Notebook zu: Mathematik für Informatiker
Band 2

Dieses Notebook ist als Ergänzung zu unserem Buch Mathematik für Informatiker, Band 2, 3. Aufl., Springer 2014, gedacht. Dort finden Sie auch weitere Erklärungen.

Elementare Funktionen

Nullstellen

Der **Solve** -Befehl in Mathematica versucht, eine Gleichung nach der angegebenen Variablen aufzulösen. Gleichungen werden mit einem doppelten Gleichheitszeichen eingegeben :

In[1]:=
```
Solve[x^3 - 7 x^2 + 7 x - 1 == 0, x]
```

Out[1]=
$$\left\{\{x \to 1\}, \left\{x \to 3 - 2\sqrt{2}\right\}, \left\{x \to 3 + 2\sqrt{2}\right\}\right\}$$

Mathematica hat also drei Nullstellen gefunden.

Faktorisierung

Polynome können mit dem Befehl

In[2]:=
```
Factor[x^3 - 1]
```

Out[2]=
$$(-1 + x)\left(1 + x + x^2\right)$$

faktorisiert und mit dem Befehl

In[3]:=
```
Expand[%]
```

Out[3]=
$$-1 + x^3$$

ausmultipliziert werden. Eine Faktorisierung erfolgt nur, wenn die Nullstellen rationale Zahlen sind.

Polynomdivison

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

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

Out[4]=
$$-3 + x + 3\,x^2$$

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

In[5]:=
```
PolynomialRemainder[3 x^4 + x^3 - 2 x, x^2 + 1, x]
```

Out[5]=
$$3 - 3\,x$$

berechnen.

Interpolationspolynome

Interpolationspolynome können mittels **InterpolatingPolynomial[**{{x_0, y_0}, . . .}, x**]** berechnet werden, wobei (x_0, y_0), ... die gewünschten Stützpunkte sind. Um uns die Eingabe der Stützpunkte zu ersparen, können wir leicht mit dem **Table**-Befehl eine Liste gleichmäßig verteilter Stützpunkte erzeugen. Zum Beispiel :

In[6]:=
```
f[x_] = 1/(1 + 25 x^2);
```

In[7]:=
```
Stuetzpunkte[n_] := Table[{j/n, f[j/n]}, {j, -n, n}]
```

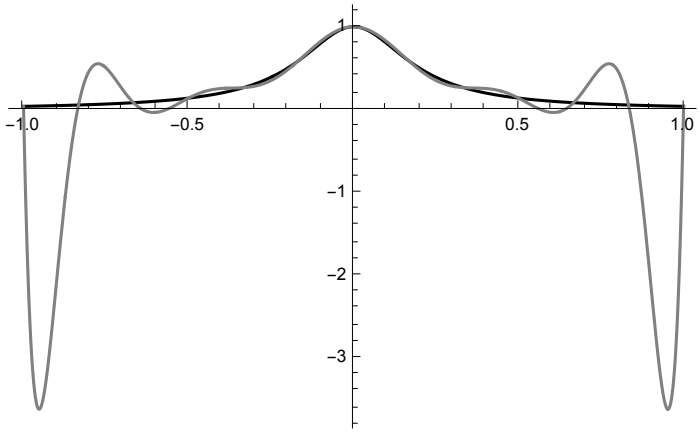
In[8]:=
```
P[x_] = InterpolatingPolynomial[Stuetzpunkte[6], x] // Expand
```

Out[8]=
$$1 - \frac{551\,599\,221\,900\,x^2}{28\,167\,484\,501} + \frac{367\,051\,586\,875\,x^4}{1\,847\,048\,164} - \frac{107\,641\,853\,578\,125\,x^6}{112\,669\,938\,004} + \frac{62\,017\,871\,484\,375\,x^8}{28\,167\,484\,501} - \frac{65\,809\,335\,937\,500\,x^{10}}{28\,167\,484\,501} + \frac{25\,628\,906\,250\,000\,x^{12}}{28\,167\,484\,501}$$

Stuetzpunkte[n**]** erzeugt eine Liste von 2 n + 1 Stützpunkten, die gleichmäßig im Intervall[–1, 1] verteilt sind. Damit wurden hier 13 Stützpunkte erzeugt und dann
das Interpolationspolynom P(x) von f(x) = $\frac{1}{1+25\,x^2}$ berechnet.

In[9]:=
```
Plot[{f[x], P[x]}, {x, -1, 1},
 PlotRange → All, PlotStyle → {Black, GrayLevel[0.5`]}]
```

Out[9]=



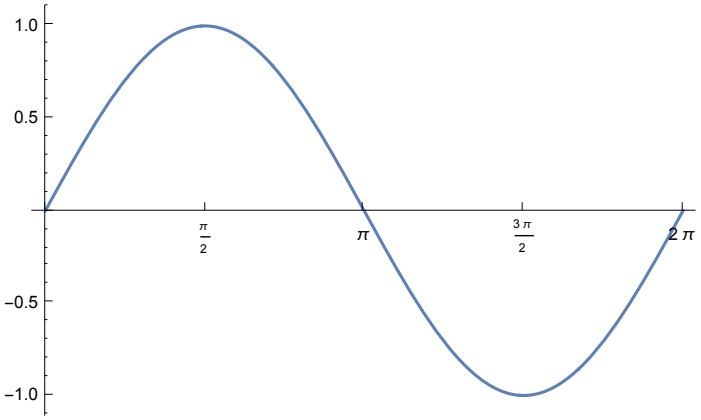
In[10]:=
```
Clear[f, P];
```

Graphische Darstellung von Funktionen

Erinnern Sie sich an den **Plot**-Befehl, mit dem Funktionen gezeichnet werden können. Es ist dabei sogar möglich mehrere Funktionen gleichzeitig zu zeichnen, indem man statt einer Funktion eine Liste von Funktionen angibt :

In[11]:=
```
Plot[Sin[x], {x, 0, 2 π}, Ticks → {{0, π/2, π, 3 π/2, 2 π}, Automatic}]
```

Out[11]=

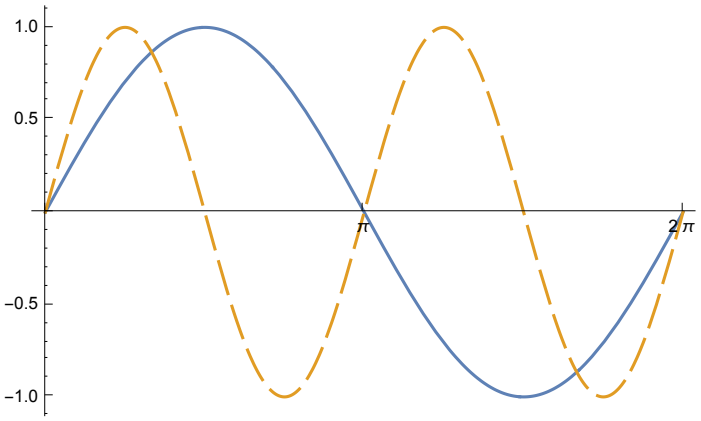


Mit der Option **Ticks** wurde hier festgelegt, dass die x - Achse an den Positionen 0, $\frac{\pi}{2}$, π, etc. zu beschriften ist (die Beschriftung der y - Achse erfolgt hier automatisch).

Es ist dabei sogar möglich mehrere Funktionen gleichzeitig zu zeichnen, indem man statt einer Funktion eine Liste von Funktionen angibt :

In[12]:=
```
Plot[{Sin[x], Sin[2 x]}, {x, 0, 2 π}, Ticks → {{0, π, 2 π}, Automatic},
 PlotStyle → {{}, Dashing[{0.05`, 0.02`}]}]
```

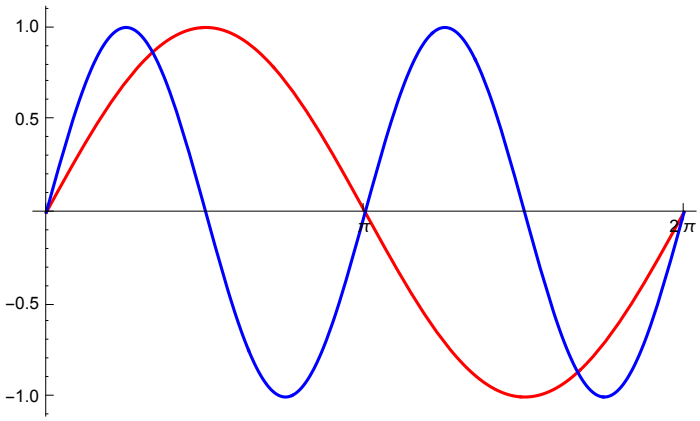
Out[12]=



Mit der Option PlotStyle wurde für die zweite Funktion durch **Dashing[**{0.05, 0.02}**]** eine strichlierte Linie gewählt. Als **PlotStyle** kann durch Angabe von **RGBColor[**r, g, b**]** auch ein Farbwert gewählt werden (z.B. **RGBColor[**1, 0, 0**]** für rot)

In[13]:=
```
Plot[{Sin[x], Sin[2 x]}, {x, 0, 2 π}, Ticks → {{0, π, 2 π}, Automatic},
 PlotStyle → {{RGBColor[1, 0, 0]}, {RGBColor[0, 0, 1]}}]
```

Out[13]=



Bogenmaß und Gradmaß

Winkelwerte werden von Mathematica defaultmäßig im Bogenmaß interpretiert:

In[14]:=
```
Sin[π/2]
```

Out[14]= 1

Haben Sie den Winkel im Gradmaß gegeben, so muss er mit dem Faktor $\frac{\pi}{180}$ ins Bogenmaß umgerechnet werden. Dafür gibt es die eingebaute Konstante **Degree**, die auch als ∘ über die Palette eingegeben werden kann :

In[15]:=
```
Sin[90 Degree]
```

Out[15]= 1

Funktionen mit komplexen Argumenten, Polardarstellung

Natürlich können wir in Mathematica die trigonometrischen (bzw. alle anderen) Funktionen auch mit komplexem Argument aufrufen (Eingabe der imaginären Einheit mit dem Großbuchstaben “I” oder über die Palette).

In[16]:=
```
Sin[I]
```

Out[16]= i Sinh[1]

Oder:

In[17]:=
```
Exp[2 + i π/3] // ComplexExpand
```

Out[17]= $\frac{e^2}{2} + \frac{1}{2}\, i\, \sqrt{3}\, e^2$

Hier mussten wir noch mit **ComplexExpand** nachhelfen, damit das Ergebnis auch in Real - und Imaginärteil aufgespalten wird.

Der Betrag einer komplexen Zahl kann mit **Abs[**z**]** und der Winkel mit **Arg[**z**]** berechnet werden. Der Winkel wird im Bogenmaß ausgegeben. Die Polarkoordinaten von $1 + i\sqrt{3}$ sind zum Beispiel

In[18]:=
```
{Abs[1 + I Sqrt[3]], Arg[1 + I Sqrt[3]]}
```
Out[18]=
$$\left\{2, \frac{\pi}{3}\right\}$$

Die n-te Wurzel einer komplexen Zahl z wird in der Form z^n oder über die Palette eingegeben :

In[19]:=
```
Sqrt[1 + i Sqrt[3]] // ComplexExpand
```
Out[19]=
$$\sqrt{\frac{3}{2}} + \frac{i}{\sqrt{2}}$$

Oder:

In[20]:=
```
Sqrt[3, 8 Exp[i π / 2]] // ComplexExpand
```
Out[20]=
$$i + \sqrt{3}$$

Den komplexen Logarithmus erhält man mit

In[21]:=
```
Log[1 + i Sqrt[3]] // ComplexExpand
```
Out[21]=
$$\frac{i\,\pi}{3} + \text{Log}[2]$$

und analog komplexe Potenzen

In[22]:=
```
{3^-i, (1 + i)^i, (1 - i)^(2+3 i)} // ComplexExpand // Simplify
```
Out[22]=
$$\left\{\text{Cos}[\text{Log}[3]] - i\,\text{Sin}[\text{Log}[3]],\ e^{-\pi/4}\left(\text{Cos}\left[\frac{\text{Log}[2]}{2}\right] + i\,\text{Sin}\left[\frac{\text{Log}[2]}{2}\right]\right),\right.$$
$$\left.2\,e^{3\pi/4}\left(-i\,\text{Cos}\left[\frac{3\,\text{Log}[2]}{2}\right] + \text{Sin}\left[\frac{3\,\text{Log}[2]}{2}\right]\right)\right\}$$

Differentialrechnung I

Grenzwert

Grenzwerte erhalten wir wie bei Folgen mit dem Befehl **Limit**:

In[23]:=
```
Limit[(x^2 - 1)/(x - 1), x → 1]
```
Out[23]=
$$2$$

Ableitungen

Ableitungen werden mit dem Befehl **D** berechnet :

In[24]:=
```
D[2 x^3 - 4 x + 1, x]
```
Out[24]=
$$-4 + 6\,x^2$$

Die n-te Ableitung nach x bekommen wir mit **D[**f(x), {x, n}**]** :

In[25]:=
```
D[2 x^3 - 4 x + 1, {x, 2}]
```
Out[25]=
```
12 x
```

Eine Ableitung kann in Mathematica auch einfach als f'[x] anstelle von **D[**f[x], x**]** eingegeben werden

In[26]:=
```
f[x_] := 2 x^3 - 4 x + 1; f'[x]
```
Out[26]=
```
-4 + 6 x^2
```

In[27]:=
```
Clear[f]
```

Splines

Mathematica stellt leider keinen Befehl zur Berechnung von Splinefunktionen zur Verfügung, man muss also selbst zur Tat schreiten und den Algorithmus aus dem Buch implementieren. (Im Paket NumericalMath'SplineFit' gibt es allerdings eine Funktion **SplineFit**, die aus gegebenen Stützpunkten eine Splinekurve berechnet.)

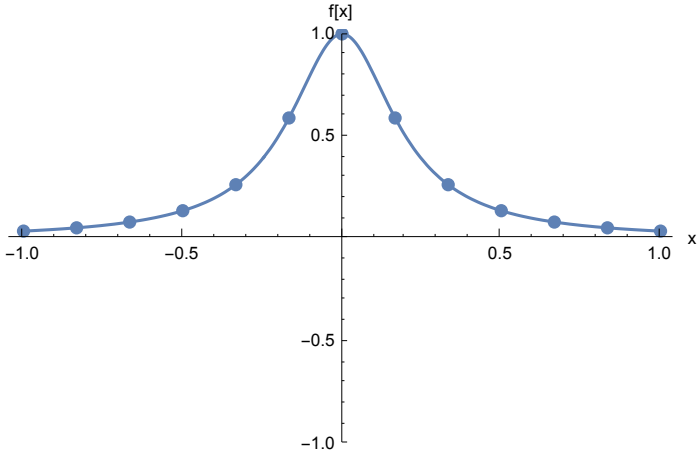
In[28]:=
```
NaturalSpline[xy_, t_] := Module[{x, y, z, n, u, v, d, k, Fh, F, spl},
  {x, y} = Transpose[xy];
  n = Length[x]; u = Table[0, {j, n}]; v = Table[0, {j, n}];
  d = Table[x[[j + 1]] - x[[j]], {j, n - 1}];
  k = Table[(y[[j + 1]] - y[[j]])/d[[j]], {j, n - 1}];
  u[[1]] = 0; u[[2]] = 3 k[[1]]; v[[1]] = 1; v[[2]] = -2;
  Fh[j_, z_] :=
    -1/d[[j - 2]] (2 (d[[j - 1]] + d[[j - 2]]) z[[j - 1]] + d[[j - 1]] z[[j - 2]]);
  F[j_, z_] := Fh[j, z] + 3/d[[j - 2]] (k[[j - 2]] d[[j - 1]] + k[[j - 1]] d[[j - 2]]);
  Do[u[[j]] = F[j, u]; v[[j]] = Fh[j, v];, {j, 3, n}];
  z = u + (3 k[[n - 1]] - u[[n - 1]] - 2 u[[n]])/(2 v[[n]] + v[[n - 1]]) v;
  spl = Flatten[Table[{t < x[[j + 1]], Expand[
        y[[j]] + k[[j]] (t - x[[j]]) + (z[[j]] - k[[j]])/d[[j]]^2 (t - x[[j]]) (t - x[[j + 1]])^2 +
          (z[[j + 1]] - k[[j]])/d[[j]]^2 (t - x[[j]])^2 (t - x[[j + 1]])]}, {j, n - 1}]];
  spl[[-2]] = True; spl[[0]] = Which;
  spl
];
```

Ein Vergleich mit dem Interpolationspolynom aus dem letzten Abschnitt:

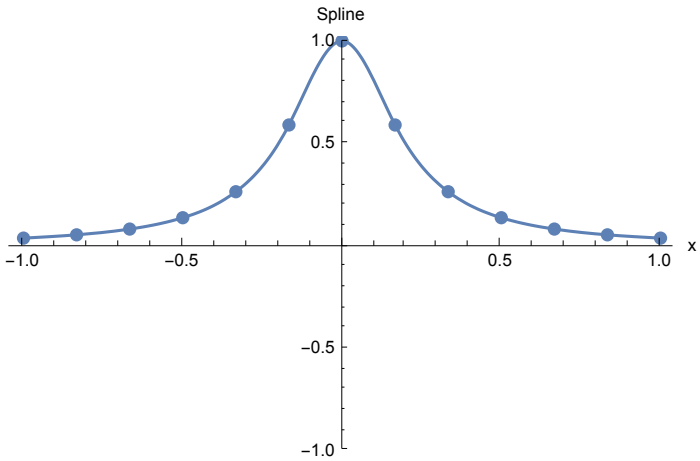
In[29]:=
```
f[x_] = 1/(1 + 25 x^2);
```

In[30]:=
```
Stuetzpunkte[n_] := Table[{j/n, f[j/n]}, {j, -n, n}];
XY = Stuetzpunkte[6];
S[x_] = NaturalSpline[XY, x];
P[x_] = InterpolatingPolynomial[XY, x];
dots = ListPlot[XY, PlotStyle → {PointSize[0.02`]}];
gr1 = Plot[f[x], {x, -1, 1}, PlotRange → {-1, 1}];
gr2 = Plot[S[x], {x, -1, 1}, PlotRange → {-1, 1}];
gr3 = Plot[P[x], {x, -1, 1}, PlotRange → {-1, 1}];
Show[gr1, dots, AxesLabel → {"x", "f[x]"}]
Show[gr2, dots, AxesLabel → {"x", "Spline"}]
Show[gr3, dots, AxesLabel → {"x", "Interpolationspolynom"}]
```

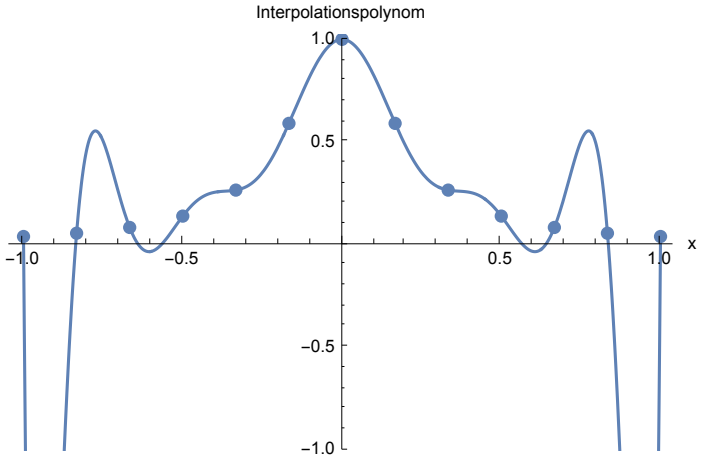
Out[38]=



Out[39]=



Out[40]=



In[41]:=
```
Clear[f, S, P, XY]
```

Differentialrechnung II

Taylorpolynome

Taylorpolynome werden mit dem Befehl **Series** berechnet. Anzugeben ist dabei das Argument der Funktion (hier x), der Entwicklungspunkt (hier 0), sowie der gewünschte Grad des Polynoms (hier 3) :

In[42]:=
```
Series[Sin[x], {x, 0, 3}]
```

Out[42]= $x - \frac{x^3}{6} + 0[x]^4$

Den “O-Term” können wir mit

In[43]:=
```
Normal[%]
```

Out[43]= $x - \frac{x^3}{6}$

entfernen.

Gauß'sche Glockenkurve

Um die Extrema und Wendepunkte der Gauß'schen Glockenkurve zu berechnen, definieren wir zunächst

In[44]:=
```
f[x_] := 1/(Sqrt[2 π] σ) e^(-((x-μ)^2)/(2 σ^2));
```

und bestimmen dann die Nullstellen der Ableitung:

In[45]:=
```
f'[x]
```

Out[45]= $-\frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}(x-\mu)}{\sqrt{2\pi}\,\sigma^3}$

Es gibt also eine mögliche Extremstelle, x = μ. Da die zweite Ableitung an dieser Stelle

In[46]:=
```
Solve[f'[x] == 0, x]
```

Solve: Inverse functions are being used by Solve, so some solutions may not be found; use Reduce for complete solution information.

Out[46]= $\{\{x \to \mu\}\}$

In[47]:=
```
f''[μ]
```

Out[47]= $-\frac{1}{\sqrt{2\,\pi}\ \sigma^3}$

negativ ist, liegt hier ein Maximum vor. Für die Wendepunkte ermitteln wir die Nullstellen der zweiten Ableitung,

In[48]:=
```
Solve[f''[x] == 0, x]
```

Out[48]= $\{\{x \to \mu - \sigma\}, \{x \to \mu + \sigma\}\}$

und werten die dritte Ableitung an diesen Stellen aus:

In[49]:=
```
{f'''[μ - σ], f'''[μ + σ]}
```

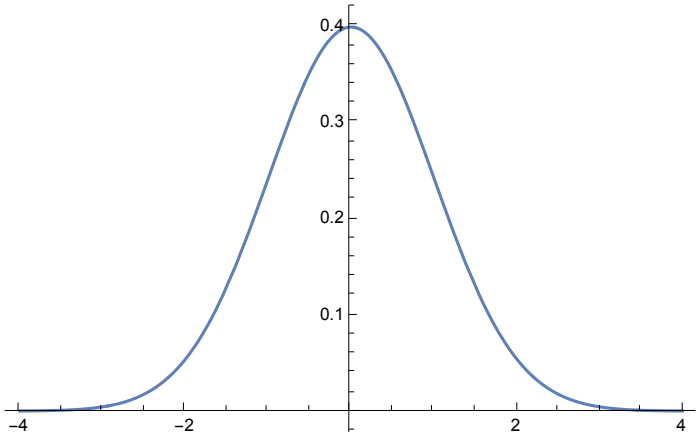
Out[49]= $\left\{-\frac{\sqrt{\frac{2}{e\,\pi}}}{\sigma^4}, \frac{\sqrt{\frac{2}{e\,\pi}}}{\sigma^4}\right\}$

Da das Ergebnis jeweils ungleich null ist, liegen hier tatsächlich Wendepunkte vor.

Und so sieht die Glockenkurve aus:

In[50]:=
```
μ = 0; σ = 1; Plot[f[x], {x, -4, 4}]
```

Out[50]=



Sekantenverfahren

Wir definieren die Funktion, deren Nullstellen wir numerisch mit dem Sekantenverfahren bestimmen möchten:

In[51]:=
```
f[x_] := x^3 + 2 x^2 + 10 x - 20;
```

In[52]:=
```
{f[1], f[2]}
```

Out[52]= $\{-7, 16\}$

Dann geben wir die Formel für die rekursive Berechnung der Näherungswerte ein:

In[53]:=
```
x[n_] := x[n - 1] - f[x[n - 1]] (x[n - 1] - x[n - 2]) / (f[x[n - 1]] - f[x[n - 2]])
```

Nun geben wir die Startwerte vor und lassen uns eine Liste der nächsten Näherungswerte ausgeben:

In[54]:=
```
x[0] = 1.; x[1] = 2.; {x[2], x[3], x[4], x[5]}
```
Out[54]=
```
{1.30435, 1.35791, 1.36901, 1.36881}
```

In[55]:=
```
Clear[f, x]
```

Newtonverfahren

Wir geben die Funktion ein, deren Nullstellen wir bestimmen möchten, und geben die rekursive Formel für das Newton-Verfahren ein :

In[56]:=
```
f[x_] := x^2 - 2;
x[n_] := x[n - 1] - f[x[n - 1]] / f'[x[n - 1]]
```

Mit dem Startwert $x_0 = 1$ lassen wir uns eine Liste der nächsten fünf Näherungswerte ausgeben :

In[58]:=
```
x[0] = 1.; {x[1], x[2], x[3], x[4], x[5]}
```
Out[58]=
```
{1.5, 1.41667, 1.41422, 1.41421, 1.41421}
```

Natürlich können Sie auch einfach den Befehl **FindRoot[**f[x], {x, x0}**]** verwenden, um mit dem Startwert $0_{\square}$ einen Näherungswert für eine Nullstelle von f(x) zu finden.

In[59]:=
```
FindRoot[f[x], {x, 1}]
```
Out[59]=
```
{x → 1.41421}
```

In[60]:=
```
Clear[f, x]
```

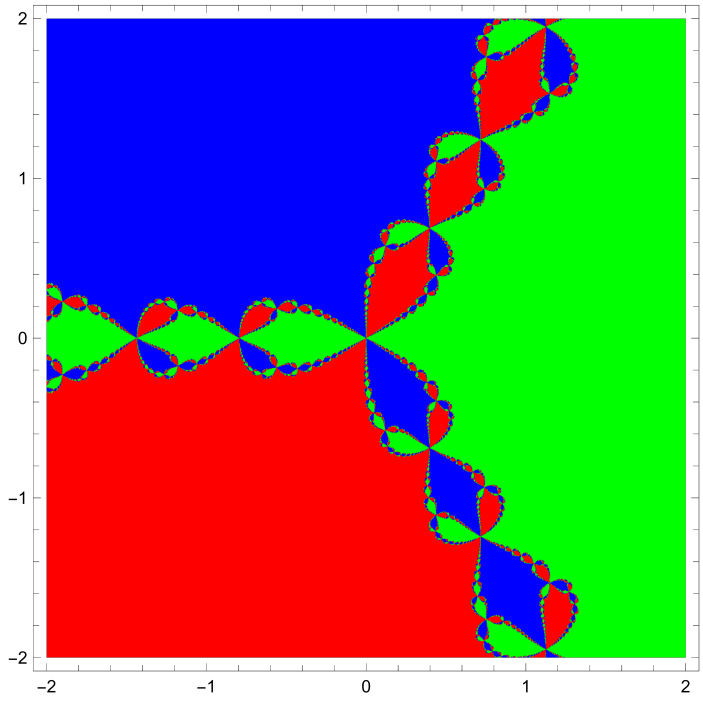
Konvergenzbereich des Newton-Verfahrens

Anbei eine Funktion die mit dem Newton-Verfahren die Nullstellen von $z^3 - 1$ sucht und jedem komplexen Startwert eine Farbe zuordnet die der Nullstelle entspricht zu der der Startwert konvergiert.

In[61]:=
```
WhichZero = Compile[{{z0, _Complex}},
   Block[{z = z0, k},
    For[k = 0, k < 200 && Abs[z^3 - 1] > 0.5, k++, z = 1/(3 z^2) + 2 z/3];
    (Round[Im[z]] + 1) / 2]
   ];
MyColor[x_] := Which[x == 0, RGBColor[1, 0, 0], x == 1/2,
   RGBColor[0, 1, 0], x == 1, RGBColor[0, 0, 1], True, RGBColor[1, 1, 1]];
```

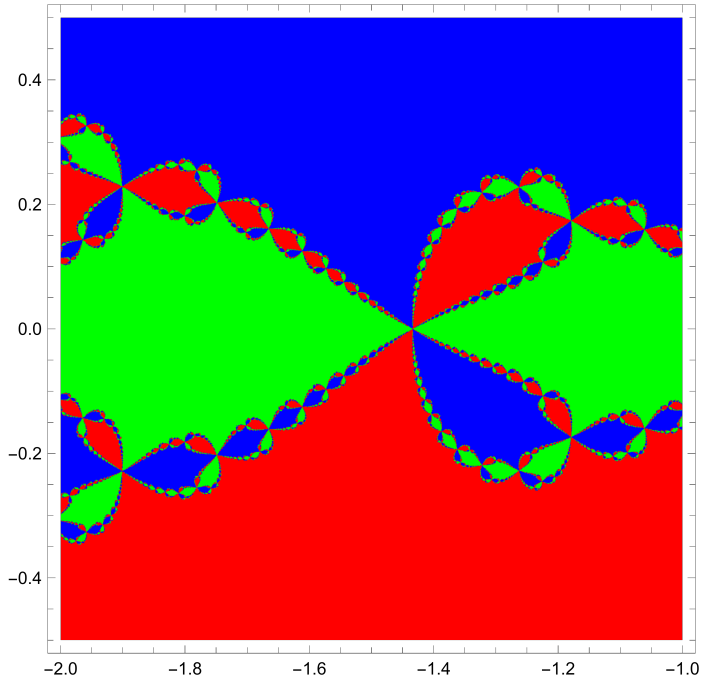
In[63]:=
```
DensityPlot[WhichZero[x + i y], {x, -2, 2}, {y, -2, 2},
 PlotPoints → 400, Mesh → False, ColorFunction → MyColor]
```

Out[63]=



In[64]:=
```
DensityPlot[WhichZero[x + i y], {x, -2, -1}, {y, -0.5`, 0.5`},
 PlotPoints → 400, Mesh → False, ColorFunction → MyColor]
```

Out[64]=



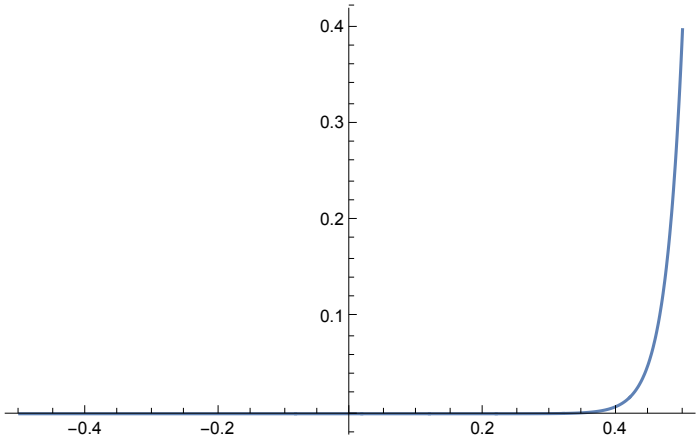
Dioden-Logik

Wir definieren zuerst $I_D(U)$

In[65]:=
```
c = 39.6;
ID[U_] = 10^-9 (Exp[c U] - 1);
```

In[67]:=
```
Plot[ID[U], {U, -0.5, 0.5}, PlotRange → All]
```

Out[67]=



und unsere Werte für die Parameter (die Widerstände)

In[68]:=
```
R0 = 10 000; R1 = 4700; R = R0 R1 / (R0 + R1);
```

Nun lösen wir mit **FindRoot** die Gleichung für U_2 und geben eine Liste mit den Werten für U_1, U_2, U_0 aus:

In[69]:=
```
U0[E1_, E2_] := {E1 - E2 + U2, U2, E2 - U2} /.
  FindRoot[U2 == E2 - R (ID[E1 - E2 + U2] + ID[U2]), {U2, -4}]
```

(Der Startwert – 4 wurde gewählt, damit das Newton - Verfahren in allen vier untenstehenden Fällen konvergiert; -) Damit erhalten wir

In[70]:=
```
{U0[0, 0], U0[5, 0], U0[0, 5], U0[5, 5]} // Chop // TableForm
```

Out[70]//TableForm=

0	0	0
0.35829	−4.64171	4.64171
−4.64171	0.35829	4.64171
0.340881	0.340881	4.65912

Der Befehl **Chop** ersetzt Werte, die innerhalb der Rechengenauigkeit null sind, durch 0. Das macht die Ausgabe oft leichter lesbar.

In[71]:=
```
Clear[T, ID, R, R0, R1, c, U0]
```

Physikalische Konstanten

Mathematica kennt viele wichtige Konstanten und kann auch mit Einheiten rechnen.

In[72]:=
```
Quantity["BoltzmannConstant"]
```

Out[72]= $1\ k$

In[73]:=
```
UnitConvert[%]
```

Out[73]= $1.38065 \times 10^{-23}\ \mathrm{kg\,m^2/(s^2 K)}$

In[74]:=
```
UnitSimplify[%]
```

Out[74]= $1.38065 \times 10^{-23}\ \mathrm{J/K}$

Zum Beispiel können wir den Wert $\frac{q}{kT}$ der in der Kennlinie einer Diode auftritt ausrechnen:

In[75]:=
```
UnitConvert[Quantity["ElectronCharge"] /
  (Quantity["BoltzmannConstant"] Quantity[293., "Kelvin"])]
```

Out[75]= $39.6059\ \mathrm{s^3 A/(kg\ m^2)}$

In[76]:=
```
UnitSimplify[%]
```

Out[76]= $39.6059/\mathrm{V}$

Übungen

In[77]:=
```
f[x_] := x / Sqrt[1 + x^2];
x[n_] := x[n - 1] - f[x[n - 1]] / f'[x[n - 1]]
```

In[79]:=
```
x[0] = 0.5; {x[1], x[2], x[3], x[4]}
```

Out[79]= $\{-0.125,\ 0.00195313,\ -7.45058 \times 10^{-9},\ 0.\}$

In[80]:=
```
x[0] = 1; {x[1], x[2], x[3], x[4]}
```

Out[80]= $\{-1,\ 1,\ -1,\ 1\}$

In[81]:=
```
x[0] = 2; {x[1], x[2], x[3], x[4]}
```

Out[81]= $\{-8,\ 512,\ -134\,217\,728,\ 2\,417\,851\,639\,229\,258\,349\,412\,352\}$

In[82]:=
```
Clear[f, x]
```

Integralrechnung

Unbestimmtes Integral

Das Integralzeichen geben wir mithilfe der Palette ein.Alternativ kann auch der Befehl **Integrate[**f, x**]** verwendet werden.

In[83]:=
```
∫ (Sin[x] + x^2) dx
```

Out[83]= $\frac{x^3}{3} - \mathrm{Cos}[x]$

Mathematica gibt keine Integrationskonstante aus.

Bestimmtes Integral

WiederverwendenwirdiePaletteoderdenBefehl **Integrate[**f, {x, xmin, xmax}**]**:

In[84]:=
$$\int_0^1 \left(2\, e^x + x\right) dx$$

Out[84]=
$$-\frac{3}{2} + 2\, e$$

Kann das Integral nicht analytisch berechnet werden, so kann ein numerischer Wert mit dem Befehl **NIntegrate[**f, {x, xmin, xmax}**]** erhalten werden.

Uneigentliches Integral

Analog berechnen wir ein uneigentliches Integral:

In[85]:=
$$\int_1^\infty \frac{1}{t^3}\, dt$$

Out[85]=
$$\frac{1}{2}$$

Gammafunktion

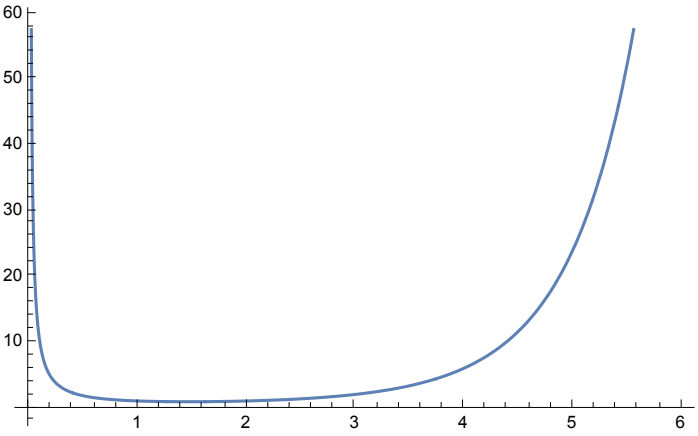
Die Gammafunktion ist eingebaut:

In[86]:=
```
Gamma[5]
```

Out[86]=
```
24
```

In[87]:=
```
Plot[Gamma[x], {x, 0, 6}]
```

Out[87]=



Fourierreihen

Fourierpolynom

Natürlich gibt es auch einen Befehl in Mathematica, um ein Fourierpolynom zu berechnen:

In[88]:=
```
FourierTrigSeries[Abs[t], t, 3, FourierParameters → {1, 2 π}]
```

Out[88]=
$$\frac{1}{4} - \frac{2\,\text{Cos}[2\,\pi\,t]}{\pi^2} - \frac{2\,\text{Cos}[6\,\pi\,t]}{9\,\pi^2}$$

Mathematica nimmt als Intervall automatisch[$-\pi$, π] an, den allgemeinen Fall [–T, T] erhält man

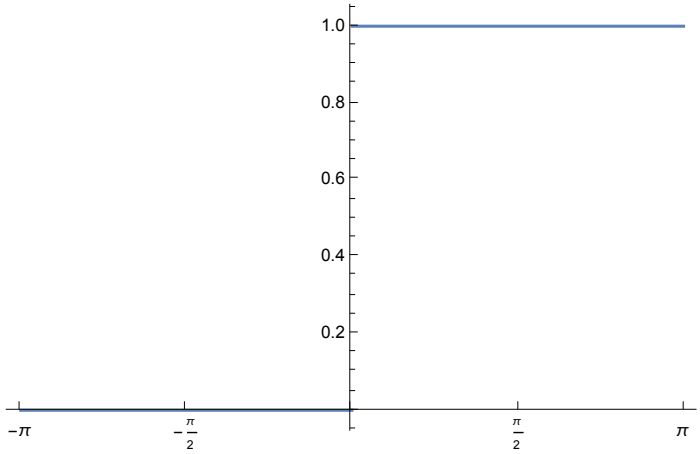
mit der Option **FourierParameters** → {1, $\frac{2\pi}{T}$}.

Fourierreihe

Wir definieren zunächst die Funktion

In[89]:=
```
f[t_] := If[t > 0, 1, 0];
Plot[f[t], {t, -π, π}, Ticks → {{-π, -π/2, 0, π/2, π}, Automatic}]
```

Out[89]=



und berechnen

In[90]:=
```
a[0] = 1/π ∫_-π^π f[t] 𝕕t
```

Out[90]=
```
1
```

Weiters gilt

In[91]:=
```
a[k_] = 1/π ∫_-π^π f[t] Cos[k t] 𝕕t
```

Out[91]=
$$\frac{\text{Sin}[k\,\pi]}{k\,\pi}$$

Das ist aber für ganzzahliges k gleich Null, wie auch mit Mathematica leicht zu sehen ist:

In[92]:=
```
Simplify[%, k ∈ Integers]
```

Out[92]=
```
0
```

(Die Option k ∈ Integers legt fest, dass k ganzzahlig ist. Eingabe entweder über die Palette oder als **Element[**k,Integers**]**.)

Analog berechnen wir

In[93]:=
```
b[k_] = 1/π ∫_-π^π f[t] Sin[k t] 𝕕t
```

Out[93]=
$$\frac{1 - \text{Cos}[k\,\pi]}{k\,\pi}$$

was sich ebenfalls noch etwas vereinfachen lässt:

In[94]:=
```
Simplify[%, k ∈ Integers]
```

Out[94]=
$$\frac{1+(-1)^{1+k}}{k\,\pi}$$

Berechnen wir noch explizit die ersten sechs Koeffizienten b_k:

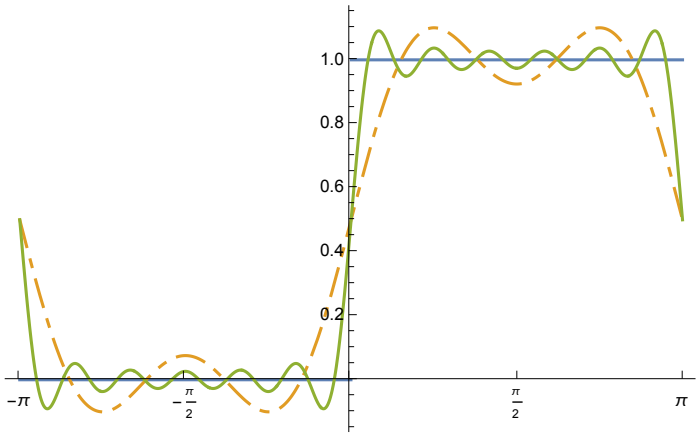
In[95]:=
```
{b[1], b[2], b[3], b[4], b[5], b[6]}
```

Out[95]=
$$\left\{\frac{2}{\pi}, 0, \frac{2}{3\,\pi}, 0, \frac{2}{5\,\pi}, 0\right\}$$

Zum Schluss noch ein Vergleich zwischen der Funktion und zweier Fourierpolynome:

In[96]:=
```
F[n_, t_] := FourierTrigSeries[f[t], t, n];
Plot[Evaluate[{f[t], F[3, t], F[12, t]}], {t, -π, π},
 PlotStyle → {{}, {Dashing[{0.04`, 0.02`, 0.01`, 0.02`}]}, {}},
 Ticks → {{-π, -π/2, 0, π/2, π}, Automatic}]
```

Out[97]=



In[98]:=
```
Clear[a, b, f, F];
```

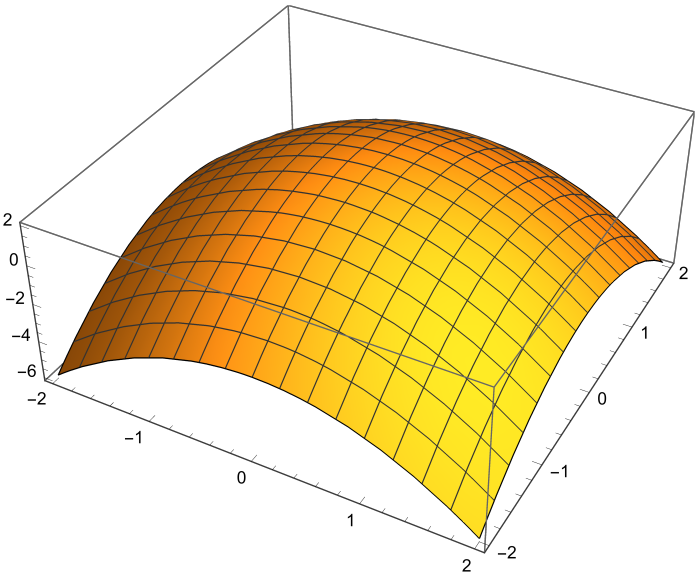
Differentialrechnung in mehreren Variablen

Graphische Darstellung

Zur Veranschaulichung einer reellwertigen Funktion von zwei Variablen kann der Befehl **Plot3D[**f[x, y], {x, xmin, xmax}, {y, ymin, ymax}**]** verwendet werden.

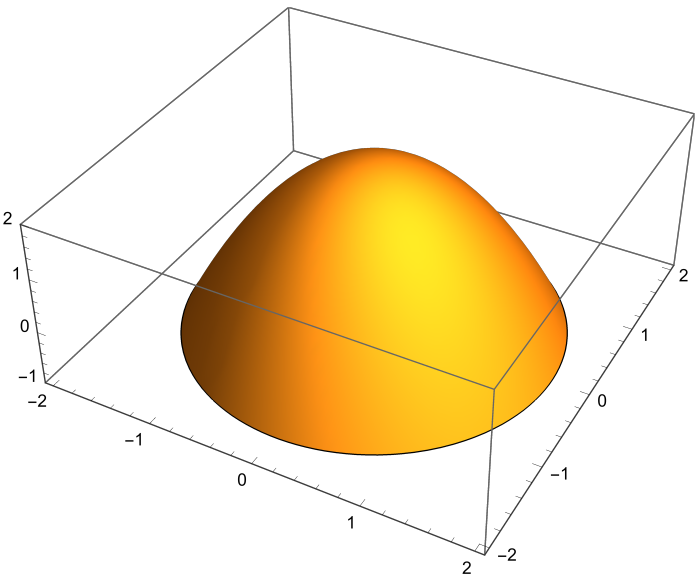
In[99]:=
```
Plot3D[2 - x^2 - y^2, {x, -2, 2}, {y, -2, 2}]
```

Out[99]=



In[100]:=
```
Plot3D[2 - x^2 - y^2, {x, -2, 2}, {y, -2, 2}, ClippingStyle → None,
 PlotPoints → 50, Mesh → False, PlotRange → {-1, 2}]
```

Out[100]=



Partielle Ableitung

Mit Mathematica können wir partielle Ableitungen wie folgt berechnen:

In[101]:=
```
D[x1 x2^3, x1, x1]
```

Out[101]=
```
0
```

oder

In[102]:=
```
D[x1 x2^3, x1, x2]
```

Out[102]=
$$3\,x_2^2$$

Hesse-Matrix

Die Hesse-Matrix erhalten wir mit

In[103]:=
```
Table[D[x1 (x2 - 1) + x1^3, xk, xj], {k, 2}, {j, 2}] // MatrixForm
```

Out[103]//MatrixForm=

$$\begin{pmatrix} 6\,x_1 & 1 \\ 1 & 0 \end{pmatrix}$$

Differentialgleichungen

Differentialgleichungen

Mit Mathematica können wir Differentialgleichungen mit dem Befehl **DSolve** lösen:

In[104]:=
```
DSolve[x'[t] == 2 Sqrt[x[t]], x[t], t]
```

Out[104]=

$$\left\{\left\{x[t] \to \frac{1}{4}\left(4\,t^2 + 4\,t\,C[1] + C[1]^2\right)\right\}\right\}$$

Die Parameter in der allgemeinen Lösung werden mit C[1], C[2], etc.bezeichnet.Da unsere Gleichung von der Ordnung eins ist, gibt es genau eine Konstante.

Auch Anfangsbedingungen können angegeben werden :

In[105]:=
```
DSolve[{x'[t] == 2 Sqrt[x[t]], x[0] == x0}, x[t], t]
```

Out[105]=

$$\left\{\left\{x[t] \to t^2 - 2\,t\,\sqrt{x0} + x0\right\}, \left\{x[t] \to t^2 + 2\,t\,\sqrt{x0} + x0\right\}\right\}$$

Beachten Sie, dass die erste von Mathematica ausgegebene Lösung keine Lösung der gegebenen Differentialgleichung ist (sie erfüllt $x'(t) = -2\sqrt{x(t)}$). Außerdem findet das Programm für $x_0 = 0$ nicht alle Lösungen, die wir im Buch gefunden haben. (Es schadet also nie, die Ergebnisse des Computers kritisch zu überprüfen; -)

Auch Differentialgleichungen höherer Ordnung sind kein Problem:

In[106]:=
```
DSolve[{x''[t] == x'[t] + 6 x[t], x[0] == 1, x'[0] == 8}, x[t], t]
```

Out[106]=

$$\left\{\left\{x[t] \to e^{-2\,t}\left(-1 + 2\,e^{5\,t}\right)\right\}\right\}$$

Parabolspiegel

Die Lösung der Differentialgleichung des Parabolspiegels:

In[107]:=
```
DSolve[y'[x] == (y'[x] * y[x] + x) / Sqrt[y[x]^2 + x^2], y[x], x]
```

Out[107]=

$$\{\{y[x] \to -x\,\text{Sinh}[C[1] - \text{Log}[x]]\}\}$$

Systeme von Differentialgleichungen

Der **DSolve**-Befehl kann zwar theoretisch auch Systeme lösen, meistens ist aber eine analytische

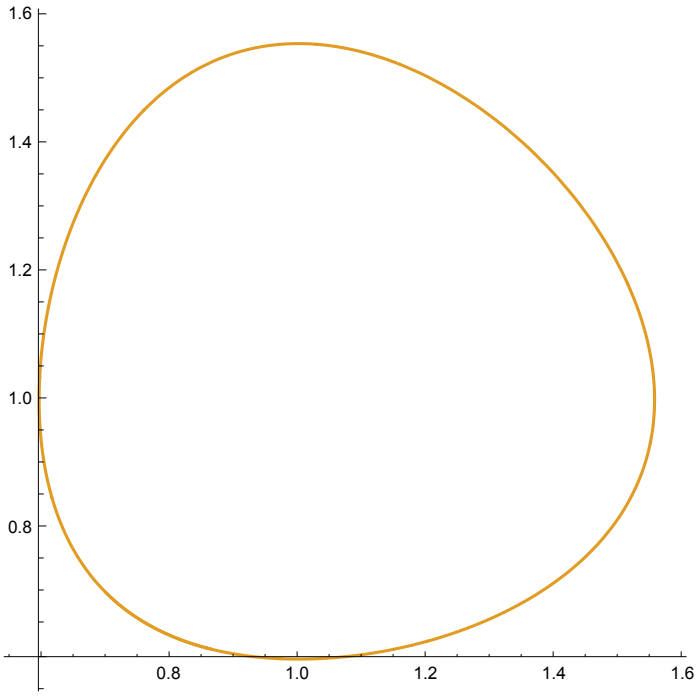
Lösung nicht möglich. Man muss dann das System numerisch mit **NDSolve** lösen.

Eine Lösungskurve x(t) = (x(t), y(t)) des Volterra-Lotka-Systems zu den Anfangsbedingungen (x(0), y(0)) = (0.7, 0.7) im Intervall t ∈ [0, 7] erhalten wir mit

In[108]:=
```
loesung = {x[t], y[t]} /. NDSolve[{x'[t] == (1 - y[t]) x[t],
      y'[t] == (x[t] - 1) y[t], x[0] == 0.7, y[0] == 0.7}, {x, y}, {t, 0, 7}][[1]];
```

In[109]:=
```
ParametricPlot[loesung, {t, 0, 7}]
```

Out[109]=



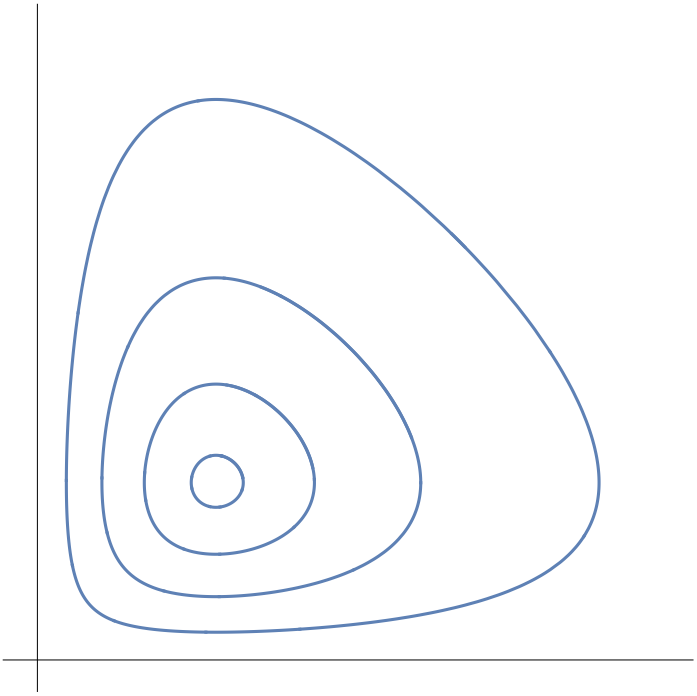
Folgendes Programm berechnet die Lösungskurven für mehrere Anfangswerte und stellt sie gemeinsam dar.

In[110]:=
```
PhasePlot[f_, ic_, tmax_, opts___] :=
  Block[{i, n = Length[ic], ff, ivp, sol, phaseplot},
   ff = f /. {x → x[t], y → y[t]};
   Off[ParametricPlot::"ppcom"];
   Do[
    ivp = {x'[t] == ff[[1]], y'[t] == ff[[2]], x[0] == ic[[i, 1]], y[0] == ic[[i, 2]]};
    sol = NDSolve[ivp, {x[t], y[t]}, {t, -tmax, tmax}];
    phaseplot[i] = ParametricPlot[{x[t], y[t]} /. sol, {t, -tmax, tmax}]
    , {i, 1, n}];
   On[ParametricPlot::"ppcom"];
   Show[Table[phaseplot[i], {i, 1, n}], opts]
  ];
```

Und so wird der Befehl verwendet (die Anfangsbedingung (x(0), y(0)) = (1, 1) entspricht übrigens dem Fixpunkt):

In[111]:=
```
PhasePlot[{(1 - y) x, (x - 1) y},
 {{0.3, 0.3}, {0.5, 0.5}, {0.7, 0.7}, {0.9, 0.9}, {1, 1}},
 3.7, PlotRange → {{0, 3.5}, {0, 3.5}}, Ticks → False]
```

Out[111]=



Analog kann die Lösung der Lorenz-Gleichung mit dem Befehl

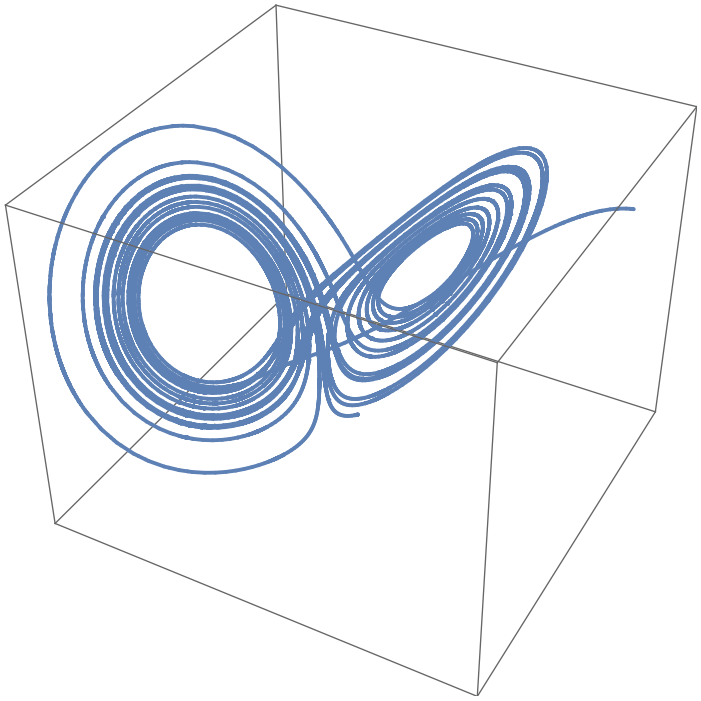
In[112]:=
```
σ = 10; ρ = 28; β = 8 / 3;
loesung = {x[t], y[t], z[t]} /. NDSolve[{x'[t] == -σ (x[t] - y[t]),
      y'[t] == -x[t] z[t] + ρ x[t] - y[t], z'[t] == x[t] y[t] - β z[t],
      x[0] == 30, y[0] == 10, z[0] == 40}, {x, y, z}, {t, 0, 20}][[1]];
```

berechnet werden.

EIne grafische Darstellung erhält man mit

In[113]:=
```
ParametricPlot3D[loesung, {t, 0, 20}, PlotPoints → 2000, Axes → False]
```

Out[113]=



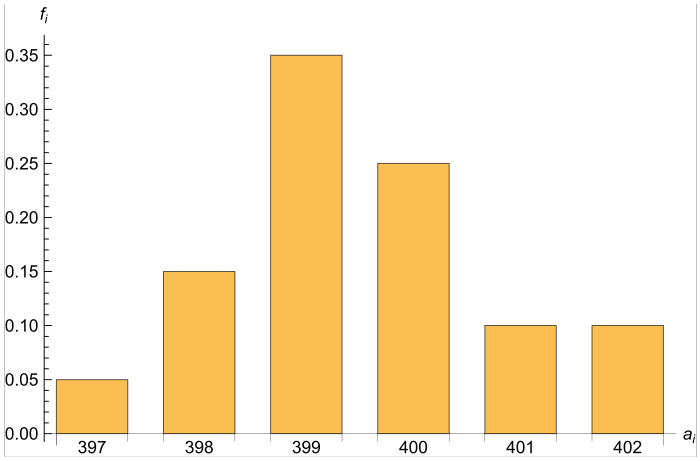
Beschreibende Statistik und Zusammenhangsanalysen

Stabdiagramm

Ein Stabdiagramm kann mit dem Befehl

In[114]:=
```
BarChart[{1 / 20, 3 / 20, 7 / 20, 1 / 4, 1 / 10, 1 / 10}, BarSpacing → 0.5,
 ChartLabels → {"397", "398", "399", "400", "401", "402"}, AxesLabel → {"aᵢ", "fᵢ"}]
```

Out[114]=



gezeichnet werden.

Verteilungsfunktion

Eine Verteilungsfunktion kann wie folgt dargestellt werden. Zuerst summieren wir die Werte der Reihe nach auf:

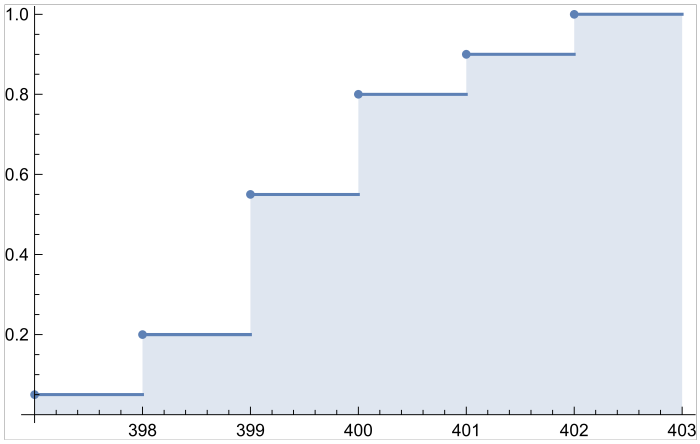
In[115]:=
```
F = Accumulate[{1 / 20, 3 / 20, 7 / 20, 1 / 4, 1 / 10, 1 / 10}]
```

Out[115]=
$$\left\{\frac{1}{20}, \frac{1}{5}, \frac{11}{20}, \frac{4}{5}, \frac{9}{10}, 1\right\}$$

Das sind die Funktionswerte an den einzelnen Punkten. Nun können wir sie wie folgt darstellen:

In[116]:=
```
DiscretePlot[F[[n - 396]], {n, 397, 402},
 ExtentSize → {None, Full}, PlotMarkers → "Point"]
```

Out[116]=

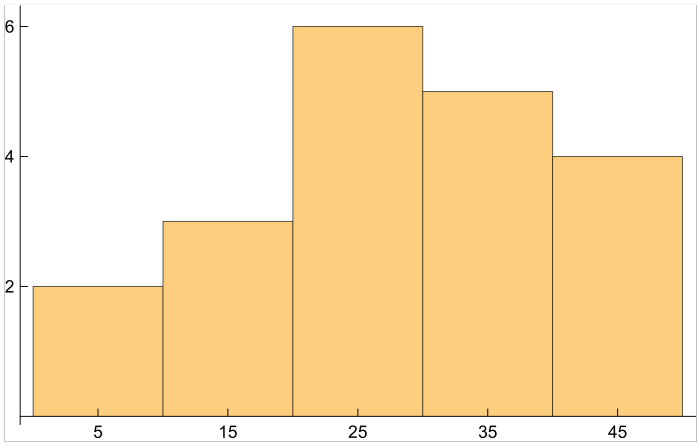


Histogramm

Ein Histogramm erhalten wir so:

In[117]:=
```
Histogram[{3, 7, 12, 18, 19, 20, 25, 25, 27, 28, 29, 31, 32, 34, 37, 38, 40, 41, 45,
  47}, {{0, 10, 20, 30, 40, 50}}, Ticks → {{5, 15, 25, 35, 45}, {2, 4, 6}}]
```

Out[117]=



Das erste Argument enthält die Urliste und das zweite (optional) die Klassengrenzen.

Arithmetische Mittel

Das arithmetische Mittel kann mit dem Befehl **Mean** berechnet werden :

In[118]:=
```
Mean[{1, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4}]
```

Out[118]=
$$\frac{30}{11}$$

Median und Quantile

Den Median berechnen wir so :

In[119]:=
```
data = {1, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4};
Median[data]
```

Out[120]= 3

Ein p-Quantil (bzw. mehrere gleichzeitig, wenn man die verschiedenen p-Werte als Liste übergibt) erhalten wir mit :

In[121]:=
```
Quantile[data, {0.25, 0.5, 0.75}]
```

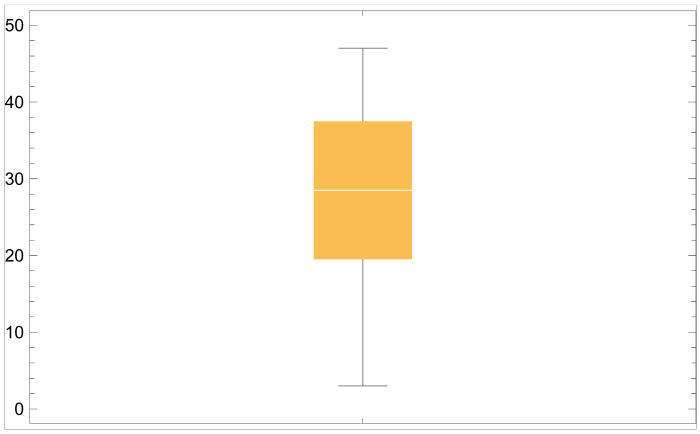
Out[121]= {2, 3, 3}

(Achtung : Mathematica verwendet eine leicht abweichende Definition für das p-Quantil.)

Eine Box Plot erhält man mit :

In[122]:=
```
BoxWhiskerChart[
 {3, 7, 12, 18, 19, 20, 25, 25, 27, 28, 29, 31, 32, 34, 37, 38, 40, 41, 45, 47}]
```

Out[122]=



Standardabweichung

Varianz und Standardabweichung können mit **Variance** bzw. **StandardDeviation** berechnet werden:

In[123]:=
```
data = {1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 7};
{Variance[data], StandardDeviation[data]}
```

Out[124]= $\left\{\frac{14}{5}, \sqrt{\frac{14}{5}}\right\}$

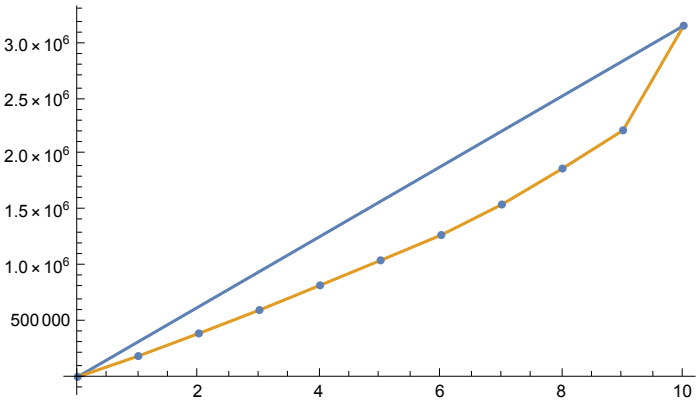
Lorenz-Kurve und Gini-Koeffizient

Lorenz-Kurve für die häufigsten Automarken in Österreich 2012

In[125]:=
```
data = {187 244, 204 044, 210 476, 222 903,
    225 007, 227 863, 275 077, 323 840, 344 505, 942 268};
n = Length[data];
adata = Join[{{0, 0}}, Transpose[{Table[i, {i, n}], Accumulate[data]}]];
f = Interpolation[adata, InterpolationOrder → 1];
```

In[129]:=
```
Show[Plot[{f[n]/n x, f[x]}, {x, 0, n}], ListPlot[adata]]
```

Out[129]=



In[130]:=
```
Gini = (n f[n] / 2 - NIntegrate[f[x], {x, 0, n}]) / (n f[n] / 2) // N
```

Out[130]=
```
0.26886
```

Lorenz - Kurve für die Einkommensverteilung in Österreich 2011

In[131]:=
```
data = {370, 667, 1049, 1317, 1540, 1733, 1933, 2182, 2537, 3200, 5444};
data = Join[{0}, Table[(data[[i]] + data[[i + 1]] - 1)/2., {i, Length[data] - 1}]]
```

Out[132]=
```
{0, 518., 857.5, 1182.5, 1428., 1636., 1832.5, 2057., 2359., 2868., 4321.5}
```

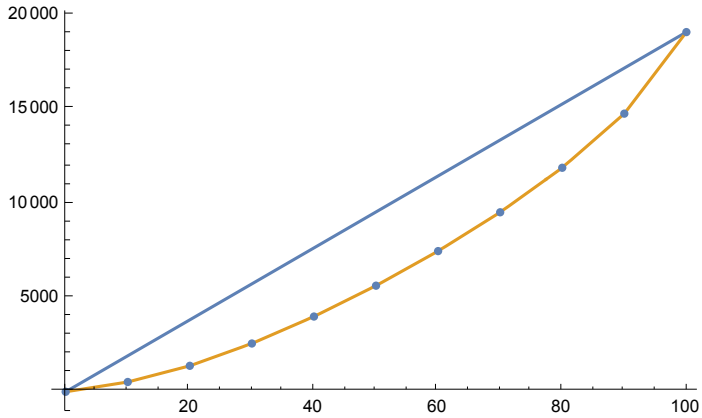
In[133]:=
```
Mean[%[[2 ;; -1]]]
```

Out[133]=
```
1906.
```

In[134]:=
```
n = Length[data];
adata = Transpose[{Table[10 (i - 1), {i, n}], Accumulate[data]}];
f = Interpolation[adata, InterpolationOrder → 1];
```

In[137]:=
```
Show[Plot[{f[100]/100 x, f[x]}, {x, 0, 100}], ListPlot[adata]]
```

Out[137]=



In[138]:=
```
Gini = (NIntegrate[f[100]/100 x - f[x], {x, 0, 100}])/(f[100] 100 / 2)
```

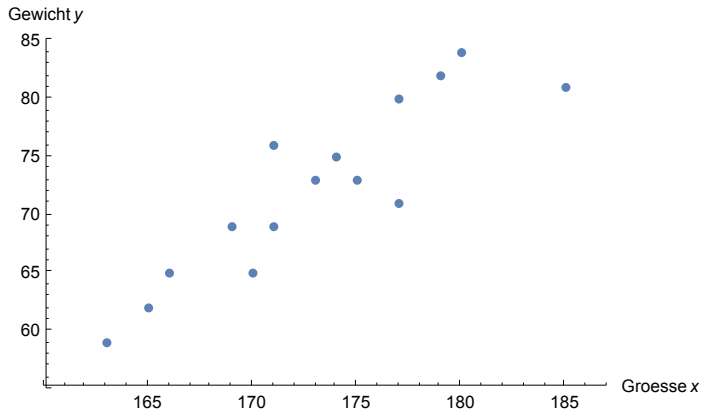
Out[138]=
```
0.295231
```

Streudiagramm

Ein Streudiagramm erhält man leicht auf folgende Weise: Die Stichprobendaten werden als Liste eingegeben (z.B. mit dem Variablennamen data), danach wird mit dem Befehl **ListPlot** die Graphik erzeugt:

In[139]:=
```
data = {{163, 59}, {165, 62}, {166, 65}, {169, 69}, {170, 65},
   {171, 69}, {171, 76}, {173, 73}, {174, 75}, {175, 73} ,
   {177, 80} , {177, 71}, {179, 82}, {180, 84}, {185, 81}};
ListPlot[data, PlotRange -> {{160, 187}, {55, 85}},
 AxesLabel → {Groesse x, Gewicht y}, PlotStyle → PointSize[0.016]]
```

Out[140]=



(Hier wurde – was nicht unbedingt notwendig ist – der dargestellte x,y-Bereich mit **PlotRange** gewählt, die Achsen wurden beschriftet und die Punkte vergrößert.)

Korrelationskoeffizient

Nach dem Laden des Zusatzpakets

In[141]:=
```
Needs["MultivariateStatistics`"]
```

kann der empirische Korrelationskoeffizient mit dem Befehl **Correlation** berechnet werden:

In[142]:=
```
xdata = {30, 45, 45, 60, 75, 80, 90, 100, 110, 120};
ydata = {6, 5, 3, 4, 3, 5, 2, 4, 2, 3};
Correlation[xdata, ydata] // N
```

Out[144]=
```
-0.629935
```

In[145]:=
```
xdata = {163, 165, 166, 169, 170, 171, 171, 173, 174, 175, 177, 177, 179, 180, 185.};
ydata = {59, 62, 65, 69, 65, 69, 76, 73, 75, 73, 80, 71, 82, 84, 81.};
Correlation[xdata, ydata] // N
```

Out[147]=
```
0.897914
```

In[148]:=
```
xdata = {163, 165, 166, 169, 170, 171, 171, 173, 174, 175, 177, 177, 179, 180, 185};
ydata = {2900, 1100, 3600, 2300, 4000, 5600,
  2100, 5100, 3400, 1800, 2100, 2600, 4600, 3600, 2300};
Correlation[xdata, ydata] // N
```

Out[148]=
```
0.0606408
```

Regressionsgerade

Der Befehl **Fit** gibt uns eine Regressionsgerade aus :

In[149]:=
```
data = {{163, 59}, {165, 62}, {166, 65}, {169, 69}, {170, 65},
   {171, 69}, {171, 76}, {173, 73}, {174, 75}, {175, 73} ,
   {177, 80} , {177, 71}, {179, 82}, {180, 84}, {185, 81}};
g = Fit[data, {1, x}, x]
```

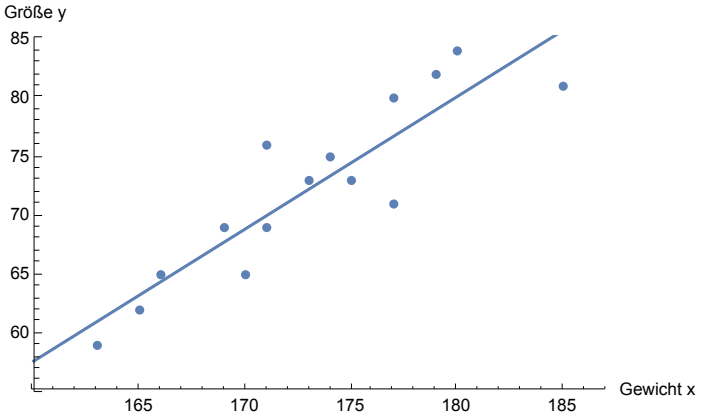
Out[150]=
```
-122.02 + 1.12305 x
```

Die Syntax ist **Fit[**Daten, Funktionen, Variablen**]**. Dadurch werden Daten mithilfe der Gauß'schen Methode der kleinsten Quadrate durch eine Linearkombination der Funktionen genähert. Hier möchten wir eine Gerade, also eine Linearkombination der Funktionen 1 und x.

Nun können wir die Regressionsgerade zusammen mit den Daten darstellen:

In[151]:=
```
g1 = ListPlot[data,
    PlotRange -> {{160, 187}, {55, 85}}, PlotStyle → PointSize[0.016]];
g2 = Plot[g, {x, 160, 190}];
Show[g1, g2, AxesLabel -> {"Gewicht x", "Größe y"}]
```

Out[152]=



Natürlich können wir auch direkt die Formeln aus dem Buch verwenden:

In[153]:=
```
xdata = Transpose[data][[1]];
ydata = Transpose[data][[2]];
{mx, my} = Mean /@ {xdata, ydata};
{sx, sy} = StandardDeviation /@ {xdata, ydata};
r = Correlation[xdata, ydata];
k = r sy/sx;
d = my - k mx;
{mx, my, sx, sy, r, k, d} // N
```

Out[160]=
```
{173., 72.2667, 6.04743, 7.5637, 0.897914, 1.12305, -122.02}
```

In[161]:=
```
Clear[Gini, F, mx, my, sx, sy, r, k, d, f, g, g1, g2, data, xdata, ydata]
```

Anscombe-Quartett

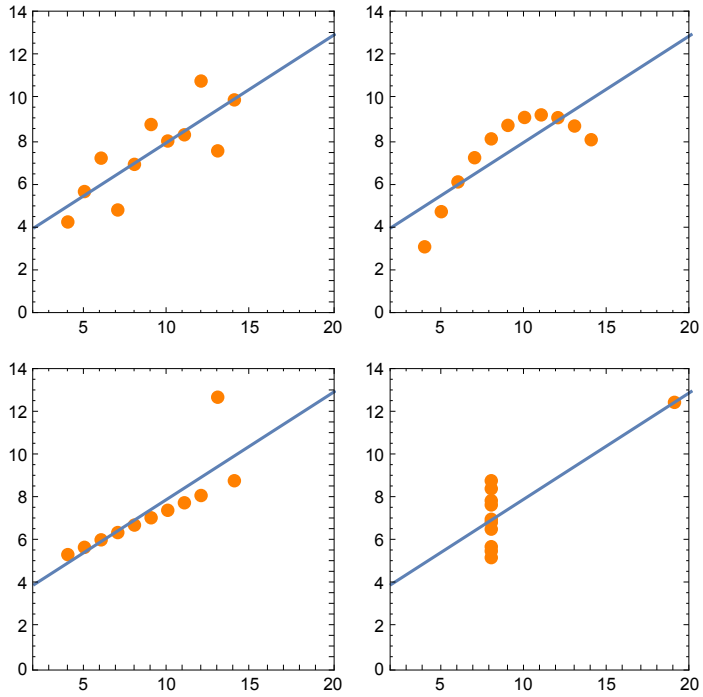
Eine Darstellung der Anscomb-Quartett:

In[162]:=
```
AnscombeA = {{10, 8.04}, {8, 6.95}, {13, 7.58}, {9, 8.81}, {11, 8.33},
   {14, 9.96}, {6, 7.24}, {4, 4.26}, {12, 10.84}, {7, 4.82}, {5, 5.68}};
AnscombeB = {{10, 9.14}, {8, 8.14}, {13, 8.74}, {9, 8.77}, {11, 9.26},
   {14, 8.1}, {6, 6.13}, {4, 3.1}, {12, 9.13}, {7, 7.26}, {5, 4.74}};
AnscombeC = {{10, 7.46}, {8, 6.77}, {13, 12.74}, {9, 7.11}, {11, 7.81},
   {14, 8.84}, {6, 6.08}, {4, 5.39}, {12, 8.15}, {7, 6.42}, {5, 5.73}};
AnscombeD = {{8, 6.58}, {8, 5.76}, {8, 7.71}, {8, 8.84}, {8, 8.47},
   {8, 7.04}, {8, 5.25}, {8, 5.56}, {8, 7.91}, {8, 6.89}, {19, 12.5}};
```

In[163]:=
```
Anscombe[data_] := Block[{g},
  g[x_] = Fit[data, {1, x}, x];
  Show[ListPlot[data, PlotRange → {{2, 20}, {0, 14}},
    PlotStyle → {Orange, PointSize[Large]}, AspectRatio → 1,
    Frame → True, Axes → False], Plot[g[x], {x, 2, 20}]]
 ]
```

In[164]:=
```
GraphicsGrid[{{Anscombe[AnscombeA], Anscombe[AnscombeB]},
  {Anscombe[AnscombeC], Anscombe[AnscombeD]}}]
```

Out[164]=



Elementare Wahrscheinlichkeitsrechnung

Optimale Stoppstrategie

Die Wahrscheinlichkeiten für die optimale Stoppstrategie

In[165]:=
$$P[n\_, j\_] := \frac{j}{n} \sum_{k=j}^{n-1} \frac{1}{k};$$

können mit Mathematica leicht berechnet werden:

In[166]:=
```
Table[N[P[12, j], 2], {j, 11}]
```

Out[166]=
```
{0.25, 0.34, 0.38, 0.40, 0.39, 0.37, 0.33, 0.28, 0.23, 0.16, 0.083}
```

In[167]:=
```
Clear[P];
```

Zufallsvariablen

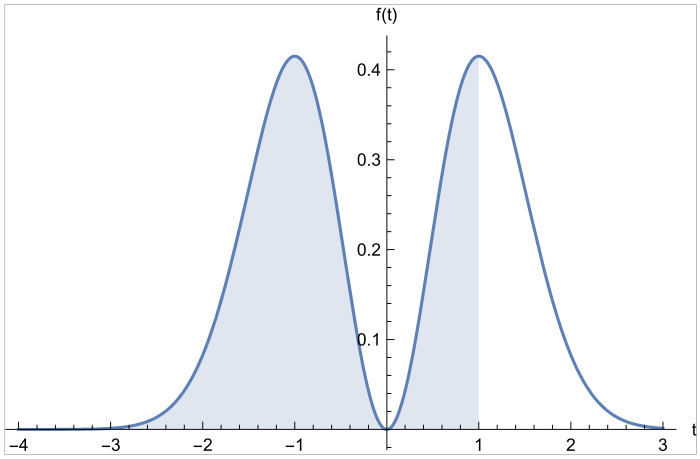
FilledPlot

Ein Bereich einer Verteilung kann wie folgt veranschaulicht werden:

In[168]:=
$$f[x\_] := \frac{2}{\sqrt{\pi}} x^2 \, \text{Exp}\left[-x^2\right];$$

In[169]:=
```
gr1 = Plot[f[x], {x, -4, 1}, Filling → Axis];
gr2 = Plot[f[x], {x, -4, 3}];
Show[gr1, gr2, AxesLabel → {"t", "f(t)"}, PlotRange → All]
```

Out[171]=



Zufallszahlen

Zur Simulation von komplexen Zufallsvorgängen (z.B. bei der Planung von Großprojekten) benötigt man Zufallszahlen, die bestimmten Verteilungen genügen. In Mathematica können im Intervall [0, 1] gleichverteilte Zufallszahlen mit dem Befehl **Random[]** (ohne Argument) erhalten werden.

Mit dem Befehl

In[172]:=
```
RandomReal[ExponentialDistribution[1.5]]
```

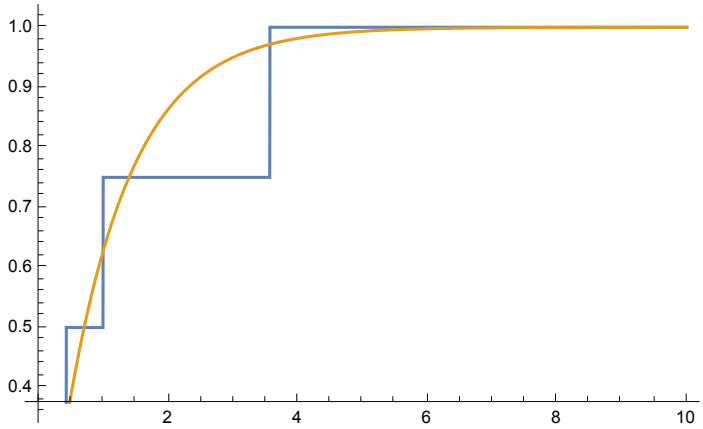
Out[172]=
```
0.573374
```

können exponentialverteilte (k = 1.5) Zufallszahlen erhalten werden. Es stehen noch weitere Verteilungen zur Verfügung, die wir später noch kennen lernen werden.

Eine Veranschaulichung des Hauptsatz der Statstik anhand eines Computerexperiments mit einer exponentialverteilten (k = 1) Zufallsvariablen erhalten wir wie folgt:

In[173]:=
```
n = 4; xlist = RandomReal[ExponentialDistribution[1], n];
F[x_] := 1 - e^-x;
Fn[x_] := Count[xlist, y_ /; (y ≤ x)] / Length[xlist];
Plot[{Fn[x], F[x]}, {x, 0, 10}]
```

Out[176]=



Der Befehl **Count** zählt die Anzahl der Elemente y in der Liste, die die Bedingung y <= x erfüllen. Division durch die gesamte Anzahl (= Länge der Liste) ergibt die empirische Verteilungsfunktion.

In[177]:=
```
Clear[f, F, Fn, n]
```

Diskrete Wahrscheinlichkeitsverteilungen

Allgemein gilt: Der Erwartungswert wird mit **Mean**, die Varianz mit **Variance** berechnet. Die Wahrscheinlichkeit P(X=x) erhalten wir mit **PDF** („ probability density function ") und die Wahrscheinlichkeit P(X<=x) wird mit dem Befehl **CDF** („cumulative distribution function ") aufgerufen.

Hypergeometrische Verteilung

Mit **HypergeometricDistribution[**n, M, N**]** erhalten wir (symbolisch) die hypergeometrische Verteilung mit den Parametern n, M, N. Wir können nun z.B. den Erwartungswert und die Varianz berechnen :

In[178]:=
```
hdist = HypergeometricDistribution[5, 4, 20];
{Mean[hdist], Variance[hdist]}
```

Out[179]=
$$\left\{1, \frac{12}{19}\right\}$$

Hier wurde als Abkürzung für die Verteilung die Variable hdist eingeführt. Die Wahrscheinlichkeit P(X=2) bzw.die Verteilungsfunktion F(2) = P(X<=2) erhalten wir mit :

In[180]:=
```
{PDF[hdist, 2], CDF[hdist, 2]} // N
```

Out[180]=
```
{0.216718, 0.968008}
```

Binomialverteilung

Mit **BinomialDistribution[**n, p**]** kann die Binomialverteilung aufgerufen werden:

In[181]:=
```
bdist = BinomialDistribution[50, 0.03]; PDF[bdist, 2]
```
Out[181]= 0.255518

Poisson-Verteilung

Der Befehl für die Poisson - Verteilung ist **PoissonDistribution[λ]** :

In[182]:=
```
pdist = PoissonDistribution[1.6];
CDF[pdist, 2]
```
Out[183]= 0.783358

Stetige Wahrscheinlichkeitsverteilungen

Wie schon bei den diskreten Verteilungen kann der Erwartungswert mit **Mean** und die Varianz mit **Variance** berechnet werden. Die Anweisung **CDF** („cumulative distribution function“) gibt die Verteilungsfunktion F und der Befehl **PDF** („probability density function“) liefert die Dichtefunktion f.

Normalverteilung

Mit **NormalDistribution[μ, σ]** erhält man (symbolisch) die Normalverteilung mit den Parametern μ, σ:

In[184]:=
```
ndist = NormalDistribution[4, 2];
CDF[ndist, 6] // N
```
Out[185]= 0.841345

(Wir haben diese Normalverteilung mit ndist abgekürzt, um sie z.B. handlicher als Argument des **CDF**-Befehls schreiben zu können.) Wenn Sie nicht numerisch rechnen (also // N weglassen), so bekommen Sie einen symbolischen Output, der hier die Errorfunktion erf(x) enthält. Sie ist nichts anderes als eine Abkürzung für das (nur numerisch berechenbare) Integral erf(x) = $\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt$.

p-Quantile

Das p - Quantil einer Verteilung erhalten wir mit dem Befehl **Quantile[**verteilung, p**]**.Für eine Normalverteilung ergibt sich zum Beispiel :

In[186]:=
```
Quantile[NormalDistribution[100, 5], 0.05]
```
Out[186]= 91.7757

Normalverteilung als Näherung

Näherung der Binomialverteilung:

In[187]:=
```
n = 10^7; p = 10^-5; μ = n p; σ = Sqrt[n p (1 - p)]; x = 90;
1 - {CDF[BinomialDistribution[n, p], x], CDF[NormalDistribution[μ, σ], x + 0.5]} //
 N
```

Out[188]= {0.828616, 0.828945}

Näherung der Poisson-Verteilung:

In[189]:=
```
λ = 20;
x = 14;
{CDF[PoissonDistribution[λ], x], CDF[NormalDistribution[λ, Sqrt[λ]], x + 0.5]} // N
```

Out[189]= {0.104864, 0.109379}

Prüfverteilungen

Das 0.9-Quantil der Chi-Quadrat-Verteilung mit m = 12 Freiheitsgraden ist

In[190]:=
```
Quantile[ChiSquareDistribution[12], 0.9]
```

Out[190]= 18.5493

Das 0.9-Quantil der t-Verteilung mit m = 12 Freiheitsgraden erhalten wir mit

In[191]:=
```
Quantile[StudentTDistribution[12], 0.9]
```

Out[191]= 1.35622

Das 0.95-Quantil der F-Verteilung mit $m_1 = 2$ und $m_2 = 12$ Freiheitsgraden berechnet sich so:

In[192]:=
```
Quantile[FRatioDistribution[2,12], 0.95]
```

Out[192]= 3.88529

In[193]:=
```
Clear[ndist, n, p, μ, σ, x]
```

Schließende Statistik

Funktionen zur Berechnung von Konfidenzintervallen stehen nach dem Laden des Zusatzpakets

In[194]:=
```
Needs["HypothesisTesting`"]
```

zur Verfügung. Die Beispieldaten aus dem Buch sind

In[195]:=
```
data = {100., 97., 101., 96., 98., 102., 96., 100., 101., 98.};
{Mean[data], Variance[data], StandardDeviation[data]}
```

Out[196]= {98.9, 4.76667, 2.18327}

Konfidenzintervall für den Erwartungswert

Das Konfidenzintervall einer Stichprobe aus einer normalverteilten Grundgesamtheit mit bekannter Varianz erhalten wir mit:

In[197]:=
```
MeanCI[data, KnownVariance → 4, ConfidenceLevel → .9]
```

Out[197]= {97.8597, 99.9403}

Ohne Angabe des Konfidenzniveaus (**ConfidenceLevel**) wird ein Niveau von $1 - \alpha = 0.95$ verwendet :

In[198]:=
```
MeanCI[data, KnownVariance → 4]
```

Out[198]=
```
{97.6604, 100.14}
```

Wird die Varianz nicht angegeben, so wird sie automatisch als unbekannt angenommen:

In[199]:=
```
MeanCI[data]
```

Out[199]=
```
{97.3382, 100.462}
```

Konfidenzintervall für den Vergleich zweier Erwartungswerte

Das Konfidenzintervall für den Vergleich zweier Erwartungswerte bekommen wir so:

In[200]:=
```
data1 = {152, 148, 149, 146, 146, 152, 149, 150};
data2 = {153, 150, 149, 149, 152, 151, 154, 152};
MeanDifferenceCI[data1, data2, EqualVariances → True, ConfidenceLevel → 0.9]
```

Out[202]=
```
{-4.09578, -0.404222}
```

Konfidenzintervall für die Varianz

Das Konfidenzintervall für die Varianz bekommen wir mit

In[203]:=
```
VarianceCI[data]
```

Out[203]=
```
{2.25519, 15.8866}
```

Konfidenzintervall für den Anteilswert

Ein Befehl zur Berechnung des Konfidenzintervalls für einen Anteilswert steht zwar nicht zur Verfügung, wir können aber leicht einen definieren. Da Mathematica die F-Verteilung kennt, können wir die exakte Formel implementieren :

In[204]:=
```
ProportionCI[n_, x_, opts___] := Block[{p, Fp},
  p = 1 - (1 - ConfidenceLevel)/2 /. {opts, ConfidenceLevel → .95};
  Fp[m1_, m2_] := Quantile[FRatioDistribution[m1, m2], p];
  {x/(x + (n - x + 1) Fp[2 (n - x + 1), 2 x]),
   (x + 1) Fp[2 (x + 1), 2 (n - x)]/(n - x + (x + 1) Fp[2 (x + 1), 2 (n - x)])}];
```

In[205]:=
```
ProportionCI[500, 60]
```

General: $0.151752^{440.}$ is too small to represent as a normalized machine number; precision may be lost.

General: $0.151752^{440.}$ is too small to represent as a normalized machine number; precision may be lost.

Out[205]=
```
{0.092834, 0.151752}
```

Hypothesentests

Funktionen zur Durchführung von Hypothesentests stehen nach dem Laden des Zusatzpakets

In[206]:=
```
Needs["HypothesisTesting`"]
```

zur Verfügung. Den Gauß-Test für H0 : μ >= 100 bekommen wir so :

In[207]:=
```
MeanTest[data, 100, KnownVariance → 4, SignificanceLevel → 0.05]
```

General: The function MeanTest is now obsolete and has been superseded by LocationTest.

Out[207]=
```
{OneSidedPValue → 0.0409952, Reject null hypothesis at significance level → 0.05}
```

In[208]:=
```
MeanTest[data, 100, TwoSided → True, FullReport → True, SignificanceLevel → 0.05]
```

Out[208]=
```
{FullReport → Mean    TestStat    Distribution
              98.9    -1.59326    StudentTDistribution[9]
 ,
 TwoSidedPValue → 0.1455670918418355,
 Fail to reject null hypothesis at significance level → 0.05}
```

Mathematica führt automatisch einen einseitigen Test durch (je nachdem, ob x > μ0 oder x < μ0 gilt) und gibt den zugehörigen p-Wert aus. Wird die Varianz nicht angegeben, so wird automatisch ein t-Test durchgeführt. Einen zweiseitigen Test können wir durch Angabe der Option **TwoSided → True** erhalten. Mit der Option **FullReport → True** werden zusätzlich Stichproben-Mittelwert, Prüfwert und die verwendete Verteilung ausgegeben.

χ^2-Anpassungstest auf Normalverteilung

Wir beginnen mit den gegeben Füllmengen:

In[209]:=
```
xlist = {102.5, 115.7, 93.8, 102.6, 110., 111.5, 98.6, 96.8, 101.7, 110.9, 103.,
    104.4, 108.4, 104.2, 97.7, 105.5, 92.1, 100.3, 97.9, 105.5, 106.1, 90.3,
    108.9, 96.4, 90.4, 91.5, 94.9, 99.9, 84.9, 102., 101.5, 96.8, 99.9, 104.6,
    92.7, 87.9, 104., 108.6, 94.7, 107.3, 98.6, 96.6, 105.4, 101.4, 104.,
    94.2, 108.3, 106.2, 101.1, 109.1, 94., 95.6, 100.1, 89.5, 101.2, 94.1,
    92., 100.1, 105.5, 105.1, 94.1, 113.1, 101.6, 86., 92.1, 91.5, 98.6,
    90.6, 101.4, 93.6, 88.3, 88.5, 88.6, 95.9, 108.2, 101.2, 101.1, 96.9,
    100.2, 104.7, 96.6, 109.2, 108.5, 108.4, 111.6, 99.2, 90.8, 111.7, 99.7,
    100.4, 96., 95.7, 90.9, 95.5, 106.6, 100.2, 114.1, 101.6, 113.6, 98.5};
n = Length[xlist]
```

Out[210]=
```
100
```

Die gesuchten Schätzwerte sind

In[211]:=
```
{m, s} = {Mean[xlist], (n - 1)/n Sqrt[Variance[xlist]]}
```

Out[211]=
```
{100.135, 6.9834}
```

und die zugehörige Normalverteilung ist

In[212]:=
```
ndist = NormalDistribution[m, s];
```

Nun berechnen wir für gegebene Klassengrenzen alle benötigten Werte:

In[213]:=
```
k = 16;
{min, max} = {Floor[Min[xlist]], Ceiling[Max[xlist]]};
clist = Table[If[j ≤ 0, -∞, If[j ≥ 16, ∞, min + (max - min) j/k]], {j, 0, k}]
```

Out[215]=
```
{-∞, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, ∞}
```

In[216]:=
```
tab = Table[{{clist[[i]], clist[[i + 1]]},
     Count[xlist, y_ /; (clist[[i]] < y ≤ clist[[i + 1]])],
     n (CDF[ndist, clist[[i + 1]]] - CDF[ndist, clist[[i]]])},
    {i, 1, Length[clist] - 1}];
tab // MatrixForm
```

Out[217]//MatrixForm=

{-∞, 86}	2	2.14807
{86, 88}	1	1.96519
{88, 90}	4	3.22158
{90, 92}	8	4.86803
{92, 94}	6	6.78043
{94, 96}	10	8.70526
{96, 98}	8	10.3021
{98, 100}	8	11.2381
{100, 102}	17	11.3001
{102, 104}	5	10.4735
{104, 106}	9	8.94792
{106, 108}	4	7.04651
{108, 110}	10	5.11501
{110, 112}	4	3.42246
{112, 114}	2	2.11081
{114, ∞}	2	2.35487

Zusammenlegung einiger Klassen:

In[218]:=
```
clist = {-∞, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, ∞};
```

In[219]:=
```
tab = Table[{{clist[[i]], clist[[i + 1]]},
     Count[xlist, y_ /; (clist[[i]] < y ≤ clist[[i + 1]])],
     n (CDF[ndist, clist[[i + 1]]] - CDF[ndist, clist[[i]]])},
    {i, 1, Length[clist] - 1}];
tab // MatrixForm
```

Out[220]//MatrixForm=

{-∞, 90}	7	7.33483
{90, 92}	8	4.86803
{92, 94}	6	6.78043
{94, 96}	10	8.70526
{96, 98}	8	10.3021
{98, 100}	8	11.2381
{100, 102}	17	11.3001
{102, 104}	5	10.4735
{104, 106}	9	8.94792
{106, 108}	4	7.04651
{108, 110}	10	5.11501
{110, ∞}	8	7.88814

Daraus folgt der gesuchte Prüfwert und das gesuchte Quantil:

In[221]:=
```
{Sum[tab[[i, 2]]^2 / tab[[i, 3]], {i, 1, Length[clist] - 1}] - n,
 Quantile[ChiSquareDistribution[Length[clist] - 4], 0.95]}
```

Out[221]= {15.4801, 16.919}