Mathematica anhand von Beispielen

Daniel Baumgartner

Fachhochschule für Technik St.Gallen, 2001

Dieses Dokument ist erhältlich auf www.mathematicus.ch

Inhalt

Die folgenden Meldungen sollen ausgeschaltet sein:

```
In[1]:= Off[General::spell1]
        Off[ParametricPlot::ppcom]
        Off[$MaxExtraPrecision::meprec]
```

1 Beispiele

ganze Zahlen

beispiel 1

Die Funktion **FactorInteger** zerlegt eine ganze Zahl in Primfaktoren:

```
In[4]:= FactorInteger[12]
Out[4]= {{2, 2}, {3, 1}}
```

Die Anzahl der verschiedenen Primfaktoren ist gleich der Länge dieser Liste

```
In[5]:= AnzPrimfaktoren[n_] := Length[FactorInteger[n]]
```

Zum Beispiel:

```
In[6]:= AnzPrimfaktoren[17! + 1]
Out[6]= 3
```

beispiel 2

Table erzeugt eine Menge, deren Elemente eine gemeinsame Eigenschaften haben. Zum Beispiel die Menge der ersten 10 Primzahlen:

```
In[7]:= Table[Prime[k], {k, 1, 10}]
Out[7]= {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}
```

beispiel 3

Select wählt mit einer Boolschen Funktion bestimmte Elemente einer Menge aus:

Select[$\{e_1, e_2, \ldots, e_n\}$, f**]** $\longrightarrow$ Alle Elemente für die $f(e_i)$ = **True**

Welche ungeraden Zahlen im Intervall [1001, 1111] haben genau zwei Primfaktoren?

```
In[8]:= Select[Range[1001, 1111], AnzPrimfaktoren[#] == 2 &]
Out[8]= {1003, 1004, 1006, 1007, 1011, 1016, 1017, 1018, 1025, 1027, 1028, 1029,
         1037, 1041, 1042, 1043, 1046, 1047, 1048, 1052, 1053, 1055, 1057,
         1058, 1059, 1067, 1072, 1073, 1075, 1076, 1077, 1079, 1081, 1082,
         1083, 1084, 1088, 1089, 1094, 1096, 1099, 1101, 1107, 1108, 1111}
```

aufgaben

Gegeben sei der Algorithmus **Primzahlen** welcher die Menge der ersten n Primzahlen erzeugt.

```
In[9]:= Primzahlen[n_] := Table[Prime[k], {k, 1, PrimePi[n]}];
```

1. Welche der ersten 100 Primzahlen sind von der Form $4k+1$, welche von der Form $4k+3$?

2. Schreibe einen Algorithmus **PrimzahlenDerForm[4k+x_≤n_]** welcher von den ersten n Primzahlen diejenigen der Form $4k+x$, $(x = 1, 3)$ bestimmt.

beispiel 4

```
In[14]:= QuadratzahlQ[n_] := IntegerQ[Sqrt[n]]
```

prüft, ob eine Quadratzahl vorliegt. Möchte man von einer Menge von Zahlen wissen, welche Quadratzahlen sind, so hat man zwei Möglichkeiten: Mit **Map** die Menge mit der Funktion **QuadratzahlQ[#]&** abbilden,

```
In[15]:= Map[QuadratzahlQ[#] &, {16, 72}]
Out[15]= {True, False}
```

oder dem Algoritmus **QuadratzahlQ** das Attribut **Listable** zuzuordnen:

```
In[16]:= SetAttributes[QuadratzahlQ, Listable]

In[17]:= QuadratzahlQ[{16, 72}]
Out[17]= {True, False}
```

beispiel 5

Ist $n = 117$ eine Summe von 2 Quadraten? Wenn ja, welche? Man muss nur die Quadrate von $k = 1$ bis $k = \sqrt{n/2}$ prüfen. Ist die Differenz $n - k^2$ eine Quadratzahl, so hat man eine Summe von Quadraten:

```
In[18]:= 117 - Table[k^2, {k, Sqrt[117 / 2]}]
Out[18]= {116, 113, 108, 101, 92, 81, 68}
```

81 ist eine Quadratzahl, d.h $117 = 81 + 36$.

Im allgemeinen Fall wählen wir mit **Select** und der Funktion **QuadratzahlQ[#]&** die Quadratzahlen aus. Die Menge A enthält (falls vorhanden) die Quadratzahlen $a^2 = n - k^2$, B die Menge $b^2 = n - a^2$:

```
In[19]:= SummeZweierQuadrate[n_] := Block[{mengeA, mengeB},
           mengeA = Select[n - Table[k^2, {k, Sqrt[n / 2]}], QuadratzahlQ[#] &];
           mengeB = n - mengeA;
           Thread[{mengeA, mengeB}]]

In[20]:= SummeZweierQuadrate[221]
Out[20]= {{196, 25}, {121, 100}}
```

Will man nur wissen, *ob* eine Zahl Summe zweier Quadrate ist:

```
In[21]:= SummeZweierQuadrateQ[n_] := Block[{a},
                a = n - Table[k^2, {k, 1, Sqrt[n]}];
                Apply[Or, QuadratzahlQ[a]]]
```

QuadratzahlQ liefert eine Liste mit den Werten True und False. Apply verknüpft sie hier mit Or. Falls mindestens eine Quadratzahl in der Liste vorkommt, erhält man True, ansonsten False.

```
In[22]:= SummeZweierQuadrateQ[117]
Out[22]= True
```

Noch das Attribut **Listable** zuordnen:

```
In[23]:= SetAttributes[SummeZweierQuadrateQ, Listable]

In[24]:= SummeZweierQuadrateQ[{117, 215}]
Out[24]= {True, False}
```

beispiel 6

Sind alle Primzahlen (≤1000) welche von der Form $4k+1$ sind, Summe zweier Quadrate?

```
In[25]:= Apply[And, SummeZweierQuadrateQ[PrimzahlenDerForm[4 k + 1 ≤ 1000]]]
Out[25]= True
```

Gibt es eine Primzahl (≤1000) von der Form $4k+3$ welche Summe zweier Quadrate ist?

```
In[26]:= Apply[Or, SummeZweierQuadrateQ[PrimzahlenDerForm[4 k + 3 ≤ 1000]]]
Out[26]= False
```

Bemerkung. Man kann allgemein zeigen, dass die Primzahlen der Form $4k+1$ genau diejenigen Primzahlen sind, welche Summe zweier Qudrate sind (Euler).

beispiel 7

Welches sind die *verschiedenen* Primteiler einer Zahl? **FactorInteger** liefert eine Liste von Primzahlen und deren Exponenten

```
In[27]:= FactorInteger[19845]
Out[27]= {{3, 4}, {5, 1}, {7, 2}}
```

Als Matrix interpretiert kann man sie Transponieren...

```
In[28]:= Transpose[%]
Out[28]= {{3, 5, 7}, {4, 1, 2}}
```

und davon das erste Element (bzw. die erste Zeile) nehmen:

```
In[29]:= Primfaktoren[n_] := Transpose[FactorInteger[n]][[1]]

In[30]:= Primfaktoren[19845]
Out[30]= {3, 5, 7}
```

aufgaben

3. Verifiziere: **Map[Range [**#, 72, #**] &, {**5, 7**}]** erzeugt alle durch 5 und 7 teilbare Zahlen (≤72). Von welcher Struktur ist das Resultat?

4. Erzeuge die Menge aller zu 72 a) nicht-teilerfremden b) teilerfremden Zahlen.

5. Schreibe einen Algorithmus, welcher die Menge aller zu n teilerfremden Zahlen erzeugt. Teste ihn mit **EulerPhi**.

beispiel 8

Eine Zahl n heisst *pseudorim* zur Basis b (ggT(b, n) = 1), wenn $b^{n-1} \equiv 1 \bmod n$ und n nicht prim ist. Welches sind die zur Basis 2 pseudoprimen Zahlen bis 1000?

```
In[36]:= Select[Range[1000], PowerMod[2, # - 1, #] == 1 && ! PrimeQ[#] &]
Out[36]= {341, 561, 645}
```

Allgemein prüft die Funktion

```
In[37]:= PseudoprimQ[b_Integer, n_Integer] := PowerMod[b, n - 1, n] == 1 && ! PrimeQ[n];
```

ob eine Zahl n pseudoprim bzgl einer Basis b ist.

aufgaben

6. Zeige: 1105 ist sowohl pseudoprim zur Basis 2 als auch zur Basis 3

7. Zeige: a) 1729 ist pseudoprim bzgl der Basen 2,3,5 b) 29341 ist pseudoprim bzgl der Basen 2,3,5,7

8. Zeige: 29341 ist die kleinste Pseudoprimzahl bzgl der Basen 2,3,5,7

9. Eine Zahl n welche bzgl allen (zu n) teilerfremden Basen pseudoprim ist, heisst Carmichael-Zahl. Zeige: 29341 ist die kleinste Carmichael-Zahl

erstellen einer graphik

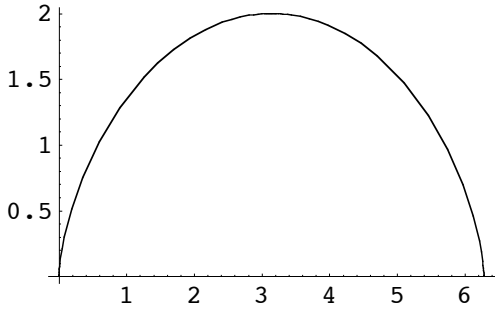
In diesem Kapitel werden die am meisten gebrauchten Elemente zur Erzeugung von Grafiken erläutert.

optionen

Die Parameterdarstellung einer Zykloide lautet

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = r \begin{pmatrix} t - \sin t \\ 1 - \cos t \end{pmatrix}$$

```
In[45]:= ParametricPlot[ {t - Sin[t], -Cos[t] + 1}, {t, 0, 2 π}];
```



Mathematica stellt eine Grafik immer in einem Rechteck gleicher Grösse dar (im Verhältnis des Goldenen Schnitts). Dies hat zur Folge, dass Kurven "verzerrt" sind.

Manchmal möchte man aber eine Kurve in einem anderen Verhältnis darstellen. Dazu ist **AspectRatio** zuständig

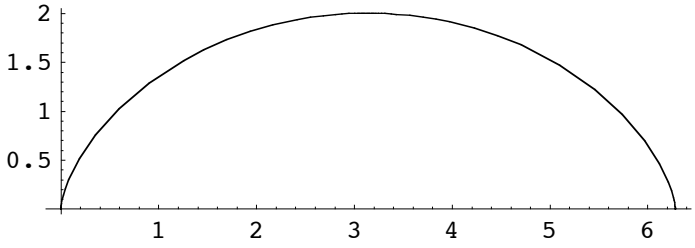
$$\text{AspectRatio} \rightarrow \frac{\text{Ordinate}}{\text{Abszisse}}.$$

Sollen in unserem Beispiel die Einheiten der Ordinate und Abszisse gleich lang sein, so müssen wir also

$$\text{AspectRatio} \rightarrow \frac{2r}{2\pi r} \quad (= \frac{1}{\pi})$$

wählen.

```
In[46]:= ParametricPlot[{t - Sin[t], -Cos[t] + 1}, {t, 0, 2 π}, AspectRatio -> 1/π];
```

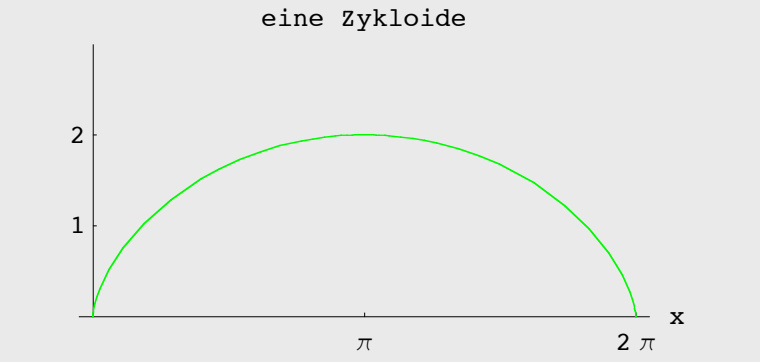


Für des Erstellen von Grafiken stehen noch viele weitere Optionen zur Verfügung.

Beispiele:

Option	Attribute	
PlotStyle	Hue[•] , RGBColor[•,•,•], Dashing[{•,•}]	Farben, gestrichelte Linien
PlotLabel	String	kleine Textzeile
PlotPoints	Integer	Genauigkeit des Graphen
PlotRange	{{*xmin*, *xmax*}, {*ymin*, *ymax*}}	Zeichenbereich
Ticks	None, Automatic, {{x_1, .. x_n}, {y_1, .. y_n}}	Einheiten auf den Achsen
AxesLabel	String	Achsen beschriften

```
In[47]:= ParametricPlot[{t - Sin[t], -Cos[t] + 1}, {t, 0, 2 π},
          PlotStyle -> {RGBColor[0, 1, 0]},
          PlotLabel -> "eine Zykloide",
          Ticks -> {{0, π, 2 π}, {1, 2}},
          AxesLabel -> {"x", None},
          PlotRange -> {Automatic, {0, 3}},
          AspectRatio -> 3 / (2 π),
          Background -> GrayLevel[0.9]];
```



grafikelemente

Rollt ein Kreis auf einer Geraden, so beschreibt ein Peripheriepunkt eine Zykloide. Wir gehen daran, diesen Sachverhalt graphisch darzustellen.

Für Punkte, Kreise und Streckenzüge etc stellt *Mathematica* u.a. die Elemente

Point[$\{x, y\}$**]**	ein Punkt
Circle[$\{x, y\}, r$**]**	ein Kreis mit Zentrum $\{x, y\}$ und Radius r
Line[$\{\{x_1, y_1\}, \ldots \{x_n, y_n\}\}$**]**	der Streckenzug $P(x_1, y_1) - \ldots P(x_n, y_n)$

sowie die Attribute

PointSize[d**]**	Punktgrösse
GreyLevel[x**]**	Graustufe $(0 \le x \le 1)$ (0 : schwarz, 1 : weiss)
Hue, RGBColor	Farben
Dashing[$\{d_1, d_2\}$**]**	gestrichelte Linie
Thickness[d**]**	Liniendicke

zur Verfügung.

Diese Elemente und die Atribute übergibt man **Graphics** in Form einer geordneten Liste.

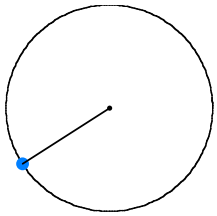
Beispiel: ein Kreis, dessen Zentrum, ein farbiger Peripheriepunkt der Grösse 0.005 und der Radius:

{ Circle[$\{0, 1\}, 1$**], Point[**$\{0, 1\}$**], Hue[**0.6**], PointSize[**0.02**], Point[{**$\varphi - \sin\varphi, 1 - \cos\varphi$**}] ,**
GreyLevel[0**] , Line[**$\{\{\varphi, 1\}, \{\varphi - \sin\varphi, 1 - \cos\varphi\}\}$**] }**

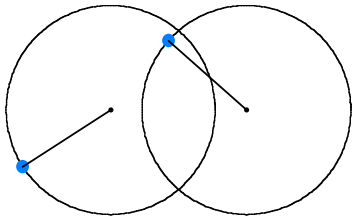
Ein auf einer Geraden abrollender Kreis:

```
In[48]:= kreis[t_] := Graphics[
              { Circle[{t, 1}, 1], Point[{t, 1}],
          Hue[0.6], PointSize[0.02], Point[{t - Sin[t], 1 - Cos[t]}],
       GrayLevel[0] , Line[{{t, 1}, {t - Sin[t], 1 - Cos[t]}}] },
              PlotRange -> {{0, 2 π}, {0, 2}},
              AspectRatio -> 1 / π
              ];
```

In[49]:=
```
Show[kreis[1]];
```



In[50]:=
```
Show[{kreis[1], kreis[2.3]}];
```



mehrere Zeichnungen übereinanderlegen

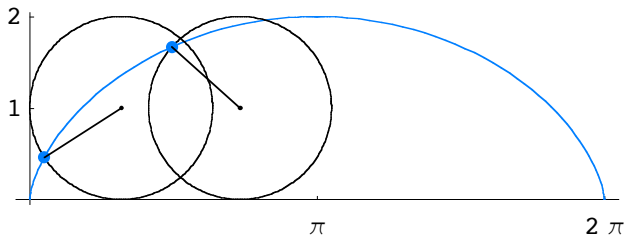
Wir möchten noch die Zykloide hinzufügen. Die Option **DisplayFunction** steuert die Anzeige einer Graphik:

DisplayFunction ⟶ Identity Die Grafik bleibt unsichtbar

DisplayFunction ⟶ $DisplayFunction Die Grafik wird sichtbar

In[51]:=
```
zykloide[] := ParametricPlot[{t - Sin[t], -Cos[t] + 1}, {t, 0, 2 π},
    PlotStyle -> Hue[0.6],
    Ticks -> {{0, π, 2 π}, {1, 2}},
    AspectRatio -> 1 / π,
    PlotPoints -> 100,
    DisplayFunction -> Identity
    ];
```

In[52]:=
```
Show[{zykloide[], kreis[1], kreis[2.3]},
    DisplayFunction -> $DisplayFunction];
```



beispiel einer Animation

ShowAnimation aus dem Paket **Graphics`Animation`** erzeugt aus einer Liste von **-**Graphics**-** -Objekten eine Filmsequenz.

Beispiel: die Zykloide und der Kreis zum Zeitpunkt *t*:

```
In[53]:= zykloide[t_] := ParametricPlot[{φ - Sin[φ], -Cos[φ] + 1}, {φ, 0, 2 π},
           PlotStyle -> Hue[0.6],
           Ticks -> {{0, π, 2 π}, {1, 2}},
           PlotPoints -> 100,
           DisplayFunction -> Identity
           ][[1]];
         kreis[t_] :=
              { Circle[{t, 1}, 1], Point[{t, 1}],
           Hue[0.6], PointSize[0.02], Point[{t - Sin[t], 1 - Cos[t]}] ,
            GrayLevel[0] , Line[{{t, 1}, {t - Sin[t], 1 - Cos[t]}}] };
```

Ein Bild zum Zeitpunkt *t*:

```
In[54]:= bild[t_] := Graphics[Flatten[{zykloide[t], kreis[t]}],
                PlotRange -> {{0, 2 π}, {0, 2}},
                AspectRatio -> 1 / π];
```

Eine Bildsequenz mit 9 Bildern:

```
In[55]:= film = Table[bild[t], {t, 0, 2 π, 2 π / 9}];
```

Man hat zwei Möglichkeiten: entweder man lädt das ganze Paket mit all seinen Prozeduren:

<<Graphics`Animation`

ShowAnimation[film**]**

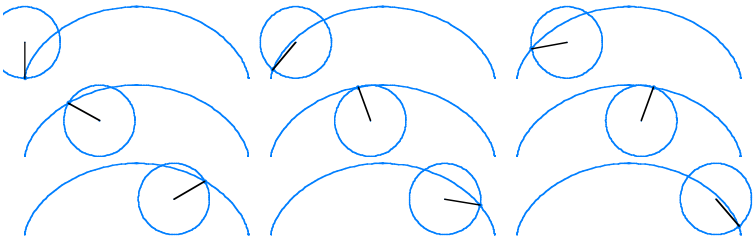
oder man greift direkt mit

Graphics`Animation`ShowAnimation[film**]**

auf den gewünschten Algorithmus zurück. **ShowAnimation** erzeugt eine Liste mit allen Bildern. Danach kann man z.B das erste Bild markieren und mit Doppelklick die Animation in Bewegung bringen. Mit den Tasten "<" und ">" kann das Tempo variiert werden.

Man kann auch die Einzelbilder mit **GraphicsArray** als "Filmstreifen" sichtbar machen:

```
In[56]:= GraphicsArray[Partition[film, 3]] // Show;
```



bemerkungen

(i) Oft möchte man eine Zeichnung im richtigen Verhältnis darstellen. Man kann dies mit der Kombination

PlotRange → $\{\{x_1, x_2\}, \{y_1, y_2\}\}$

AspectRatio → $(y_2 - y_1)/(x_2 - x_1)$

erreichen. Mit **FullOptions** können die Werte von Optionen gelesen werden, z.B.

g = **Graphics[Point[**$\{0, 0\}$**]];**
FullOptions[g , **PlotRange]** ⟶ $\{\{-1.05, 1.05\}, \{-1.05, 1.05\}\}$

Dies können wir nutzen, um einen Algorithmus **graphics** zu schreiben, welcher eine Graphik immer im richtigen Verhältnis ausgibt. Man erstellt die Graphik mit

Graphics[x , **PlotRange → All]**

und liest den Zeichenbereich mit **FullOptions** ein.Dann wird das Verhältnis des y- und x-Bereichs berechnet:

```
In[57]:= graphics[x__] := Block[{xMax, xMin, yMax, yMin},
           g = Graphics[{x}, PlotRange -> All];
           {{xMin, xMax}, {yMin, yMax}} = FullOptions[g, PlotRange];
           Graphics[{x},
               PlotRange -> All,
               AspectRatio -> (yMax - yMin) / (xMax - xMin)]];
```

Wir werden gelegentlich davon gebrauch machen.

(ii) Verschiedene Grafiken g_1, ..., g_n mit der Option **DisplayFunction→ Identity** können mit

Show[$\{g_1, ..., g_n\}$, **DisplayFunction →**$DisplayFunction]

in einem einzigen Diagramm angezeigt werden. Für Grafiken welche mit **Plot**, **ParametricPlot** etc erstellt werden, bietet das Paket **Graphics`Graphics** die Funktion **DisplayTogether**:

DisplayTogether[g_1, ..., g_n **]**

Das Paket einlesen:

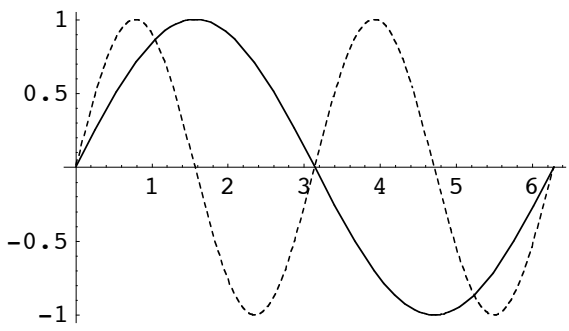
```
In[58]:= << Graphics`Graphics`
```

(iii) **Plot[**$\{f(x), g(x)\}, \{x, x_1, x_2\}$**]** zeichnet die Graphen der Funktionen $f(x)$ und $g(x)$ ins gleiche Diagramm. Mit der Option **PlotStyle** können den Graphen von f und g verschiedene Attribute zugewiesen werden:

PlotStyle ⟶{{Attribute für f}, {Attribute für g}}

Beispiel:

```
In[59]:= Plot[{Sin[x], Sin[2 x]}, {x, 0, 2 π},
           PlotStyle -> {{}, {Dashing[{0.01, 0.01}]}}];
```



(iv) Dreidimensionale Graphiken erstellt man analog zu den zweidimensionalen. Man hat dort **Plot3D**, **Graphics3D** etc.

aufgaben

1. Es sei π(x) die Anzahl Primzahlen $\leq x$. Bekanntlich gilt $\pi(x) \sim \frac{x}{\log(x)}$. Erstelle eine Grafik welche dies veranschaulicht. Verwende dazu **PrimePi**, **Plot** und **ListPlot**.

2. Zeichne den Kegelschnitt $x^2 - 2\,y^2 + x\,y - 3\,x = 0$ und seine Asymptoten. Verwende dazu **ImplicitPlot** aus der Packung Graphics`ImplicitPlot`.

berechnungszeiten

die fibonaccifolge

Der folgende Algorithmus gibt die Zeit aus für die er braucht, um die *n*-te Fibonaccizahl zu berechnen:

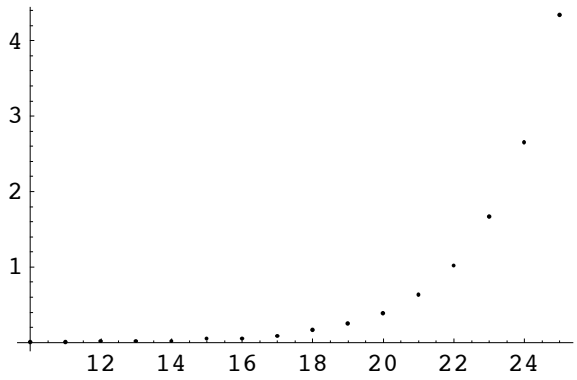
```
In[66]:= fib[n_] := Module[{f},
              f[1] = f[2] = 1;
              f[k_] := f[k - 1] + f[k - 2];
              First[Timing[f[n]]][[1]]
          ];

In[67]:= fib[15]
Out[67]= 0.0333333
```

Wie steigt der Zeitaufwand in Abhängigkeit von *n*?. Wir erstellen eine Liste von Rechenzeiten für $n = 10, \ldots 25$:

```
In[68]:= berechnungszeit = Table[{n, fib[n]}, {n, 10, 25}]
Out[68]= {{10, 0.}, {11, 0.}, {12, 0.0166667}, {13, 0.0166667},
          {14, 0.0166667}, {15, 0.05}, {16, 0.05}, {17, 0.0833333},
          {18, 0.166667}, {19, 0.25}, {20, 0.383333}, {21, 0.633333},
          {22, 1.01667}, {23, 1.66667}, {24, 2.65}, {25, 4.33333}}
```

Mit **ListPlot[...]** den Graphen zeichnen:

```
In[69]:= ListPlot[berechnungszeit];
```



Es könnte eine Exponentialfunktion sein. Wenn ja, welche? **Fit** erzeugt eine Regressionskurve:

Fit[*Daten, {Grundfumktionen}, unabhänige Variable***]**

erzeugt aufgrund der *Daten* eine Regressionskurve, welche eine Linearkombination der *Grundfunktionen* ist.

Speziell lautet die lineare Regression:

Fit[*Daten,* $\{1, x\}, x$**]**.

In unserem Fall nehmen wir eine lineare Regression an den logarithmierten Zeitwerten vor. Die Zeitwerte 0 Sekunden schliessen wir mit **DeleteCases** aus dem Experiment aus. **$MinNumber** ist kleinste Zahl grösser 0, welche *Mathematica* darstellen kann.

```
In[70]:= berechnungszeit =
           DeleteCases[berechnungszeit, {_, x_Real /; x <= $MinNumber}]
Out[70]= {{12, 0.0166667}, {13, 0.0166667}, {14, 0.0166667}, {15, 0.05},
          {16, 0.05}, {17, 0.0833333}, {18, 0.166667}, {19, 0.25}, {20, 0.383333},
          {21, 0.633333}, {22, 1.01667}, {23, 1.66667}, {24, 2.65}, {25, 4.33333}}
```

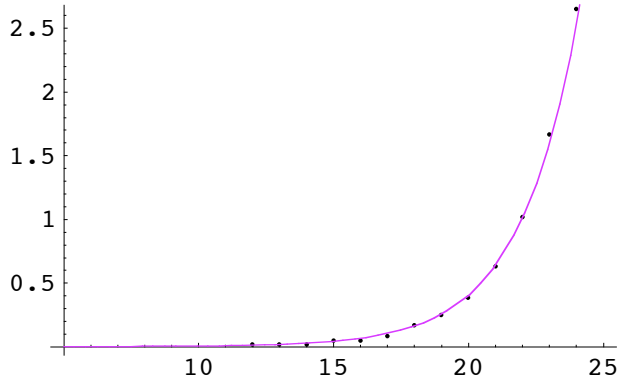
Die logarithmierten Werte:

```
In[71]:= List[#[[1]], Log[#[[2]]]] & /@ berechnungszeit
Out[71]= {{12, -4.09434}, {13, -4.09434}, {14, -4.09434},
          {15, -2.99573}, {16, -2.99573}, {17, -2.48491}, {18, -1.79176},
          {19, -1.38629}, {20, -0.95885}, {21, -0.456758},
          {22, 0.0165293}, {23, 0.510826}, {24, 0.97456}, {25, 1.46634}}
```

Die Regressionskurve:

```
In[72]:= h = Exp[Fit[%, {1, x}, x]]
Out[72]= E^(-10.0665+0.457709 x)
```

```
In[73]:= DisplayTogether[
             ListPlot[berechnungszeit],
             Plot[h, {x, 5, 25}, PlotStyle -> Hue[0.8]]];
```



bemerkung

Die Berechnungszeit für f_{50} dauert also etwa $e^{13} = 4.42\ 10^5$ s = 123 h. Definiert man f[n_] "mit Gedächtnis", so reduziert sich die Berechnungszeit wesentlich (allerdings steigt der Speicheraufwand):

```
In[74]:= f[1] = f[2] = 1;
         f[n_] := f[n] = f[n - 1] + f[n - 2];
```

Bereits berechnete Werte sind jederzeit abrufbar.

```
In[75]:= f[50] // Timing
Out[75]= {0. Second, 12586269025}
```

aufgaben

1.Wie hängt die Berechnungzeit von n ab?

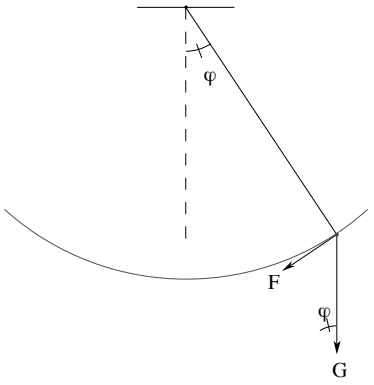
a) Determinante einer $n \times n$-Matrix

b) Sortieren einer n-elementigen Liste

differentialgleichungen

das pendel

Die Differentialgleichung des starren Pendels der Länge 1:



$$m\,\varphi'' = -m\,g\,\sin(\varphi)$$

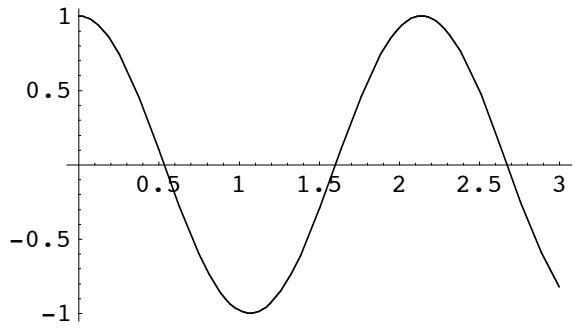
dh. $\varphi'' + g\,\sin(\varphi) = 0$

Die Differentialgleichung in Abhängigkeit der Anfangsbedingung $\varphi(0) = \varphi_0$:

```
In[88]:= diffgl[φ0_] := {
                          D[φ[t], {t, 2}] + 9.81 Sin[φ[t]] == 0,
                 φ'[0] == 0, φ[0] == φ0};
```

Ein Pendel mit Auslenkung $\varphi_0 = 1$:

```
In[89]:= pendel[1] = φ /. NDSolve[diffgl[1], φ, {t, 0, 3}][[1]]
         Plot[pendel[1][t], {t, 0, 3}];
Out[89]= InterpolatingFunction[{{0., 3.}}, <>]
```

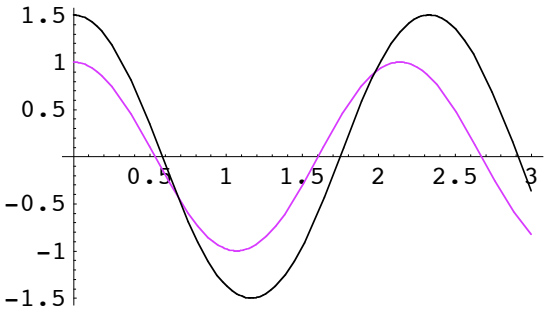


Die Schwingungsdauer:

```
In[91]:= FindRoot[pendel[1][t] == 1, {t, 2.5}]
Out[91]= {t → 2.14022}
```

Die Schwingungsdauer ist abhängig von der Auslenkung:

```
In[92]:= pendel[1.5] = φ /. NDSolve[diffgl[1.5], φ, {t, 0, 3}][[1]];
         Plot[{pendel[1][t], pendel[1.5][t]}, {t, 0, 3},
             PlotStyle -> {{Hue[0.8]}, {}}];
```



der harmonische oszillator

DSolve löst eine Differentialgleichung exakt.

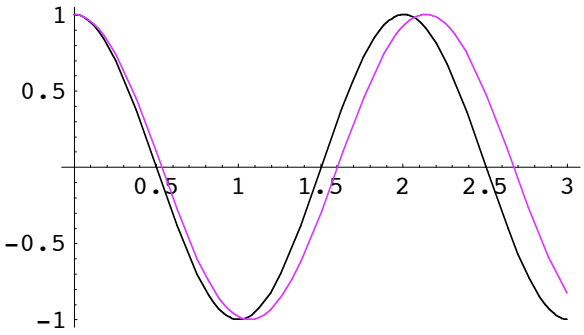
```
In[93]:= DSolve[{x''[t] + ω^2 x[t] == 0}, x, t]
Out[93]= {{x → (C[2] Cos[ω #1] + C[1] Sin[ω #1] &)}}
```

Die Lösung wird als reine Funktion ausgegeben. C[1] und C[2] sind Integrationskonstanten. Sie sind bestimmt, wenn Anfangsbedingungen gegeben sind:

```
In[94]:= s = x /. DSolve[{x''[t] + ω^2 x[t] == 0, x[0] == 1, x'[0] == 0}, x, t] // First
Out[94]= Cos[ω #1] &
```

Ein vergleich mit dem starren Pendel:

```
In[95]:= DisplayTogether[
             Plot[s[t] /. ω -> Sqrt[9.81], {t, 0, 3}],
             Plot[pendel[1][t], {t, 0, 3},
                 PlotStyle -> Hue[0.8]]];
```



die planetenbewegung

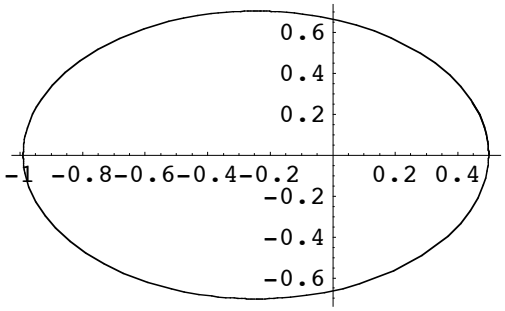
Wir setzen $\gamma M = 1$ (γ:Gravitationskonstante, M: Sonnenmasse). Der Planet habe im Punkt $\{x_0 \mid y_0\} = \{0.5 \mid 0\}$ die Anfangsgeschwindigkeit $\{v_x , v_y\} = \{0, 1.63\}$. Die Differentialgleichung:

```
In[96]:= diffgl = {
            x''[t] == - x[t] / (x[t]^2 + y[t]^2)^(3/2),
            y''[t] == - y[t] / (x[t]^2 + y[t]^2)^(3/2),
              x[0] == 0.5, y[0] == 0,
              x'[0] == 0, y'[0] == 1.63
                          };
```

```
In[97]:= {x, y} = {x, y} /. NDSolve[diffgl, {x, y}, {t, 0, 4.2}][[1]]
Out[97]= {InterpolatingFunction[{{0., 4.2}}, <>],
          InterpolatingFunction[{{0., 4.2}}, <>]}
```

Da es sich um eine zweidimensionale Bewegung handelt, nehmen wir **ParametricPlot**:

```
In[98]:= ParametricPlot[{x[t], y[t]}, {t, 0, 4.2}];
```



Wie lange dauert ein Umlauf?

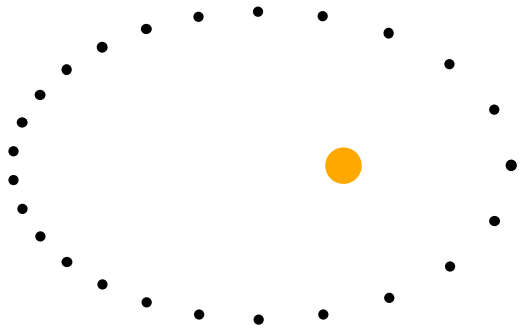
```
In[99]:= T = t /. FindRoot[x[t] == 0.5, {t, 4.1}]
Out[99]= 4.03801
```

Der Planet zum Zeitpunkt t und die Sonne:

```
In[100]:= planet[t_] := {PointSize[0.02], GrayLevel[0], Point[{x[t], y[t]}]};
          sonne = {PointSize[0.07], Hue[0.1], Point[{0, 0}]};
```

Die Position des Planeten in gleichen Zeitabständen:

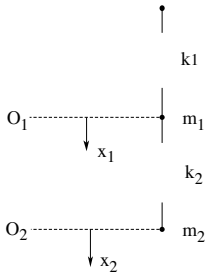
In[101]:=
```
Show[Graphics[
        {Table[planet[t], {t, 0, T, T / 25}], sonne} // Flatten
        ]];
```



Man sieht deutlich, dass sich der Planet in Sonnennähe schneller bewegt.

ein federpendel

Zwei Massen m_1, m_2, sind an Federn aufgehängt



Die Bewegung wird durch das Gleichungssystem

$$m_1\, x_1'' = -k_1\, x_1 + k_2 (x_2 - x_1)$$
$$m_2\, x_2'' = -k_2 (x_2 - x_1)$$

beschrieben.

Wir nehmen $m_1 = 20$, $m_2 = 5$ und die Federkonstanten $k_1 = 0.5$, $k_2 = 1$.

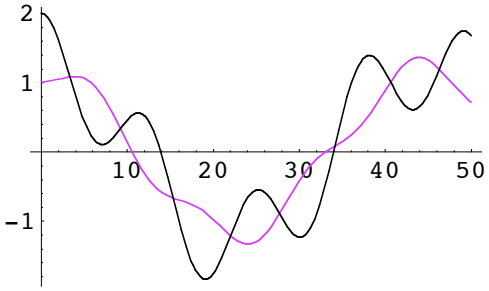
Die Differentialgleichungen und Anfangsbedingungen…

In[102]:=
```
federpendel = {
        20 x1''[t] == - 0.5 x1[t] + (x2[t] - x1[t]),
        5 x2''[t] == - (x2[t] - x1[t]),
        x1[0] == 1, x1'[0] == 0, x2[0] == 2, x2'[0] == 0
        };
```

In[103]:=
```
{x1, x2} = {x1, x2} /. NDSolve[federpendel, {x1, x2}, {t, 0, 50}][[1]]
```

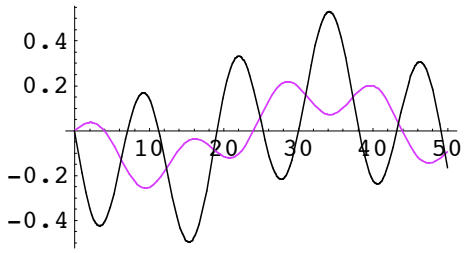
Out[103]=
```
{InterpolatingFunction[{{0., 50.}}, <>],
 InterpolatingFunction[{{0., 50.}}, <>]}
```

```
In[104]:= Plot[{x1[t], x2[t]}, {t, 0, 50},
            PlotStyle -> {{Hue[0.8]}, {}}];
```



das Geschwindigkeit-Zeit Diagramm

```
In[105]:= Plot[{x1'[t], x2'[t]}, {t, 0, 50},
            PlotStyle -> {{Hue[0.8]}, {}}];
```



aufgaben

1. Erstelle eine Bildsequenz der Planetenbewegung und simuliere sie mit **ShowAnimation**.

2. Löse die Differentialgleichung des schiefen Wurfs:

$$x''(t) = -r\,x'(t)^2 \;,\; y''(t) = \begin{cases} -g - r\,y'(t)^2 & \text{im aufsteigenden Teil der Bahn} \\ -g + r\,y'(t)^2 & \text{im absteigenden Teil der Bahn} \end{cases}$$

und stelle die Lösungsfunktion grafisch dar.

3. Die Differentialgleichung

$$x''(t) - g + \frac{k(t)}{m}\,x'(t)^2$$

beschreibt den freien Fall mit Luftwiderstand. Dabei ist

$$k(t) = 0.5\,\rho(t)\,q(t)\,c_w(t)$$
$$\rho(t) = 1.25\,e^{-0.00014\,x(t)}$$

(ρ:Luftdichte, q:Querschnittsfläche, c_w:Luftwiderstandsbeiwert).

Simuliere die Bewegung eines Fallschirmspringers mit den Werten
$m = 70, c_w = \begin{Bmatrix} 0.25 \\ 0.9 \end{Bmatrix}, q = \begin{Bmatrix} 3 \\ 25 \end{Bmatrix} \begin{Bmatrix} \text{vor} \\ \text{nach} \end{Bmatrix}$ dem Öffnen des Fallschirms.

```
In[106]:= Clear[x1, x2, diffgl, x, y, T, s, pendel]
```

wahrscheinlichkeitsrechnung

ein kartenspiel

A hat die Karte mit den Seiten 7 und 2. *B* hat die Karte mit den Seiten 1 und 8. Beide legen ihre Karte gleichzeitig auf den Tisch, wobei sie selbst wählen dürfen, welche Seite der Karte zu sehen ist. *B* gewinnt, wenn die Farben übereinstimmen, *A* wenn sie nicht übereinstimmen. In jeden Fall ist der Gewinn in Fr gleich dem Wert der Karte des Gewinners.

Welche Karten würden Sie nehmen? Mit welcher Strategie spielen Sie?

A spiele die 7 mit Wahrscheinlichkeit p, *B* die 1 mit q.

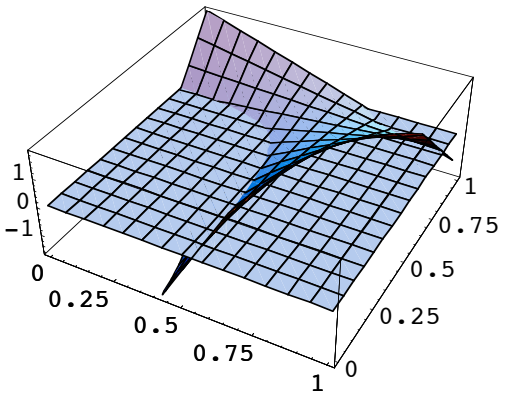
Erwartungswert für *A*:

```
In[119]:= 7 p (1 - q) + 2 (1 - p) q - p q - 8 (1 - p) (1 - q) // Expand
Out[119]= -8 + 15 p + 10 q - 18 p q
```

```
In[120]:= ewA[p_, q_] := -8 + 15 p + 10 q - 18 p q
```

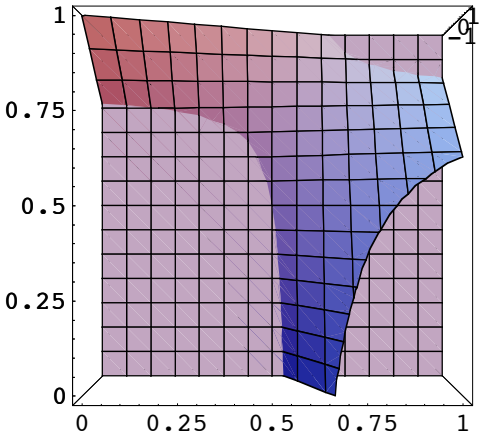
Für welche Werte von p ist er positiv?

```
In[121]:= DisplayTogether[
            Plot3D[ewA[p, q], {p, 0, 1}, {q, 0, 1}],
            Plot3D[0, {x, 0, 1}, {y, 0, 1}]];
```



Besser von oben betrachten:

```
In[122]:= DisplayTogether[
            Plot3D[ewA[p, q], {p, 0, 1}, {q, 0, 1}, ViewPoint -> {0, 0, 2}],
            Plot3D[0, {x, 0, 1}, {y, 0, 1}]];
```



Es gibt auch einen Bereich, in welchem für jedes q der Erwartungswert für A grösser 0 ist. Man sieht die Hyperbel $-8 + 15\,x + 10\,y - 18\,x\,y = 0$. Sie ist der Graph der Funktion

```
In[123]:= y[x_] = y /. Solve[ewA[x, y] == 0, y] // First
```

$$\text{Out[123]=}\quad -\frac{8 - 15\,x}{2\,(-5 + 9\,x)}$$

Sie hat einen Pol bei $x = \frac{5}{9}$. Dort hat B keine Wahl.

```
In[124]:= ewA[5 / 9, p]
```

$$\text{Out[124]=}\quad \frac{1}{3}$$

A gewinnt bei 3 Spielen durchschnittlich 3Fr.

das spiel simulieren

Ein Zufallsgenerator für A und B (B bekommt eine Münze):

```
In[125]:= karteA := If[Random[] < 5 / 9, 7, 2];
          karteB := If[Random[Integer] == 0, 8, 1];
```

Ein paar Spiele:

```
In[126]:= spiel = Table[{karteA, karteB}, {j, 1, 8000}];
```

```
In[127]:= gewinnA = 2 Count[spiel, {2, 1}] + 7 Count[spiel, {7, 8}];
          gewinnB = 8 Count[spiel, {2, 8}] + Count[spiel, {7, 1}];
          (gewinnA - gewinnB) / 8000.
Out[127]= 0.35325
```

nützliche algorithmen

Oft ist es nützlich, einen Zufallsgenerator zur Hand zu haben, welcher die Ereignisse 0, 1, 2, ..., n mit verschiedenen Wahrscheinlichkeiten eintreten lässt.

Zum Beispiel sollen die Zahlen 0, 1, 2 mit den Wahrscheinlichkeiten $p_0 = \frac{1}{4}$, $p_1 = \frac{1}{2}$, $p_2 = \frac{1}{4}$ eintreten.

Idee: Man bestimmt mit **Random[ ]** eine reelle Zufallszahl zwischen 0 und 1 und prüft, in welches der Teilintervalle $I_1 = (0, \frac{1}{4})$, $I_2 = (\frac{1}{4}, \frac{3}{4})$, $I_3 = (\frac{3}{4}, 1)$ die Zahl gefallen ist.

Wie man dies algorithmisch mit *Mathematica* bewerkstelligen kann, soll das folgende Beispiel aufzeigen. Wir nehmen an **Random[ ]** hätte die Zufallszahl 0.678 erzeugt. Sie liegt im Intervall I_2 . Den Index 2 finden man 0.678 zur Liste $\{\frac{1}{4}, \frac{3}{4}\}$ hinzufügt und sortiert. Die Position der Zufallszahl 0.678 in dieser Liste ist dann gleich dem Index 2.

Union vereinigt und sortiert zwei Mengen:

Union[$\{\frac{1}{4}, \frac{3}{4}\}$,{0.678}**]** $\longrightarrow \{\frac{1}{4}, 0.678, \frac{3}{4}\}$

Die Position erhalten wir mit dem Kriterium **_Real**:

Position[$\{\frac{1}{4}, 0.678, \frac{3}{4}\}$,**_Real] ⟶**{{2}}.

Zum Algorithmus im Allgemeien: Mit der rekursive Folge

$$x_0 = p_0$$
$$x_k = x_{k-1} + p_k$$

und **Array** wird die Liste der Stützpunkte erstellt. Von der Position muss man noch 1 subtrahieren, wenn man Zufallszahlen von 0 bis $n - 1$ will.

Es ist sinnvoll, diesen Algorithmus auch **Random** zu nennen. Da *Mathematica*-Funktionen "geschützt" sind, müssen wir zuerst den Schutz aufheben

```
In[128]:= Unprotect[Random];
          Random[p_List] := Block[{x, w, m},
                  x[1] = p[[1]];
                  x[k_] := x[k - 1] + p[[k]];
                  m = Array[x, Length[p]];
                  w = Union[m, {Random[]}];
                  Position[w, _Real][[1, 1]] - 1
                  ];
```

Sind die Wahrscheinlichkeiten vom Typ **_Real**, so kann man sie mit **Rationalize** rational machen und die gleiche Methode anwenden.

```
In[129]:= Random[p_] := Random[Rationalize[p, 0]] /; Select[p, _Real] =!= {}
          Protect[Random];
```

Ein Zufallsversuch mit diesem Generator:

```
In[131]:= wurf = Table[Random[{1 / 4, 1 / 2}], {k, 1, 7000}];
          wurf // Shallow
Out[131]//Shallow=
          {2, 1, 1, 1, 1, 2, 1, 1, 1, 1, <<6990>>}
```

Für das Auszählen nehmen wir **Count**. **Count[***folge , x***]** zählt, wieviel mal das Element x in der f*olge* vorkommt:

```
In[132]:= Count[wurf, #] & /@ {0, 1, 2}
Out[132]= {1754, 3544, 1702}
```

die relative Häufigkeit:

```
In[133]:= % / 7000 // N
Out[133]= {0.250571, 0.506286, 0.243143}
```

Allgemein:

```
In[134]:= Häuf[folge_, elemente_] := Count[folge, #] & /@ elemente;
```

Soll der Algorithmus gleich selber feststellen soll, welche Elemente in der Folge vorkommen: bildet man einfach die Menge **Union[***folge***]** ab.

```
In[135]:= Häuf[folge_] := Count[folge, #] & /@ Union[folge];
```

Die relative Häufigkeit:

```
In[136]:= RelHäuf[folge_] := Häuf[folge] / Length[folge] // N
          RelHäuf[folge_, elemente_] := Häuf[folge, elemente] / Length[folge] // N

In[138]:= RelHäuf[wurf]
Out[138]= {0.250571, 0.506286, 0.243143}
```

aufgabe

1. Löse das Kartenspiel mit **ContourPlot** und **ImplicitPlot** (ImplicitPlot liegt in der Packung Graphics`ImplicitPlot`).

vollständige induktion

der algorithmus

Es liege eine Aufgabe von folgendem Typ vor: Gegeben ist eine rekursive Folge (a_n) und eine Vermutung der expliziten Form für a_n. Beweise die Vermutung mit der Methode der vollständigen Induktion.

Wir gehen daran, einen Algorithmus zu schreiben, welcher eine solche Aufgabe löst.

Die geometrische Folge soll zum Beispiel so eingegeben werden können:

BeweisDurchInduktion[{$g[n] == g[n-1] + a\,q^n$ **,**$g[0] == a$**}** **,** $g[n] == a\,\frac{1-q^{n+1}}{1-q}$ **]**

Die Argumente sind lauter Gleichungen: die rekursive Form, die Anfangswerte und die Vermutung. Dies führt zu folgendem Muster:

```
BeweisDurchInduktion[{x_Equal, y__Equal}, z_]:=
    Module[{lokale Variablen},
        Algorithmus]
```

Bei rekursiven Folgen welche auf mehrere Glieder zurückgreifen, hat man mehrere Anfangswerte. Deshalb wählen wir für *y* das Muster __ (mindestens ein Element). *y* enthält also immer *alle* Anfangswerte. Das Muster **_Equal** wäre im Grunde genommen nicht nötig. Wir können aber so sicher sein, dass der Algorithmus nur aufgerufen wird, wenn Gleichungen vorhanden sind.

Algorithmisch gedacht, verläuft der Induktionsschritt wie folgt (zur Illustration ziehen wir die geometrische Folge *g* her). Wir interpretieren die Vermutung als reine Funktion (**Function**):

$$f(\#) = a\,\frac{1-q^{\#+1}}{1-q}$$

und substituieren in der rekursiven Form g durch diese Funktion:

$$f(n) == f(n-1) + a\,q^n$$

$$a\,\frac{1-q^{n+1}}{1-q} == a\,\frac{1-q^n}{1-q} + a\,q^n$$

Jetzt muss noch auf Gleichheit getestet werden. Dies tut dann **Simplify** oder **Factor**.

Der Algorithmus in *Mathematica*. In einem ersten Schritt muss festgestellt werden, welches die Folgenvariable und die Induktionsvariable sind (in unserem Fall *g* und *n*). Dazu gibt die linke Seite der Vermutung (das Argument *z*) Auskunft. Der Kopf ist *g*, und das erste Element ist *n* (lokal jetzt *a* und *k*):

a=**Head[lhs[***z***]];**

k=**lhs[***z***][[1]];**

Dabei ist **lhs** (**rhs**) die linke (rechte) Seite einer Gleichung. Die explizite und die rekursive Form:

explizit=rhs[z];

rekursiveForm=rhs[x];

Die Funktion *f* erhalten wir, wenn wir in der expliziten Form die Induktionsvariable durch den Platzhalter **#** ersetzen:

f=**explizit /.** $k \longrightarrow$ **#**

Die Induktionsannahme geht aus der rekursiven Form hervor, wenn die Folgenvariable *a* durch die Funktion *f* substituiert wird:

InduktionsAnnahme = rekursiveForm /. $a \longrightarrow f$

Der Induktionsschritt:

InduktionsSchritt = InduktionsAnnahme - rekursiveForm = = 0

Mit **Simplify** kann jetzt noch vereinfachet werden.

Noch die Verankerung. *y* enthält alle Anfangswerte. Auch dort substituieren wir die Folgenvariable durch die Funktion *f*.

$$\{y\} /. a \longrightarrow f$$

Dies liefert eine Liste mit den Werten **True** und **False**. Die Verankerung ist wahr, wenn eine Liste von **True** vorliegt.d.h

Verankerung= Apply[And , $\{y\} /. a \longrightarrow f$]

Der Induktionsbeweis steht oder fällt dann mit

And[Verankerung , InduktionsSchritt]

Hier der vollständige Algorithmus:

```
In[142]:= BeweisDurchInduktion[{x_Equal, y__Equal}, z_] :=
            Module[
                {a, k, expliziteForm, rekursiveForm, f,
                    InduktionsAnnahme, InduktionsSchritt},
                lhs[h_] := h[[1]];
                rhs[h_] := h[[2]];

                a = Head[lhs[z]];
                k = lhs[z][[1]];

                expliziteForm = rhs[z];
                rekursiveForm = rhs[x];

                f = Function[expliziteForm /. k → #];

                InduktionsAnnahme = rekursiveForm /. a → f;
                InduktionsSchritt =
            InduktionsAnnahme - expliziteForm == 0 // Simplify;

                Verankerung = And @@ (Simplify[#] & /@ ({y} /. a → f));

                InduktionsSchritt && Verankerung
                ];
```

Beispiel; die geometrische Folge und die Fibonaccifolge:

```
In[143]:= BeweisDurchInduktion[
              {g[n] == g[n - 1] + a q^n, g[0] == a}, g[n] == a (1 - q^(n+1)) / (1 - q)]
Out[143]= True

In[144]:= fibonacci = {f[n] == f[n - 1] + f[n - 2], f[1] == 1, f[2] == 1};

In[145]:= BeweisDurchInduktion[
              fibonacci, f[n] == ((1/2 (1 + √5))^n - (1/2 (1 - √5))^n) / √5]
Out[145]= True
```

beispiel

(a_n) sei eine arithmetische Folge. Zeige

$$\frac{1}{a_1\,a_2} + \frac{1}{a_2\,a_3} + \ldots + \frac{1}{a_{n-1}\,a_n} = \frac{n-1}{a_1\,a_n}$$

```
In[146]:= BeweisDurchInduktion[
            {s[n] == s[n - 1] + 1/(a[n - 1] a[n]), s[2] == 1/(a[1] a[2])}, s[n] == (n - 1)/(a[1] a[n])]
```

$$\text{Out[146]=}\quad \frac{a[1] - (-1+n)\,a[-1+n] + (-2+n)\,a[n]}{a[1]\,a[-1+n]\,a[n]} == 0$$

Wir haben hier noch die Nebenbedingung $a_n = a_1 + (n-1)\,d$.

```
In[147]:= % /. a[n_] -> a[1] + (n - 1) d // Simplify
Out[147]= True
```

gegenbeispiele

Wir brauchen aber noch zwei Gegenbeispiele. Eines in welchem der Induktionsschritt und eines in welchem die Verankerung misslingt. Wir wollen auch wissen, welcher Schritt nicht gelungen ist. Dazu machen wir von **Message** Gebrauch. Wir fügen noch die Zeilen

If[! Verankerung, Message[Induktion::"verank"]];

If[! InduktionsSchritt, Message[Induktion::"schritt"],

_ , Message[Induktion::"!entscheidb"]]

hinzu. Die Messages:

```
In[148]:= Induktionsbeweis::"verank"
                = "Die Verankerung misslingt";
          Induktionsbeweis::"schritt" = "Der Induktionsschritt misslingt";
          Induktionsbeweis::"!entscheidb" =
            "Der Induktionsschritt konnte nicht zu Ende geführt werden";
```

```
In[149]:= BeweisDurchInduktion[{x_Equal, y__Equal}, z_] :=
            Module[
                {a, k, expliziteForm, rekursiveForm, f,
                    InduktionsAnnahme, InduktionsSchritt},
                lhs[h_] := h[[1]];
                rhs[h_] := h[[2]];
                a = Head[lhs[z]];
                k = lhs[z][[1]];
                expliziteForm = rhs[z];
                rekursiveForm = rhs[x];
                f = Function[expliziteForm /. k → #];
                InduktionsAnnahme = rekursiveForm /. a → f;
                InduktionsSchritt =
            InduktionsAnnahme - expliziteForm == 0 // Simplify;
                Verankerung = And @@ (Simplify[#] & /@ ({y} /. a → f));

                If[! Verankerung, Message[Induktionsbeweis::"verank"]];
                If[! InduktionsSchritt, Message[Induktionsbeweis::"schritt"],
            _, Message[Induktionsbeweis::"!entscheidb"]];

                InduktionsSchritt && Verankerung
                ];
```

Ein Beispiel:

```
In[150]:= BeweisDurchInduktion[
              {s[n] == s[n - 1] + n, s[1] == 1, s[2] == 3}, s[n] == n^3/2 - 5 n^2/2 + 6 n - 3]

Induktionsbeweis::!entscheidb :
 Der Induktionsschritt konnte nicht zu Ende geführt werden

Out[150]= -3/2 (6 - 5 n + n^2) == 0

In[151]:= BeweisDurchInduktion[
             {s[n] == s[n - 1] + n, s[1] == 1, s[2] == 3}, s[n] == 1/2 (n^2 + n + 6)]

Induktionsbeweis::verank : Die Verankerung misslingt

Out[151]= False

In[152]:= BeweisDurchInduktion[{s[n] == s[n - 1] + n, s[1] == 1}, s[n] == 1/2 n (n + 1)]

Out[152]= True
```

aufgaben

1. Es sei $\varphi = \frac{\sqrt{5}-1}{2}$. Zeige:

   a) $g_n = (\varphi + 1)^n$ kann rekursiv durch $g_n = g_{n-1} + g_{n-2}$ berechnet werden.

   b) $h_n = (-\varphi)^n$ kann rekursiv durch $h_n = h_{n-1} + h_{n-2}$ berechnet werden.

Folgere daraus die explizite Form der Fibonacciolge.

2. Schreibe einen Algorithmus **ntesGlied[**{*x_* == *y_* , *z__*Equal} , *n_*Integer**]** welcher das *n*-te Glied einer rekursiven Gleichung *x* == *y* mit den Anfangswerten *z* berechnet.

3. Erweitere den Algorithmus **BeweisDurchInduktion** so, dass sich die Verankerung nicht auf die Anfangswerte beziehen muss, sondern bei einem beliebigen $n_0 \geq 1$ beginnen kann.

noch einen Schritt weiter

Zeige: Für die Fibonaccizahlen f_n gilt

$$\frac{1}{f_n} \leq (\frac{2}{3})^n \qquad n \geq 11$$

Unser Algorithmus soll auch solche Aufgaben bewältigen. Der Einfachheit halber soll die Vermutung aber in der Form $f[n] \geq (\frac{3}{2})^n$ eingegeben werden.

Head gibt uns Auskunft über die Relation. In unserem Beispiel wäre

Head[*z*] ⟶ GreaterEqual

Wir fügen noch die lokale Variable *relation* hinzu und setzen

relation=Head[*z*]

Der verallgemeinerte Induktionsschritt lautet jetzt

InduktionsSchritt = relation[*InduktionsAnnahme - expliziteForm* **, 0]**

```
In[160]:= BeweisDurchInduktion[{x_Equal, y__Equal}, z_, w_GreaterEqual] :=
           Module[
               {a = Head[lhs[z]], k = lhs[w], k0 = rhs[w],
                   expliziteForm, rekursiveForm, f, relation,
                   InduktionsAnnahme, InduktionsSchritt},
               lhs[h_] := h[[1]];
               rhs[h_] := h[[2]];
               expliziteForm = rhs[z];
               rekursiveForm = rhs[x];
               relation = Head[z];
               f = Function[expliziteForm /. k → #];
               InduktionsAnnahme = rekursiveForm /. a → f;
               InduktionsSchritt =
           relation[InduktionsAnnahme - expliziteForm, 0] // Simplify;

               Verankerung = relation[ntesGlied[{x, y}, k0], f[k0]] // Simplify;

               If[! Verankerung, Message[Induktionsbeweis::"verank"]];
               If[! InduktionsSchritt, Message[Induktionsbeweis::"schritt"],
           _, Message[Induktionsbeweis::"!entscheidb"]];

               InduktionsSchritt && Verankerung
               ];
```

```
In[161]:= BeweisDurchInduktion[fibonacci, f[n] ≥ (3/2)^n, n ≥ 11]

          Induktionsbeweis::!entscheidb :
           Der Induktionsschritt konnte nicht zu Ende geführt werden

Out[161]= 2^-n 3^-2+n ≥ 0
```

Mathematica kann hier nicht entscheiden, ob $2^{-n}\ 3^{-2+n} \geq 0$ ist (*n* ist einfach ein Symbol). Der Induktionsbeweis wurde aber soweit ausgeführt, dass wir die Richtigkeit sofort feststellen können.

bemerkung

Durch

```
In[162]:= n /: IntegerQ[n] := True
```

wird das Symbol *n* von *Mathematica* als Integer interpretiert (man sagt auch, das Symbol werde mit IntegerQ[n]:=True assoziiert).

```
In[163]:= IntegerQ[n]

Out[163]= True
```

Ist *n* eine ganze Zahl, so auch die Summe und das Produkt:

```
In[164]:= Unprotect[IntegerQ, Plus, Times];
          IntegerQ[Plus[a_?IntegerQ, b__?IntegerQ]] := True;
          IntegerQ[Times[a_?IntegerQ, b__?IntegerQ]] := True;

In[165]:= IntegerQ[#] & /@ {2 n + 1, 4 n, n - 1, (n + 1) (3 n + 1)}

Out[165]= {True, True, True, True}
```

Eine Potenz mit einem ganzzahligen Exponenten und einer positiven Basis ist grösser als 0:

```
In[166]:= Unprotect[GreaterEqual, Integer, Power];
          GreaterEqual[Power[x_ /; x ≥ 0, y_?IntegerQ], 0] := True

In[167]:= {2^n ≥ 0, 3^(2 n-1) ≥ 0, 2.718^-n ≥ 0}
Out[167]= {True, True, True}
```

Eine Zahl grösser als 0 ist positiv:

```
In[168]:= Unprotect[Positive, Negative];
          Positive[x_ /; x ≥ 0] := True
          Negative[x_ /; -x ≥ 0] := True
```

Das Produkt von positiven Zahlen ist positiv:

```
In[170]:= Unprotect[Times];
          GreaterEqual[Times[x_?Positive, y__?Positive], 0] := True

In[171]:= {2^n 3^-4 n 2.718^-n ≥ 0, 7^n ≥ 0}
Out[171]= {True, True}

In[172]:= Protect[GreaterEqual, IntegerQ,
            Integer, Power, Plus, Times, Positive, Negative];
```

Jetzt gelingt der Induktionsbeweis:

```
In[173]:= BeweisDurchInduktion[fibonacci, f[n] ≥ (3/2)^n, n ≥ 11]
Out[173]= True
```

Da *Mathematica*-Funktionen "geschützt" sind, muss bei Änderungen an solchen Funktionen der Schutz mit **Unprotect** aufgehoben werden. Mit **Protect** kann man sie wieder "schützen". Natürlich kann man auch eigene Funktionen schützen.

Aufgabe

4. Ein Induktionsschritt misslingt, wenn Ausdrücke wie $n^2 - n + 1 == 0$ entsehen. Erweitere die Regeln so, dass auch diese Fälle erfasst werden

2 Zufallszahlen

ein zufallsgenerator

John v. Neumann und S.Ulam hatten in den 50er Jahren folgenden Zufallszahlgenerator vorgeschlagen:

$$z_0 = 0.731$$
$$z_k = 4\, z_{k-1}(1 - z_{k-1})$$

Für die Berechnung der Zahlen $z_1, z_2,$ ... eignet sich **NestList** :

Nest[$z, x_0, 3$**]** $\longrightarrow z(z(z(x_0)))$

NestList[$z, x_0, 3$**]** $\longrightarrow \{x_0\ , z(x_0), z(z(x_0))\ , z(z(z(x_0)))\}$

In unserem Beispiel ist

```
In[37]:= z[x_] := 4 x (1 - x);
```

Die ersten 7000 Zufallszahlen:

```
In[38]:= zufallszahlen = NestList[z, 0.731, 7000];
         zufallszahlen // Shallow
Out[38]//Shallow=
         {0.731, 0.786556, 0.671543, 0.882292, 0.41541,
          0.971378, 0.111211, 0.395373, 0.956213, 0.16748, <<6991>>}
```

Wir möchten mit diesem Zufallsgenerator eine Urne mit den drei Kugeln 0,1,2 simulieren. Dazu müssen wir für jede Zufallszahl entscheiden, in welchem der Teilintervalle $[0, \frac{1}{3})$, $(\frac{1}{3}, \frac{2}{3})$, $(\frac{2}{3}, 1]$ sie liegt. Die Funktion *kugelziehen* nimmt ein z_k zur Menge $\{\frac{1}{3}, \frac{2}{3}\}$ hinzu, sortiert, und bestimmt die Position:

```
In[39]:= kugelziehen := (Position[Union[{1 / 3, 2 / 3}, {#}], _Real][[1, 1]] - 1) &

In[40]:= ereignisse = kugelziehen /@ zufallszahlen;
         ereignisse // Shallow
Out[40]//Shallow=
         {2, 2, 2, 2, 1, 2, 0, 1, 2, 0, <<6991>>}
```

Die relativev Häufigkeiten:

```
In[34]:= RelHäuf[ereignisse]
Out[34]= {0.384802, 0.221397, 0.393801}
```

Die Elementarereignisse treten nicht gleichwahrscheinlich ein. Allenfalls wird so eine Urne mit den fünf Kugeln 0,0,1,2,2 simuliert.

verteilung

Wie sieht die Verteilung aus? Wir müssen z.B. zu $X = \frac{1}{5}, \ldots, \frac{4}{5}, 1$ alle Zufallsahlen $x \leq X$ zählen. Hierzu wenden wir die gleiche Methode wie in Beispiel xx an, indem wir die Menge der Stützpunkte $\{0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1\}$ mit der Menge $\{z_1, \ldots, z_n\}$ von n Zufallszahlen vereinigen und sortieren. Man hat dann z.B die Menge

$$\{0, a_1, \ldots, a_7, \tfrac{1}{5}, b_1, \ldots, b_{13}, \tfrac{2}{5}, c_1, \ldots, c_{11}, \tfrac{3}{5}, d_1, \ldots, d_6, \tfrac{4}{5}, e_1, \ldots, e_{15}, 1\}$$

In diesem Fall sind wir an den Positionen der rationalen Zahlen interessiert. Da auch 0 und 1 dazugehören, lautet das Muster **_Rational | _Integer** (rational *oder* integer):

Position[$\{0, a_1, \ldots, a_7, \frac{1}{5}, b_1, \ldots, b_{13}, \frac{2}{5}, c_1, \ldots, c_{11}, \frac{3}{5}, d_1, \ldots, d_6, \frac{4}{5}, e_1, \ldots, e_{15}, 1\}$, **_Rational | _Integer]**

$\longrightarrow$ {{1}, {9}, {23}, {35}, {42}, {58}}

Flatten[{{1}, {9}, {23}, {35}, {42}, {58}}**]** $\longrightarrow$ {1, 9, 23, 35, 42, 58}

{1, 9, 23, 35, 42, 58} – {1, 2, 3, 4, 5, 6} $\longrightarrow$ {0, 7, 20, 31, 37, 52}

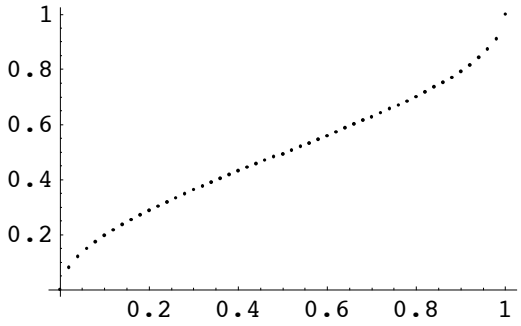
{0, 7, 20, 31, 37, 52} / 52 // **N** $\longrightarrow$ {0, 0.13461, 0.38461, 0.59615, 0.71153, 1}

Um diese Verteilungsfunktion mit **ListPlot** zu zeichnen müssen wir noch die beiden Mengen $\{0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1\}$ und {0, 0.13461, 0.38461, 0.59615, 0.71153, 1} zu einer Liste der Form $\{\{x_1, y_1\}, \ldots \{x_6, y_6\}\}$ verknüpfen. Dies tut **Thread**.

Der Algorithmus (x ist die Liste der Zufallszahlen z_k und Δx die Schrittweite):

```
In[93]:= Verteilung[x_, △x_Rational] := Block[{stpkte, w, p},
           stpkte = Range[0, 1, △x];
           w = Union[stpkte, x];
           p = Position[w, _Rational | _Integer] // Flatten;
           p = (p - Range[1 / △x + 1]) / Length[x];
           Thread[{stpkte, p}] // N
           ]
```

```
In[36]:= ereignisse = NestList[z, 0.731, 7000];
         ListPlot[Verteilung[ereignisse, 1 / 50]];
```



aufgabe

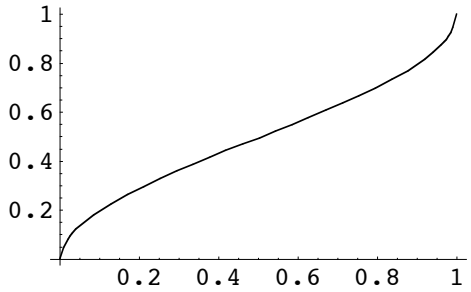
1. Verallgemeinere den Algorithmus **Verteilung** so, dass Zufallszahlen im Intevall $[x_1, x_2]$ ausgezählt werden.

interpolation

Wie muss man die Grenzen der Teilintervalle wählen, damit die Kugeln 0,1,2 mit gleicher relativer Häufigkeit auftreten? Es sind die Lösungen der Gleichung $F(x) = \frac{1}{3}$ und $F(x) = \frac{2}{3}$, wobei F die kummulative Verteilungsfunktion ist.

Dazu gehen wir mit **Interpolation** von der diskreten zur stetigen Funktion über:

```
In[37]:= F = Interpolation[Verteilung[ereignisse, 1 / 50]]
Out[37]= InterpolatingFunction[{{0, 1.}}, <>]

In[38]:= Plot[F[x], {x, 0, 1}];
```



Die Lösungen der Gleichungen $F(x) = \frac{1}{3}$ und $F(x) = \frac{2}{3}$:

```
In[39]:= FindRoot[F[x] == 1 / 3, {x, 0.3}]
         FindRoot[F[x] == 2 / 3, {x, 0.3}]
Out[39]= {x → 0.260893}
Out[40]= {x → 0.75584}
```

Wir nehmen $\frac{1}{4}$ und $\frac{3}{4}$ (sonst mache man sie mit **Rationalize** rational). Wieder die Funktion *kugelziehen*:

```
In[41]:= kugelziehen := (Position[Union[{1 / 4, 3 / 4}, {#}], _Real][[1, 1]] - 1) &;
         zufallszahlen = NestList[z, 0.731, 7000];
```

Die Ereignisse und die relative Häufigkeit

```
In[42]:= ereignisse = kugelziehen /@ zufallszahlen;
         RelHäuf[ereignisse]
Out[42]= {0.325239, 0.33738, 0.33738}
```

aufgabe

2. Der Algorithmus **Verteilung** soll ein viertes optionales Argument haben, welches erlaubt, die kummulative Verteilungsfunktion als Interploationsfunktion, als Graph oder als Schaubild zu erhalten. (Default: **Function[#]**)

bedingte wahrscheinlichkeit

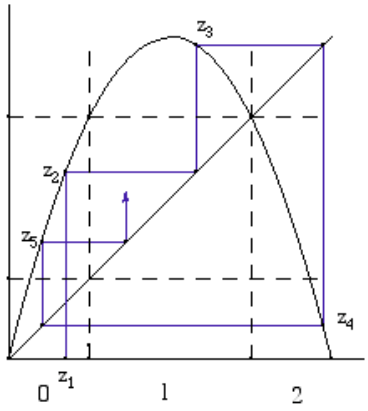
Es werden immer zwei Zahlen hintereinander gezogen. Wie gross sind die Wahrscheinlichkeiten für die Ereignisse 0 – 0, 0 – 1, ... 2 – 1, 2 – 2? Die Ereignisse in Zweierpaare aufteilen:

```
In[45]:= ereignisse = Partition[ereignisse, 2];
         ereignisse // Shallow
Out[45]//Shallow=
         {{1, 2}, {1, 2}, {1, 2}, {0, 1}, {2, 0},
          {1, 2}, {0, 0}, {1, 2}, {1, 2}, {1, 2}, <<3490>>}
```

die realativen Häufigkeiten:

```
In[46]:= RelHäuf[ereignisse,
          {{0, 0}, {0, 1}, {0, 2}, {1, 0}, {1, 1}, {1, 2}, {2, 0}, {2, 1}, {2, 2}}]
Out[46]= {0.158571, 0.166571, 0, 0, 0, 0.330857, 0.166571, 0.177429, 0}
```

Diese Elementarereignisse treten nicht mit gleicher Wahrscheinlichkeit auf. Was ist der tiefere Grund? Die folgende graphische Darstellung des Zufallsgenerators gibt Auskunft. Es sind die Parabel $4x(1-x)$, die Diagonale und die ersten vier Zufallszahlen gezeichnet.



Jetzt kann man die Wahrscheinlichkeiten ablesen:

$$P(00) = \tfrac{1}{6}, \quad P(01) = \tfrac{1}{6}, \quad P(02) = 0$$
$$P(10) = 0, \quad P(11) = 0, \quad P(12) = \tfrac{1}{3}$$
$$P(20) = \tfrac{1}{6}, \; P(21) = \tfrac{1}{6}, \quad P(22) = 0$$

Man hat hier also bedingte Wahrscheinlichkeiten.

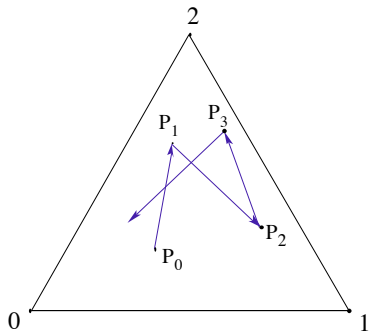
Liegt eine Folge von Zufallszahlen vor, so ist es oft schwierig, ja gar unmöglich, einen Grund anzugeben, warum bedingte oder unbedingte Wahrscheinlichkeiten vorliegen. (z.B. wenn man mit den Ziffern von π einen Zufallsgenerator konstruiren will).

Genau genommen sind Zufallszahlen nie unabhängig. Ein Algorithmus erzeugt eine Zufallszahl immer rekursiv aus den vorhergehenden nach einer deterministischen Formel (man spricht daher auch von Pseudozufallszahlen). Die Forderung nach Unabhängigkeit kann daher nur lauten, dass die Berechnungsart derart erfolgen soll, dass sich die Zahlen "für alle praktischen Zwecke" wie unabhängige Zufallszahlen verhalten.

Interessanterweise kann man mit Hilfe eines Sierpinskidreiecks bedingte Wahrscheinlichkeiten "sichtbar" machen.

erzeugung eines sierpinskidreiecks

Auch unter dem Begriff "Chaos-Spiel" ist die folgende Methode zur Erzeugung eines Sierpinskidreiecks bekannt:



Man beginnt mit irgend einem Punkt P_0 im Innern eines gleichseitigen Dreiecks, verbindet ihn zufällig mit einem der Eckpunkte 0,1,2, und konstruiert die Mitte P_1 . Mit P_1 als Ausgangspunkt wird das Verfahren wiederholt.

Wir gehen daran, ein Sierpinskidreieck auf diese Weise zu erzeugen.

Die Eckpunkte:

```
In[59]:= p[0] = {0, 0};
         p[1] = {2, 0};
         p[2] = {1, 1.732};
```

Die Mitte zweier Punkte:

```
In[60]:= mitte[x_, y_] := (x + y) / 2;
```

P_n ist die Mitte zwischen P_{n-1} und einem zufällig ausgewählten Eckpunkt:

```
In[61]:= p[3] = {0.5, 0.866};
         p[n_] := p[n] = mitte[p[n - 1], zufälligerEckpunkt[]];
```

wobei der zufällige Eckpunkt gegeben ist durch

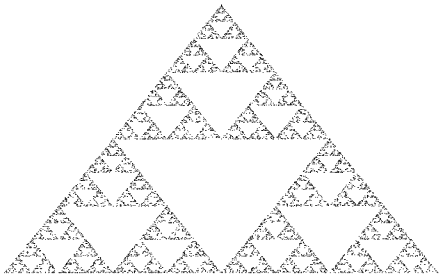
```
In[62]:= zufälligerEckpunkt[] := p[Random[Integer, 2]]
```

Jetzt haben wir alles bereitgestellt, und können beginnen. Für ein Sierpinskidreieck braucht man etwa 8000 Punkte der Grösse 10^{-5}.

```
In[65]:= punktfolge = Table[Point[p[k]], {k, 0, 8000}];
```

Die Punktgrösse fügen wir noch mit **PrependTo[...]** als erstes Element ein.

```
In[66]:= PrependTo[punktfolge, PointSize[10^-5]];
         Show[Graphics[punktfolge]];
```



beispiel

Wie wirken sich die bedingten Wahrscheinlichkeiten des Zufallsgenerators von v.Neumann auf die Erzeugung eines Sierpinskidreiecks aus?

Die Zufallszahlen:

```
In[67]:= z[x_] := 4 x (1 - x);
         zufallszahlen = NestList[z, 0.731, 7000];
```

Gleichverteilung der Ereignisse 0,1,2::

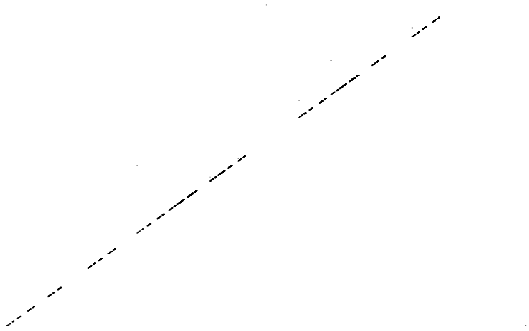
```
In[68]:= kugelziehen := (Position[Union[{1 / 4, 3 / 4}, {#}], _Real][[1, 1]] - 1) &;
         ereignisse = kugelziehen /@ zufallszahlen;
```

Der zufällige Eckpunkt ist jetzt das *n*-te Element der Folge *ereignisse*

```
In[69]:= zufälligerEckpunkt[n_] := ereignisse[[n]];
```

Den Rest können wir tel quel übernehmen:

```
In[70]:= Clear[p];
         p[0] = {0.5, 0.866};
         p[n_] := p[n] = mitte[p[n - 1], zufälligerEckpunkt[n]];
         punktfolge = Table[Point[p[k]], {k, 0, 4000}];
         PrependTo[punktfolge, PointSize[10^-5]];
         Show[Graphics[punktfolge]];
```



Man sieht jetzt sehr deutlich, dass dieser Zufallsgenerator nicht geeignet ist.

aufgabe

3. Schreibe einen Algorithmus, welcher zu einer vorgegebener Folge von Zahlen 0,1,2 ein Sierpinskidreieck zeichnet

mit π würfeln

Wir versuchen mit den Ziffern von π Zufallszahlen zu erzeugen. Jemand hat folgende Idee:

3.1415926 `53589793 `23846264` ... ⟶ 0.31415926, 0. 53589793, 0.23846264, ...

Man teilt die Ziffern von π in Blöcke (z.B immer acht Ziffern). Aus einem Block $a_1\, a_2 \cdots a_8$ wird dann die Zahl $0.\, a_1\, a_2 \cdots a_8$ konstruiert.

Die Berechnung der Zufallszahlen:

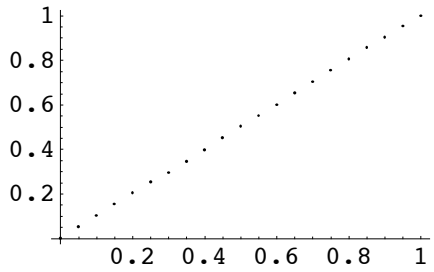
π auf n Stellen genau:	**N[**π **,** n**]**
die Ziffern einer reellen Zahl:	**RealDigits[**{3.141 ...}**]** ⟶{{3, 1, 4, 1, ...}, 1}
partitionieren:	**Partition[**{ ...} , 8**]** ⟶ {{ ...}, { ...}, ...}
eine ganze Zahl erzeugen:	**FromDigits[**{3, 1, 4, 1, 5, ..}**]** ⟶ 31415..
eine Zufallszahl	**N[**31415926 10^{-8} , 8**]** ⟶ 0.31415926

Da wir bereits ein File mit den ersten 50`000 Ziffern von π erstellt haben, lesen wir es hier ein:

```
In[72]:= ziffernPi = << PiDigit50;
         zufallszahlen = N[FromDigits[#] 10^-8, 8] & /@ Partition[ziffernPi, 8];
```

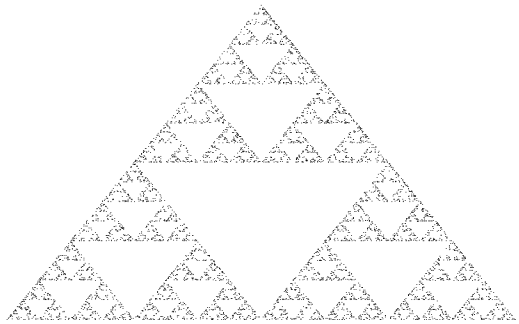
Als erstes werfen wir einen Blick auf die Verteilungsfunktion. Wir benutzen den Algorithmus aus Aufgabe 1:

```
In[100]:= Verteilung[zufallszahlen, {0, 1}, 1 / 20, ListPlot];
```



Wir dürfen davon ausgehen, dass sie gleichverteilt sind. Jetzt müssen wir aber noch das Experiment mit dem Sierpinskidreieck durchführen, um zu prüfen, ob allenfalls bedingte Wahrscheinlichkeiten vorhanden sind.

```
In[102]:= kugelziehen := (Position[Union[{1 / 3, 2 / 3}, {#}], _Real][[1, 1]] - 1) &;
          ereignisse = kugelziehen /@ zufallszahlen;

In[108]:= sierp[ereignisse, Length[ereignisse]]
```



Der Zufallsgenerator scheint brauchbar zu sein. Wohlgemerkt: es handelt sich hier um ein Experiment. Ein Ja ist provisorisch, ein Nein wäre definitiv gewesen.

3 Fixmengen

Die Ideen zu diesem Kapitel sind aus dem Buch "Bausteine des Chaos, Fraktale" von H.O.Peitgen, H.Jürgens und D. Saupe, Kapitel 5 und 6 entnommen. Dort findet man eine ausführliche Darstellung und Beschreibung.

affine Abbildungen

Beispiel 1

Eine Drehung wird durch die Matrix $\begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}$ und eine Streckung durch $\begin{pmatrix} r & 0 \\ 0 & r \end{pmatrix}$ beschrieben (beide bzgl {0, 0}).

Als reine Funktion (**Function**) definiert lauten sie:

```
In[108]:= Dreh[φ_] := {{Cos[φ], -Sin[φ]}, {Sin[φ], Cos[φ]}}.# &;
          Streck[r_] := {{r, 0}, {0, r}}.# &;
```

Den Punkt {0, 0} abbilden:

```
In[49]:= Dreh[π / 2][{1, 1}]
Out[49]= {-1, 1}
```

Das Einheitsquadrat abbilden:

```
In[50]:= quadrat = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
         Dreh[π / 2] /@ quadrat
Out[50]= {{0, 0}, {0, 1}, {-1, 1}, {-1, 0}}
```

Beispiel 2

Eine Drehstreckung kann jetzt mit **Composition** als zusammengesetzte Abbildung definiert werden:

```
In[109]:= Dreh[φ_, r_] := Composition[Dreh[φ], Streck[r]];
```

Das Einheitsquadrat abbilden:

```
In[52]:= Dreh[π / 2, 0.5] /@ quadrat
Out[52]= {{0., 0.}, {0., 0.5}, {-0.5, 0.5}, {-0.5, 0.}}
```

Bemerkung

Wir hätten die Abbildungen auch so definieren können:

dreh[φ_,x_**]:=**{{cos φ, −sin φ}, {sin φ, cos φ}} .x

streck[r_,x_**]:=**{{r, 0}, {0, r}}.x

Sie hat den Nachteil, dass Kompositionen von Abbildungen verschachtelt auftreten:

dreh[φ_ , r_ , x_**]:=dreh[**φ , **streck[**r , x**]]**

Das Einheitsquadrat abbilden:

dreh[$\frac{\pi}{2}$, 0.5, #**]**& /@ quadrat

Für lineare Abbildungen ist auch die Definition als Matrix geeignet:

dreh[φ_**]:=**{{cos φ, −sin φ}, {sin φ, cos φ}}

streck[r_**]:=**{{r, 0}, {0, r}}

Eine Komposition von Abbildungen ist dann einfach das Matrizenprodukt:

dreh[φ_ , r_ **]:=dreh[**φ**].streck[**r**]**

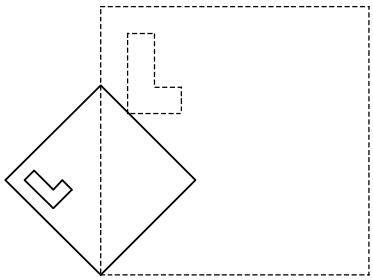
Das Einheitsquadrat abbilden:

dreh[$\frac{\pi}{2}$, 0.5**].**#& /@ quadrat

Bemerkung

Für die Illustration einer Abbildung ist ein Quadrat mit einem $\mathbb{L}$ nützlich. Der folgende Algorithmus **ShowAbb** bildet dieses Quadrat ab, und zeichnet das Urbild gestrichelt. **ShowAbb** erhält als Argumente eine Folge w__ von Abbildungen. **Composition[**w**]** bildet dann das Quadrat und das $\mathbb{L}$ ab. Damit keine Verzerrung auftritt verwenden wir **graphics** aus dem ersten Kapitel.

```
In[53]:= ShowAbb[w__] := Block[{l, q, urbild, bild},
              l =
           {{.1, .9}, {.1, .6}, {.3, .6}, {.3, .7}, {.2, .7}, {.2, .9}, {.1, .9}};
              q = {{0, 0}, {1, 0}, {1, 1}, {0, 1}, {0, 0}};
              urbild = Sequence[Dashing[{0.01, 0.01}], Line[l], Line[q]];
              bild = Sequence[
                      Line[Composition[w] /@ l], Line[Composition[w] /@ q]];
              Show[graphics[bild, urbild]];];

In[54]:= ShowAbb[Dreh[π / 4], Streck[.5]]
```



In Kapitel xx werden wir darauf zurückkommen

Aufgabe

1. Definiere die folgenden Abbildungen:

a) **Spieg[**{a_, b_, 0}**]** Spiegelung an der Geraden $a\,x + b\,y = 0$

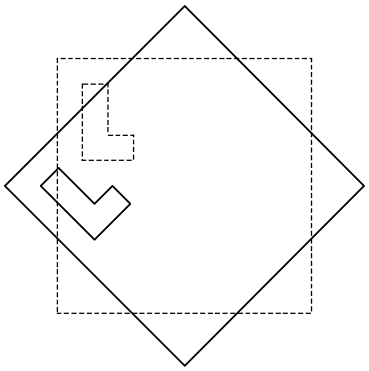
b) **Streck[***r_***,***s_***]** Streckung in X -Richtung mit Faktor r , in Y-Richtung mit s.

c) **Scher[***v_***,X] bzw Scher[***v_***,Y]** Scherung um v in X bzw Y-Richtung

Beispiel 3

Affine Abbildungen können jetzt leicht gewonnen werden. Z.B. die Drehung um einen Punkt x:

```
In[110]:= Dreh[φ_, x_?VectorQ] := (x + Dreh[φ][# - x]) &

In[56]:= ShowAbb[Dreh[π / 4, {0.5, 0.5}]]
```

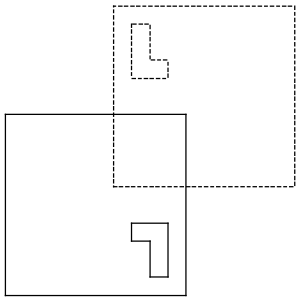


Bei der Punktspiegelung **Spieg[***x_***]** brauchen wir noch ein Unterscheidungsmerkmal zur Geradenspiegelung **Spieg** aus Aufgabe 1a. Wir können dazu die Länge des Vektors x nehmen

```
In[57]:= Spieg[x_] := (x + {{-1, 0}, {0, -1}}.(# - x) &) /; Length[x] == 2;
```

Spieg wird hier nur unter der Bedingung dass x ein Vektor der Länge 2, also ein Punkt ist aufgerufen.

```
In[58]:= ShowAbb[Spieg[{0.2, 0.2}]]
```



Aufgaben

2. Definiere die Abbildungen

a) **Spieg[{***a_* , *b_* , *c_***}]** Spiegelung an der Geraden $a\,x + b\,y + c = 0$

b) **Streck[***r_* , *x_***?VectorQ]** Streckung am Punkt *x*

c) **Dreh[**φ**_** , *r_* , *x_***?VectorQ]** Drehstreckung um *x*

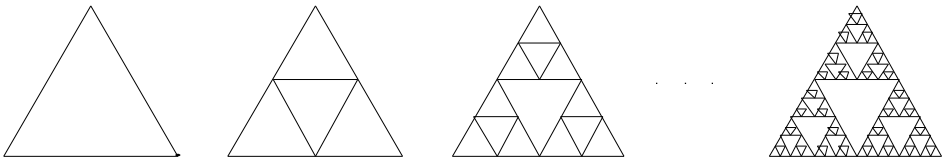
d) **Transl[***v_***]** Translation um $\vec{v}$

3. Schreibe einen Algorithmus welcher ein Polygon und sein Bild unter einer affinen Abbildung zeichnet.

Fixmengen

fixmengen

Das Sierpinskidreieck ist die Grenzfigur der Folge



Mit den Abbildungen

w_0 : Streckung mit Faktor 0.5 am Punkt {0, 0}
w_1 : Streckung mit Faktor 0.5 am Punkt {2, 0}
w_2 : Streckung mit Faktor 0.5 am Punkt $\{1, \sqrt{3}\}$

und der Anfangsmenge $D_0 = \{\{0, 0\}, \{2, 0\}, \{1, 1.732\}\}$ kann man die n-te Stufe rekursiv aufbauen:

$$D_1 = w_0(D_0) \bigcup w_1(D_0) \bigcup w_2(D_0)$$
$$D_2 = w_0(D_1) \bigcup w_1(D_1) \bigcup w_2(D_1)$$
$$\vdots$$
$$D_n = w_0(D_{n-1}) \bigcup w_1(D_{n-1}) \bigcup w_2(D_{n-1})$$

Die Folge D_n strebt gegen eine Grenzmenge D_∞, dem Sierpinskidreieck.

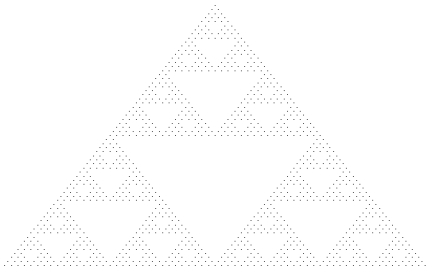
In[70]:=
```
w[0] = Streck[0.5];
w[1] = Streck[0.5, {2, 0}];
w[2] = Streck[0.5, {1, Sqrt[3]}];
```

Die rekursive Folge:

In[71]:=
```
d[0] = {{0, 0}, {2, 0}, {1, Sqrt[3]}};
d[n_] := Union[w[0] /@ d[n - 1], w[1] /@ d[n - 1], w[2] /@ d[n - 1]];
```

D_7 erzeugen:

```
In[73]:= Show[Graphics[
              Prepend[Point[#] & /@ d[6], PointSize[0.001]]]];
```



Für die Grenzmenge gilt

$$D_\infty = w_0(D_\infty) \bigcup w_1(D_\infty) \bigcup w_2(D_\infty)$$

d.h. das Sierpinskidreieck wird unter w_0 , w_1 , oder w_2 auf sich selbst abbgebildet.

Dies ist auch der Grund, warum die Methode mit dem Zufallsgenerator funktioniert. Man bleibt in der Menge D_∞ . Man nennt sie deshalb auch Fixmenge. Im "Chaos-Spiel" war

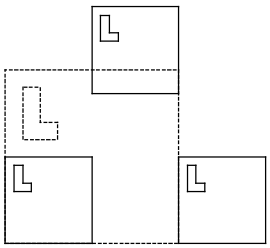
P_n = **mitte[** P_{n-1} , $P_{0,1,\text{oder } 2}$ **]**

eine der Abbildungen w_0 , w_1 , oder w_2

Von jetzt an werden wir Fixmengen mit einem Zufallsgenerator erzeugen

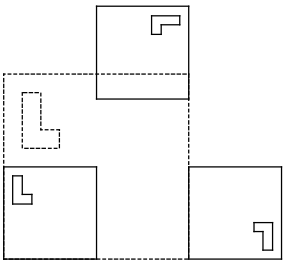
weitere fixmengen

Die drei Abbildungen, welche zum Sierpinskidreieck führen, kann man auch so darstellen:



Man nennt sie den Bauplan des Sierpinskidreiecks.

variieren:



die zugehörigen Abbildungen

```
In[74]:= w[0] = Streck[0.5];
         w[1] = Composition[Streck[0.5, {2, 0}], Dreh[π, {0.5, 0.5}]];
         w[2] = Composition[Streck[0.5, {1, 1.732}], Dreh[-π / 2, {0.5, 0.5}]];
```

Um allgemein eine Fixmenge zu zeichnen, muss der Algorithmus **sierp** aus Kapitel xx nur geringfügig geändert werden. Die Zeile

$p(k)$ = **mitte**[$p(k-1)$, p**[Random[Integer** , 2**]]**

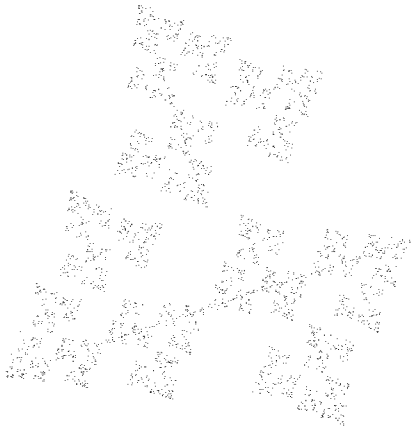
wird ersetzt durch

$p(k) = w$ [[*zufallszahl*]] [$p(k-1)$]

wobei w eine Liste $\{w_0, w_1, \ldots w_n\}$ von Abbildungen ist. *zufallszahl* wird mit **Random[Integer** , n**]** erzeugt.

verallgemeinern.

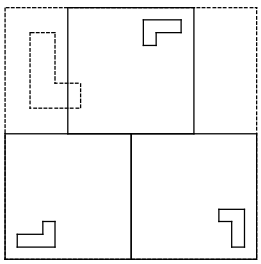
```
In[75]:= Fixmenge[w_List, n_] := Block[{p, sd, h = Length[w]},
           p[0] = {0.5, 0.5};
           p[k_] := p[k] = w⟦Random[Integer, {1, h}]⟧[p[k - 1]];
           sd = Table[Point[p[k]], {k, 0, n}];
           PrependTo[sd, PointSize[10^-4]];
           Show[graphics[sd]];];

In[77]:= Fixmenge[{w[0], w[1], w[2]}, 3000]
```



aufgaben

4. Schreibe einen Algorithmus **Bauplan[***w*__**]**, welcher zu gegebenen Abbildungen den Bauplan erstellt

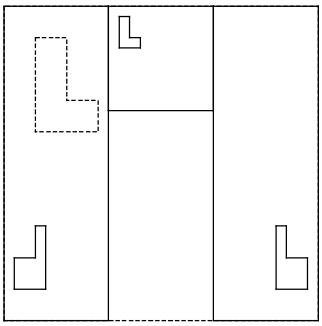
5. Welche Fixmenge wird von der folgenden Abbildungen erzeugt?



beispiel: der Cantor-Irrgarten

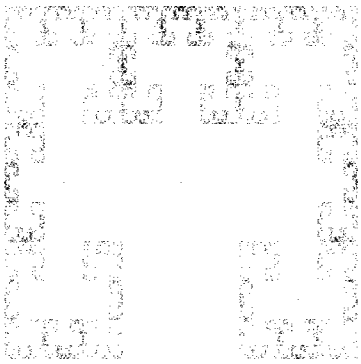
Der sog. Cantorirrgarten wird durch die Abbildungen:

```
In[88]:= w[0] = Composition[Streck[1 / 3, 1], Dreh[π / 2, {0.5, 0.5}]];
         w[1] = Streck[1 / 3, {0.5, 1}];
         w[2] =
           Composition[Spieg[{1, 0, -0.5}], Streck[1 / 3, 1], Dreh[π / 2, {0.5, 0.5}]];
         Bauplan[w[0], w[1], w[2]]
```



erzeugt:

```
In[89]:= Fixmenge[{w[0], w[1], w[2]}, 5000]
```



Wie kann man das Bild verbessern? Idee: die Wahl einer Abbildung soll nicht mehr mit gleicher Wahrscheinlichkeit erfolgen, sondern im Verhältnis der Inhalte im Bauplan (d.h dem Verhältnis der Determinanten). Bei der Cantormenge ist

$$p_0 : p_1 : p_2 = 3:1:3 = \tfrac{3}{7} : \tfrac{1}{7} : \tfrac{3}{7}$$

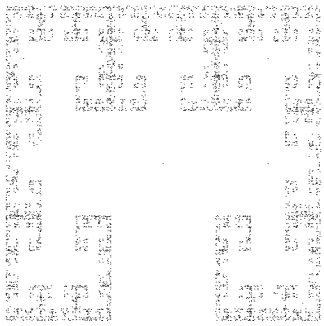
Die Zeile

$p(k) = w$ [[*zufallszahl*]] [$p(k-1)$]

im Algoritmus **Fixmenge** bleibt dieselbe, ausser dass *zufallszahl* mit **Random[**{p_0, ... p_n}**]** aus Bsp xx erzeugt wird:

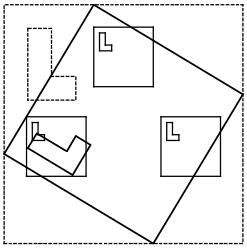
```
In[90]:= Fixmenge[abb_List, wkeit_List, n_] := Block[{p, sd},
           p[0] = {0.5, 0.5};
           p[k_] := p[k] = abb⟦Random[wkeit] + 1⟧[p[k - 1]];
           sd = Table[Point[p[k]], {k, 0, n}];
           PrependTo[sd, PointSize[10^-4]];
           Show[graphics[sd]];];
```

```
In[92]:= Fixmenge[{w[0], w[1], w[2]}, {3 / 7, 1 / 7, 3 / 7}, 5000]
```



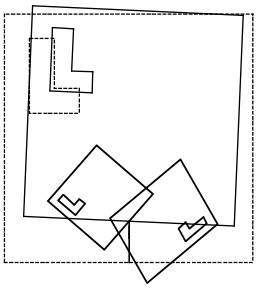
Aufgabe

6. Erzeuge die Fixmenge von



beispiel: der Barnsleyfarn

Barnsley hat mit Abbildungen experimentiert und einen Bauplan für eine Fixmenge gefunden welche einen Farn assoziiert:



```
In[69]:= w[0] = ({0.075, 0.1830} + {{0.849, 0.037}, {-0.037, 0.849}}.#) &;
         w[1] = ({0.400, 0.0490} + {{0.197, -0.226}, {0.226, 0.197}}.#) &;
         w[2] = ({0.575, -0.0840} + {{-0.15, 0.283}, {0.260, 0.237}}.#) &;
         w[3] = ({0.500, 0} + {{0, 0}, {0, 0.16}}.#) &;
```

Auch mit den Wahrscheinlichkeiten muss experimentiert werden. Als Anhaltspunkte dienen die Determinanten der Abbildungen.

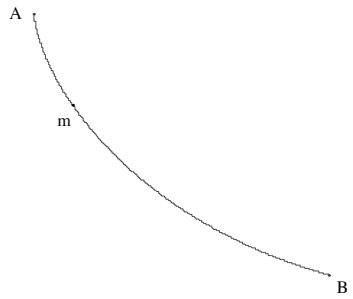
```
In[83]:= Fixmenge[{w[0], w[1], w[2], w[3]}, {1 / 2, 4 / 16, 3 / 16}, 12000]
```

4 Zykloidenpendel

Die Brachistochrone

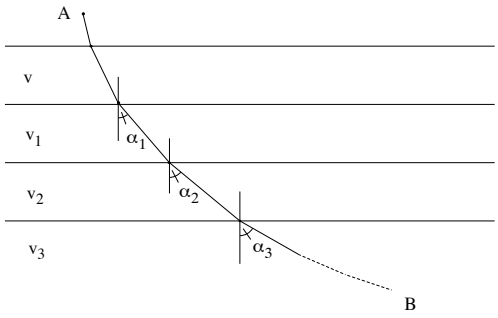
Bernoulli's Idee

Wie muss die Bahn beschaffen sein, damit ein Punkt in minimaler Zeit von *A* nach *B* gelangt? Es ist eine Zykloide. Man nennt sie auch die Brachistochrone (Bahn minimalster Zeit).



Diese Aufgabe wurde im Jahre 17xx (anlässlich eines Wettbewerbes) gestellt. Auch Johann Bernoulli beteiligte sich, und seine Lösung besticht durch ihre Eleganz. Eine schöne Darstellung findet man in *Induktion und Analogie in der Mathematik* von G.Polya.

Bernoulli interpretiert die gesuchte Bahn als Weg eines Lichtstrahls, welcher durch Schichten verschiedener Dichte hindurch von *A* nach *B* gelangt: nach dem Fermatschen Prinzip durchläuft Licht immer den Weg kürzester Zeit.



Welcher Zusammenhang besteht zwischen den Winkeln α und den Geschwindigkeiten? Das Snelliussche Brechungsgesetz gibt Auskunft: $\frac{\sin(\alpha_1)}{v_1} = \frac{\sin(\alpha_2)}{v_2} = \ldots = \text{const} = k$. Grenzübergang:

(1) $$\sin(\alpha) = k\, v$$

Man bringt jetzt die Steigung $y' = \tan(90 - \alpha)$ ins Spiel und zieht das Gesetz von Toriccelli bei:

$$\sin(\alpha) = \frac{1}{\sqrt{1+\tan^2(90-\alpha)}} = \frac{1}{\sqrt{1+y'^2}} \qquad v = \sqrt{2\,g\,y}$$

In der Gleichung (1) einsetzen und umformen:

$$y'\sqrt{y} = \sqrt{c-y} \qquad \text{mit} \quad c = \frac{1}{2\,g\,k^2}$$

Bernoulli war an dieser Stelle fertig, denn er hatte die Differentialgleichung gekannt. Sie beschreibt eine Zykloide. Wenn man sie nicht kennt, kann man mit Variation der Variablen und geeigneter Substitution nachweisen, dass die Parameterform der Lösungsfunktion eine Zykloide darstellt. Wir wollen dies aber hier nicht tun, sondern im nächsten Kapitel einen geometrischen Zugang dazu finden. Die Konstante c entpuppt sich als Höhe der Zykloide (Durchmesser des sie erzeugenden Kreises). Des weiteren sind wir an der Berechnung der minimalen Zeit interessiert.

Simulation des Bewegungsablaufs ?

Wie kann man bei vorgegebenem A und B die Bahn zeichnen? Man kommt nicht umhin aus den Koordinaten von B die Höhe $2r$ der Zykloide zu bestimmen. Selbst für eine Simulation des physikalischen Bewegungsvorgangs mit **NDSolve** muss r (wegen $c = 2\,r$) bekannt sein.

Man kommt vielleicht auf die Idee, die Bewegung mit Hilfe des Bernoullischen Lösungsansatzes zu simulieren: Man unterteilt die Fallhöhe in kleine Schichten Δy und berechnet sukzessiv Winkel und Geschwindigkeit. Die Gleichungen lauten:

$$v(y) = \sqrt{2\,g\,y}$$
$$\sin\alpha(y + \Delta y) = \frac{v(y+\Delta y)}{v(y)}\,\sin\alpha(y)$$
$$x(y + \Delta y) = \Delta y\,\tan\alpha(y + \Delta y)$$

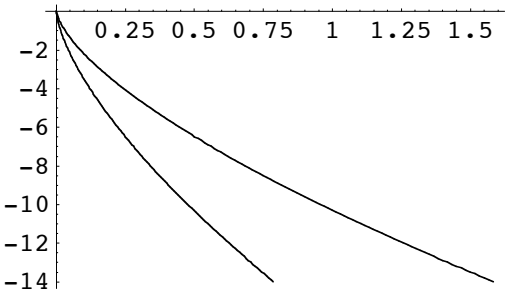
Das Problem ist aber die Singularität im Nullpunkt. Man kann nicht mit den Anfangsbedingungen y[0]=0 und sin(α)=0 beginnen.Möchte man trotzdem weiterfahren, so kann man nur "mogeln": zB y[0]=0.001, sinα:=0.001. Aber schon eine kleine Änderung dieser Anfangswerte ergibt wesentlich unterschiedliche Bahnen:

```
In[114]:= simulation[sinusα_] := Block[
            {△y = 0.1, y = .002, x = .001, y1 = 14,
                sinα, tanα, v, bahn, g = 9.81},
            v[y_] := Sqrt[2 g y];
            tanα := sinα / Sqrt[1 - sinα^2];
            sinα = sinusα;
            bahn = {{x, -y}};
            While[y < y1,
                sinα = sinα v[y + △y] / v[y];
                x = x + △y tanα;
                y = y + △y;
                AppendTo[bahn, {x, -y}]];
          Graphics[Line[bahn], Axes -> True, DisplayFunction -> Identity]];
```

Als Eingabeparameter erhält **simulation** den Anfangswert von sinα. Die **While**-Schlaufe simuliert den Bewegungsvorgang mit obigen Gleichungen.

Die Bahnen mit den Anfangswerten $\alpha = 0.001$ und $\alpha = 0.002$:

```
In[115]:= Show[{simulation[0.001], simulation[0.002]},
          DisplayFunction -> $DisplayFunction];
```



Diese Simulation ist also unbrauchbar. Abgesehen davon versagt bei vorgegebenem Endpunkt B diese Methode vollends.

Ein numerisches Beispiel

Wir setzen $A(0\,|\,0)$ und wählen $B(12\,|\,-14)$. (Der Massenpunkt fällt also in negativer y-Richtung). Die Zykloide hat dann die Darstellung

$$\begin{pmatrix} x(\varphi) \\ y(\varphi) \end{pmatrix} = r\begin{pmatrix} \varphi - \mathrm{Sin}(\varphi) \\ -1 + \mathrm{Cos}(\varphi) \end{pmatrix}$$

Um r zu berechnen müssen wir das Gleichungssystem

$$12 = r(\varphi - \mathrm{Sin}(\varphi))$$
$$-14 = r(-1 + \mathrm{Cos}(\varphi))$$

lösen. Da es sich um transzendete Gleichungen handelt, ist **FindRoot** zuständig. **FindRoot** braucht Startwerte. Wir versuchen r=10 und φ=1.

```
In[116]:= lösung = FindRoot[{r (φ - Sin[φ]) == 12, r (-1 + Cos[φ]) == -14}, {r, 10}, {φ, 1}]
Out[116]= {r → 8.97182, φ → 2.16572}
```

gelungen (man versuche andere Startwerte). Die Lösungen den Variablen r,φ zuordnen:

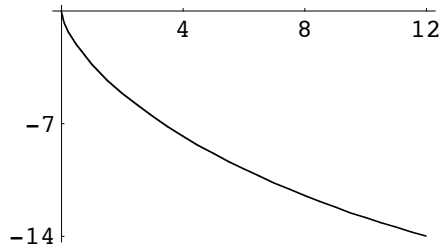
```
In[117]:= {r, φ} = {r, φ} /. lösung
Out[117]= {8.97182, 2.16572}
```

Jetzt können wir die Brachistochrone als Lösung der Differentialgleichung

$$y'\sqrt{y} = \sqrt{c - y}$$

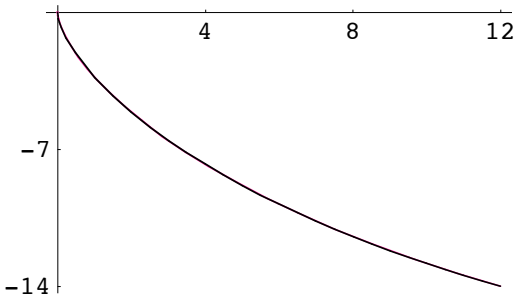
mit **NDSolve** zeichnen:

```
In[118]:= brachistochrone = {y'[x] Sqrt[y[x]] == Sqrt[c - y[x]], y[0] == 0.0001} /. c -> 2 r;
          y = y /. NDSolve[brachistochrone, y, {x, 0, 12}] // First;

In[119]:= Plot[-y[x] // Evaluate, {x, 0, 12},
            Ticks -> {{4, 8, 12}, {-7, -14}}];
```



Wie genau ist diese Lösung? Wir ziehen einen Vergleich mit der Zykloide (rot markiert):

```
In[120]:= DisplayTogether[ParametricPlot[r {t - Sin[t], -1 + Cos[t]}, {t, 0, φ},
                PlotStyle -> Hue[.9]],
            Plot[-y[x], {x, 0, 12}],
          Ticks -> {{4, 8, 12}, {-7, -14}}];
```



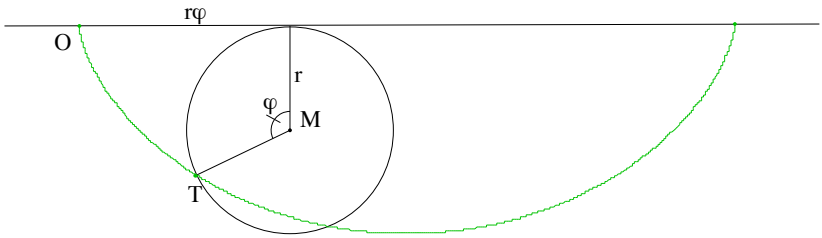
Die beiden Kurven stimmen genügend genau überein.

```
In[121]:= Clear[r, φ, c, y]
```

Die Zykloide

Die Zykloide als Rollkurve

Rollt ein Kreis auf einer Geraden, so beschreibt ein fester Peripheriepunkt eine Kurve. Man nennt sie eine Zykloide.



Ihre Parameterdarstellung lautet:

$$\begin{pmatrix} x \\ y \end{pmatrix}(t) = \overrightarrow{\mathrm{OM}} + \overrightarrow{\mathrm{MT}} = \begin{pmatrix} r\varphi \\ -r \end{pmatrix} + \begin{pmatrix} -r\sin\varphi \\ r\cos\varphi \end{pmatrix} = r\begin{pmatrix} t - \mathrm{Sin}\,t \\ -1 + \mathrm{Cos}\,t \end{pmatrix}$$

Die Zykloide als Funktion (oBdA $r = 1$):

```
In[122]:= z[t_, r_: 1] := r {t - Sin[t], -1 + Cos[t]};
```

Länge eines Zykloidenstücks

Allgemein ist die Länge s einer stetigen Kurve $(x(t) \mid y(t))$ gegeben durch $s = \int \sqrt{(\frac{\mathrm{dx}}{\mathrm{dt}})^2 + (\frac{\mathrm{dy}}{\mathrm{dt}})^2}\ \mathrm{dt}$

Für die Zykloide lautet der Ausdruck $(\frac{\mathrm{dx}}{\mathrm{dt}})^2 + (\frac{\mathrm{dy}}{\mathrm{dt}})^2$:

```
In[123]:= Plus @@ (z'[t]^2)
Out[123]= (1 - Cos[t])^2 + Sin[t]^2
```

Vereinfachen:

```
In[124]:= TrigFactor[%]
Out[124]= 4 Sin[t/2]^2
```

Die Bogenlänge einer Zykloide vom Nullpunkt aus ist also gleich

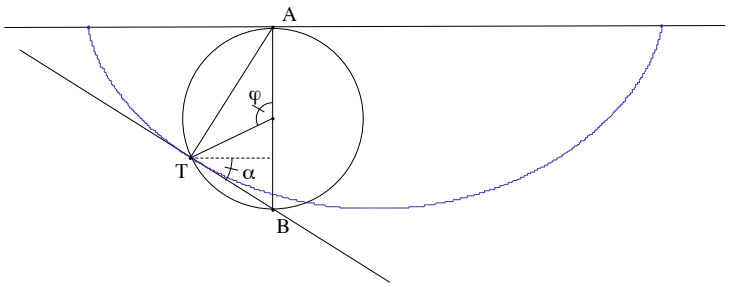
```
In[125]:= 2 r ∫_0^φ Sin[t/2] dt // Factor
```

```
In[126]:= s[φ]
Out[126]= -4 r (-1 + Cos[φ/2])
```

Im folgenden werden wir vor allem $\frac{\mathrm{d}s}{\mathrm{d}\varphi} = 2\sin(\frac{t}{2})$ brauchen.

Die Differentialgleichung

Wir betrachten eine Zykloide mit abrollendem Kreis. *AB* ist der Durchmesser, $T(x \mid y)$ der die Zykloide beschreibende Punkt.



Satz: *BT* ist Tangente an die Zykloide.

Dazu zeichnen wir die Tangente *t* in *T* und zeigen, dass sie mit *AT* einen rechten Winkel bildet. Es ist

$$\sin\alpha = \frac{dy}{ds} = \frac{dy}{d\varphi}\frac{d\varphi}{ds} = \frac{dy}{d\varphi}\frac{1}{\frac{ds}{d\varphi}} = \sin\varphi\,\frac{1}{2\sin\frac{\varphi}{2}}$$

$$= \cos\frac{\varphi}{2} = \sin(90 - \frac{\varphi}{2})$$

d.h. $\alpha = 90 - \frac{\varphi}{2}$. Also verläuft *t* nach dem Satz von Thales durch *B*.

Diese Eigenschaft nutzen wir aus. Man beachte, dass die Höhe $HT = \sin\varphi$ ist.

(i) Die Dreiecke *BHT* und *THA* sind ähnlich:

$$y' = \frac{dy}{dx} = \frac{BH}{HT} = \frac{HT}{HA} = \frac{\sin\varphi}{y} \quad \text{d.h} \quad \sin\varphi = y'\,y$$

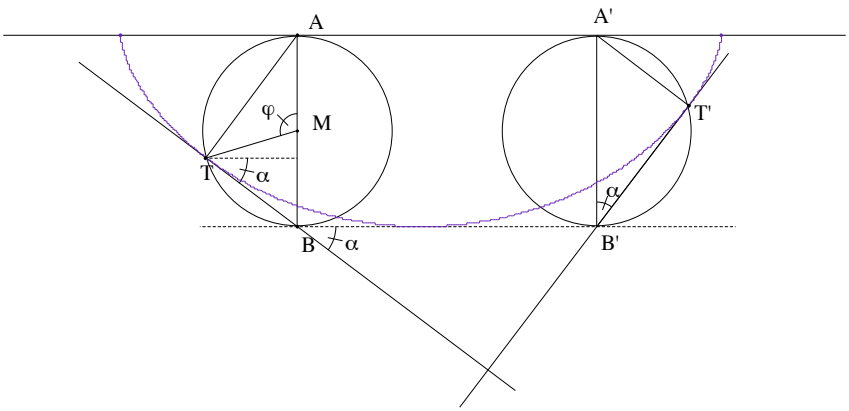
(ii) den Höhensatz in *ATB* angewandt ergibt

$$y(2r - y) = \sin^2\varphi \quad \text{d.h.} \quad \sin\varphi = \sqrt{y}\,\sqrt{2r - y}$$

Damit gewinnt man jetzt die Differentialgleichung für die Zykloide:

$$y'\sqrt{y} = \sqrt{2r - y}$$

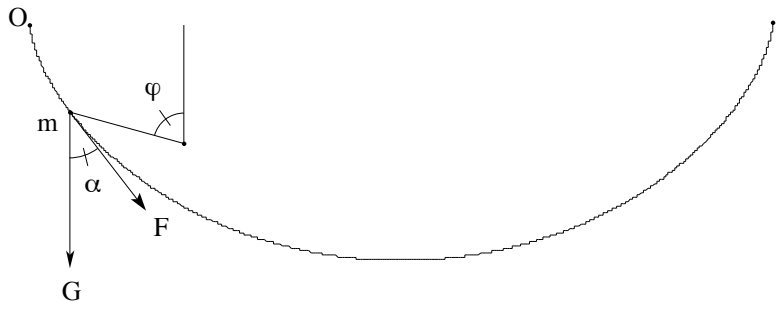
Weiter unten werden wir noch von einer weiteren geometrischen Eigenschaft gebrauch machen: Hat man zwei orthogonale Tangenten, so sind die Dreiecke *ATB* und *A'T'B'* kongruent. Wir nennen sie die "Gegendreiecke":



Die Isochrone

Ein Pendel mit konstanter Schwingungsdauer

Wir denken uns einen Massenpunkt, der in *O* losgelassen wird, und in einer zykloidenförmigen Wanne hin und her schwingt. Wie lautet die Bewegungsgleichung?



Aus der Newtonschen Definition der Kraft und der Winkeleigenschaft der Zykloide folgt

$$(1) \qquad F = -m\,g\cos\alpha = m\,g\cos\tfrac{\varphi}{2}$$

$$(2) \qquad v = \tfrac{\mathrm{ds}}{\mathrm{dt}} = \tfrac{\mathrm{ds}}{\mathrm{d}\varphi}\,\tfrac{\mathrm{d}\varphi}{\mathrm{dt}} = 2\,r\sin\tfrac{\varphi}{2}\,\tfrac{\mathrm{d}\varphi}{\mathrm{dt}} = -4\,r\,\tfrac{\mathrm{d}}{\mathrm{dt}}\cos\tfrac{\varphi(t)}{2}$$

Gleichung (2) liefert

$$(3) \qquad F = m\,\tfrac{\mathrm{dv}}{\mathrm{dt}} = -4\,m\,r\,\tfrac{d^2}{\mathrm{dt}^2}\cos\tfrac{\varphi(t)}{2}$$

Jetzt haben wir die Differentialgleichung:

$$(4) \qquad m\,g\cos\tfrac{\varphi}{2} = -4\,m\,r\,\tfrac{d^2}{\mathrm{dt}^2}\cos\tfrac{\varphi(t)}{2}$$

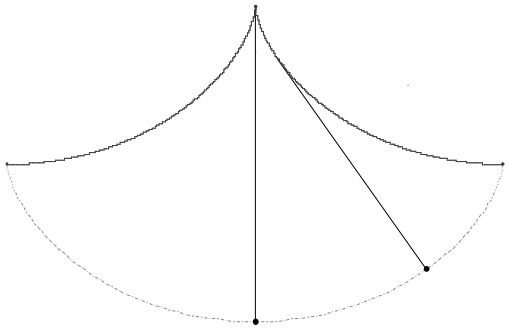
Mit $u(t) = \cos\tfrac{\varphi(t)}{2}$ lautet sie:

$$(5) \qquad 4\,r\,u''(t) + g\,u(t) = 0$$

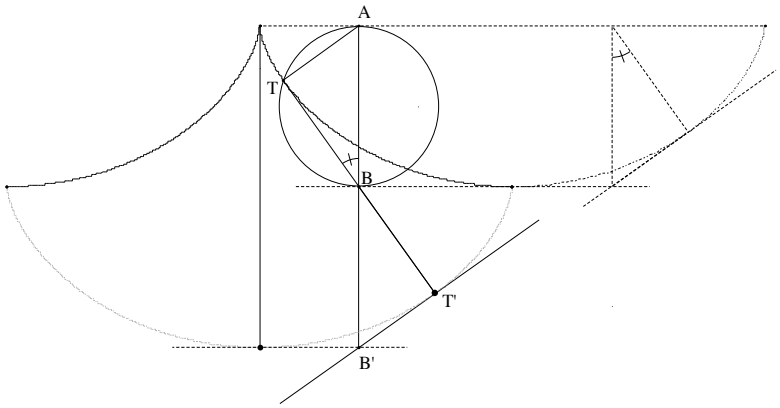
Dies ist aber die Bewegungsgleichung des harmonischen Oszillators! Die Schwingungsdauer ist also unabhängig von der Auslenkung:

$$T = 2\,\pi\sqrt{\tfrac{4\,r}{g}}$$

Genau dies hatte Huygens ausgenutzt, um ein Pendel mit konstanter Schwingungsdauer zu bauen: Man muss die Pendelführung so konstruieren, dass der Massenpunkt sich auf einer Zykloide bewegt. Es stellt sich heraus, dass die Backen wiederum zykloidenförmig sind:



Um dies zu sehen, gehen wir den umgekehrten Weg: wir zeigen, dass bei zykloidenförmigen Backen die Masse sich auf einer Zykloide bewegt. Es sei r=1, d.h. das Pendel habe die Länge $l = 4\,r = 4$. Wir lassen die Skizze sprechen.:



T' werde wie folgt konstruiert: AB verdoppeln, und von B' aus das Lot auf die Tangente in T fällen. Dann ist $BT'B'$ kongruent zum Gegendreieck von ATB. T' beschreibt also eine Zykloide, wenn T auf der Backe wandert. Wir müssen nur noch sicherstellen, dass die Kurve OTT' konstante Länge hat:

$$OT = -4\cos\tfrac{\varphi}{2} + 4$$
$$TT' = 2\,BT' = 2\,(2\cos\tfrac{\varphi}{2}) = 4\cos\tfrac{\varphi}{2}$$
$$OT+TT' = 4$$

simulation

Wir haben gesehen, dass die Bewegung eines Zykloidenpendels durch die Differentialgleichung

$$4\,r\,u''(t) + g\,u(t) = 0 \quad \text{mit} \qquad u(t) = \cos\tfrac{\varphi(t)}{2}$$

modelliert wird. Die Lösung gibt uns Auskunft über den Zustand des Pendels zu einem Zeitpunkt t. Damit können wir eine Folge von Bildern erstellen; z.B zu den Zeitpunkten $t = 0,\ \Delta t,\ 2\,\Delta t,\ ....,\ T$ mit $\Delta t = \frac{T}{25}$

Wir wollen zwei Pendel A und B mit den Auslenkungen $\varphi_0 = 1$ und $\varphi_0 = 1.7$ schwingen lassen.

Die halbe Schwingunsdauer und die Lösungsfunktionen $\varphi_A(t)$ und $\varphi_B(t)$:

```
In[9]:= T = 2 π Sqrt[1 / 9.81];
        φA =
            z /. NDSolve[{4 D[Cos[z[t] / 2], {t, 2}] + 9.81 Cos[z[t] / 2] == 0,
              z'[0] == 0, z[0] == 1}, z, {t, 0, T}][[1]];
        φB =
            z /. NDSolve[{4 D[Cos[z[t] / 2], {t, 2}] + 9.81 Cos[z[t] / 2] == 0,
              z'[0] == 0, z[0] == 1.7}, z, {t, 0, T}][[1]];
```

Die Position der Kugel wird durch die Lösungsfunktion der Differentialgleichung bestimmt. Der Faden ist Tangente an die Backe, der Berührpunkt kann leicht mit Hilfe der Gegendreiecke bestimmt werden.

```
In[26]:= kug[φ_] := {PointSize[0.05], Point[{φ - Sin[φ], -1 + Cos[φ]}]}
         faden[φ_] := {Line[{{φ - Sin[φ], -1 + Cos[φ]}, {φ + Sin[φ], 1 - Cos[φ]}}],
                ParametricPlot[{t + Sin[t], 1 - Cos[t]},
             {t, φ, π}, DisplayFunction -> Identity][[1, 1, 1]]}
         backen = ParametricPlot[{t + Sin[t], 1 - Cos[t]}, {t, 1, 2 π - 1},
             PlotStyle -> Hue[0.9], DisplayFunction -> Identity][[1]];
```

Ein Bild zum Zeitpunkt *t*:

```
In[14]:= bild[t_] := Graphics[Flatten[
                    {backen, GrayLevel[0], faden[φA[t]], faden[φB[t]],
                        kug[φA[t]], GrayLevel[0.4], kug[φB[t]]}],
          PlotRange -> {{0, 2 π}, {-2.5, 2}},
               AspectRatio -> 4.5 / (2 π)];
```

Die Bildsequenz (man braucht nur die eine Richtung):

```
In[46]:= sequenz = Table[bild[t], {t, 0, T, T / 25}];
         pendel = Flatten[{sequenz, Reverse[sequenz]}];
```

Mit

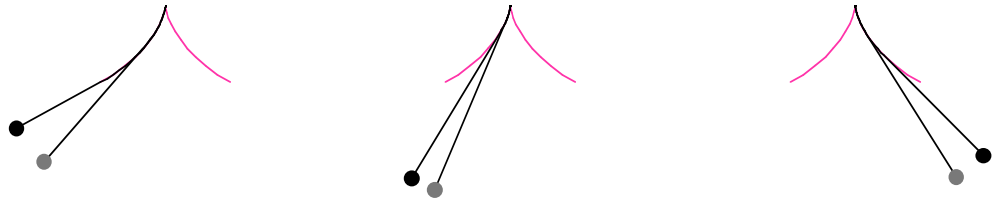
<<Graphics`Animation`

ShowAnimation[pendel]

kann jetzt die Bildfolge animiert werden.

Bild Nr 1,7 und 17:

```
In[43]:= GraphicsArray[{pendel[[1]], pendel[[7]], pendel[[17]]}] // Show;
```



nochmals die Brachistochrone

Wir sind jetzt in der Lage, die minimale Zeit die ein Massenpunkt braucht, um von *A* nach *B* zu gelangen, zu berechnen. Im Spezialfall, wenn *B* gerade der tiefste Punkt der Zykloide ist (also die Koordinaten $B(r\pi \mid 2r)$ hat), beträgt die mimimale Zeit

$$t = \tfrac{1}{4}\, T = \pi \sqrt{\tfrac{r}{g}}$$

Für den allgemeinen Fall müssen wir $y(t)$ kennen. Wegen $\cos\varphi = 1 - 2\cos^2 \tfrac{\varphi}{2}$ ist

$$(6) \qquad y(t) = r(-1 + \cos\varphi(t)) = -2\, r \cos^2 \tfrac{\varphi}{2} = -2\, r\, u(t)^2$$

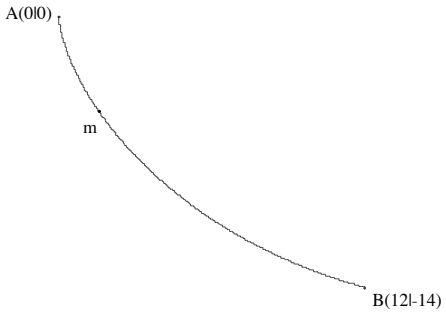
Jetzt brauchen wir noch $u(t)$. Mit $\omega := \sqrt{g/(4\,r)}$ und den Anfangsbedingungen u(0)=1, u'(0)=0 lautet die Lösung der Differentialgleichung (5)

$$u(t)=\cos(\omega\, t)$$

Einsetzen in (6):

$$y(t) = -2\, r \cos^2(\omega\, t).$$

Aufgabe. Gegeben sind die Punkte $A(0\,|\,0)$ und $B(12\,|\,-14)$. Berechne die minimale Zeit, die ein Massenpunkt braucht, um von A nach B zu gelangen.



Lösung.

Erster Schritt: bestimme die Höhe $2r$ der Zykloide mit Anfangspunkt A und Kurvenpunkt B.

Das Gleichungssystem

$$\begin{aligned} r(\varphi - \sin(\varphi)) &= 12 \\ r(-1 + \cos(\varphi)) &= -14 \end{aligned}$$

lösen:

```
In[127]:= lösung = FindRoot[
              {r (φ - Sin[φ]) == 12,
                   r (-1 + Cos[φ]) == -14}, {r, 10}, {φ, 1}]
Out[127]= {r → 8.97182, φ → 2.16572}
```

Die Parameter:

```
In[128]:= r = r /. lösung; g = 9.81; ω = 0.5 Sqrt[g/r];
```

Zweiter Schritt: berechne mit $y(t) = -2\,r\cos^2(\omega\,t)$ die Zeit.

```
In[129]:= FindRoot[-14 == -2 r Cos[ω t]^2, {t, 1}]
Out[129]= {t → 0.933256}
```

Die minimale Zeit beträgt also 0.93 Sekunden.

aufgabe

Simuliere das Fallen eines Punktes von A nach B auf der Brachistochrone. Es soll gleichzeitig ein anderer Punkt auf der schiefen Ebene von A nach B gelangen. A habe die Koordinaten $(0\,|\,0)$, B die Koordinaten $(\pi\,|\,2)$.

5 Lösungen

beispiele

ganze Zahlen

Aufgabe 1

```
In[10]:= Select[Primzahlen[100], Mod[#, 4] == 1 &]
         Select[Primzahlen[100], Mod[#, 4] == 3 &]
Out[10]= {5, 13, 17, 29, 37, 41, 53, 61, 73, 89, 97}
Out[11]= {3, 7, 11, 19, 23, 31, 43, 47, 59, 67, 71, 79, 83}
```

Aufgabe 2

```
In[12]:= PrimzahlenDerForm[4 k + x_ ≤ n_] := Select[Primzahlen[n], Mod[#, 4] == x &]

In[13]:= PrimzahlenDerForm[4 k + 1 ≤ 100]
Out[13]= {5, 13, 17, 29, 37, 41, 53, 61, 73, 89, 97}
```

Aufgabe 3

Liste der Tiefe 2. erstes Element: alle Vielfachen von 5, zweites Element: alle Vielfachen von 7

Aufgabe 4

a) mit **Apply**

```
In[31]:= Apply[Union, Map[Range[#, 72, #] &, Primfaktoren[72]]]
Out[31]= {2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24,
          26, 27, 28, 30, 32, 33, 34, 36, 38, 39, 40, 42, 44, 45, 46, 48,
          50, 51, 52, 54, 56, 57, 58, 60, 62, 63, 64, 66, 68, 69, 70, 72}
```

oder mit **Map**

```
In[32]:= Map[Range[#, 72, #] &, Primfaktoren[72]]
         // Flatten // Union
Out[32]= {2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24,
          26, 27, 28, 30, 32, 33, 34, 36, 38, 39, 40, 42, 44, 45, 46, 48,
          50, 51, 52, 54, 56, 57, 58, 60, 62, 63, 64, 66, 68, 69, 70, 72}
```

b) die Komplementärmenge von a)

```
In[33]:= Complement[Range[72], %]
Out[33]= {1, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31,
          35, 37, 41, 43, 47, 49, 53, 55, 59, 61, 65, 67, 71}
```

Aufgabe 5

```
In[34]:= TeilerfremdeZahlen[n_] :=
           Complement[Range[n],
             Apply[Union, Map[Range[#, n, #] &, Primfaktoren[n]]]];
```

Die Eulersche φ-Funktion ordnet jeder Zahl n die Anzahl der zu n teilerfremden Zahlen zu welche kleiner als n sind.

```
In[35]:= Length[TeilerfremdeZahlen[72]] == EulerPhi[72]
Out[35]= True
```

Aufgabe 6

```
In[36]:= Apply[And, Map[PseudoprimQ[#, 1105] &, {2, 3}]]
Out[36]= True
```

Aufgabe 7

den Algorithmus **PseudoprimQ** erweitern:

```
In[49]:= PseudoprimQ[b_List, n_Integer] := Apply[And, Map[PseudoprimQ[#, n] &, b]]

In[50]:= PseudoprimQ[{2, 3, 5}, 1729]
Out[50]= True

In[51]:= PseudoprimQ[{2, 3, 5, 7}, 29341]
Out[51]= True
```

Aufgabe 8

Alle zur Basis 2 pseudoprimen Zahlen:

```
In[52]:= Select[Range[29341], PseudoprimQ[2, #] &]
Out[52]= {341, 561, 645, 1105, 1387, 1729, 1905, 2047, 2465, 2701, 2821,
          3277, 4033, 4369, 4371, 4681, 5461, 6601, 7957, 8321, 8481, 8911,
          10261, 10585, 11305, 12801, 13741, 13747, 13981, 14491, 15709,
          15841, 16705, 18705, 18721, 19951, 23001, 23377, 25761, 29341}
```

davon noch dijenigen zu den Basen 3,5,7 prüfen:

```
In[53]:= Select[%, PseudoprimQ[{3, 5, 7}, #] &]
Out[53]= {29341}
```

Aufgabe 9

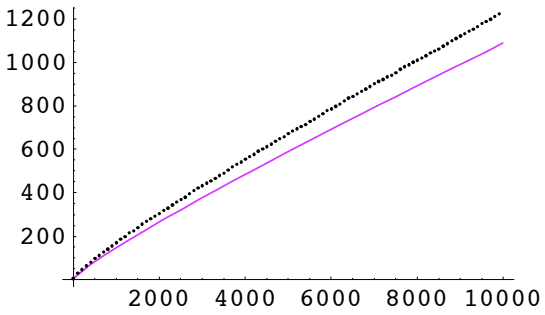
```
In[54]:= PseudoprimQ[TeilerfremdeZahlen[29341], 29341]
Out[54]= True
```

graphik

Aufgabe 1

PrimePi[x**]** liefert die Anzahl Primzahlen $\le x$. **ListPlot** erhält die Liste $\{\{x_1,$ **PrimePi[**x_1**]**$\}$, ..., $\{x_n,$ **PrimePi[**x_n**]**$\}\}$ mit $x_k = 10 + k\,100$.

```
In[70]:= DisplayTogether[
           Plot[x / Log[x], {x, 10, 10000},
             PlotStyle -> Hue[0.8]],
           ListPlot[Table[{k, PrimePi[k]}, {k, 10, 10000, 100}]]
           ];
```



Aufgabe 2

In projektiven Koordinaten lautet der Kegelschnitt $x^2 - 2\,y^2 + x\,y - 3\,x\,z = 0$. Die Asymptoten sind die Tangenten in den Fernpunkten ($z = 0$).

Koordinaten der Fernpunkte lauten

```
In[71]:= Solve[x^2 - 2 y^2 + x y - 3 x z == 0 /. z -> 0, y]
```

$$\text{Out[71]=}\quad \left\{\left\{y \to -\frac{x}{2}\right\}, \{y \to x\}\right\}$$

d.h. $(2\,|\,-1\,|\,0)$ und $(1\,|\,1\,|\,0)$. Die Geraden mit Steigung $-\frac{1}{2}$ und 1, welche keinen endlichen Schnittpunkt mit der Hyperbel haben:

```
In[72]:= Solve[{x^2 - 2 y^2 + x y - 3 x == 0, y == -x / 2 + q}, {x, y}]
         Solve[{x^2 - 2 y^2 + x y - 3 x == 0, y == x + q}, {x, y}]
```

$$\text{Out[72]=}\quad \left\{\left\{y \to \frac{-3\,q + 2\,q^2}{3\,(-1+q)},\ x \to \frac{2\,q^2}{3\,(-1+q)}\right\}\right\}$$

$$\text{Out[73]=}\quad \left\{\left\{y \to \frac{3\,q + q^2}{3\,(1+q)},\ x \to -\frac{2\,q^2}{3\,(1+q)}\right\}\right\}$$

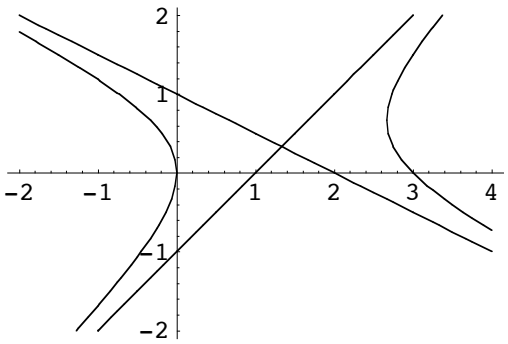
Also lauten die Gleichungen der Asymptoten:

$$y = -\tfrac{1}{2} + 1 \text{ und } y = x - 1$$

Die Packung laden

```
In[74]:= << Graphics`ImplicitPlot`
```

```
In[75]:= ImplicitPlot[
           {x^2 - 2 y^2 + x y - 3 x == 0, y == -x / 2 + 1, y == x - 1}, {x, -2, 4}, {y, -2, 2}];
```



berechnungszeiten

Aufgabe 1

Eine Zufallsmatrix der Grösse $n \times n$:

```
In[87]:= zufallsmatrix[n_] :=
          Table[Random[Integer, {-2^16 - 1, 2^16 - 1}], {k, 1, n}, {j, 1, n}]
```

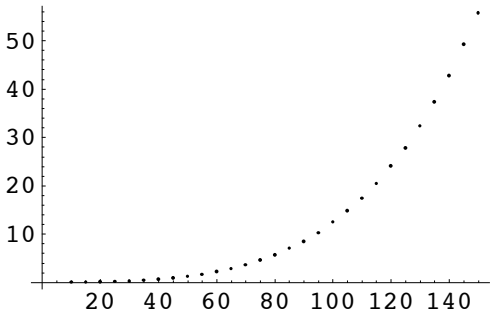
Zeit für die Berechnung einer Determinante:

```
In[88]:= det[n_] := Timing[Det[zufallsmatrix[n]]][[1, 1]]
```

```
In[89]:= berechnungszeit = Table[{k, det[k]}, {k, 10, 150, 5}]
```

```
Out[89]= {{10, 0.0333333}, {15, 0.0333333}, {20, 0.1}, {25, 0.166667}, {30, 0.283333},
          {35, 0.4}, {40, 0.633333}, {45, 0.883333}, {50, 1.21667}, {55, 1.65},
          {60, 2.16667}, {65, 2.81667}, {70, 3.6}, {75, 4.56667}, {80, 5.66667},
          {85, 7.03333}, {90, 8.46667}, {95, 10.2333}, {100, 12.5}, {105, 14.7833},
          {110, 17.3667}, {115, 20.4333}, {120, 24.0833}, {125, 27.8},
          {130, 32.3667}, {135, 37.3}, {140, 42.7}, {145, 49.2167}, {150, 55.75}}
```

```
In[90]:= ListPlot[berechnungszeit];
```

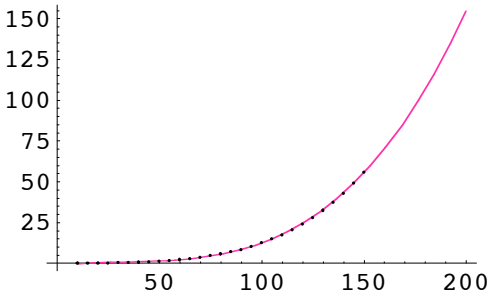


Eine Determinante kann in polynomialer Zeit der Ordung 3 berechnet werden:

```
In[91]:= f = Fit[berechnungszeit, {1, x, x^2, x^3}, x]
```

```
Out[91]= -1.57225 + 0.138992 x - 0.00322141 x^2 + 0.0000321432 x^3
```

```
In[92]:= DisplayTogether[
             Plot[f, {x, 10, 200}, PlotStyle -> Hue[.9]],
             ListPlot[berechnungszeit]];
```



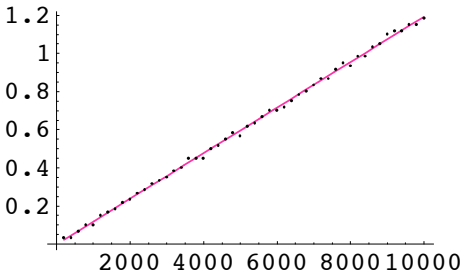
Aufgabe 2

Eine Liste mit Zufallszahlen sortieren:

```
In[93]:= sort[n_] := Sort[Table[Random[Integer, {-2^16 - 1, 2^16 - 1}], {k, 1, n}]];

In[94]:= berechnungszeit = Table[{k, Timing[sort[k]][[1, 1]]}, {k, 200, 10000, 200}];

In[95]:= ListPlot[berechnungszeit];
```

die Berechnungszeit wächst linear:

```
In[96]:= f = Fit[berechnungszeit, {1, x}, x]
Out[96]= -0.00353741 + 0.000119452 x

In[97]:= DisplayTogether[
             Plot[f, {x, 200, 10000}, PlotStyle -> Hue[.9]],
             ListPlot[berechnungszeit]];
```




```
In[98]:= Clear[f]
```

differentialgleichungen

Aufgabe 1

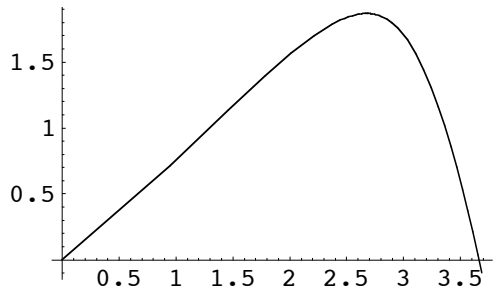
```
In[116]:= bild[t_] := Flatten[{sonne, planet[t]}];

In[117]:= film = Table[
            Graphics[bild[t],
                PlotRange -> {{-1.3, 0.6}, {-0.8, 0.8}},
                AspectRatio -> 1.6 / 1.9], {t, 0, T, T / 25}];

      ShowAnimation[film]

In[118]:= Clear[x, y, T, s]
```

Aufgabe 2

Sei r=1.

Die Gleichungen:

```
In[27]:= schieferWurf = {
            x''[t] == -x'[t]^2, y''[t] == If[y'[t] ≥ 0, -g - y'[t]^2, -g + y'[t]^2],
            x[0] == 0, y[0] == 0, x'[0] == 30, y'[0] == 20} /. g -> 9.81;

In[28]:= bahn = NDSolve[schieferWurf, {x, y}, {t, 0, 2}]
Out[28]= {{x → InterpolatingFunction[{{0., 2.}}, <>],
          y → InterpolatingFunction[{{0., 2.}}, <>]}}

In[29]:= ParametricPlot[{x[t], y[t]} /. bahn,
          {t, 0, 1.3}, AxesOrigin -> {0, 0}, PlotRange -> All];
```



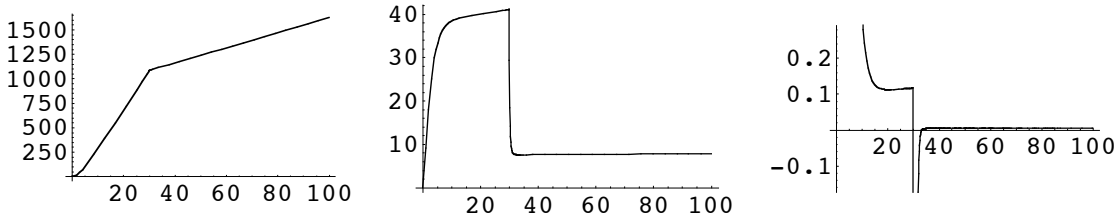
Aufgabe 3

```
In[30]:= k[t_] := 0.5 ρ[t] q[t] cw[t];
         ρ[t_] := 1.25 Exp[-.00014 x[t]];
         cw[t_] := If[t <= t1, 0.25, 0.9];
         q[t_] := If[t <= t1, 3, 25];
```

dabei wird der Fallschirm zum Zeitpunkt t_1 geöffnet

```
In[31]:= diffgl := x''[t] == g - k[t] / m x'[t]^2 /. {m -> 70, g -> 9.81, t1 -> 30};

In[32]:= s = x /. NDSolve[{diffgl, x[0] == 0, x'[0] == 0}, x, {t, 0, 100}][[1]]
Out[32]= InterpolatingFunction[{{0., 100.}}, <>]
```

In[33]:=
```
GraphicsArray[{
        Plot[s[t], {t, 0, 100}, DisplayFunction -> Identity],
        Plot[s'[t], {t, 0, 100}, DisplayFunction -> Identity],
        Plot[s''[t], {t, 0, 100}, DisplayFunction -> Identity]}] // Show;
```



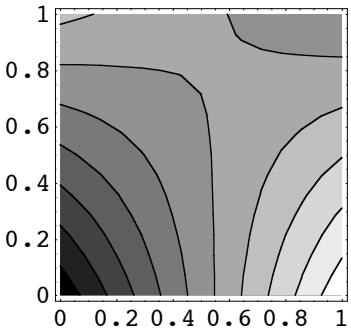
In[35]:=
```
Clear[s, k, ρ, cw, q, bahn]
```

wahrscheinlichkeitsrechnung

Aufgabe 1

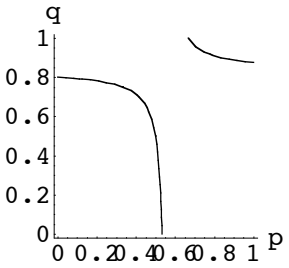
Mit **ContourPlot** erhält man Höhenlinien. Dunkle Stellen entsprechen tiefen, helle Stellen hohen Werten.

In[138]:=
```
ContourPlot[ewA[p, q], {p, 0, 1}, {q, 0, 1}];
```

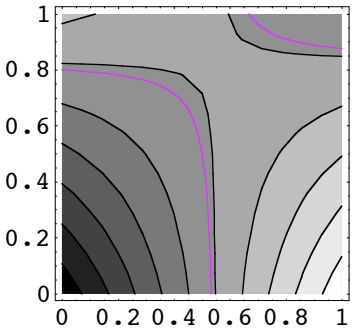


Wo ist der Erwartungswert gleich 0?

In[139]:=
```
ImplicitPlot[ewA[p, q] == 0, {p, 0, 1}, {q, 0, 1}, AxesLabel -> {"p", "q"}];
```



```
In[140]:= DisplayTogether[
            ContourPlot[ewA[p, q], {p, 0, 1}, {q, 0, 1}],
            ImplicitPlot[ewA[p, q] == 0, {p, 0, 1}, {q, 0, 1},
                PlotStyle -> Hue[0.8]]];
```



Der Rest geht genau gleich.

vollständige induktion

Aufgabe 1

```
In[154]:= phi = 1/2 (Sqrt[5] - 1);
```

```
In[155]:= BeweisDurchInduktion[{g[n] == g[n - 1] + g[n - 2],
            g[1] == phi + 1, g[2] == (phi + 1)^2}, g[n] == (phi + 1)^n]
          BeweisDurchInduktion[{h[n] == h[n - 1] + h[n - 2], h[1] == -phi, h[2] == phi^2},
           h[n] == (-phi)^n]
```

```
Out[155]= True
```

```
Out[156]= True
```

```
In[157]:= Clear[phi]
```

Die Folge $a_n = g_n - h_n$ hat die beiden Anfangswerte $a_1 = a_2 = \sqrt{5}$, d.h $f_n = a_n \frac{1}{\sqrt{5}}$ mit $a_n = (\varphi + 1)^n - (-\varphi)^n$.

Aufgabe 2

a ist die lokale Folgenvariable. Im Beispiel der Fibonaccifolge ist *index* ={1, 2} und *wert* = {1, 1}.

```
In[158]:= ntesGlied[{x_ == y_, z__Equal}, n_Integer] :=
            Module[{a, index, wert},
                index = #[[1, 1]] & /@ {z};
                wert = #[[2]] & /@ {z};
                Do[a[index[[k]]] = wert[[k]], {k, 1, Length[{z}]}];
                a[m_] := a[m] = y /. {Head[x] -> a, x[[1]] -> m};
            a[n]];
```

```
In[159]:= ntesGlied[fibonacci, 7]
```

```
Out[159]= 13
```

Aufgabe 3

Man nimmt ein weiteres Argument *w* mit dem Muster **GreaterEqual** hinzu. Die Induktionsvariable *k* kann man jetzt auch an diesem Argument ablesen. Für die Verankerung wird **ntesGlied** aufgerufen.

```
In[160]:= BeweisDurchInduktion[{x_Equal, y__Equal}, z_, w_GreaterEqual] :=
            Module[
                {a = Head[lhs[z]], k = lhs[w], k0 = rhs[w],
                    explizit, rekursiveForm, f,
                    InduktionsAnnahme, InduktionsSchritt},
                lhs[h_] := h[[1]];
                rhs[h_] := h[[2]];

                expliziteForm = rhs[z];
                rekursiveForm = rhs[x];
                f = Function[expliziteForm /. k → #];
                InduktionsAnnahme = rekursiveForm /. a → f;
                InduktionsSchritt =
              InduktionsAnnahme - expliziteForm == 0 // Simplify;

                Verankerung = ntesGlied[{x, y}, k0] == f[k0] // Simplify;

                If[! Verankerung, Message[Induktionsbeweis::"verank"]];
                If[! InduktionsSchritt, Message[Induktionsbeweis::"schritt"],
              _, Message[Induktionsbeweis::"!entscheidb"]];

                InduktionsSchritt && Verankerung
                ];
```

Aufgabe 4

```
In[175]:= Unprotect[Equal, PolynomialQ];
          Equal[x_, 0] := False /; PolynomialQ[x, n_?IntegerQ] && Variables[x] == {n}

In[176]:= Protect[Equal, PolynomialQ];

In[177]:= Equal[3 n^2 + 2 n - 1, 0]
Out[177]= False

In[178]:= BeweisDurchInduktion[
           {s[n] == s[n - 1] + n, s[1] == 1, s[2] == 3}, s[n] == n^3/2 - 5 n^2/2 + 6 n - 3]
          Induktionsbeweis::schritt : Der Induktionsschritt misslingt
Out[178]= False
```

zufallszahlen

Aufgabe 1

Man nimmt noch ein weiteres Argument mit den Intervallgrenzen hinzu:

```
In[94]:= Verteilung[x_, {x1_, x2_}, △x_Rational] :=
           Block[{stpkte, w, p},
               stpkte = Range[x1, x2, △x];
               w = Union[stpkte, x];
               p = Position[w, _Rational | _Integer] // Flatten;
               p = (p - Range[1 / △x + 1]) / Length[x];
               Thread[{stpkte, p}] // N
               ]
```

Aufgabe 2

```
In[95]:= Verteilung[x_, {x1_, x2_}, △x_Rational, op_: Function[#]] :=
           Block[{stpkte, w, p, g},
               stpkte = Range[x1, x2, △x];
               w = Union[stpkte, x];
               p = Position[w, _Rational | _Integer] // Flatten;
               p = (p - Range[1 / △x + 1]) / Length[x];
               g = Thread[{stpkte, p}] // N;
               op[g]
               ]
```

Aufgabe 3

```
In[107]:= sierp[folge_, n_] := Block[{p, pktfolge},
              {p[0], p[1], p[2]} = {{0, 0}, {2, 0}, {1, 1.732}};
              p[3] = {0.5, 0.866};
              p[k_] := p[k] = mitte[p[k - 1], p[folge[[k]]]];
              pktfolge = Table[Point[p[j]], {j, 0, n}];
              PrependTo[pktfolge, PointSize[10^-5]];
              Show[Graphics[pktfolge]];]
```

fixmengen

affine Abbildungen

Aufgabe 1a)

Die Spiegelung an einer Geraden $a\,x + b\,y = 0$ ist durch die Matrix

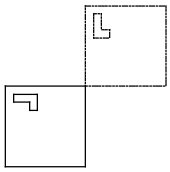
$$\frac{1}{a^2+b^2}\begin{pmatrix} -a^2+b^2 & -2\,a\,b \\ -2\,a\,b & a^2-b^2 \end{pmatrix}$$

gegeben.

```
In[59]:= Spieg[{a_, b_, 0}] :=
           1 / (a^2 + b^2) {{-a^2 + b^2, -2 a b}, {-2 a b, a^2 - b^2}}.# &;
```

Beispiel:

```
In[60]:= ShowAbb[Spieg[{1, 1, 0}]];
```

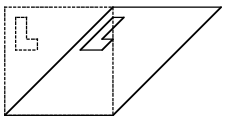


Aufgabe 1b),c)

```
In[87]:= Streck[r_, s_] := {{r, 0}, {0, s}}.# &;
```

```
In[62]:= Scher[v_, X] := Transpose[{{1, 0}, v}].# &;
         Scher[v_, Y] := Transpose[{v, {0, 1}}].# &;
```

```
In[63]:= ShowAbb[Scher[{1, 1}, X]];
```

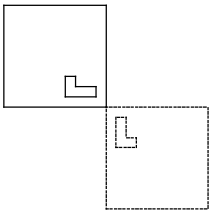


Aufgabe 2a

Die Gerade um den Vektor $-\frac{c}{a^2+b^2}\begin{pmatrix} a \\ b \end{pmatrix}$ durch den Nullpunkt verschieben:

```
In[64]:= Spieg[{a_, b_, c_}] :=
           (-c / (a^2 + b^2) {a, b} + Spieg[{a, b, 0}][# + c / (a^2 + b^2) {a, b}]) &;
```

```
In[65]:= ShowAbb[Spieg[{1, -1, 1}]]
```



Eine Variante mit Fallunterscheidung:

Spieg[{a_, 0 ,c_}**]:=**$\{-\frac{c}{a}, 0\}$ + **Spieg[**$\{a, 0, 0\}$**] [# +** $\{-\frac{c}{a}, 0\}$**]**&

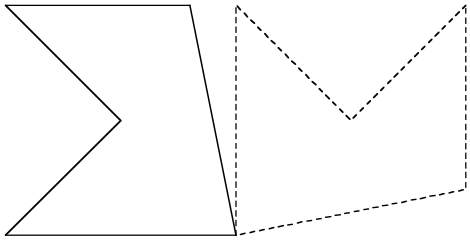
Spieg[{a_,b_,c_}**]:=**$\{0, -\frac{c}{b}\}$ + **Spieg[**$\{a, b, 0\}$**] [# +** $\{0, -\frac{c}{b}\}$**]**&

Aufgaben 2b), c),d):

```
In[111]:= Streck[r_, x_?VectorQ] := (x + Streck[r][# - x]) &;
          Dreh[φ_, r_, x_?VectorQ] := (x + Dreh[φ, r][# - x]) &;
          Transl[v_] := (v + #) &;
```

Aufgabe 3

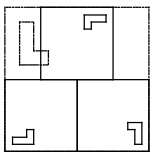
```
In[67]:= polygonabbilden[w__, x_List] :=     graphics[{
                  Line[Composition[w] /@ x], Dashing[{0.01, 0.01}], Line[x]}];

In[69]:= polygonabbilden[Dreh[π / 2],
           {{1, 1}, {0.5, 0.5}, {0, 1}, {0, 0}, {1, 0.2}, {1, 1}}] // Show;
```



fixmengen

Aufgabe 4

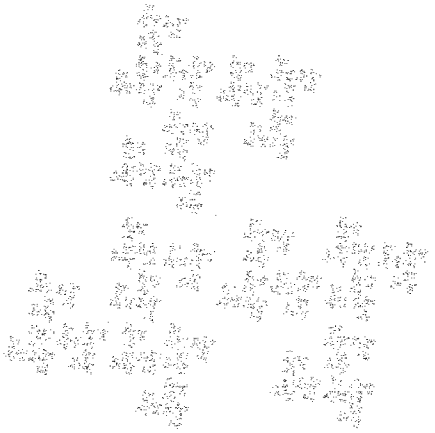
```
In[78]:= Bauplan[w__] := Block[{l, q},
              l =
           {{.1, .9}, {.1, .6}, {.3, .6}, {.3, .7}, {.2, .7}, {.2, .9}, {.1, .9}};
              q = {{0, 0}, {1, 0}, {1, 1}, {0, 1}, {0, 0}};
              urbild = Sequence[Dashing[{0.01, 0.01}], Line[l], Line[q]];
              bild = Table[
                       {
                       Line[{w}[[k]] /@ l], Line[{w}[[k]] /@ q]},
                       {k, Length[{w}]}];
              Show[graphics[Flatten[{bild, urbild}]]];
              ]

In[79]:= w[0] = Composition[Streck[0.5], Dreh[π / 2, {0.5, 0.5}]];
         w[1] = Composition[Streck[0.5, {1, 0}], Dreh[π, {0.5, 0.5}]];
         w[2] = Composition[Streck[0.5, {0.5, 1}], Dreh[-π / 2, {0.5, 0.5}]];

In[80]:= Bauplan[w[0], w[1], w[2]]
```



Aufgabe 5

```
In[81]:= w[0] = Composition[Streck[0.5], Dreh[π / 2, {0.5, 0.5}]];
         w[1] = Composition[Streck[0.5, {1, 0}], Dreh[π, {0.5, 0.5}]];
         w[2] = Composition[Streck[0.5, {0.5, 1}], Dreh[-π / 2, {0.5, 0.5}]];
```

```
In[83]:= Fixmenge[{w[0], w[1], w[2]}, 5000]
```



Aufgabe 6

```
In[101]:= w[0] = Streck[0.25, {0.125, 0.375}];
          w[1] = Streck[0.25, {0.875, 0.375}];
          w[2] = Streck[0.25, {0.5, 0.875}];
          w[3] = Dreh[ArcTan[5 / 3.], Sqrt[34.] / 8, {0.5, 0.5}];

In[113]:= Fixmenge[{w[0], w[1], w[2], w[3]}, {4 / 46, 4 / 46, 4 / 46, 34 / 46}, 5000]
```