[145]:=
```
r[t_] = Part[r /. Solve[qform[r, t] == 1, r], 2]
```

Out[145]=
$$\frac{1}{\sqrt{4 - 2\,\text{Cos}[2\,t] + 3\,\text{Sin}[2\,t]}}$$

In[146]:=
```
g = ParametricPlot[{r[t] * Cos[t], r[t] * Sin[t]}, {t, 0, 2 Pi}]
```



Out[146]= - Graphics -

Schließlich kann man noch die Eigenvektoren von m einblenden. Wie man sieht, sind die Eigenvektoren gerade die Hauptachsen der Ellipse, und die Länge der Hauptachsen ist jeweils der Kehrwert der Wurzel aus dem zugehörigen Eigenwert (warum?).

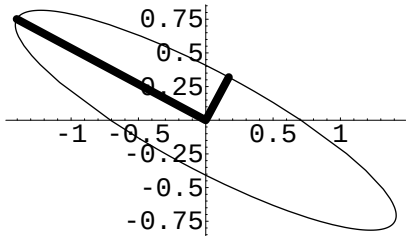
In[147]:=
```
{ew, ev} = Eigensystem[m]
```

Out[147]= $\left\{\left\{4-\sqrt{13},\ 4+\sqrt{13}\right\},\ \left\{\left\{\frac{1}{3}\left(-2-\sqrt{13}\right),\ 1\right\},\ \left\{\frac{1}{3}\left(-2+\sqrt{13}\right),\ 1\right\}\right\}\right\}$

In[148]:=
```
ev2 = Map[# / Sqrt[#.#] &, ev];(* Eigenv. normieren *)
```

In[149]:=
```
li = Line[{ev2[[1]] / Sqrt[ew[[1]]], {0, 0}, ev2[[2]] / Sqrt[ew[[2]]]}];
```

In[150]:=
```
Show[{g, Graphics[{AbsoluteThickness[3], li}]}]
```



Out[150]= - Graphics -