Notebook zu: Mathematik für Informatiker
Band 1

Dieses Notebook ist als Ergänzung zu unserem Buch Mathematik für Informatiker, Band 1, 4. Aufl., Springer 2013, gedacht. Dort finden Sie auch weitere Erklärungen.

Logik und Mengen

Mengen

Mengen können in Mathematica als Listen eingegeben werden:

```
In[1]:= A = {a, b, c}; B = {c, d, e};
```

Vereinigung

```
In[2]:= Union[A, B]
Out[2]= {a, b, c, d, e}
```

Durchschnitt

```
In[3]:= Intersection[A, B]
Out[3]= {c}
```

Zur Vereinfachung (Sortieren und Entfernen von Mehrfachelementen) kann man einfach folgendes verwenden:

```
In[4]:= Union[{a, c, b, c}]
Out[4]= {a, b, c}
```

```
In[5]:= A = {a, c, b, c}; B = {c, b, a};
```

Achtung: Unsere Mengen sind Listen und werden auch also solche behandelt. Um z.B auf Gleichheit zu testen muss man sich daher eines kleinen Tricks bedienen:

```
In[6]:= {A == B, Union[A] == Union[B]}
Out[6]= {False, True}
```

Schaltalgebra

Mathematica verwendet **False**, **True** für 0, 1 und kennt eine Reihe logischer Verknüpfungen: Nega-

tion **Not[**a**]** (oder ! a*)*, Und **And[**a,b**]**} (oder a && b), Oder **Or**[a,b] (oder a || b).

In[7]:=
```
a = True; b = False;
{Not[a], And[a, b], Or[a, b]}
```
Out[8]=
```
{False, False, True}
```

In[9]:=
```
{! a, a && b, a || b}
```
Out[9]=
```
{False, False, True}
```

In[10]:=
```
Clear[a, b]
```

Mit folgendem Programm können wir leicht Wertetabellen erstellen:

In[11]:=
```
LogicTable[f_, v_List] := Module[{n = Length[v], tabl, values, rules},
  tabl = {Flatten[{v, f}]};
  Do[
   values = IntegerDigits[i, 2, n] /. {0 -> False, 1 -> True};
   rules = Table[Rule[v[[i]], values[[i]] ], {i, n}];
   tabl = Append[tabl, Flatten[{values, f /. rules}]];
   , {i, 0, 2^n - 1}];
  TableForm[tabl]
 ]
```

Grübeln Sie nicht darüber, wie dieses Programm funktioniert, sondern rufen Sie es einfach mit einem logischem Ausdruck (oder einer Liste von logischen Ausdrücken) und einer Liste der Variablen auf:

In[12]:=
```
LogicTable[{! a, ! (a || b), ! a && ! b}, {a, b}]
```
Out[12]//TableForm=

a	b	! a	! (a || b)	! a && ! b
False	False	True	True	True
False	True	True	False	False
True	False	False	False	False
True	True	False	False	False

Mathematica kann übrigens auch logische Ausdrücke vereinfachen :

In[13]:=
```
LogicalExpand[! a && ! b || ! a && b || a && b]
```
Out[13]=
```
b || ! a
```

In[14]:=
```
LogicalExpand[Implies[a, b]]
```
Out[14]=
```
b || ! a
```

Zahlenmengen und Zahlensysteme

Approximation von $\sqrt{2}$

Das auf im Buch beschriebene Programm zur Annäherung der Wurzel aus 2 kann mit Mathematica wie folgt implementiert werden:

In[15]:=
```
d[q_] := Module[{p = q},
  While[(p/q)^2 < 2, p = p + 1];
  {(p - 1)/q, p/q}]
```

Der Befehl **Module** fasst mehrere Befehle zusammen. Das erste Argument ist dabei eine Liste von lokalen Variablen.

In[16]:=
```
d[100]
```

Out[16]=
$$\left\{\frac{141}{100}, \frac{71}{50}\right\}$$

In[17]:=
```
Clear[d]
```

Ungleichungen

Mathematica kann auch mit Ungleichungen umgehen. Der **Simplify**-Befehl kann zum Überprüfen von Ungleichungen verwendet werden :

In[18]:=
```
Simplify[x/(x^2 + y^2) < 1/y, x > 0 && y > 0]
```

Out[18]=
```
True
```

Mit **Reduce** können Ungleichungen sogar aufgelöst werden:

In[19]:=
```
Reduce[1 - x^2 > 0, x, Reals]
```

Out[19]=
$$-1 < x < 1$$

Komplexe Zahlen

Mit komplexen Zahlen rechnet man folgendermaßen :

In[20]:=
```
z1 = 1 + 2 I; z2 = 3 - I; z1/z2
```

Out[20]=
$$\frac{1}{10} + \frac{7\,i}{10}$$

Die imaginäre Einheit kann entweder über die Tastatur (als großes I) oder über die Palette (als i) eingegeben werden. Real - bzw. Imaginärteil, komplexe Konjugation und Absolutbetrag erhält man mit

In[21]:=
```
{Re[z1], Im[z1], Conjugate[z1], Abs[z1]}
```

Out[21]=
$$\left\{1, 2, 1 - 2\,i, \sqrt{5}\right\}$$

Manchmal muss man mit dem Befehl **ComplexExpand** noch nachhelfen, damit das Ergebnis in Real - und Imaginärteil aufgespalten wird :

In[22]:= $\sqrt{1 + I\sqrt{3}}$

Out[22]= $\sqrt{1 + i\sqrt{3}}$

In[23]:= `ComplexExpand[%]`

Out[23]= $\sqrt{\frac{3}{2}} + \frac{i}{\sqrt{2}}$

Mehr noch, Mathematica geht bei allen Variablen standardmäßig davon aus, dass sie komplexwertig sind. Deshalb wird zum Beispiel der Ausdruck

In[24]:= $\text{Simplify}\left[\frac{\sqrt{a\,b}}{\sqrt{a}}\right]$

Out[24]= $\frac{\sqrt{a\,b}}{\sqrt{a}}$

nicht zu $\sqrt{b}$ vereinfacht, denn das stimmt im Allgemeinen nur für a> 0! Abhilfe schafft in so einem Fall die Möglichkeit, im **Simplify**-Befehl die Zusatzinformation a > 0 zu geben :

In[25]:= $\text{Simplify}\left[\frac{\sqrt{a\,b}}{\sqrt{a}}, a > 0\right]$

Out[25]= $\sqrt{b}$

Achtung : Nicht alle Wurzelgesetzte bleiben im Komplexen richtig :

In[26]:= $\sqrt{-I}\,\sqrt{-I} == \sqrt{(-I)\,(-I)}$

Out[26]= False

In[27]:= `Clear[z1, z2]`

Summen- und Produktzeichen

Das Summenzeichen kann entweder direkt über die Palette eingegeben werden,

In[28]:= $\sum_{k=1}^{n} k$

Out[28]= $\frac{1}{2} n\,(1 + n)$

oder auch als **Sum[**k, {k, 1, n}**]**. Wie Sie sehen, wertet Mathematica (falls möglich) Summen sofort aus. Analog für Produkte mit **Product[**k, {k, 1, n}**]** oder über die Palette:

In[29]:= $\prod_{k=1}^{n} k$

Out[29]= $n!$

Umwandlung zwischen Zahlensystemen

Der Mathematica-Befehl **BaseForm[**x, b**]** wandelt die Dezimalzahl *x* in eine Zahlendarstellung mit Basis *b* um. Zum Beispiel wird die Zahl $(0.1)_{10}$ mit

```
In[30]:= BaseForm[0.1, 2]
Out[30]//BaseForm=
0.0001100110011001100110012
```

vom Dezimalsystem ins Dualsystem umgewandelt. Die Umwandlung einer Zahl *x* von einem System mit Basis *b* ins Dezimalsystem erhält man mit b^^x:

```
In[31]:= 16^^FAD
Out[31]= 4013
```

wandelt die Hexadezimalzahl $(FAD)_{16}$ ins Dezimalsystem um oder

```
In[32]:= 8^^67
Out[32]= 55
```

wandelt die Oktalzahl $(67)_8$ ins Dezimalsystem um.

Teilbarkeit und Primzahlen

Mit dem Mathematica-Befehl **PrimeQ** kann man feststellen, ob eine Zahl eine Primzahl ist :

```
In[33]:= PrimeQ[4]
Out[33]= False
```

Das „Q "steht dabei für „question ".Mit einer **Do**-Schleife können wir zum Beispiel die Liste aller Primzahlen bis 5 ausgeben lassen :

```
In[34]:= Do[
  If[PrimeQ[n], Print[n]],
  {n, 1, 5}];

2
3
5
```

Der Befehl zur Primfaktorzerlegung heißt **FactorInteger** und liefert die Liste aller Primfaktoren, zusammen mit der zugehörigen Vielfachheit:

```
In[35]:= FactorInteger[180]
Out[35]= {{2, 2}, {3, 2}, {5, 1}}
```

also $180 = 2^2 \cdot 3^2 \cdot 5^1$. Der größte gemeinsame Teiler kann mit dem Befehl **GCD** (“greatest common divisor”) berechnet werden:

```
In[36]:= GCD[75, 38]
Out[36]= 1
```

Die Zahlen 75 und 38 sind also teilerfremd.Der Rest der Division einer ganzen Zahl x durch die natürliche Zahl m wird mit **Mod[**x, m**]** erhalten :

In[37]:=
```
Mod[22, 5]
```
Out[37]= 2

Der Quotient der Division wird mit

In[38]:=
```
Quotient[22, 5]
```
Out[38]= 4

berechnet. Also ist 22 = 4·5 + 2.

Elementare Begriffe der Zahlentheorie

Rest modulo m

Der Rest von *a* modulo *m* wird mit **Mod[**a, m**]** berechnet :

In[39]:=
```
Mod[17, 5]
```
Out[39]= 2

Multiplikatives Inverses

Das multiplikative Inverse 1 in $\mathbb{Z}^m$ kann mit **PowerMod[**e, –1, m**]** berechnet werden :

In[40]:=
```
PowerMod[4, -1, 9]
```
Out[40]= 7

also 1 = 7 in $\mathbb{Z}^9$. Allgemein berechnet **PowerMod[**e, k, m**]** die Potenz e^k modulo *m*. In Mathematica kann man nicht einfach *e* – 1 schreiben, denn woher soll das arme Programm wissen, ob Sie in $\mathbb{R}$ oder in $\mathbb{Z}^m$ rechnen wollen!

Euklidscher Algorithmus

Der Euklid' sche Algorithmus kann wie folgt implementiert werden :

In[41]:=
```
Euklid[a_Integer, b_Integer] := Module[{r = a, rr = b},
   While[ rr != 0, {r, rr} = {rr, Mod[r, rr]}];
   r];
```

In[42]:=
```
Euklid[75, 38]
```
Out[42]= 1

(„ != "bedeutet „ungleich "). Der ggT (75, 38) ist also 1. Natürlich hätten wir auch gleich den internen Mathematica-Befehl **GCD** für den größten gemeinsamen Teiler verwenden können :

In[43]:=
```
GCD[75, 38]
```
Out[43]= 1

Alternativ können wir den Algorithmus auch rekursiv implementieren:

In[44]:=
```
Euklid[a_Integer, 0] = a;
Euklid[a_Integer, b_Integer] := Euklid[b, Mod[a, b]];
```

In[45]:=
```
Euklid[75, 38]
```
Out[45]= 1

Analog kann der erweiterte Euklid'sche Algorithmus so programmiert werden:

In[46]:=
```
ExtendedEuklid[a_Integer, b_Integer] :=
  Module[{r = a, rr = b, x = 1, xx = 0, y = 0, yy = 1, Q},
   While[ rr != 0,
    Q = Quotient[r, rr];
    {r, rr, x, xx, y, yy} = {rr, Mod[r, rr], xx, x - Q xx, yy, y - Q yy}
   ];
   {r, x, y}];
```

In[47]:=
```
ExtendedEuklid[75, 38]
```
Out[47]= {1, -1, 2}

Ausgegeben werden also der ggT (75, 38) = 1 und ganzzahlige Lösungen x = –1 und y = 2 der diophantischen Gleichung 75 x + 38 y = ggT (75, 38). Wieder gibt es einen internen Mathematica-Befehl dazu : **ExtendedGCD[**a, b**]** gibt die Liste {g, {x, y}} aus, wobei g = ggT (a, b) und x, y ganzzahlige Lösungen von ax + by = g sind.

In[48]:=
```
ExtendedGCD[75, 38]
```
Out[48]= {1, {-1, 2}}

RSA

Die Verschlüsselung mittels RSA-Algorithmus ist natürlich zu aufwendig, um sie von Hand durchzuführen (aber auch für langsame Computer – Stichwort Chipkarten – kann die Geschwindigkeit bei RSA zu einem Problem werden). Wir wollen uns hier von Mathematica helfen lassen : Der Befehl **ToCharacterCode** wandelt ein Zeichen (Buchstabe, Ziffer, . . .) und sogar eine ganze Zeichenkette in eine Liste von Zahlen gemäß dem ASCII-Code um :

In[49]:=
```
ToCharacterCode["KLEOPATRA"]
```
Out[49]= {75, 76, 69, 79, 80, 65, 84, 82, 65}

Da im ASCII-Code der Buchstabe A der Zahl 65, B der Zahl 66, usw. entspricht, müssen wir – um die gewünschte Zuordnung A = 0, B = 1 usw.zu erhalten – noch 65 subtrahieren :

In[50]:=
```
x = % - 65
```
Out[50]= {10, 11, 4, 14, 15, 0, 19, 17, 0}

Das ist nun der in Zahlen codierte Klartext, der verschlüsselt werden soll. Für die Verschlüsselung von x benötigen wir den öffentlichen Schlüssel

In[51]:=
```
n = 1147; e = 29;
```

Nun können wir mit der Vorschrift $y = x^e \pmod n$ verschlüsseln :

In[52]:=
```
y = PowerMod[x, e, n]
```
Out[52]= {803, 730, 132, 547, 277, 0, 979, 42, 0}

Hier ist **PowerMod[**x, e, n**]** eine effektivere Variante von Mod[x^e, n**]**. Der Empfänger kann mit dem

geheimen Schlüssel d und der Vorschrift $x = y^d \pmod n$ entschlüsseln:

```
In[53]:= d = 149; PowerMod[y, d, n]
Out[53]= {10, 11, 4, 14, 15, 0, 19, 17, 0}
```

Bei unserem kurzen Spielzeugschlüssel ist es natürlich für einen Angreifer kein Problem den Algorithmus zu knacken, d.h.n zu faktorisieren :

```
In[54]:= FactorInteger[n]
Out[54]= {{31, 1}, {37, 1}}
```

zerlegt den Modul n = 1147 in seine Primfaktoren p = 31 und q = 37. Damit können wir m berechnen :

```
In[55]:= m = (31 - 1) (37 - 1)
Out[55]= 1080
```

m kann auch alternativ mittels m = **EulerPhi[**n**]** berechnet werden:

```
In[56]:= EulerPhi[n]
Out[56]= 1080
```

Der geheime Schlüssel d ist nun die Lösung der Gleichung e d = 1 (mod m), wobei e = 29 der öffentliche Schlüssel ist und m = 1080 gerade vom Angreifer gefunden wurde. d kann mit dem Befehl **PowerMod** berechnet werden :

```
In[57]:= PowerMod[e, -1, m]
Out[57]= 149
```

```
In[58]:= Clear[x, y, n, e, d, m]
```

Chinesischer Restsatz

Das System von Kongruenzen x = $a_1 \pmod{m_1}$, ..., x = $a_k \pmod{m_k}$ kann mit dem Befehl **ChineseRemainder[**{$a_1, \ldots, a_k$}, {$m_1, \ldots, m_k$}**]** gelöst werden :

```
In[59]:= ChineseRemainder[{2, 3, 2}, {3, 5, 7}]
Out[59]= 23
```

Ausgegeben wird die kleinste nichtnegative Lösung x, hier x = 23.

Polynomringe und endliche Körper

Faktorisierung

Polynome können mit dem Befehl

```
In[60]:= Factor[x^3 - 1]
Out[60]= (-1 + x) (1 + x + x^2)
```

faktorisiert und mit dem Befehl

In[61]:=
```
Expand[%]
```
Out[61]= $-1+x^3$

ausmultipliziert werden. Eine Faktorisierung erfolgt nur, wenn die Nullstellen rationale Zahlen sind. Beispielsweise wird die Faktorisierung $x^2-2=(x-\sqrt{2})(x-+\sqrt{2})$ von Mathematica nicht durchgeführt:

In[62]:=
```
Factor[x^2 - 2]
```
Out[62]= $-2+x^2$

In[63]:=
```
Factor[x^2 - 2, Extension → Sqrt[2]]
```
Out[63]= $-\left(\sqrt{2}-x\right)\left(\sqrt{2}+x\right)$

Polynomdivision

Den Quotienten s(x) der Polynomdivision p(x) = s(x) q(x) + r(x) können wir mit **PolynomialQuotient[**p (x), q (x), x**]**

In[64]:=
```
PolynomialQuotient[3 x^4 + x^3 - 2 x, x^2 + 1, x]
```
Out[64]= $-3+x+3\,x^2$

und den Rest r(x) mit **PolynomialRemainder[**p(x), q(x), x**]**

In[65]:=
```
PolynomialRemainder[3 x^4 + x^3 - 2 x, x^2 + 1, x]
```
Out[65]= $3-3\,x$

berechnen.

Euklid'scher Algorithmus

In Mathematica können wir den Euklid' schen Algorithmus analog wie für ganze Zahlen definieren :

In[66]:=
```
PolynomialEuklid[p_, q_] := Module[{r = p, rr = q},
  While[rr =!= 0, {r, rr} = {rr, PolynomialRemainder[r, rr, x]}];
  r/CoefficientList[r, x][[-1]]];
```
(Im letzten Schritt wird dabei nur noch der höchste Koeffizient von r auf eins normiert.)

In[67]:=
```
PolynomialEuklid[x^3 - 2 x + 1, x^2 - 1]
```
Out[67]= $-1+x$

Wir hätten übrigens auch gleich den internen Befehl **PolynomialGCD[**a(x), b(x), x**]** verwenden können. Ich wollte aber zeigen, dass bei der Implementierung nur die Division ganzer Zahlen durch die Polynomdivision ersetzt werden muss. Außerdem normiert Mathematica den ggT nicht : So gibt es als ggT von 5 (x3 – 2 x + 1) und 5 (x2 – 1) das Polynom 5 (x – 1) anstelle des zugehörigen normierten Polynoms x – 1 aus.

Analog wird der erweiterte Euklid' sche Algorithmus implementiert :

```
In[68]:= PolynomialExtendedEuklid[p_, q_] :=
           Module[{r = p, rr = q, s = 1, ss = 0, t = 0, tt = 1, Q},
            While[ rr =!= 0,
             Q = PolynomialQuotient[r, rr, x];
             {r, rr, s, ss, t, tt} =
              {rr, PolynomialRemainder[r, rr, x], ss, s - Q ss, tt, t - Q tt}
            ];
            {r, s, t}
           ----------------------];
           CoefficientList[r, x][[-1]]
```

Als Ergebnis werden also die drei Polynome r(x), s(x), t(x) ausgegeben, wobei r(x) der größte gemeinsame Teiler von p(x) und q(x) ist und s(x), t(x) die Beziehung s(x)p (x) + t(x) q(x) = r(x) erfüllen. Damit berechnen wir

```
In[69]:= PolynomialExtendedEuklid[x^3 - 2 x + 1, x^2 - 1]
Out[69]= {-1 + x, -1, x}
```

Also sind r(x) = x – 1, s(x) = –1 und t(x) = x.

Die analogen internen Befehle lauten:

```
In[70]:= PolynomialGCD[x^3 - 2 x + 1, x^2 - 1]
Out[70]= -1 + x

In[71]:= PolynomialExtendedGCD[x^3 - 2 x + 1, x^2 - 1]
Out[71]= {-1 + x, {-1, x}}
```

Multiplikatives Inverses

In Mathematica kann der Befehl **PolynomialRemainder[**$a\,(x)^{-1}$, m(x), x, Modulus → p**]** verwendet werden, um das multiplikative Inverse von a(x) in $\mathbb{Z}_p[x]_{m(x)}$ zu berechnen :

```
In[72]:= PolynomialRemainder[(x + 1)^-1, x^2 + x + 1, x, Modulus → 2]
Out[72]= x
```

Faktorisierung

Mathematica kann natürlich auch Polynome aus $\mathbb{Z}_m[x]$ faktorisieren. Ein Beispiel aus $\mathbb{Z}_2[x]$:

```
In[73]:= Factor[x^8 + x^4 + x^3 + x + 1, Modulus → 2]
Out[73]= 1 + x + x^3 + x^4 + x^8
```

Das Polynom $x^8 + x^4 + x^3 + x + 1$ ist also über$\mathbb{Z}_2$ irreduzibel. Über $\mathbb{Z}_3$ ist es reduzibel:

```
In[74]:= Factor[x^8 + x^4 + x^3 + x + 1, Modulus → 3]
Out[74]= (1 + x^2) (2 + x + x^2) (2 + x + x^2 + 2 x^3 + x^4)
```

Folgen und Reihen

Folgen

Mit Mathematica können wir eine Folge einfach als eine Funktion definieren,

In[75]:=
```
a[n_] := 1/2^n
```

und damit leicht Folgenglieder berechnen:

In[76]:=
```
Table[a[n], {n, 4}]
```

Out[76]=
$$\left\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}\right\}$$

gibt uns die ersten Folgenglieder aus. Analog können rekursiv definierte Folgen eingegeben werden, wie hier die Heron' sche Folge mit Startwert $a_1 = 2$,

In[77]:=
```
a[1] = 2.; a[n_] := 1/2 (a[n - 1] + 2/a[n - 1])
```

und die Folgenglieder berechnet werden:

In[78]:=
```
Table[a[n], {n, 5}]
```

Out[78]=
```
{2., 1.5, 1.41667, 1.41422, 1.41421}
```

Bei rekursiven Folgen muss aber unbedingt die Definition mit ":= "erfolgen, damit die rechte Seite der Definition nicht sofort, sondern erst bei Angabe einer konkreten Zahl n ausgewertet wird. Außerdem darf auch der Startwert a1 nicht vergessen werden! Wenn Mathematica a_n berechnet, so drückt es gemäß dem Bildungsgesetz a_n durch a_{n-1} aus. Das geschieht so lange, bis es auf den Startwert trifft. Wurde dieser nicht angegeben (oder wurde der Doppelpunkt vergessen), so schicken Sie Mathematica in eine Endlosschleife. Sie können diese durch den Menüpunkt "Kernel/Abort Evaluation" oder durch die Tastenkombination "ALT"+"."abbrechen.

Die Folge für die Euler' sche Zahl lautet

In[79]:=
```
a[n_] := (1 + 1/n)^n
```

und die im Buch gesuchten Folgenglieder sind:

In[80]:=
```
N[{a[2], a[3], a[10], a[100], a[1000], a[10 000]}]
```

Out[80]=
```
{2.25, 2.37037, 2.59374, 2.70481, 2.71692, 2.71815}
```

Erinnern Sie sich, dass der Befehl **N[...]** bewirkt, dass das Ergebnis numerisch angezeigt wird. Der Grenzwert einer Folge kann mit dem Befehl **Limit** berechnet werden :

In[81]:=
```
Limit[a[n], n → ∞]
```

Out[81]=
$$e$$

Der Pfeil →, das Symbol ∞ und die Konstante e werden in Mathematica entweder über die Palette oder über die Tastatur als ->, Infinity bzw. E eingegeben.

Einen numerische Wert von e auf 30 Stellen genau erhalten wir mit dem Befehl **N[**E, 30**]**.

In[82]:=
```
N[E, 30]
```
Out[82]=
```
2.71828182845904523536028747135
```

In[83]:=
```
Clear[a]
```

Heron'sche Folge

Ein Programm zur Berechnung von Wurzeln mithilfe der Heron' schen Folge kann in Mathematica so implementiert werden:

In[84]:=
```
MySqrt[x_] := Module[{h = N[x], hh = N[x] + 1},
  While[hh - h > 0, hh = h; h = 1/2 (h + x/h)];
  hh]
```

Dabei wird h mit dem numerischen Wert von x initialisiert (rechnen wir symbolisch, so würden wir endlos iterieren; -). Dann wird in jedem Schritt der letzte Wert als hh gespeichert und ein neuer Wert h berechnet, bis hh > h gilt.

In[85]:=
```
MySqrt[2]
```
Out[85]=
```
1.41421
```

Reihen

Mit Mathematica berechnet man den Grenzwert einer Reihe mit:

In[86]:=
$$\sum_{k=0}^{\infty} \frac{1}{2^k}$$
Out[86]= 2

Das kann auch ohne Palette als

In[87]:=
```
Sum[1 / 2^k, {k, 0, Infinity}]
```
Out[87]= 2

eingegeben werden. Die n-te Teilsumme erhalten wir mit

In[88]:=
$$s[n\_] = \sum_{k=1}^{n} \frac{1}{k^2};$$

bzw.numerische Werte davon mit

In[89]:=
```
N[{s[1], s[3], s[10], s[100], s[1000], s[10000]}]
```
Out[89]=
```
{1., 1.36111, 1.54977, 1.63498, 1.64393, 1.64483}
```

Der Grenzwert dieser Reihe ist

In[90]:=
$$\sum_{k=1}^{\infty} \frac{1}{k^2}$$
Out[90]=
$$\frac{\pi^2}{6}$$

bzw. als Kommazahl ausgegeben:

```
In[91]:= N[%]
Out[91]= 1.64493
```

Kombinatorik

Fakultät und Bimomialkoeffizient

In Mathematica wird die Fakultät mit

```
In[92]:= 6!
Out[92]= 720
```

und der Binomialkoeffizient mit

```
In[93]:= Binomial[45, 6]
Out[93]= 8 145 060
```

berechnet.

Rekursionen und Wachstum von Algorithmen

Iteration und Chaos

Um Folgenglieder einer rekursiv definierten Folge effektiv zu berechnen, bietet sich eine **Do**-Schleife an:

```
In[94]:= f[μ_, x_] := x + μ (1 - x) x;
         F[x_, μ_, n_] := Module[{xx = x}, Do[xx = f[μ, xx], {n}];
           xx];
```

Hier wurde die Rekursion $x_n = x_n - 1 + \mu\,(1 - x_{n-1})\,x_{n-1}$ definiert.Den Buchstaben μ können wir (wie jeden griechischen Buchstaben) mithilfe der Palette oder über die Tastatur mit der Tastenkombination “ESC m ESC” eingeben.

Noch schneller, aber dafür auch noch kryptischer, würde es mit folgender Variante gehen:

```
In[96]:= f[μ_, x_] := x + μ (1 - x) x;
         F[x_, μ_, n_] := Nest[f[μ, #1] &, x, n];
```

Folgenglieder kann man sich nun bequem mit

```
In[98]:= Table[F[0.1, 2.2, n], {n, 1, 10}]
Out[98]= {0.298, 0.758231, 1.16153, 0.748766, 1.16262,
          0.746676, 1.16281, 0.746316, 1.16284, 0.746258}
```

ausgeben lassen. Ein “Spinnennetz” entsteht mit folgendem kleinen Programm :

In[99]:=
```
ShowWeb[f_, xstart_, nmax_, opts___] :=
 Module[{x, xmin, xmax, delta, graph, web, lines},
   lines = Type /. {opts, Type → Line};
   x[0] := xstart;
   x[n_] := x[n] = f[x[n - 1]];
   If[x[0] == x[1], Return[Print["Fixed Point!"]]];
   web = Flatten[Table[{{x[n], x[n]}, {x[n], x[n + 1]}}, {n, 0, nmax}], 1];
   xmax = Max[web]; xmin = Min[web]; delta = 0.1 (xmax - xmin);
   graph = Plot[{f[x], x}, {x, xmin - delta, xmax + delta}];
   Show[graph, Graphics[lines[web]]]
 ] /; (IntegerQ[nmax] && Positive[nmax])
```

Gehen wir es kurz Schritt für Schritt durch. Es übernimmt als Argument die zu iterierende Funktion f, den Startwert xstart, die Anzahl der durchzuführenden Iterationen nmax und weitere Optionen opts (der dreifache Unterstrich steht für eine beliebige Anzahl (inkl. keiner) weiterer Argumente). Als Erstes wird in der Variable lines der Verbindungstyp festgelegt. Der Defaultwert ist Line und kann mit einer optionalen Ersetzungsregel Type → wert (als letztes Argument unseres Programms) geändert werden. Nun werden rekursiv die zu verbindenden Punkte (x_n, x_n) und (x_n, x_{n+1}) berechnet (der **Flatten**-Befehl wirft nur ein paar überschüssige Klammern weg). Über das Minimum bzw. Maximum der berechneten Werte x_n wird der darzustellende Bereich ermittelt (dieser Wert wird noch um 10 % vergrößert, damit nichts Wertvolles abgeschnitten wird). Nun wird noch die Funktion f(x) zusammen mit x gezeichnet und die Grafik mit den durch Linien verbundenen Punkten zusammengefügt.

Wenn wir die Linien durch Pfeile ersetzen möchten, dann können wir als Verbindungstyp folgende Funktion, die eine Liste von Punkten durch Pfeile verbindet, verwenden:

In[100]:=
```
Arrows[l_List] :=
  {Arrowheads[Small], Table[Arrow[{l[[i]], l[[i + 1]]}], {i, Length[l] - 1}]};
```

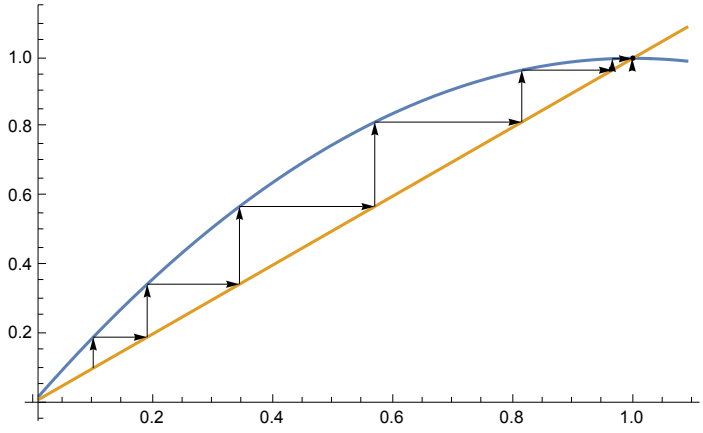
Setzen wir

In[101]:=
```
f[x_] := x + μ x (1 - x);
```

dann zeichnet

In[102]:=
```
μ = 1; ShowWeb[f, 0.1, 6, Type → Arrows]
```

Out[102]=

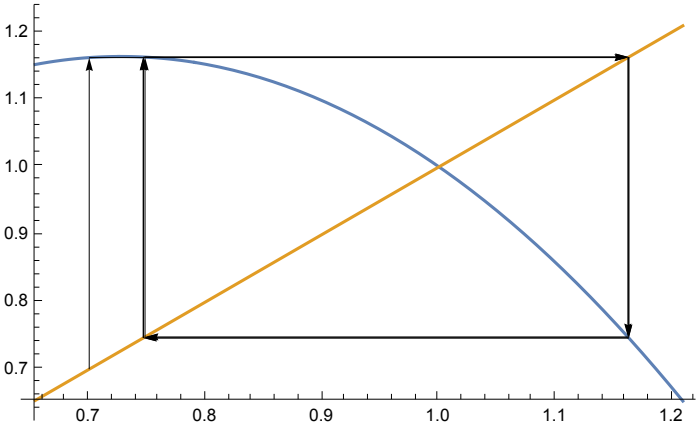


das Spinnennetz der ersten 12 Folgenglieder für $\mu = 1$ mit dem Startwert $x_0 = 0.1$, das im Buch abgebildet ist.

Einige weitere Grafiken:

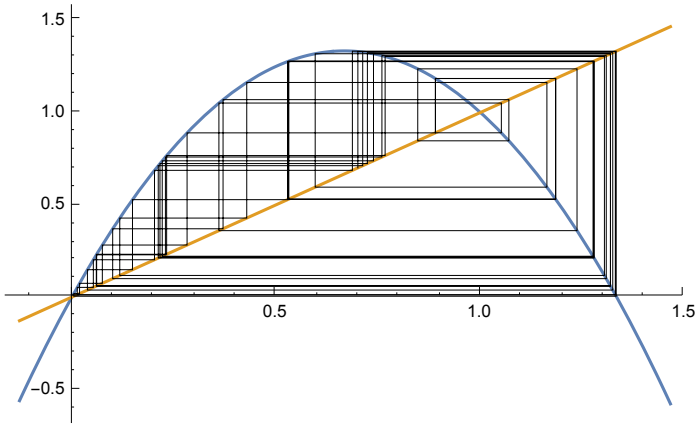
In[103]:=
```
μ = 2.2; ShowWeb[f, 0.7, 6, Type → Arrows]
```

Out[103]=



In[104]:=
```
μ = 3; ShowWeb[f, 0.7, 50]
```

Out[104]=



In[105]:=
```
Clear[f, F]
```

Lösung einer Rekursion

In Mathematica können Rekursionen mit dem Befehl **RSolve** gelöst werden.

In[106]:=
```
RSolve[{a[n] == 3 a[n - 1] - 4, a[0] == 5}, a[n], n]
```

Out[106]=
$$\{\{a[n] \to 2 + 3^{1+n}\}\}$$

Analog für Rekursionen höherer Ordnung:

In[107]:=
```
RSolve[{a[n] == a[n - 1] + 6 a[n - 2], a[0] == 1, a[1] == 8}, a[n], n]
```

Out[107]=
$$\{\{a[n] \to -(-2)^n + 2 \times 3^n\}\}$$

Landausymbol

Zuerst die schlechte Nachricht : Mathematica kann nicht mit Landausymbolen rechnen (es kennt zwar das Landausymbol und verwendet es bei der Darstellung von Potenzreihen, kann aber nur sehr eingeschränkt damit rechnen).Nun die gute Nachricht : Mathematica ist eine Programmiersprache, also kann uns niemand daran hindern, Mathematica etwas auf die Sprünge zu helfen, und dem Programm die Regeln für das Landausymbol beizubringen!

Wir definieren zunächst die Funktion **LandauO** für das Landausymbol:

In[108]:=
```
LandauO[f_ + g_] :=
  LandauO[If[tmp == 0, g, f] ] /; FreeQ[tmp = Limit[Abs[f/g], n → ∞], n];
LandauO[c_ f_] := LandauO[f] /; FreeQ[c, n];
```

Die erste Definition **LandauO[**f_ + g_**]** versucht festzustellen, welcher der beiden Summanden schneller wächst. Dazu wird der Grenzwert $\lim_{n\to\infty} = \frac{|f|}{|g|}$ berechnet und getestet, ob es Mathematica gelungen ist, den Grenzwert zu berechnen (in diesem Fall darf das Ergebnis kein n mehr enthalten). Die bedingte Definition F[x_] := expr /; *Bedingung* wendet die Definition für F nur an, falls die Bedingung erfüllt ist. Die zweite Definition wirft Konstanten weg.

Jetzt noch eine handliche Funktion

In[110]:=
```
LandauExpand[expr_] :=
  LandauO[Expand[expr]] /. Log[x_] /; PolynomialQ[x, n] -> Log[n];
```

die den übergebenen Ausdruck expandiert (alle Produkte ausmultipliziert) und berücksichtigt, dass $\log(p_k n^k + p_{k-1} n^{k-1} - 1 + \ldots + p_0) = O(\log(n))$ ist. Die Ersetzungsregel Alt /; Bedingung → Neu wird nur ausgeführt, wenn die Bedingung erfüllt ist.

Nun unser erster Test:

In[111]:=
```
LandauExpand[(n + 1)/(n^2 + 1) + Exp[-n + 3] + 11 n Log[3 n + 1]]
```

Out[111]=
```
LandauO[n Log[n]]
```

Sieht doch ganz gut aus! Ganz ehrlich, hätten Sie das mit der Hand auch geschafft?

Vektorräume

Vektoren

In Mathematica können Vektoren einfach mit geschwungenen Klammern eingegeben werden :

In[112]:=
```
a = {1, 2, 0, -1}; b = {3, 0, 1, 2};
```

Wir erhalten dann zum Beispiel 3 a – b mit

In[113]:=
```
3 a - b
```

Out[113]=
```
{0, 6, -1, -5}
```

Die Länge eines Vektors wird mit

In[114]:=
```
Norm[a]
```

Out[114]=
$$\sqrt{6}$$

berechnet.

In[115]:=
```
Clear[a, b]
```

Matrizen und Lineare Abbildungen

Matrizen

Matrizen werden in Mathematica als verschachtelte Listen eingegeben:

In[116]:=
```
A = {{1, 3, 2}, {-1, 4, 5}}; B = {{-2, 4, 0}, {6, 2, -8}};
```

Manche Großbuchstaben (wie z.B.C, D oder N) sind in Mathematica bereits vordefiniert, und können daher nicht als Variablennamen verwendet werden.

Summe und Multiplikation mit einem Skalar werden wie erwartet gebildet:

In[117]:=
```
2 (A + B)
```

Out[117]=
```
{{-2, 14, 4}, {10, 12, -6}}
```

Das können wir noch in die vertraute Matrixschreibweise bringen:

In[118]:=
```
MatrixForm[%]
```

Out[118]//MatrixForm=

$$\begin{pmatrix} -2 & 14 & 4 \\ 10 & 12 & -6 \end{pmatrix}$$

Wir können sogar

$PrePrint = MatrixForm;

setzen, dann wird jedes Ergebnis automatisch in Matrixform dargestellt.

Matrizen können mithilfe einer Palette bereits in Matrixform eingegeben werden. Von der Palette wird dabei eine (2, 2)-Matrix als Schablone vorgegeben; mit den Tastenkombinationen “Ctrl”+“Enter” bzw. “Ctrl”+“, “ kann eine Zeile bzw.eine Spalte hinzugefügt werden. Alternativ können Sie eine Schablone mit der gewünschten Zeilen und Spaltenanzahl auch mit der rechten Maustaste über das Menü “Create Table/Matrix/Palette” eingeben (engl. : Spalte = column, Zeile = row). Mit der Tabulatortaste kommen Sie in der Schablone von einem Platzhalter zum nächsten.

Die Dimension einer Matrix kann mit dem Befehl **Dimensions** überprüft werden:

In[119]:=
```
Dimensions[A]
```

Out[119]=
```
{2, 3}
```

Die transponierte Matrix erhalten wir mit

In[120]:=
```
Transpose[{{3, 2}, {5, -1}, {0, 7}}] // MatrixForm
```

Out[120]//MatrixForm=

$$\begin{pmatrix} 3 & 5 & 0 \\ 2 & -1 & 7 \end{pmatrix}$$

Die (n, n)-Einheitsmatrix wird mit **IdentityMatrix[**n**]** erhalten:

In[121]:=
```
IdentityMatrix[2] // MatrixForm
```

Out[121]//MatrixForm=

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

Eine Diagonalmatrix diag($a_1, \ldots, a_n$) erhalten Sie mit **DiagonalMatrix[**{ $a_1, \ldots, a_n$}**]** und das Kronecker Delta δ_{jk} mit **KroneckerDelta[**j, k**]**.

In[122]:=
```
DiagonalMatrix[{1, 2, 3}] // MatrixForm
```

Out[122]//MatrixForm=

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

In[123]:=
```
KroneckerDelta[1, 2]
```

Out[123]=
```
0
```

Multiplikation von Matrizen

Achtung : Das Matrixprodukt wird mit einem Punkt "." (und nicht mit dem üblichen Mal-Symbol " * " oder mit Leerzeichen) berechnet!

In[124]:=
```
A = {{4, 2, 0}, {-1, 3, 5}}; B = {{2, 1}, {3, 7}, {1, 0}}; A.B // MatrixForm
```

Out[124]//MatrixForm=

$$\begin{pmatrix} 14 & 18 \\ 12 & 20 \end{pmatrix}$$

Die Eingabe A * B (oder A B) würde als Resultat eine Matrix liefern, die durch elementweises Ausmultiplizieren von A und B entsteht. In diesem Sinn muss das Quadrat M^2 der Matrix

In[125]:=
```
M = {{7, 1}, {2, 5}};
```

mit Punkt

In[126]:=
```
M.M // MatrixForm
```

Out[126]//MatrixForm=

$$\begin{pmatrix} 51 & 12 \\ 24 & 27 \end{pmatrix}$$

eingegeben werden.Im Gegensatz dazu wird

In[127]:=
```
M^2 // MatrixForm
```

Out[127]//MatrixForm=

$$\begin{pmatrix} 49 & 1 \\ 4 & 25 \end{pmatrix}$$

von Mathematica wieder als M * M (elementweises Quadrieren) interpretiert. Höhere Potenzen können wir effizient mit **MatrixPower[**A, n**]** berechnen.

In[128]:=
```
MatrixPower[M, 2] // MatrixForm
```

Out[128]//MatrixForm=

$$\begin{pmatrix} 51 & 12 \\ 24 & 27 \end{pmatrix}$$

Achtung : Die Eingabe A = {{1, 2}, {3, 4}} // MatrixForm kann zu unerwarteten Ergebnissen führen, da MatrixForm vor der Zuweisung (=) angewendet wird:

In[129]:=
```
A = {{1, 2}, {3, 4}} // MatrixForm
```

Out[129]//MatrixForm=

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

In[130]:=
```
A + M
```

Out[130]=

$$\left\{\left\{7 + \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, 1 + \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}\right\}, \left\{2 + \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, 5 + \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}\right\}\right\}$$

Um das zu vermeiden, verwenden Sie Klammern : (A = {{1, 2}, {3, 4}}) // MatrixForm.

In[131]:=
```
Clear[A, B, M]
```

Eingabe von Gleichungen mit Matrizen

Lineare Gleichungen können in der Form Ax = b eingegeben

In[132]:=
```
A = {{5, 3}, {2, -1}}; x = {x1, x2}; b = {2, 3};
A.x == b
```

Out[133]=
```
{5 x1 + 3 x2, 2 x1 - x2} == {2, 3}
```

und gelöst werden:

In[134]:=
```
Solve[%, {x1, x2}]
```

Out[134]=
```
{{x1 → 1, x2 → -1}}
```

In[135]:=
```
Clear[A, B, b, x]
```

Inverse Matrix

Die inverse Matrix wird mit dem Befehl **Inverse** berechnet

In[136]:=
```
A = {{2, 4}, {-1, 3}}; Inverse[A] // MatrixForm
```

Out[136]//MatrixForm=

$$\begin{pmatrix} \frac{3}{10} & -\frac{2}{5} \\ \frac{1}{10} & \frac{1}{5} \end{pmatrix}$$

Matrizen größerer Dimension sind für Mathematica kein Problem:

In[137]:=
```
Inverse[{{0.30, 0.59, 0.11}, {0.60, -0.28, -0.32}, {0.21, -0.52, 0.31}}] // MatrixForm
```

Out[137]//MatrixForm=

$$\begin{pmatrix} 1. & 0.948262 & 0.624013 \\ 1. & -0.276066 & -0.63981 \\ 1. & -1.10545 & 1.72986 \end{pmatrix}$$

In[138]:=
```
Clear[A]
```

In[139]:=
```
$PrePrint = .;
```

Lineare Gleichungen

Gauß-Algorithmus

Wir wissen bereits, dass in Mathematica der Befehl **Solve** zur Lösung eines Gleichungssystems zur Verfügung steht:

In[140]:=
```
Solve[{y + 3 z == 2, x + 2 y + 5 z == 0, 2 x + 5 y + 13 z == 2}, {x, y, z}]
```

Solve: Equations may not give solutions for all "solve" variables.

Out[140]=
```
{{y → -10 - 3 x, z → 4 + x}}
```

Alternativ kann Mathematica ein lineares Gleichungssystem auch auf die Endform des Gauß-Jordan-Algorithmus (reduzierte Zeilenstufenform) bringen. Wir geben dazu die erweiterte Koeffizientenmatrix ein, und verwenden dann die Anweisung **RowReduce**. Für das Beispiel aus dem Buch ergibt sich dann:

In[141]:=
```
A = {{1, -1, -1, 0}, {-1, 0, -3, -11}, {4, -1, 2, 15}};
```

In[142]:=
```
RowReduce[A] // MatrixForm
```

Out[142]//MatrixForm=

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

Einige weitere Beispiele aus dem Buch:

In[143]:=
```
RowReduce[{{1, -1, 7, 6}, {-1, 4, -13, 3}, {2, 1, 8, 17}}] // MatrixForm
```

Out[143]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 5 & 0 \\ 0 & 1 & -2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In[144]:=
```
RowReduce[{{0, 1, 3, 2}, {1, 2, 5, 0}, {2, 5, 13, 2}}] // MatrixForm
```

Out[144]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & -1 & -4 \\ 0 & 1 & 3 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

In[145]:=
```
RowReduce[{{1, 2, -1, 3}, {2, 4, -2, 6}, {3, 6, -3, 9}}] // MatrixForm
```

Out[145]//MatrixForm=

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

In[146]:=
```
Clear[A]
```

Rang, Kern und Bild

In Mathematica kann eine Basis für den Kern mit dem Befehl **NullSpace[**A**]** berechnet werden.

In[147]:=
```
A = {{1, 1, 2}, {0, 1, 1}, {1, 0, 1}}; NullSpace[A]
```

Out[147]=
```
{{-1, -1, 1}}
```

Eine Basis für das Bild von A kann zwar nicht direkt berechnet werden, aber es gibt einen Trick : Um die lineare Hülle der Spaltenvektoren nicht zu verändern, müssten wir statt elementarer Zeilenumformungen elementare Spaltenumformungen machen.Der Befehl **RowReduce** führt aber nur elementare Zeilenumformungen durch. Da aber Transponieren Zeilen und Spalten vertauscht, ist eine elementare Spaltenumformung von A gleich einer elementaren Zeilenumformung von A^T. Somit erhalten wir mit dem Befehl

In[148]:=
```
Transpose[RowReduce[Transpose[A]]] // MatrixForm
```

Out[148]//MatrixForm=

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

Vektoren, die das Bild aufspannen.Der letzte Vektor ist der Nullvektor und kann natürlich weggeworfen werden. Die beiden ersten Vektoren (1, 0, 1) und (0, 1, –1) sind linear unabhängig und bilden somit eine Basis für das Bild. Daher ist insbesondere die Dimension des Bildes (der Rang der Matrix) gleich 2. Das kann auch direkt mit

In[149]:=
```
MatrixRank[A]
```

Out[149]=
```
2
```

berechnet werden.

In[150]:=
```
Clear[A]
```

Determinante

Determinanten werden mit dem Befehl **Det** berechnet:

In[151]:=
```
A = {{1, -1, -1}, {-1, 0, -3}, {4, -1, 2}}; Det[A]
```

Out[151]=
```
6
```

In[152]:=
```
A = {{0, 1, 3}, {1, 2, 5}, {2, 5, 13}}; Det[A]
```

Out[152]=
```
0
```

In[153]:=
```
Clear[A]
```

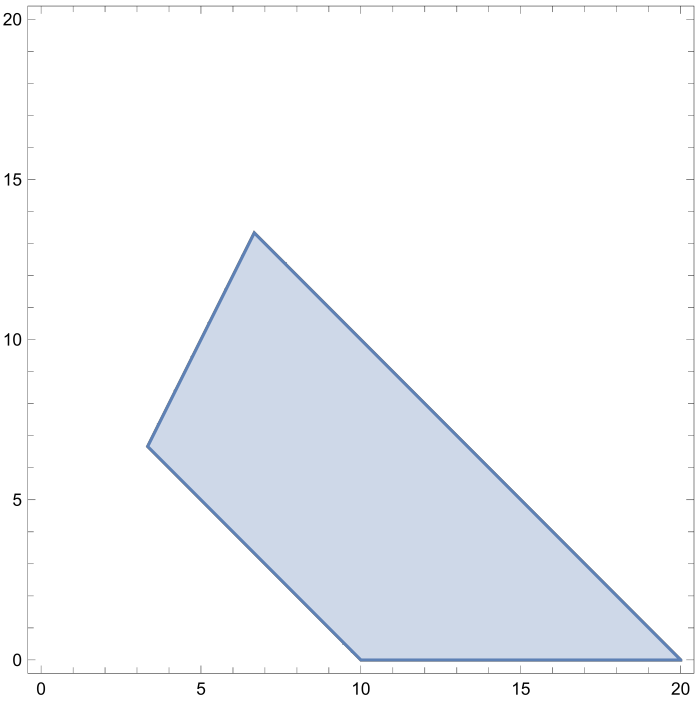
Lineare Optimierung

Lineare Ungleichungen

Mit dem Befehl **RegionPlot** können Ungleichungen dargestellt werden:

In[154]:=
```
RegionPlot[x ≥ 0 && y ≥ 0 && x + y ≤ 20 && x + y ≥ 10 && y ≤ 2 x, {x, 0, 20}, {y, 0, 20}]
```

Out[154]=



Lineare Optimierung

Mit Mathematica kann die Lösung eines linearen Optimierungsproblems mit dem Befehl **LinearProgramming[**c, A, b**]** gefunden werden. Im Argument des Befehls werden zuerst die Koeffizienten c = (c_1, ..., c_n) der Zielfunktion f eingegeben (da die Konstante d für das Finden des Maximums unerheblich ist, wird sie weggelassen); danach die Koeffizientenmatrix A = (a_{ij}) aller Variablen, und zuletzt die zugehörigen rechten Seiten b = (b_1, ..., b_m) der Ungleichungen. Es wird dabei immer das Minimum von f gesucht. Benötigt man das Maximum, so ist einfach f durch – f zu ersetzen. Zum Beispiel:

In[155]:=
```
LinearProgramming[{4, 3}, {{-1, -1}, {1, 1}, {2, -1}}, {-20, 10, 0}]
```

Out[155]=
$$\left\{\frac{10}{3}, \frac{20}{3}\right\}$$

Wir haben hier im Vergleich zur Angabe aus dem Buch zuvor alle Ungleichungszeichen auf die Richtung >= gebracht und das Vorzeichen der Zielfunktion geändert, da wir ja das Maximum suchen.

In[156]:=
```
LinearProgramming[{-3, -4}, {{-1, -2}, {-2, -1}}, {-20, -28}]
```

Out[156]=
```
{12, 4}
```

Wenn man zu faul ist, das Problem auf die Standardform zu bringen, kann man auch die Befehle

Maximize und Minimize verwenden:

In[157]:=
```
Maximize[1.8 - 0.04 x1 - 0.03 x2, x1 ≥ 0 && x2 ≥ 0 && 20 - x1 - x2 ≥ 0 &&
 x1 + x2 ≥ 10 && x2 ≤ 2 x1, {x1, x2}]
```

Out[157]=
```
{1.46667, {x1 → 3.33333, x2 → 6.66667}}
```

Dieser Befehl kann auch für nichtlineare Optimierungsprobleme verwendet werden. Allerdings ist dann meist nur noch eine numerische Lösung möglich.

Skalarprodukt und Orthogonalität

Skalarprodukt

Das Skalarprodukt wird mit einem Punkt berechnet:

In[158]:=
```
a = {1, 2, 3}; b = {2, -4, 1}; a.b
```

Out[158]=
```
-3
```

Die Länge eines Vektors können wir also auch mit

In[159]:=
```
√(a.a)
```

Out[159]=
$$\sqrt{14}$$

erhalten.

Kreuzprodukt

Das Kreuzprodukt kann mithilfe des Befehls **Cross[**a, b**]** berechnet werden:

In[160]:=
```
Cross[{1, 2, 0}, {3, 4, 5}]
```

Out[160]=
```
{10, -5, -2}
```

Ray Tracing

Der reflektiere Strahl erhält man durch lösen folgender Gleichung:

In[161]:=
```
solr = Solve[(e.n + kr)^2 == (e.n)^2 (1 + 2 kr (e.n) + kr^2), kr]
```

Out[161]=
```
{{kr → 0}, {kr → -2 e.n}}
```

Einsetzten der Lösung ergibt

In[162]:=
```
er = Simplify[e + kr n /. solr[[2]] ]
```

Out[162]=
```
e - 2 n e.n
```

und konkret für die Beispielvektoren:

In[163]:=
```
% /. {n -> {0, 0, 1}, e -> {1, 0, -1}/√2} // Simplify
```

Out[163]=
$$\left\{\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right\}$$

Analog erhält man den gebrochenen Strahl:

In[164]:=
```
solb =
 Solve[1 - (((e.n + kb ) / Sqrt[(1 + 2 (e.n) kb  + kb ^ 2)])) ^ 2 == nb ^ 2 (1 - (e.n) ^ 2),
   kb] // Simplify
```

Out[164]=
$$\left\{\left\{kb \to -\frac{-nb^2\, e.n + nb^2\, (e.n)^3 + \sqrt{nb^2\, \left(-1 + (e.n)^2\right)^2\, \left(1 - nb^2 + nb^2\, (e.n)^2\right)}}{nb^2\, \left(-1 + (e.n)^2\right)}\right\},\right.$$
$$\left.\left\{kb \to \frac{nb^2\, e.n - nb^2\, (e.n)^3 + \sqrt{nb^2\, \left(-1 + (e.n)^2\right)^2\, \left(1 - nb^2 + nb^2\, (e.n)^2\right)}}{nb^2\, \left(-1 + (e.n)^2\right)}\right\}\right\}$$

In[165]:=
```
kb  = kb /. solb[[1]]
```

Out[165]=
$$-\frac{-nb^2\, e.n + nb^2\, (e.n)^3 + \sqrt{nb^2\, \left(-1 + (e.n)^2\right)^2\, \left(1 - nb^2 + nb^2\, (e.n)^2\right)}}{nb^2\, \left(-1 + (e.n)^2\right)}$$

In[166]:=
```
1 + 2 kb (e.n) + kb ^ 2  // Simplify
```

Out[166]=
$$\frac{1}{nb^2}$$

In[167]:=
```
Clear[solr, solb, er, kb]
```

Gram-Schmidt-Verfahren

Das Gram-Schmidt-Verfahren kann mit folgendem Befehl durchgeführt werden :

In[168]:=
```
Orthogonalize[{{1, 0, 1}, {0, 1, 1}, {1, 2, 4}}]
```

Out[168]=
$$\left\{\left\{\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right\}, \left\{-\frac{1}{\sqrt{6}}, \sqrt{\frac{2}{3}}, \frac{1}{\sqrt{6}}\right\}, \left\{-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right\}\right\}$$

Diskrete Kosinustransformation

Die diskrete Kosinustransformation können wir wie folgt berechnen:

In[169]:=
```
DCT[n_] := Table[Sqrt[(2 - If[k == 1, 1, 0])/n] Cos[((2 j - 1) (k - 1) π)/(2 n)], {j, n}, {k, n}]
```

In[170]:=
```
DCT[2] // MatrixForm // Simplify
```

Out[170]//MatrixForm=
$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

In[171]:=
```
DCT[3] // MatrixForm // Simplify
```

Out[171]//MatrixForm=

$$\begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & 0 & -\sqrt{\frac{2}{3}} \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \end{pmatrix}$$

Eigenwerte und Eigenvektoren

Koordinatentransformationen

Die Drehmatrix um die z-Achse ist:

In[172]:=
```
U[ϕ_] = {{Cos[ϕ], -Sin[ϕ], 0}, {Sin[ϕ], Cos[ϕ], 0}, {0, 0, 1}};
```

Konkret für eine Drehung um $\pi/4$

In[173]:=
```
U[π / 4] // MatrixForm
```

Out[173]//MatrixForm=

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

Ein gedrehter Vektor

In[174]:=
```
U[-π / 4].{1, 3, 4} // Simplify // MatrixForm
```

Out[174]//MatrixForm=

$$\begin{pmatrix} 2\sqrt{2} \\ \sqrt{2} \\ 4 \end{pmatrix}$$

und eine gedrehte Matrix:

In[175]:=
```
A = {{0, 1, 0}, {1, 0, 0}, {0, 0, 1}};
```

In[176]:=
```
Inverse[U[π / 4]].A.U[π / 4] // MatrixForm
```

Out[176]//MatrixForm=

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

In[177]:=
```
Clear[A, U]
```

Eigenwerte und Eigenvektoren

In Mathematica können die Befehle **Eigenvalues** und **Eigenvectors** zur Berechnung von Eigenwerten und Eigenvektoren verwendet werden:

In[178]:=
```
A = {{0, -1}, {1, 0}}; Eigenvalues[A]
```

Out[178]=
```
{i, -i}
```

In[179]:=
```
Eigenvectors[A]
```

Out[179]=
```
{{i, 1}, {-i, 1}}
```

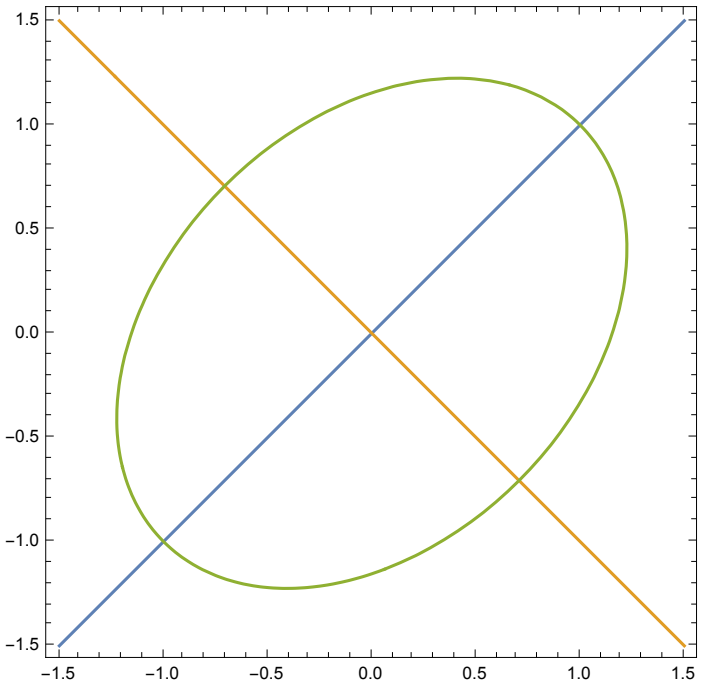
In[180]:=
```
Clear[A]
```

Implizite Kurven

Implizit gegebene Kurven, wie die Ellipse $3\,x_1^2 + 2\,x_1\,x_2 + 3\,x_2^2 = 4$, kann man mit folgendem Befehl zeichnen:

In[181]:=
```
ContourPlot[{x == y, x == -y, 3 x^2 - 2 x y + 3 y^2 == 4}, {x, -1.5, 1.5}, {y, -1.5, 1.5}]
```

Out[181]=



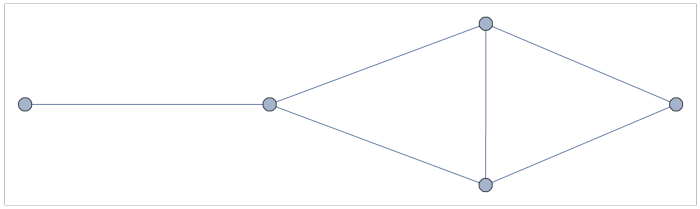
Grundlagen der Graphentheorie

Darstellung von Graphen

Graphen können mit dem **Befehl AdjacencyGraph[**A**]** eingegeben werden. Dabei ist A die Adjazenzmatrix und P eine Liste der gewünschten Koordinaten der Knoten im $\mathbb{R}^2$.

In[182]:=
```
A = {{0, 1, 0, 1, 1}, {1, 0, 1, 1, 0},
    {0, 1, 0, 1, 0}, {1, 1, 1, 0, 0}, {1, 0, 0, 0, 0}};
gr = AdjacencyGraph[
  A]
```

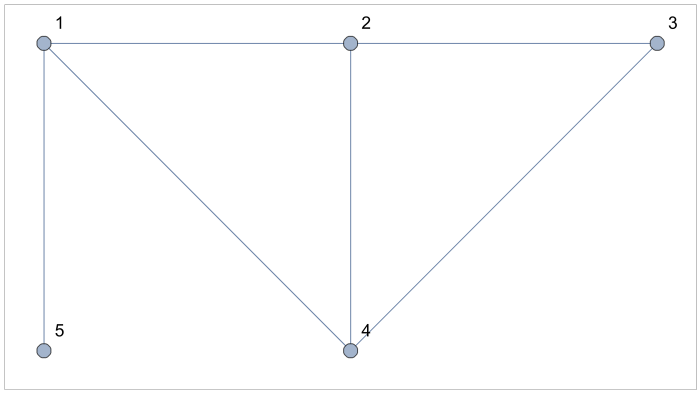
Out[183]=



Mathematica wählt dabei die Koordinaten der Knoten automatisch. Man kann sie aber auch über die Option **VertexCoordinates** angeben. Die Knoten können mit der Option **VertexLabels** bezeichnet werden und die Option **ImagePadding** stellt sicher, dass die Namen am Rand des Bildes nicht abgeschnitten werden.

In[184]:=
```
AdjacencyGraph[A, VertexCoordinates → {{0, 1}, {1, 1}, {2, 1}, {1, 0}, {0, 0}},
 VertexLabels → "Name", ImagePadding → 10]
```

Out[184]=



Zusammenhang

Mit dem Befehl

In[185]:=
```
ConnectedGraphQ[gr]
```

Out[185]=
```
True
```

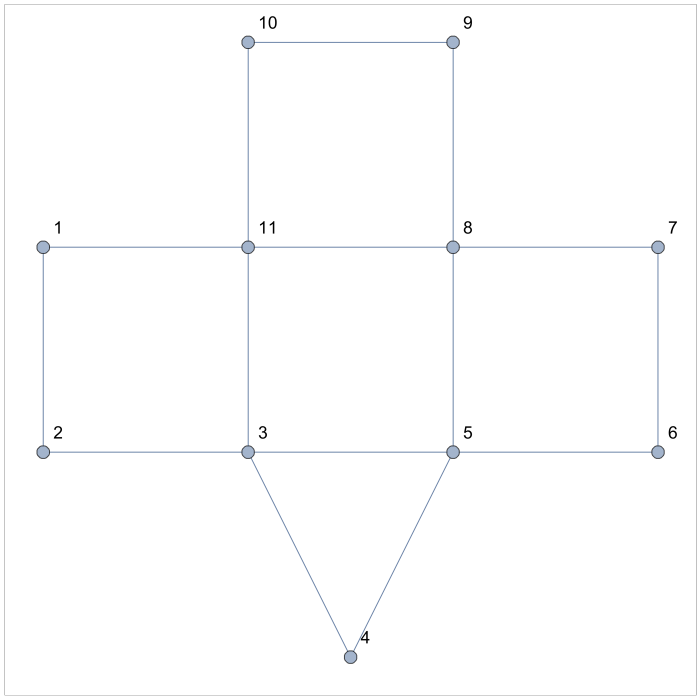
können wir auf Zusammenhang testen.

EulerZug

In Mathematica könnten wir das Beispiel aus dem Buch wie folgt lösen:

In[186]:=
```
gr1 = AdjacencyGraph[{ {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1},
    {0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0}, {0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0},
    {0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0},
    {0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1}, {0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1}, {1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0}},
  VertexCoordinates → { {0, 2}, {0, 1}, {1, 1}, {1.5, 0}, {2, 1}, {3, 1}, {3, 2},
    {2, 2}, {2, 3}, {1, 3}, {1, 2}}, VertexLabels → "Name", ImagePadding → 10]
```

Out[186]=



In[187]:=
```
FindEulerianCycle[gr1]
```

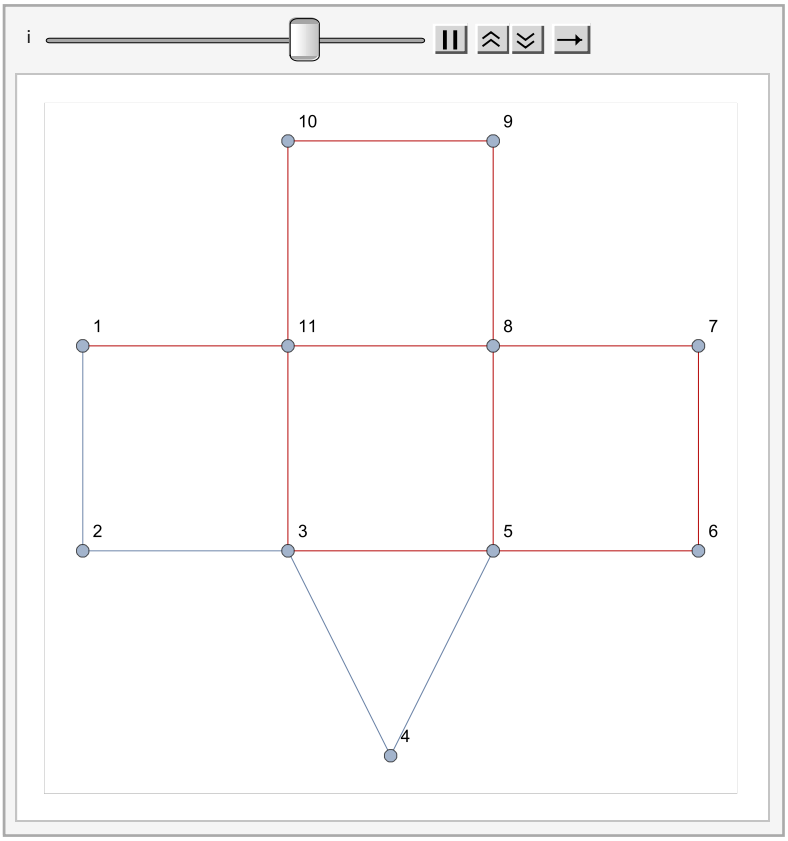
Out[187]=
```
{{1 ⟷ 11, 11 ⟷ 10, 10 ⟷ 9, 9 ⟷ 8, 8 ⟷ 7, 7 ⟷ 6,
  6 ⟷ 5, 5 ⟷ 8, 8 ⟷ 11, 11 ⟷ 3, 3 ⟷ 5, 5 ⟷ 4, 4 ⟷ 3, 3 ⟷ 2, 2 ⟷ 1}}
```

Mathematica gibt uns einen Euler-Zug mit Start im Knoten 1 (= a) aus. (Die Knoten sind entsprechend der Adjazenzmatrix, über die der Graph in Mathematica eingegeben wurde, durchnummeriert.)

Wir können den Euler-Zug auch hervorheben und animieren:

In[188]:=
```
Animate[HighlightGraph[gr1, Part[First[%], 1 ;; i]], {i, 1, Length[First[%]], 1}]
```

Out[188]=



Hamilton Kreis

Einen Hamilton-Kreis erhalten wir mit

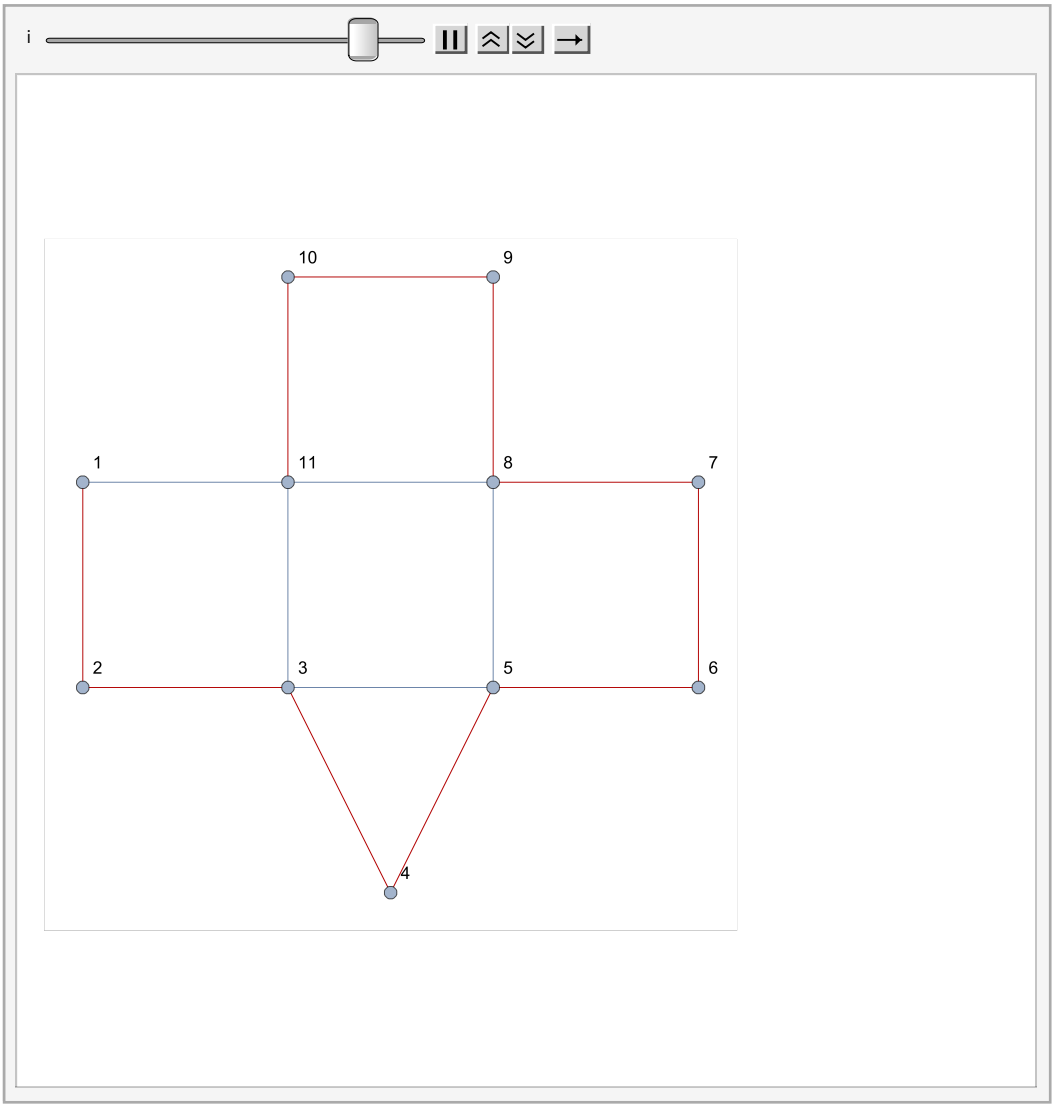
In[189]:=
```
FindHamiltonianCycle[gr1]
```

Out[189]=
```
{{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 5 ↔ 6, 6 ↔ 7, 7 ↔ 8, 8 ↔ 9, 9 ↔ 10, 10 ↔ 11, 11 ↔ 1}}
```

Wiederum können wir die Lösung veranschaulichen:

In[190]:=
```
Animate[HighlightGraph[gr1, Part[First[%], 1 ;; i]], {i, 1, Length[First[%]], 1}]
```

Out[190]=



Bäume und kürzeste Wege

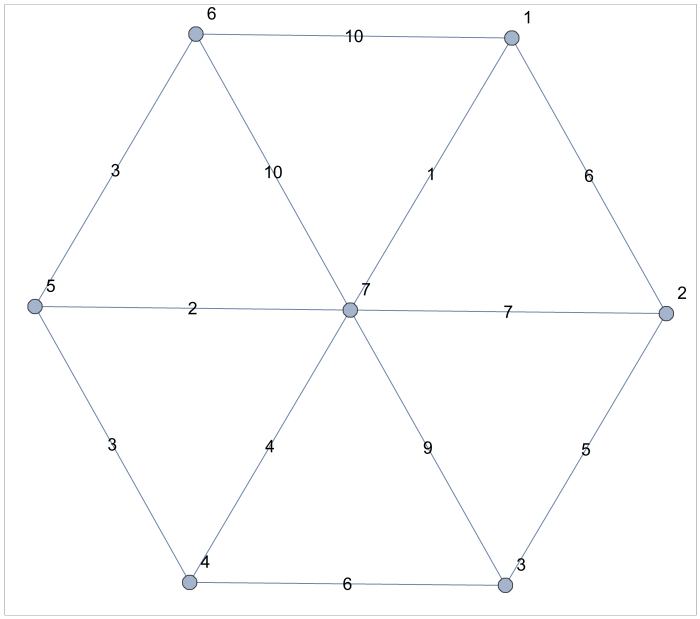
Gewichtete Graphen

Gewichtete Graphen können analog wie ungewichtete Graphen mit dem Befehl **WeightedAdjacencyGraph** eingegeben werden. Dabei müssen nicht-vorhandene Kanten durch ein Gewicht ∞ (Eingabe als **Infinite** oder über die Palette) gekennzeichnet werden.

In[191]:=
$$A = \begin{pmatrix} \infty & 6 & \infty & \infty & \infty & 10 & 1 \\ 6 & \infty & 5 & \infty & \infty & \infty & 7 \\ \infty & 5 & \infty & 6 & \infty & \infty & 9 \\ \infty & \infty & 6 & \infty & 3 & \infty & 4 \\ \infty & \infty & \infty & 3 & \infty & 3 & 2 \\ 10 & \infty & \infty & \infty & 3 & \infty & 10 \\ 1 & 7 & 9 & 4 & 2 & 10 & \infty \end{pmatrix};$$

```
gr2 = WeightedAdjacencyGraph[A, VertexLabels → "Name",
  ImagePadding → 5, EdgeLabels → "EdgeWeight"]
```

Out[191]=



Mit der Option **EdgeLabels** wird bei der Darstellung das Gewicht neben jede Kante geschrieben.

Travelling Salesman Problem

Eine billigste Rundreise in einem Graphen erhalten wir mit

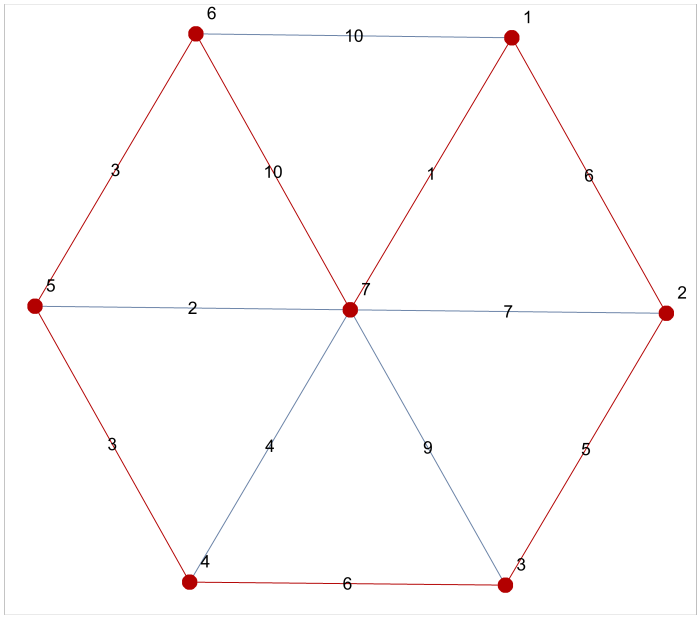
In[192]:=
```
ts = FindShortestTour[Range[Length[A]], DistanceFunction → (A[[#1, #2]] &)]
```

Out[192]=
```
{34, {1, 2, 3, 4, 5, 6, 7, 1}}
```

Die Ausgabe enthält die Länge zusammen mit der Rundreise in Form einer Liste der durchlaufenen Knoten. Um das Ergebnis zu veranschaulichen, können wir den gefundenen Weg mit folgendem Befehl hervorheben:

In[193]:=
```
HighlightGraph[gr2, PathGraph[ts[[2]]]]
```

Out[193]=

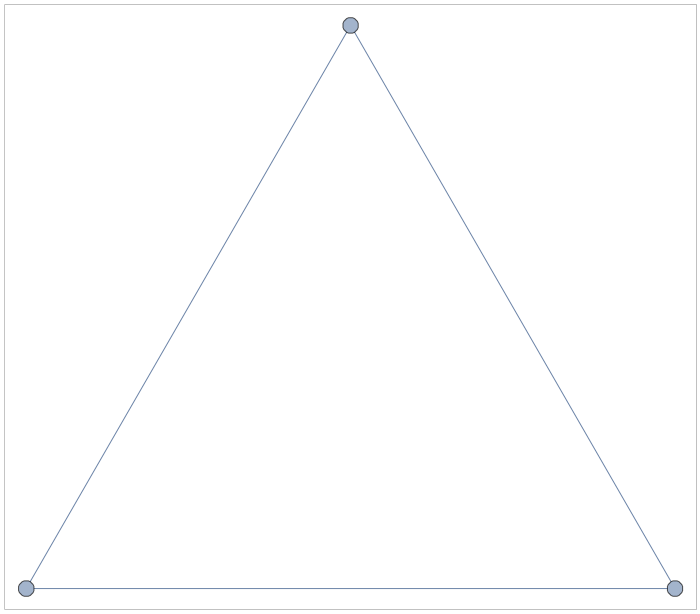


Der Befehl **PathGraph** macht aus der Knotenliste den zugehörigen Graphen, der diese Knoten verbindet.

Den vollständige Graphen K_n bekommen Sie übrigens mit **CompleteGraph[**n**]**.

In[194]:=
```
CompleteGraph[3]
```

Out[194]=



In[195]:=
```
Clear[A, ts]
```

Minimal aufspannende Bäume

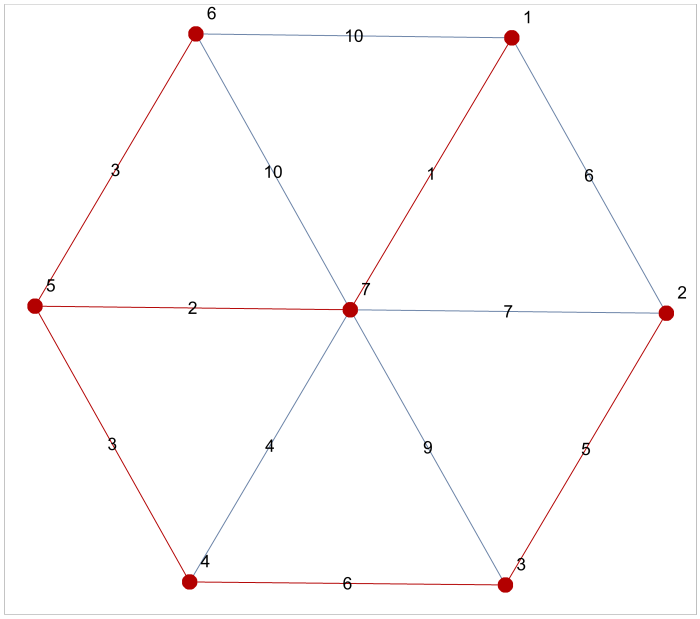
Mathematica implementiert keinen Algorithmus für minimal aufspannende Bäume, aber das können wir leicht selbst beheben:

In[196]:=
```
Kruskal[g_Graph] := Module[{el, i = 2, n = VertexCount[g], mst, cmst, mstn, cmstn},
  el = Sort[EdgeList[g],
    PropertyValue[{g, #1}, EdgeWeight] < PropertyValue[{g, #2}, EdgeWeight] &];
  mst = Graph[VertexList[g], {el[[1]]}]; cmst = n - 1;
  While[i ≤ n && cmst ≠ 1,
   mstn = GraphUnion[mst, Graph[{el[[i]]}]];
   cmstn = Length[ConnectedComponents[mstn]];
   If[cmstn < cmst, mst = mstn; cmst = cmstn];
   i++];
  mst
 ]
```

Zum Beispiel

In[197]:=
```
mst = Kruskal[gr2];
HighlightGraph[gr2, mst]
```

Out[198]=



In[199]:=
```
Clear[mst]
```

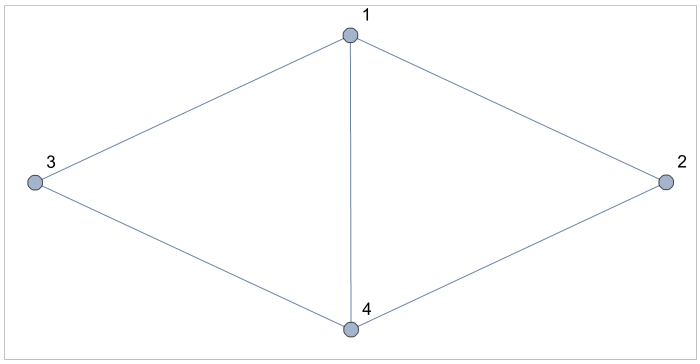
Kürzeste Wege

Mit Mathematica kann der kürzeste Weg zwischen zwei Knoten i, j mit dem Befehl **ShortestPath[**-graph, i, j**]** berechnet werden.

In[200]:=
$$A = \begin{pmatrix} \infty & 4 & 6 & 7 \\ 4 & \infty & \infty & 1 \\ 6 & \infty & \infty & 2 \\ 7 & 1 & 2 & \infty \end{pmatrix};$$

```
gr3 = WeightedAdjacencyGraph[A, VertexLabels → "Name", ImagePadding → 5]
```

Out[200]=



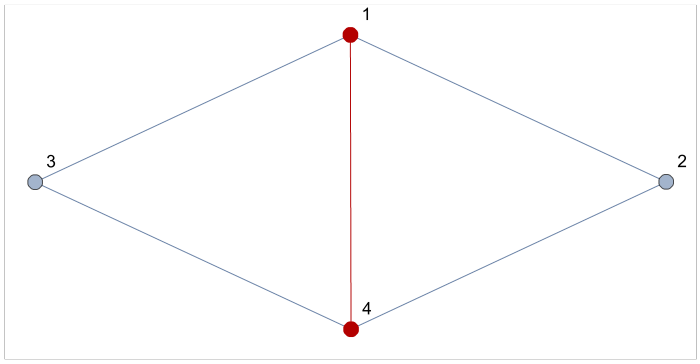
In[201]:=
```
FindShortestPath[gr, 1, 4]
```

Out[201]=
```
{1, 4}
```

Das Ergebnis können wir natürlich wieder mit

In[202]:=
```
HighlightGraph[gr3, PathGraph[%]]
```

Out[202]=



veranschaulichen.

In[203]:=
```
Clear[A, gr3]
```

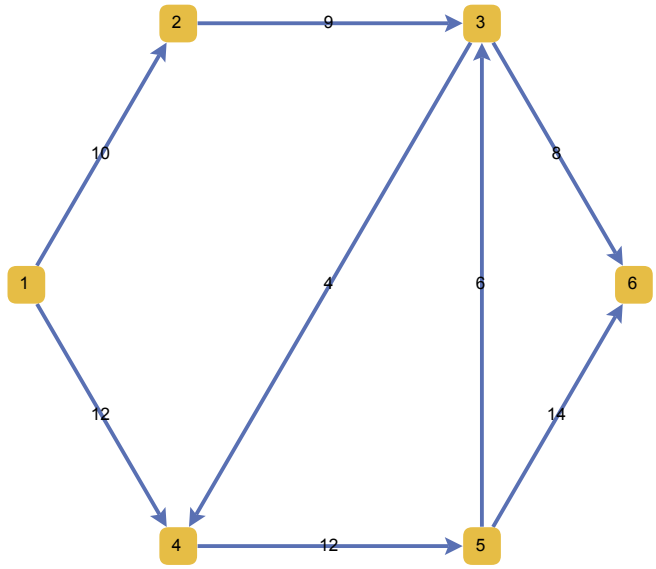
Flüsse in Netzwerken und Matchings

Netzwerke

Definieren wir zunächst einen Graphen

In[204]:=
```
A = {{∞, 10, ∞, 12, ∞, ∞}, {∞, ∞, 9, ∞, ∞, ∞}, {∞, ∞, ∞, 4, ∞, 8},
   {∞, ∞, ∞, ∞, 12, ∞}, {∞, ∞, 6, ∞, ∞, 14}, {∞, ∞, ∞, ∞, ∞, ∞}};
v = {{-1, 0}, {-0.5, 0.86}, {0.5, 0.86}, {-0.5, -0.86}, {0.5, -0.86}, {1, 0}};
gr = WeightedAdjacencyGraph[A, VertexCoordinates → v,
   ImagePadding → 5, GraphStyle → "SmallNetwork", EdgeLabels → "EdgeWeight"]
```

Out[205]=



Nun kann der maximale Fluss mittels

In[206]:=
```
mf = FindMaximumFlow[gr, 1, 6, "OptimumFlowData", EdgeCapacity → EdgeWeight];
```

berechnet werden. Mit

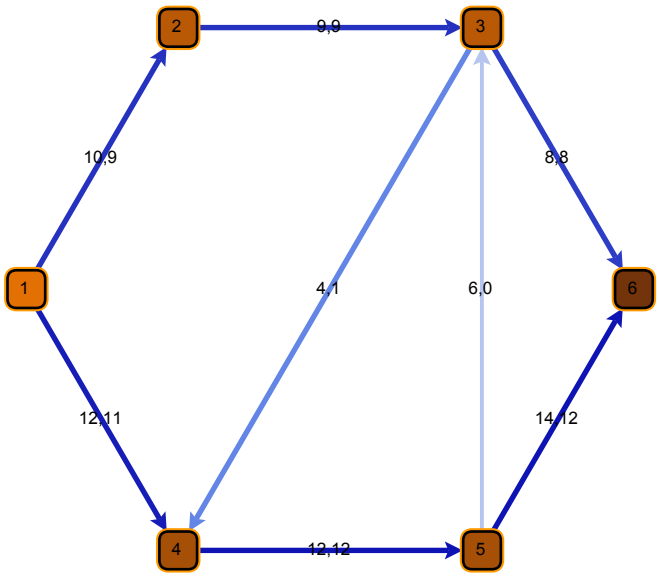
In[207]:=
```
mf["FlowValue"]
```

Out[207]=
```
20
```

erhalten wir den maximalen Fluss und mit mf[" FlowGraph"] den zugehörigen Graphen. In diesem können wir die Kantenbezeichnungen auf die Angabe von “Kapazität, Fluss“ erweitern:

In[208]:=
```
SetProperty[mf["FlowGraph"], {EdgeLabels →
    (# → Row[{PropertyValue[{gr, #}, EdgeWeight], ",", mf[#]}] & /@ EdgeList[gr])}]
```

Out[208]=



Matchings

Definieren wir zunächst einen Graphen

In[209]:=
```
gr = Graph[{1 → 5, 1 → 6, 2 → 6, 2 → 7, 2 → 8, 3 → 7, 4 → 6}, VertexCoordinates ->
    {{0, 3}, {2, 3}, {2, 1}, {0, 2}, {2, 2}, {2, 0}, {0, 1}, {0, 0}},
  VertexLabels → "Name", ImagePadding → 10];
```

Hier haben wir zur Abwechslung den Graphen als eine Liste von gerichteten Kanten angegeben (die Knotenliste wird automatisch aus den Kanten generiert). Nun kann mit dem Befehl **FindIndependentEdgeSet** ein maximales Matching gefunden werden:

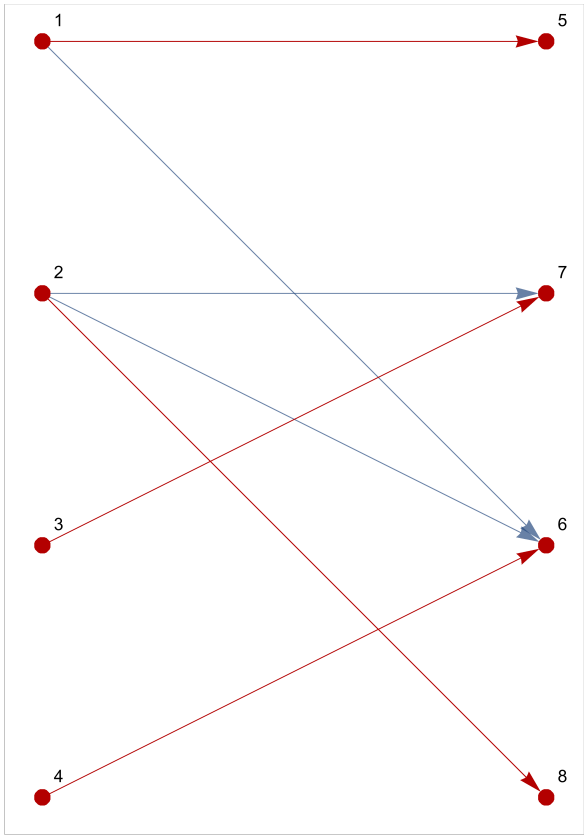
In[210]:=
```
FindIndependentEdgeSet[gr]
```

Out[210]=
```
{1 ↔ 5, 4 ↔ 6, 2 ↔ 8, 3 ↔ 7}
```

Veranschaulichen können wir es mit

In[211]:=
```
HighlightGraph[gr, Graph[% /. {a_, b_} → (a → b)]]
```

Out[211]=



In[212]:=
```
Clear[gr];
```