

Mathematica Kompaktkurs

Inhaltsübersicht

1. Einführung
2. Arithmetik
3. Fortgeschrittene Anwendungen
4. Analysis
5. Lösen von Gleichungen
6. Lineare Algebra
7. Grafik
8. Einige vollständige Beispiele

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:

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

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

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*
```

```
Equal          EqualColumns EqualRows    EquatedTo
```

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

```
FactorList      FactorTermsList    FixedPointList    FoldList
FactorSquareFreeList FindList
```

■ 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* vordefinierten Funktionen, Prozeduren und Konstanten beginnen mit einem Großbuchstaben.

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

Auf die letzten 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:

Infinity, ∞	Symbol für ∞
Degree, °	Umrechnungsfaktor Grad \rightarrow 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[19]:= ? 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}.

In[20]:= ? Arc*

ArcCos ArcCot ArcCoth ArcCsc ArcCsch ArcSec ArcSech ArcSin ArcSinh
ArcTan ArcTanh
ArcCosh

Neben diesen Standardfunktionen existieren viele spezielle Funktionen, wie die Gamma-Funktion Gamma, die Fehlerfunktion Erf, verschiedene Bessel-Funktionen (BesselJ, BesselY, BesselI, BesselK) oder die Kugelflächenfunktionen 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[21]:= **v = {1, 2, 3, 4}**

Out[21]= {1, 2, 3, 4}

In[22]:= **w = -v * 2**

Out[22]= {-2, -4, -6, -8}

In[23]:= **w + v**

Out[23]= {-1, -2, -3, -4}

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

```
In[24]:= Table[Sin[2], {5}]
Out[24]= {Sin[2], Sin[2], Sin[2], Sin[2], Sin[2]}
```

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

```
In[25]:= potenzen = Table[{n, n^2, n^3}, {n, 10}]
Out[25]= {{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[26]:= qzahlen = Part[Transpose[potenzen], 2]
Out[26]= {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[27]:= {potenzen[[3]], potenzen[[4, 3]]}
Out[27]= {{3, 9, 27}, 64}
```

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

```
In[28]:= Apply[Plus, qzahlen] (* Summiere alle Elemente *)
Out[28]= 385
```

Map dagegen wendet eine Funktion auf jedes Listenelement *einzel*n an

```
In[29]:= Map[Sin, qzahlen]
Out[29]= {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[30]:= rand = Random[] (* setze rand auf eine feste Zufallszahl *)
Out[30]= 0.52907
```

```
In[31]:= rand * 2
Out[31]= 1.05814
```

Eigene Funktionen können auf ähnliche Weise definiert werden:

```
In[32]:= randint[min_, max_] := Random[Integer, {min, max}]
```

```
In[33]:= Table[randint[2, 10], {10}]
Out[33]= {4, 10, 9, 5, 10, 3, 6, 6, 6, 6}
```

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[34]:= Clear[v] (* ein Symbol löschen *)
In[35]:= Clear[r1, r2] (* mehrere Symbole löschen *)
In[36]:= 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[37]:= f[x_] = Sin[x]^2
Out[37]= Sin[x]^2
In[38]:= N[Map[f, Range[1, 10]]]
Out[38]= {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[39]:= N[Map[Sin[#]^2 &, Range[1, 10]]]
Out[39]= {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[40]:= Sum[q^n, {n, 1, Infinity}]
Out[40]= -\frac{q}{-1 + q}
In[41]:= Integrate[Tan[c * y], y]
Out[41]= -\frac{\text{Log}[\text{Cos}[c y]]}{c}
```

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

```
In[42]:= Simplify[Sin[x]^2 + Cos[x]^2]
Out[42]= 1
In[43]:= Simplify[(a^2 - b^2) / (a - b)]
Out[43]= a + b
```

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

```
In[44]:= Simplify[Sqrt[x^2]]
```

```
Out[44]=  $\sqrt{x^2}$ 
```

```
In[45]:= Simplify[Sqrt[x^2], x < 0]
```

```
Out[45]= -x
```

```
In[46]:= Simplify[Sin[x + 2 Pi n], n ∈ Integers]
```

```
Out[46]= Sin[x]
```

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

```
In[47]:= FullSimplify[(I/4) / E^(2 * I * x) - I/4 * E^(2 * I * x)]
```

```
Out[47]= Cos[x] Sin[x]
```

$\text{Sqrt}[x^2]$ 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[48]:= PowerExpand[{ $\sqrt{x^2}$ , (a * b)^c, (a^b)^c}]
```

```
Out[48]= {x, a^c b^c, a^b^c}
```

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

```
In[49]:=  $\sqrt{a}$  /. a → x^2 + y^2
```

```
Out[49]=  $\sqrt{x^2 + y^2}$ 
```

```
In[50]:= a + b /. {a → b^2, b → c^3}
```

```
Out[50]= 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[51]:= Do[Print["Hallo Welt!"], {5}]
```

```
Hallo Welt!
```

```
Hallo Welt!
```

```
Hallo Welt!
```

```
Hallo Welt!
```

```
Hallo Welt!
```

```
In[52]:= liste = {};
Do[list = Append[list, Prime[n]], {n, 1000, 1005}]
liste
```

```
Out[54]= {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[55]:= Table[Prime[i], {i, 1000, 1005}]
Out[55]= {7919, 7927, 7933, 7937, 7949, 7951}
```

```
In[56]:= Map[Prime, Range[1000, 1005]]
Out[56]= {7919, 7927, 7933, 7937, 7949, 7951}
```

```
In[57]:= Nest[Sin, x, 3]
Out[57]= Sin[Sin[Sin[x]]]
```

```
In[58]:= Nest[Sin, 1.2, 3]
Out[58]= 0.71933
```

```
In[59]:= NestList[Sin, x, 3]
Out[59]= {x, Sin[x], Sin[Sin[x]], Sin[Sin[Sin[x]]]}
```

If-Anweisungsfunktionieren ganz ähnlich wie in anderen Programmiersprachen auch

```
In[60]:= wuerfel[] := Random[Integer, {1, 6}]

In[61]:= 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 "≐" steht dabei für die `ESC`-Taste, das "+" deutet an, dass die Tasten *gleichzeitig* zu drücken sind.

<code>TAB</code>	zum nächsten Platzhalten (■ oder □) springen
<code>CTRL</code> + 2	Wurzel – Zeichen einfügen \sqrt{a}
<code>CTRL</code> + <code>SHIFT</code> + 6	Bruch einfügen $\frac{a}{b}$
<code>CTRL</code> + ^	Exponenten einfügen a^b
<code>CTRL</code> + <code>SPACE</code>	zur nächsthöheren Schachtelungsebene gehen
≐a≐, ..., ≐Z≐	α, \dots, ζ
≐inf≐	∞
≐elem≐	\in
->	\rightarrow

Einen Ausdruck wie

$$\text{In[62]:= } \frac{1}{\pi \sqrt{x(1-x)} + y};$$

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[63]:= `D[x Log[x], x]`

Out[63]= $1 + \text{Log}[x]$

Zweifache Ableitung nach x

In[64]:= `D[Exp[x^2], {x, 2}]`

Out[64]= $2 e^{x^2} + 4 e^{x^2} x^2$

Gemischte Ableitung $\partial_y \partial_x$

In[65]:= `D[Sin[x y], x, y]`

Out[65]= $\text{Cos}[x y] - x y \text{Sin}[x y]$

Unbestimmtes Integral...

In[66]:= `Integrate[1/(1+x^2), x]`

Out[66]= $\text{ArcTan}[x]$

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

In[67]:= `Integrate[Sin[Sin[x]], {x, 0, 10}]`

Out[67]= $\int_0^{10} \text{Sin}[\text{Sin}[x]] dx$

Numerische Integration löst das Problem

In[68]:= `NIntegrate[Sin[Sin[x]], {x, 0, 10}]`

Out[68]= 1.6296

Mehrfache Integrale lassen sich einfach durch die Angabe mehrere Integrationsvariablen berechnen

In[69]:= `Integrate[Exp[-(x^2+y^2)], {x, 0, x1}, {y, 0, x2}]`

Out[69]= $\frac{1}{4} \pi \text{Erf}[x1] \text{Erf}[x2]$

■ 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[70]:= Series[Sin[x], {x, 0, 6}]
```

```
Out[70]= x -  $\frac{x^3}{6}$  +  $\frac{x^5}{120}$  + O[x]7
```

```
In[71]:= Series[Log[x + 1], {x, 0, 4}]
```

```
Out[71]= 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[72]:= % + %%
```

```
Out[72]= 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[73]:= Limit[Sin[x] / x, x → 0]
```

```
Out[73]= 1
```

```
In[74]:= Sum[Binomial[n, m], {m, 0, n}]
```

```
Out[74]= 2n
```

```
In[75]:= Sum[1 / n ^ 2, {n, 1, ∞}]
```

```
Out[75]=  $\frac{\pi^2}{6}$ 
```

```
In[76]:= Product[1 + 1 / k, {k, 1, n}]
```

```
Out[76]= 1 + n
```

```
In[77]:= Product[1 + 1 / k ^ 2, {k, 1, ∞}]
```

```
Out[77]=  $\frac{\text{Sinh}[\pi]}{\pi}$ 
```

5. Lösen von Gleichungen

■ i. Gewöhnliche Gleichungen

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

```
In[78]:= FindRoot[Sin[x] == x ^ 2, {x, 1}]
```

```
Out[78]= {x → 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[79]:= FindRoot[Exp[x] == 5 x^3, {x, 1, 0, 2}]
```

```
Out[79]:= {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[80]:= Solve[x^2 + p x + q == 0, x]
```

```
Out[80]:= {{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[81]:= Solve[x^6 + x == 10, x]
```

```
Out[81]:= {{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[82]:= N[%]
```

```
Out[82]:= {{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[83]:= NSolve[x^6 + x == 10, x]
```

```
Out[83]:= {{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 Variablen zu lösen. Ein Beispiel

```
In[84]:= FindRoot[{Sin[x] == Cos[x], x^2 + y^2 == 1}, {x, .5}, {y, .2}]
```

```
Out[84]:= {x -> 0.785398, y -> 0.618991}
```

■ ii. Differentialgleichungen

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

```
In[85]:= DSolve[y''[x] == omega^2 y[x], y[x], x]
```

```
Out[85]:= {{y[x] -> e^{-x omega} C[1] + e^{x omega} C[2]}}
```

```
In[86]:= DSolve[{y'[0] == a, y''[x] == omega^2 y[x]}, y[x], x]
```

```
Out[86]:= {{y[x] -> e^{-x omega} (-a/omega + C[2] + e^{2 x omega} C[2])}}
```

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[87]:= {y[x], y'[x]} /. DSolve[{y[0] == 0, y'[0] == a, y''[x] == omega^2 y[x]}, y, x]
```

```
Out[87]:= {{e^{-x omega} (-a/(2 omega) + a e^{2 x omega}/(2 omega)), a e^{x omega} - e^{-x omega} (-a/(2 omega) + a e^{2 x omega}/(2 omega)) omega}}
```

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

```
In[88]:= NDSolve[{y'[x] == Cos[x * y[x]], y[0] == 1}, y[x], {x, -5, 5}]
Out[88]:= {{y[x] → InterpolatingFunction[{{-5., 5.}}, <>][x]}}
```

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

```
In[89]:= 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[90]:= a = {a1, a2, a3}; b = {b1, b2, b3};
```

```
In[91]:= a.b (* Skalarprodukt *)
```

```
Out[91]= a1 b1 + a2 b2 + a3 b3
```

```
In[92]:= Cross[a, b] (* Kreuzprodukt *)
```

```
Out[92]= {-a3 b2 + a2 b3, a3 b1 - a1 b3, -a2 b1 + a1 b2}
```

```
In[93]:= m = {{2, 1, 0}, {0, 3, -1}, {0, 0, 3}}
```

```
Out[93]= {{2, 1, 0}, {0, 3, -1}, {0, 0, 3}}
```

```
In[94]:= % // MatrixForm (* Äquivalent zu MatrixForm[%] *)
```

```
Out[94]/MatrixForm=

$$\begin{pmatrix} 2 & 1 & 0 \\ 0 & 3 & -1 \\ 0 & 0 & 3 \end{pmatrix}$$

```

```
In[95]:= m.a (* Lineare Abbildung eines Vektors *)
```

```
Out[95]= {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[96]:= b = Table[i * j, {i, 3}, {j, 3}]
```

```
Out[96]= {{1, 2, 3}, {2, 4, 6}, {3, 6, 9}}
```

... oder eine 3×3 Diagonalmatrix...

```
In[97]:= d = DiagonalMatrix[{x1, x2, x3}]
```

```
Out[97]= {{x1, 0, 0}, {0, x2, 0}, {0, 0, x3}}
```

... oder eine 3×3 Einheitsmatrix

```
In[98]:= IdentityMatrix[3]
Out[98]:= {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}
```

b ist symmetrisch, stimmt als mit der eigenen Transponierten überein

```
In[99]:= Transpose[b] == b
Out[99]:= True
```

Andere klassische Funktionen einer Matrix sind die Determinante und die Inverse

```
In[100]:= Det[d]
Out[100]:= x1 x2 x3

In[101]:= Inverse[d]
Out[101]:= {{1/x1, 0, 0}, {0, 1/x2, 0}, {0, 0, 1/x3}}
```

```
In[102]:= 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[103]:= LinearSolve[{{1, 2}, {3, 4}}, {a, b}]
Out[103]:= {-2 a + b, 1/2 (3 a - b)}
```

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

```
In[104]:= NullSpace[{{2, -2}, {4, -4}}]
Out[104]:= {{1, 1}}
```

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

```
In[105]:= lu = LUdecomposition[{{1, 2}, {3, 4}}]
Out[105]:= {{{1, 2}, {3, -2}}, {1, 2}, 1}

In[106]:= LUBackSubstitution[lu, {a, b}]
Out[106]:= {-2 a + b, 1/2 (3 a - b)}
```

■ 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[107]:= m = {{2, 1, 0}, {0, 3, -1}, {0, 0, 3}};
Out[107]:= {{2, 1, 0}, {0, 3, -1}, {0, 0, 3}}

In[108]:= Eigenvalues[m]
Out[108]:= {2, 3, 3}
```

```
In[109]:= Eigenvectors[m]
```

```
Out[109]:= {{1, 0, 0}, {1, 1, 0}, {0, 0, 0}}
```

Eigensystem berechnet gleichzeitig die Eigenwerte und –vektoreiner Matrix und gibt sie als Liste zurück.

```
In[110]:= {ew, ev} = Eigensystem[m]
```

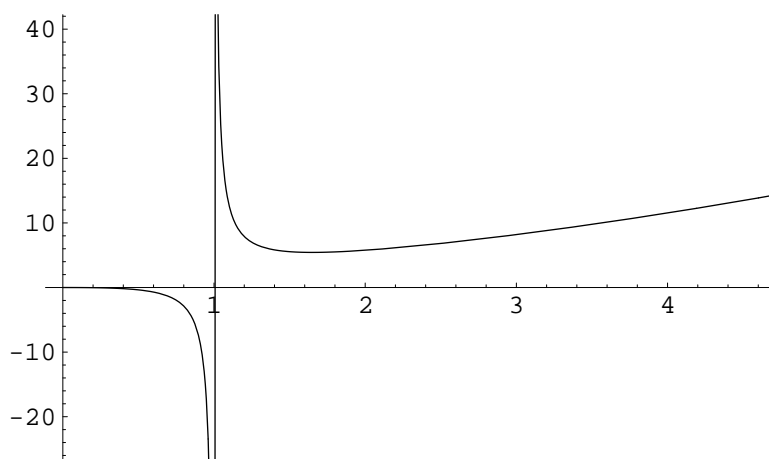
```
Out[110]:= {{2, 3, 3}, {{1, 0, 0}, {1, 1, 0}, {0, 0, 0}}}
```

5. Grafik

■ i. Zweidimensional

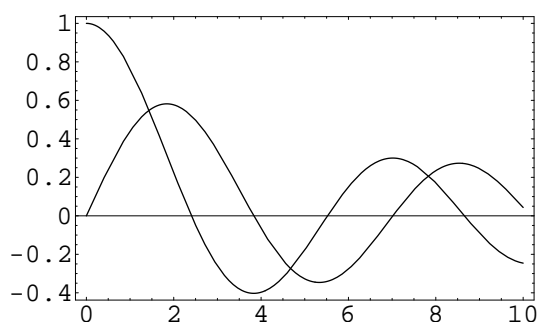
Plot stellt eine Funktion auf einem reellen Intervall dar...

```
In[111]:= Plot[x2 / Log[x], {x, 0, 10}];
```



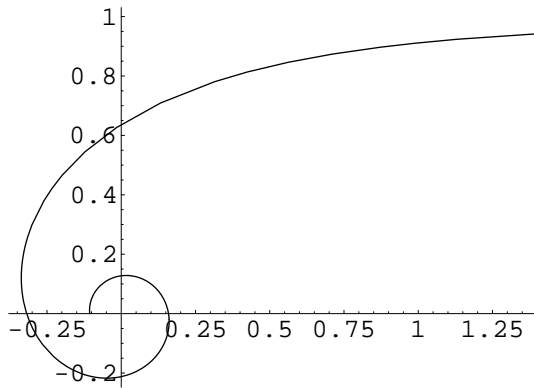
... oder auch mehrere Funktionen. Das folgende Beispiel zeigt auch, wie man eine *Option* an Plot übergibt (hier `Frame → True`).

```
In[112]:= Plot[{BesselJ[0, x], BesselJ[1, x]}, {x, 0, 10}, Frame → True];
```



Parametrische Kurven in der Ebene werden von ParametricPlot dargestellt

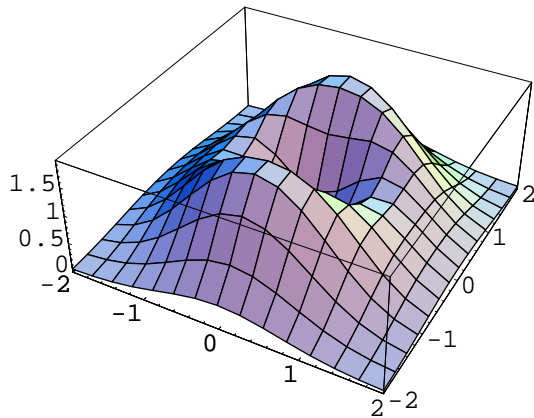
```
In[113]:= ParametricPlot[{Cos[t] / t, Sin[t] / t},
  {t, 0, 3 Pi}, AspectRatio -> Automatic, ImageSize -> 200];
```



■ ii. Dreidimensional

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

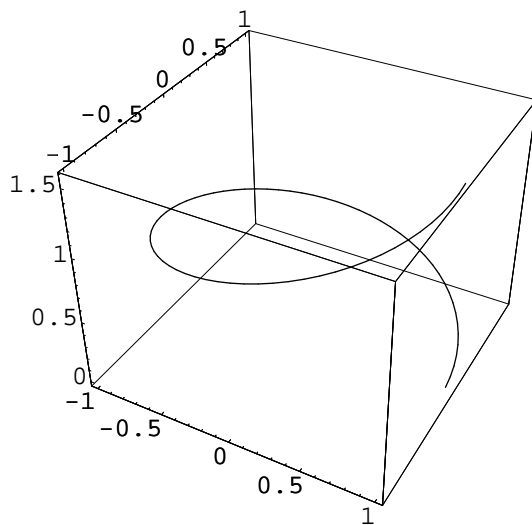
```
In[114]:= Plot3D[(x^2 + 2 y^2) E^{1-x^2-y^2}, {x, -2, 2}, {y, -2, 2}, ImageSize -> 200];
```



```
Out[114]= - SurfaceGraphics -
```

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}

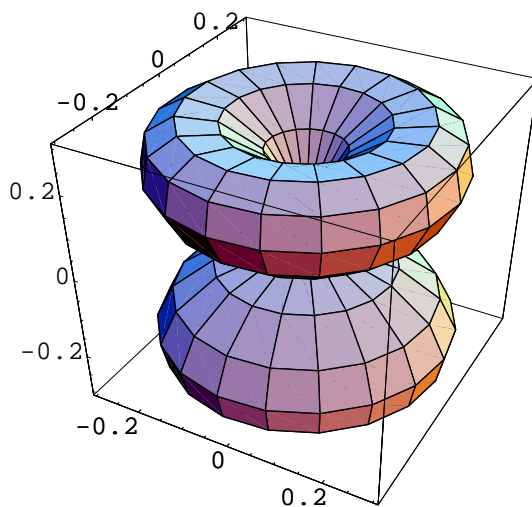
```
In[115]:= ParametricPlot3D[{Cos[t], Sin[t], t/4}, {t, 0, 2 Pi}, ImageSize -> 200];
```



```
Out[115]= - Graphics3D -
```

```
In[116]:= r[t_, p_] := Abs[SphericalHarmonicY[2, -1, t, p]];
```

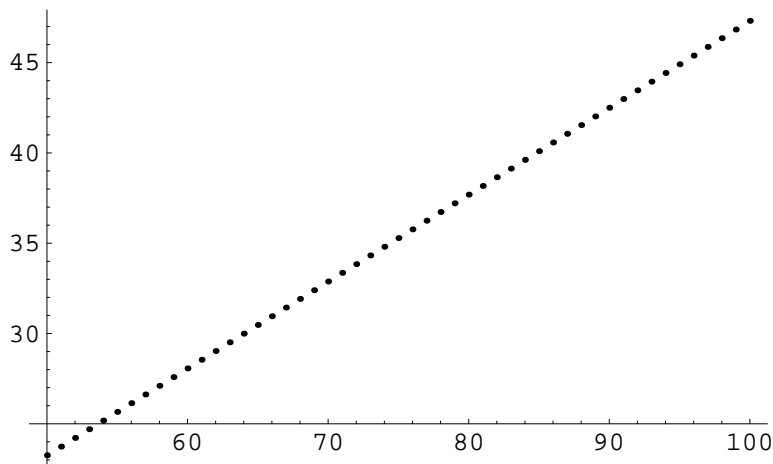
```
In[117]:= ParametricPlot3D[r[θ, φ] {Sin[θ] Cos[φ], Sin[θ] Sin[φ], Cos[θ]},
  {θ, 0, Pi}, {φ, 0, 2 Pi}, AspectRatio -> 1, ImageSize -> 200];
```



■ iii. Andere Visualisierungsmöglichkeiten

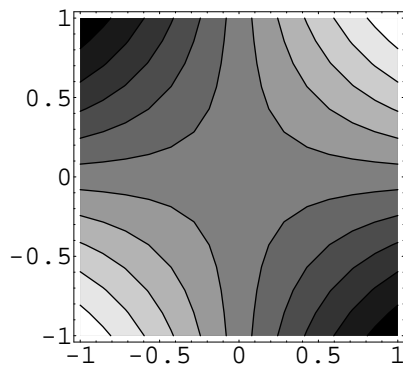
Diskrete Punktmengen werden mit `ListPlot` gezeichnet. Die Option `PlotJoined -> True` verbindet die einzelnen Punkte.

```
In[118]:= ListPlot[Table[{i, Log[Fibonacci[i]]}, {i, 50, 100}]];
```



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

```
In[119]:= 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[120]:= SetOptions[{ListPlot, ParametricPlot, ContourPlot},
  ImageSize -> 150, AspectRatio -> Automatic];
```

■ i. Darstellung der n-ten Einheitswurzeln

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

```
In[121]:= Solve[z^4 == 1, z]
```

```
Out[121]= {{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

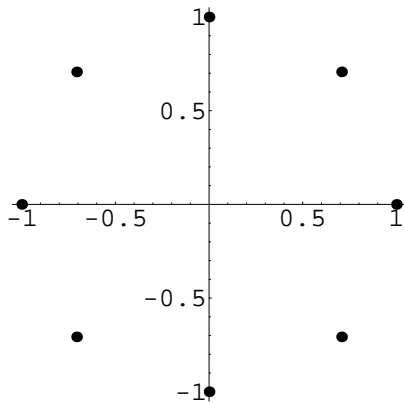
```
In[122]:= einheitsw[n_] := Module[{z}, z /. Solve[z^n == 1, z]]
```

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

```
In[123]:= ewxy = Map[{Re[#], Im[#]} &, einheitsw[8]]
```

```
Out[123]= {{-1, 0}, {0, -1}, {0, 1}, {1, 0},
  {-1/√2, -1/√2}, {1/√2, 1/√2}, {1/√2, -1/√2}, {-1/√2, 1/√2}}
```

```
In[124]:= ListPlot[ewxy, PlotStyle -> {AbsolutePointSize[4]}];
```



```
Out[124]= - Graphics -
```

■ ii. Koch-Kurve(zum Knobeln)

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

```
In[125]:= 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 von `verfeinerung` berechnet, die obiges Dreieck in `n` rekursiven Schritten "verfeinert" und eine Kantenliste zurückliefert.

```
In[126]:= kochkurve[n_Integer] := Graphics[Map[Line, verfeinerung[dreieck, n]]]
```

Diese Verfeinerung läuft wie folgt ab:

```
In[127]:= verfeinerung[kanten_, 0] := kanten;
verfeinerung[kanten_List, n_Integer] := verfeinerung[
  Flatten[Map[auswoelben, kanten], 1],
  n - 1]
```

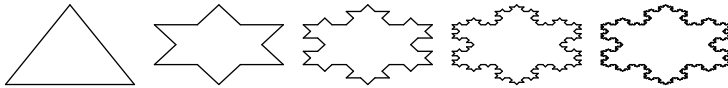
Schließlich das Auswölben einer Kante: Das mittlere Drittel jeder Kante wird entfernt und in die Lücke die Spitze eines kleineren gleichseitigen Dreiecks eingesetzt

```
In[129]:= << Geometry`Rotations`
rotmat = RotationMatrix2D[60 Degree]
```

```
Out[130]= {{1/2, √3/2}, {-√3/2, 1/2}}
```

```
In[131]:= 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[132]:= GraphicsArray[Table[kochkurve[i], {i, 0, 4}]] // Show
```



```
Out[132]:= - GraphicsArray -
```

■ iii. Symmetrische Matrizen

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

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

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

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

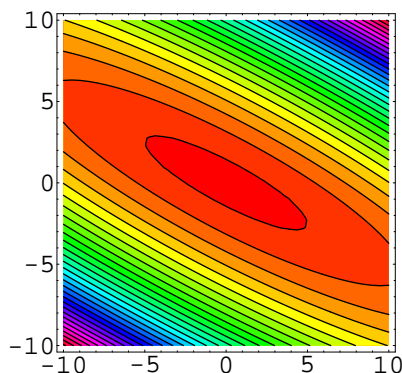
In beiden Fällen sind die mithilfe der Matrizen gebildeten Ausdrücke positiv definite *quadratische Formen*. Diese haben die besondere Eigenschaft, dass Punkte mit konstantem Funktionswert auf Hyperellipsoiden im jeweiligen Vektorraum liegen (etwa der Trägheitsellipsoid der ω -Vektoren mit konstanter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

```
In[133]:= quadform[mtx_, vec_] := vec.mtx.vec
```

```
In[143]:= 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[135]:= 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[136]:= qform[r_, t_] = Simplify[quadform[m, {r Cos[t], r Sin[t]}]]
```

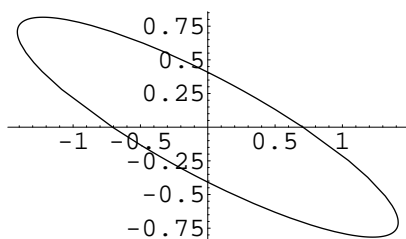
```
Out[136]:= -r^2 (-4 + 2 Cos[2 t] - 3 Sin[2 t])
```

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

```
In[137]:= r[t_] = Part[r /. Solve[qform[r, t] == 1, r], 2]
```

```
Out[137]:= 1 / Sqrt[4 - 2 Cos[2 t] + 3 Sin[2 t]]
```

```
In[138]:= g = ParametricPlot[{r[t] * Cos[t], r[t] * Sin[t]}, {t, 0, 2 Pi}]
```



```
Out[138]= - 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?).

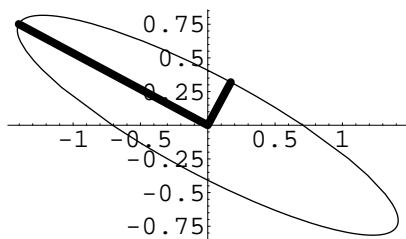
```
In[139]:= {ew, ev} = Eigensystem[m]
```

```
Out[139]= {{4 - Sqrt[13], 4 + Sqrt[13]}, {{1/3 (-2 - Sqrt[13]), 1}, {1/3 (-2 + Sqrt[13]), 1}}}
```

```
In[140]:= ev2 = Map[# / Sqrt[#.#] &, ev]; (* Eigenv. normieren *)
```

```
In[141]:= li = Line[{ev2[[1]] / Sqrt[ew[[1]]], {0, 0}, ev2[[2]] / Sqrt[ew[[2]]]}];
```

```
In[142]:= Show[{g, Graphics[{AbsoluteThickness[3], li}]]]
```



```
Out[142]= - Graphics -
```