Aufgabenvariation mit CAS

Die Variation von Aufgabenstellungen und Problemen hat schon immer zum mathematischen Handwerk gehört. In jüngster Zeit wurde dieses Verfahren auch für den Mathematikunterricht verstärkt propagiert. Dabei können die Lehrenden oder auch die Lernenden Aufgaben variieren. Durch CAS-Unterstützung erweitern sich die Möglichkeiten dieses Ansatzes erheblich. Dies wird an Beispielen erläutert.

I Aufgabenvariation

Beim BLK-Projekt Sinus gehörte das Modul zur Weiterentwicklung der Aufgabenkultur zu denen, die am meisten Aufmerksamkeit unter Lehrern auf sich gezogen habe. Das ist nicht verwunderlich, denn Aufgaben sind einer der wichtigsten Anstöße, mit denen Schüler zum Mathematiktreiben aufgefordert werden. Eine gute Übersicht über die moderne Sichtweise gibt das Buch von Büchter und Leuders. Darin wird allerdings nicht auf die Nutzung von CAS eingegangen. Großen Einfluss hatte und hat immer noch das Buch von Schupp, indem eine Vielzahl von interessanten Aufgaben mit zahlreichen Variationsmöglichkeiten dargestellt werden. Dort findet man auch eine systematische Übersicht zu den Strategien des Variierens:

- geringfügig ändern („wackeln“)
- analogisieren
- verallgemeinern (Bedingungen weglassen)
- spezialisieren (Bedingungen hinzufügen)
- Grenzfälle betrachten
- Lücken beheben
- in Beziehung setzen
- umorientieren (Ziel ändern, Zielumkehr, inverses Problem)
- zerlegen
- kombinieren
- umzentrieren
- Kontext ändern
- visualisieren
- konkretisieren
- extremalisieren (nach Optimum suchen)
- anwenden

Auch bei Schupp finden CAS kaum Beachtung. Dies ist schade, denn die Variation von Aufgaben, besonders wenn diese durch Schüler durchgeführt wird, kann leicht zu umfangreichen Rechnung oder zu unlösbaren Aufgaben führen. Bei der Exploration, was geht und was nicht mehr, kann deshalb ein CAS wertvolle Hilfe leisten. Dieser Beitrag möchte an einigen Beispielen erläutern, welche zusätzlichen Variationsmöglichkeiten sich durch CAS eröffnen.

II Numerische Verfahren untersuchen

Auch wenn die Algorithmisierung als eigenständige fundamentale Idee der Mathematik es nicht geschafft hat, in den Bildungsstandards angemessen vertreten zu sein, kann man hoffen, dass in der Praxis dieses Thema auf gewissen Inseln weiter gedeihen kann.

Eine verallgemeinernde Variation, die nur mit symbolischer Mathematik möglich ist, besteht darin, die Verfahren zum Aufstellen von Termen zu nutzen. Das folgende MuPAD-Fragment zeig, wie dies beim Heron-Verfahren aussehen kann. Dabei kann natürlich nicht eine übliche aposteiori-Abbruchbedingung verwendet werden, sondern man muss a priori die Zahl der Iterationen (oder hier Rekursionen) festlegen.

```
•heron:=   /* Wurzel aus a, Startwert x, n Schritte */
      (a,x,n) -> if n=0 then x else heron(a, (x+a/x)/2, n-1) end_if:
•heron(2,1.0,2); float(sqrt(2))
```

1.416666667

1.414213562

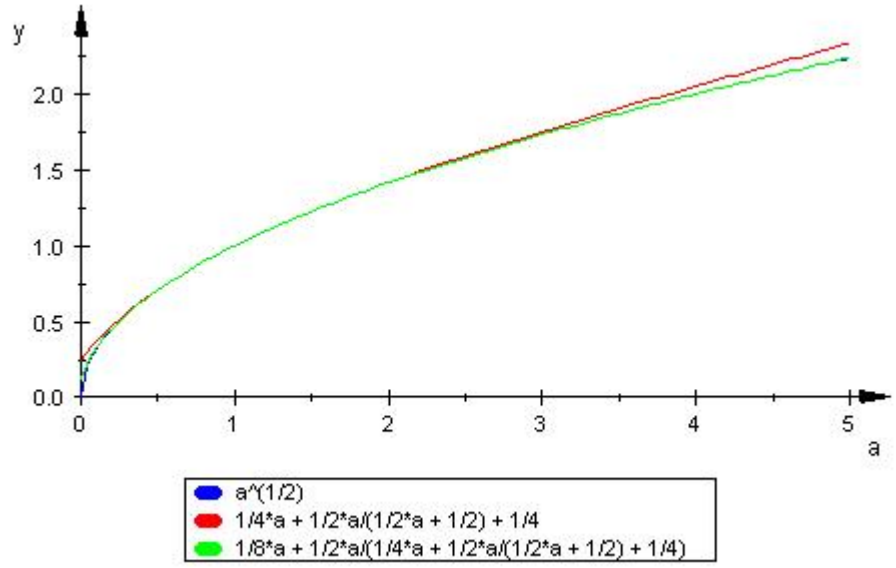
```
•  w2:=heron(a,1,2); w3:=heron(a,1,3)
```

$$\frac{a}{4}+\frac{a}{2(a/2+1/2)\in}+\frac{1}{4}$$

$$\frac{a}{8}+\frac{a}{2\left(\frac{a}{4}+\frac{a}{a/2+1/2}+\frac{1}{4}\right)}+\frac{a}{4(a/2+1/2)}+\frac{1}{8}$$

Der folgende Graph zeigt die hervorragende Übereinstimmung mit dem Graph der Wurzelfunktion.

```
•plotfunc2d(sqrt(a),w2,w3,a=0..5)
```



Die gefundene Approximation erweist sich als eine Pade-Approximation:

```
•normal(w3)
```

$$\frac{a^4+28a^3+70a^2+28a+1}{8a^3+56a^2+56a+8}$$

```
•pade(sqrt(a),a=1,[4,3])
```

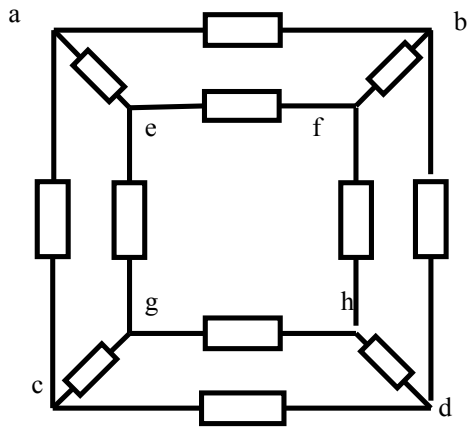
$$\frac{a^4+28a^3+70a^2+28a+1}{8a^3+56a^2+56a+8}$$

III Variationen zum Solve-Befehl

Der solve-Befel kann für vielerlei Dinge verwendet werden, die teilweise auch etwas abseits des Üblichen liegen. Die hier gezeigten Beispiele sind deshalb selbst Variationen, die aber selbst wieder leicht variiert werden können.

Gleichstromnetzwerke

Die Berechnung von Widerstandsnetzwerken bereitet den Schülern vor allem dann Schwierigkeiten, wenn es nicht möglich ist, allein durch die Reduktion von Reihen- und Parallelschaltungen zum Ziel zu kommen. Dies ist etwa bei dem beliebten Würfel aus 1Ω-Widerständen der Fall.



Die Verfügbarkeit eines CAS erlaubt es, sich ganz auf das Erstellen der Gleichungen zu konzentrieren. Dabei kann man sich sogar noch etwas Denkarbeit abnehmen lassen, wenn man die Symmetrie der Spannung in einer Funktion kodiert:

```
U:=proc(x,y)
begin
if x=y then 0 elseif sysorder(x,y) then u(x,y) else -u(y,x) end_if
end_proc;
```

Ähnlich geht man auch für die Ströme vor. Dann schreibt man ohne weiteres Nachdenken (wohl aber beachtend, dass I in MuPAD für $\sqrt{-1}$ steht) auf:

```
Maschenregeln:={U(a,b)+U(b,f)+U(f,e)+U(e,a),...};
Knotenregeln:={II(a,e)+II(a,b)+II(a,c)=0,...);
OhmGesetze:={U(a,b)=R(a,b)*II(a,b),...};
Variablen:={U(a,b),....});
solve(Maschenregeln union Knotenregeln union OhmGesetze, Variablen);
```

Das Beispiel bietet viele Ausbaumöglichkeiten. Warum genügt es, die Maschenregeln für die elementaren Maschen aufzuschreiben? Ist das System immer lösbar? Was verändert sich, wenn die Widerstände nicht gegeben, sondern gesucht sind?

Funktionsanpassung

Lineare Systeme treten in unzählbaren weiteren Zusammenhängen auf. Als besonders schön empfinde ich ihr Auftreten bei der Bestimmung von Interpolationsfunktionen und allgemeiner bei Funktionsanpassungen.

Die folgende Bestimmung eines Polynoms vierten Grades ist weitgehend selbsterklärend und kann schon von Schülern der Sekundarstufe I bearbeitet werden:

```
f:=a*x^3+b*x^2+c*x+d:
daten:={[1,4],[2,2],[3,4],[4,0]}:
solve(map(daten,func(subs(f,x=p[1])=p[2],p)),{a,b,c,d});
                    {{a = -5/3, b = 12, c = -79/3, d = 20}}
```

Auch zur Konvertierung zwischen bestimmten Notationsformen von Funktionen lassen sich lineare Gleichungssysteme einsetzen. Dabei kann man einfach eine Reihe von Werten für die

Funktionsvariable einsetzen. Die Gültigkeit der erhaltenen Formeln kann dann ja noch explizit geprüft werden.

```
f:=a*x^2+b*x+c:
g:=q*(x-x0)^2+p:
solve({subs(f-g,x=1)=0,subs(f-g,x=2)=0,subs(f-g,x=3)=0},{x0,q,p});
                { --        2                                -- }
                { |       b  - 4 a c                  b     | }
                { |  p = - ----------, q = a, x0 = - ---    | }
                { --          4 a                    2 a    -- }
```

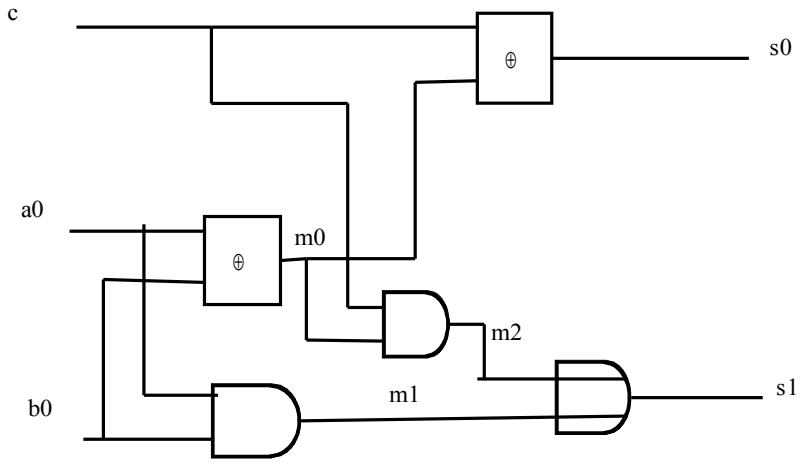
Boolesche Ausdrücke

Zwischen der digitalen Welt von 0 und 1 und dem Kontinuum des Zahlenstrahls liegt eine Denkbarriere, deren Überwindung die Schüler und Schülerinnen verblüfft. Dabei ist es ganz einfach zu bewerkstelligen, dass eine Variable als boolesche angesehen werden kann: Die Gleichung x*(x-1)=0 besitzt nur die Lösungen 0 und 1. Als Teil eines Gleichungssysteme deklariert sie damit x als boolesch! Dann ist es auch nicht mehr schwer, die anderen elementaren booleschen Funktionen darzustellen. Hier nur die wichtigsten:

Boolescher Ausruck	**Gleichung**
z=x y	`z=x*y`
z=x y	`z=1-(1-x)*(1-y)`
z=x y	`z=-(x+y)*(x+y-2)`
z= x	`z=1-x`

Wenn das Computeralgebra-System Gröbnerbasen verwendet (MuPAD, Maple, Mathematica, Reduce), um Polynomialgleichungssysteme zu lösen, ist algorithmisch verbürgt, dass diese Übersetzungmethode Erfüllbarkeits- bzw. Unerfüllbarkeitsbeweise liefert.
Als Beispiel diene ein Volladdierer für Ein-Bit-Zahlen.

```
sol:=solve({c*(c-1)=0,b0*(b0-1)=0,a0*(a0-1)=0,s0*(s0-1)=0,
            s1*(s1-1)=0,m0*(m0-1)=0,m1*(m1-1)=0,m2*(m2-1)=0,
            a0*b0=m1,c*m0=m2, s1=1-(1-m1)*(1-m2),
            m0=-(a0+b0)*(a0+b0-2),s0=-(c+m0)*(c+m0-2)},
            {a0,b0,c,s0,s1,m0,m1,m2}):
for i from 1 to nops(sol) do
       print(i,subs([a0,b0,c,s1,s0],op(sol,i)));
end_for;
```



IV q-Analysis

Die folgenden beiden Absätze stammen aus „Unfertige Mathematik mit q“, Praxis der Mathematik 2004.

Analysis ohne Grenzwerte – geht das? Die Definition der Ableitung als $f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$ ist Allgemeingut. Das Schwere daran ist der Grenzwert. Warum macht man ihn? Warum gerade so? Wer die erste Frage stellt, hat nichts von der Analysis begriffen, oder er hat zu viel Phantasie. Wer die zweite stellt, kann dagegen schon eher auf Verständnis hoffen. In der Tat ist auch der symmetrische Quotient $\lim_{h\to 0} \frac{f(x+h)-f(x-h)}{2h}$ ausführlich untersucht worden. Er hat die schöne Eigenschaft, dass auch die Betragsfunktion überall differenzierbar ist. Aber es gibt noch mehr Möglichkeiten: Im Grunde braucht man ja nur zwei benachbarte Stellen, und wählt sie als x und $x+h$. Aber warum h addieren? Genauso gut kann man multiplikativ arbeiten: qx und x als Stellen. Der interessante Grenzwert wäre dann $q \to 1$, aber (wir sind so mutig, die erste oben gestellte Frage ernst zu nehme) er soll gar nicht ausgeführt werden. Das qDifferential einer Funktion wird also definiert als $d_q f(x) := f(qx) - f(x)$, was zusammen mit $d_q x := qx - x = (q-1)x$ die qAbleitung festlegt: $D_q f(x) := \frac{d_q f(x)}{d_q x} := \frac{f(qx)-f(x)}{(q-1)x}$

Mit CAS kann man die Eigenschaft dieser Operation wunderbar untersuchen. Zu einer echten Variation kommt man, wenn man den Grenzwert doch wieder durchführt – dann man eine Variation der Ableitungsdefinition, die sich sowohl für händisches wie für CAS-gestütztes rechnen eignet.

```
•qDIF:= (f,x) ->  limit((subs(f,x=q*x)-f)/(q*x-x),q=1):
```

```
•qDIF(x^2,x),qDIF(x^3,x),qDIF(x^4,x)
```

$$2x, 3x^2, 4x^3$$

```
•qDIF(1/x,x)
```

$$-\frac{1}{x^2}$$

Händische Rechnung erhellt die Ableitung der Potenzen: $f(x) = x^n$:

$$f'(x) = \lim_{q\to 1} \frac{f(qx)-f(x)}{qx-x} = \lim_{q\to 1} \frac{q^n x^n - x^n}{x(q-1)} = x^{n-1} \lim_{q\to 1} \frac{q^n - 1}{q-1}.$$

Mittels $q^n - 1 = (q-1)\cdot(q^{n-1} + q^{n-2} + \cdots + q + 1)$ sieht man, dass sich der ganze Bruch zu $q^{n-1} + q^{n-2} + \cdots + q + 1$ vereinfacht. Im Grenzwert $q \to 1$ erhält man n als Faktor.

Mit MuPAD gehen auch noch einige andere Anwendungen gut durch:

```
•assume(q>0): factor(simplify(Dq(ln(x),x)))
```

$$\frac{\ln(q)}{x(q-1)}$$

```
•qDIF(sin(x),x)
  cos(x)
```

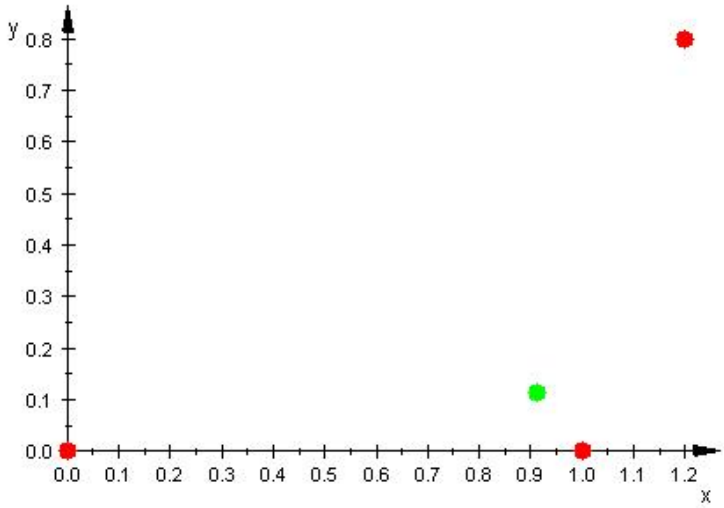
V Optimierung

Verschiedene Inhalte können sich unterscheiden in der Leichtigkeit, mit der sie sich variieren lassen. Besonders schnell geht das mit numerischer Optimierung. Zunächst ist Schülern recht schnell klar, was es bedeutet, nach einem Minimum eine Funktion in zwei (und evtl. auch mehr) Variablen zu suchen. Damit eröffnen sich Modellierungsmöglichkeiten, die in mannigfaltiger Weise variiert werden können. Die Ideen dazu liegen bei etwas mathematischer und physikalischer Allgemeinbildung auf der Straße.

Problem 1: Fermat-Punkt im Dreieck

Gesucht ist der Punkt P im Dreieck ABC, dessen Entfernungssumme von den Eckpunkten minimalen Abstand hat.

```
•dist:= (u,v) -> sqrt((u[1]-v[1])^2+(u[2]-v[2])^2):

•A:=[ax,ay]: B:=[bx,by]: C:=[cx,cy]: P:=[x,y]:
•f:=dist(A,P)+dist(B,P)+dist(C,P)

•coords:=[ax=0,ay=0,by=0,bx=1,cx=1.2,cy=0.8];
  f1:=subs(f,coords)


•P:=NOptimize(f1,[x,y],[1,1])
     [0.9123638952,0.1154156369]

•plot(plot::Point2d(P,PointSize=3,PointColor=RGB::Green),
       plot::PointList2d(subs([A,B,C],coords),
      PointSize=3,PointColor=RGB::Red),Scaling=Constrained)
```



Leichte Variationen: Viereck, gewichtete Punkte, etc...

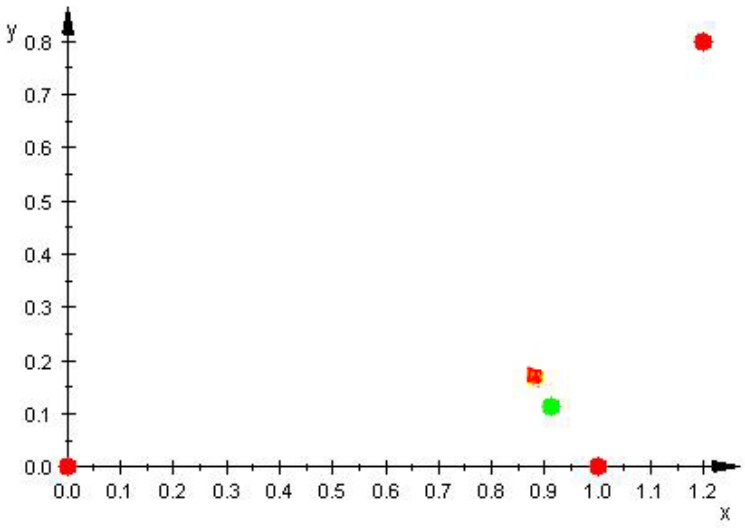
Anspruchsvollere Variation: Lemoine-Punkt: Minimierung des Abstandes zu den Dreiecksseiten.

Die folgende Funktion berechnet den Abstand des Punktes P von der Geraden durch A und B.

```
•  distPG:= (P,A,B) -> abs((B[2]-A[2])*(P[1]-A[1])+(A[1]-
B[1])*(P[2]-A[2]))/sqrt((B[2]-A[2])^2+(A[1]-B[1])^2):

•distPG([1,4],[-1,0.5],[2,1])
     3.123580759

  f2:= distPG([x,y],A,B)+distPG([x,y],C,B)+distPG([x,y],A,C) :

•Q:=NOptimize(subs(f2,coords),[x,y],[3,4])
```

```
[0.8820383905, 0.1675103622]
```

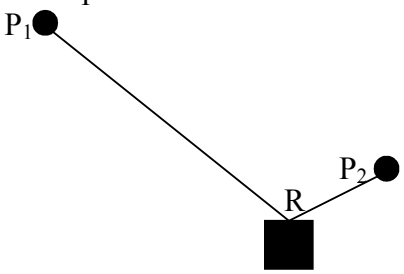
```
•plot(plot::Point2d(Q,PointSize=3,PointColor=RGB::Yellow),
       plot::Point2d(P,PointSize=3,PointColor=RGB::Green),
       plot::PointList2d(subs([A,B,C],coords),
      PointSize=3,PointColor=RGB::Red),Scaling=Constrained)
```



Auf der Ebene der numerischen Optimierung sind Fermat- und Lemoine-Punkt vergleichbar leicht zu erhalten. Elementargeometrisch ist aber ersterer wesentlich leichter zugänglich.

Problem 2: Energieminimierung

Ein dünner Faden der festen Länge L wird an zwei Haken P1 und P2, die (damit es nicht zu einfach wird) nicht gleich hoch sind, befestigt. An den Faden wird ein Massestück gehängt, das auf ihm reibungsfrei gleiten kann. Es zieht den Faden nach unten und verleiht ihm einen Knickpunkt R. Bildlich sieht das dann folgendermaßen aus:



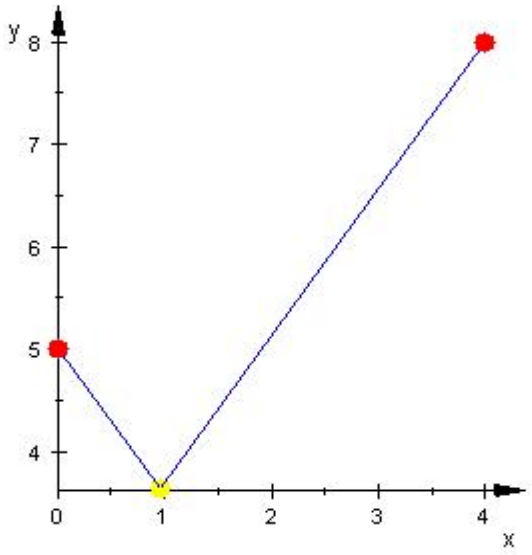
Wenn man sich nun Koordinaten für die Aufhängepunkte (z.B. P1(0|5), P2(4|8)) und einen Wert für die Fadenlänge (z.B. L=7) vorgibt, wohin rutscht dann R? Natürlich wird die potentielle Energie minimiert, d.h. die y-Koordinate von R wird minimal unter der Nebenbedingung, dass die Strecken P_1R und P_2R zusammen die vorgegebene Länge L besitzen. Damit ist die Modellbildung abgeschlossen und wir können im Modell nach einer Lösung suchen.

```
•P1:=[0,5]: P2:=[4,8]: L:=7:
•R:=NOptimize(y,[x,y],[5,5],[dist(P1,[x,y])+dist(P2,[x,y])=L])
```

[0.9555340638, 3.627718677]

```
•plot(plot::Point2d(R,PointSize=3,PointColor=RGB::Yellow),
       plot::Line2d(P1,R),plot::Line2d(P2,R),
```

```
plot::PointList2d(subs([P1,P2],coords),
PointSize=3,PointColor=RGB::Red),Scaling=Constrained)
```



Berechung eines Kranauslegers

Stäbe werden als Federn modelliert, Gesamtenergie wird minimiert Federendpunkte sind (x[i],y[i])
Energie einer Feder mit Feder-Konstante K und Ruhelänge L

•
```
feder:= (i,j,K,L) -> 1/2*K*(L-sqrt((x[i]-x[j])^2+(y[i]-y[j])^2))^2
```

•
```
vars:=[x[3],y[3],x[4],y[4],x[5],y[5],x[6],y[6],x[7],y[7]]:
  vals:=[1,0,      2,0,      3,0,      2,1,      1,1]:

  zeichne:=proc(v)
  begin
  plot(plot::PointList2d([[0,0],[0,1],[v[1],v[2]],[v[3],v[4]],
    [v[5],v[6]],[v[7],v[8]],[v[9],v[10]]],PointSize=5),
  plot::Line2d([0,0],[0,1],LineWidth=1.5),
  plot::Line2d([0,0],[v[1],v[2]],LineWidth=1.5),
  plot::Line2d([v[3],v[4]],[v[1],v[2]],LineWidth=1.5),
  plot::Line2d([v[3],v[4]],[v[5],v[6]],LineWidth=1.5),
  plot::Line2d([v[7],v[8]],[v[5],v[6]],LineWidth=1.5),
  plot::Line2d([v[7],v[8]],[v[9],v[10]],LineWidth=1.5),
  plot::Line2d([0,1],[v[9],v[10]],LineWidth=1.5),
  plot::Line2d([v[1],v[2]],[0,1],LineWidth=1.5),
  plot::Line2d([v[9],v[10]],[v[1],v[2]],LineWidth=1.5),
  plot::Line2d([v[9],v[10]],[v[3],v[4]],LineWidth=1.5),
  plot::Line2d([v[7],v[8]],[v[5],v[6]],LineWidth=1.5),
  plot::Line2d([v[7],v[8]],[v[3],v[4]],LineWidth=1.5), Axes=Boxed,
    Scaling=Constrained
  )
  end_proc:
```

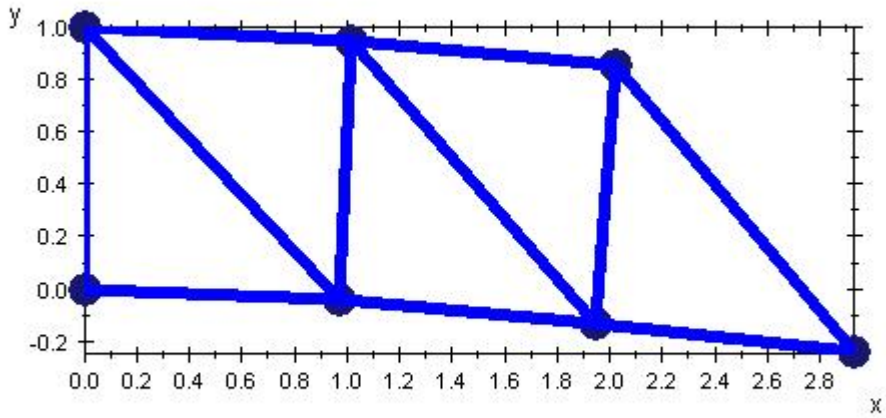
•
```
gesEnergie:=
    feder(2,3,100,1)+feder(4,3,100,1)+feder(4,5,100,1)+feder(1,7,100
    ,1)+feder(7,6,100,1)+ // horizontal
    feder(3,7,100,1)+feder(4,6,100,1)+ // vertikal
```

```
    feder(1,3,100,1.41)+feder(7,4,100,1.41)+feder(6,5,100,1.41)+ //
    schräg
    9.81*m*y[5]:  // m Kg Gewicht am Auslenker
```

Die zu minimierende Gesamtenergie ist die Summe aus allen Federenergien und der Lageenergie der Last.

Jetzt werden einige Punkte fest gesetzt, sie können sich nicht bewegen.

```
•gesE:= mm -> subs(gesEnergie,m=mm,x[1]=0,x[2]=0,y[1]=1,y[2]=0):
•valsBelastet:=NOptimize (gesE(0.1),vars,vals):
zeichne(valsBelastet)
```



Literatur

A. Büchter, T. Leuders: Mathematikaufgaben selbst entwickeln. Cornelsen 2005.

R. Oldenburg: The q-Way of doing analysis. International Journal of Computer Algebra in Mathematical Education.

R. Oldenburg: Ableitungen alternativ berechnen, *MNU* 58 (2005) 343-344

R. Oldenburg: Numerische Optimierung – ein schneller Weg zu komplexer Modellbildung. erscheit bei Istron.

R. Oldenburg: Variationen zum solve-Befehl von Computer-Algebra-Systemen, *Mathematik in der Schule*, 2/2000

H. Schupp: Thema mit Variationen. Franzbecker 2002.