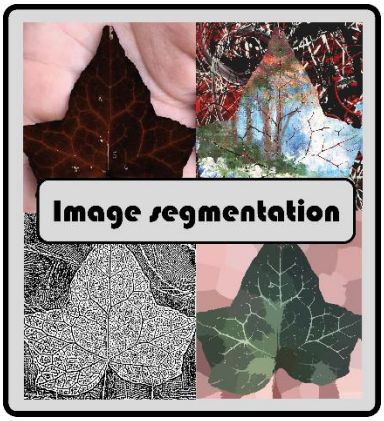
Steirische
Modellierungswoche

17. Woche der Modellierung

mit Mathematik



Dokumentationsbroschüre 11.2. - 17.02.2023

WOCHE DER MODELLIERUNG MIT MATHEMATIK

JUFA Leibnitz, 11.2. – 17.2.2023

Weitere Informationen:

https://imsc.uni-graz.at/modellwoche/2023/

Sponsoren und Organisatoren

virtual vehicle

UNI GRAZ

Bildungsdirektion Steiermark

Pädagogische Hochschule Steiermark

LV-EV STMK

Koordination

Mag. DDr. Patrick-Michel Frühmann

Anna Legenstein, BSc

Alexander Sekkas

Vorwort

Mathematik hat eine einzigartige Doppelrolle als Jahrtausende alte Kulturleistung und als Schlüsseltechnologie für die modernsten Entwicklungen der heutigen Zeit. Darüberhinaus kommt ihre Entwicklung ohne komplexe Apparate und nur mit Papier und Bleistift aus. Trotzdem empfinden die meisten Menschen heutzutage Mathematik als etwas Unzugängliches und für ihr Leben wenig Relevantes; dies beginnt bereits in der Schule, wo trotz engagierter Lehrkräfte die Zeit fehlt, die Vielfalt, Schönheit, und Nützlichkeit dieser Disziplin begreifbar zu machen.

Daher soll als Ergänzung des Schulunterrichts die Woche der Modellierung mit Mathematik Schüler:innen die Möglichkeit geben, Mathematik als lebendiges Fach zu erfahren: Die jungen Leute arbeiten und forschen in kleinen Gruppen mit Wissenschafter:innen an realen Problemen aus den verschiedensten Bereichen und versuchen, mit Hilfe mathematischer Modelle neue Erkenntnisse zu gewinnen. Sie arbeiten freiwillig ohne Leistungsdruck, dafür mit Eifer und Enthusiasmus, rechnen, diskutieren, recherchieren -- oft auch noch am späten Abend -- in einer entspannten und kreativen Umgebung, die den Schüler:innen und betreuenden Wissenschafter:innen gleichermaßen Spaß macht. Die Projektbetreuer konnten auch in diesem Jahr wieder erleben, wie eigenes Entdecken und Selbstmotivation das Verhalten der Schüler:innen während der ganzen Modellierungswoche bestimmen. Sie lernen dadurch eine Arbeitsmethode kennen, die dem tatsächlichen Forschungsleben näherkommt, als das im Rahmen des Schulunterrichts möglich wäre.

Ähnliche Modellierungswochen gab bzw. gibt es zum Beispiel auch in den USA, in Deutschland oder in Italien. Wir verdanken Herrn Univ.-Prof. Dr. Stephen Keeling den Vorschlag, auch durch die Universität Graz so eine Woche zu veranstalten, und seiner unermüdlichen Organisationsarbeit das tatsächliche Zustandekommen. Er leitet nun bereits -- trotz wohlverdienten Ruhestand -- zum 17. Mal diese inzwischen zur Institution gewordene Veranstaltung. Ihm sei an dieser Stelle noch einmal ausdrücklich und herzlich gedankt. Besonders wichtig war in den vergangenen Jahren auch die Unterstützung durch den langjährigen Mentor der Modellierungswoche, Herrn o.Univ.-Prof. Dr. Franz Kappel, der oft auch eine eigene Gruppe mit interessanten Problemstellungen betreut hat. Leider verstarb Prof. Kappel Anfang 2020, so dass er nicht erleben konnte, wie die Woche der Modellierung mit Mathematik nach der Pandemie-bedingten Pause wieder mit großem Zuspruch starten konnte.

Wir danken der Bildungsdirektion für Steiermark, und hier insbesondere Frau Fachinspektorin Mag.a Michaela Kraker, für die Hilfe bei der Organisation und die kontinuierliche Unterstützung der Idee einer Modellierungswoche. Finanzielle Unterstützung erhielten wir von der Karl-Franzens-Universität Graz durch Vizerektorin Univ.-Prof. Dr. Catherine Walter-Laager und Dekan Univ.-Prof. Dipl.-Ing. Dr.techn. Klemens Fellner, vom Fachdidaktikzentrums für Mathematik und Geometrie der Uni Graz, Leiterin Ass.-Prof. Dr. Christina Krause, vom Forschungsmanagement der Uni Graz, und vom Forschungszentrum "Virtual Vehicle".

Ohne den idealistischen, unentgeltlichen und engagierten Einsatz der direkten Projektbetreuer, Dr. Stefan Reiterer, Julius Baumhakel, Gabriel Pichlbauer und Mag. Florian Thaler, hätte diese Modellierungswoche nicht stattfinden können.

Besonderer Dank gebührt ferner Herrn Mag. DDr. Patrick-Michel Frühmann, der die ganze Veranstaltung betreut und auch die Gestaltung dieses Berichtes übernommen hat, Frau Anna Legenstein, BSc für die tatkräftige Hilfe bei der organisatorischen Vorbereitung, und Herrn Alexander Sekkas für die Hilfe bei der Betreuung der Hard- und Software.

Leibnitz, 17. Februar 2023

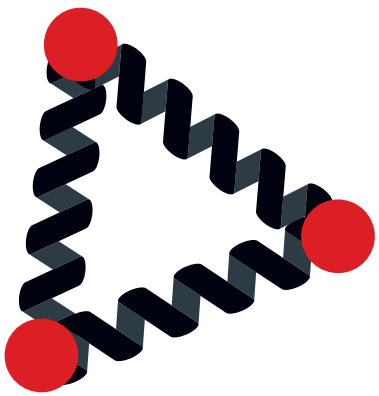
Univ.-Prof. Dr. Christian Clason
Institut für Mathematik und Wissenschaftliches Rechnen
Karl-Franzens-Universität Graz

Schwingungsdynamik

Liam Elsnig
Ferdinand Hackl
Julian Wildt
Manuel Lazian
Jakob Sengstbratl
Olivia Ringhofer

Betreuer: Stefan Reiterer

11.2. - 17.2.2023

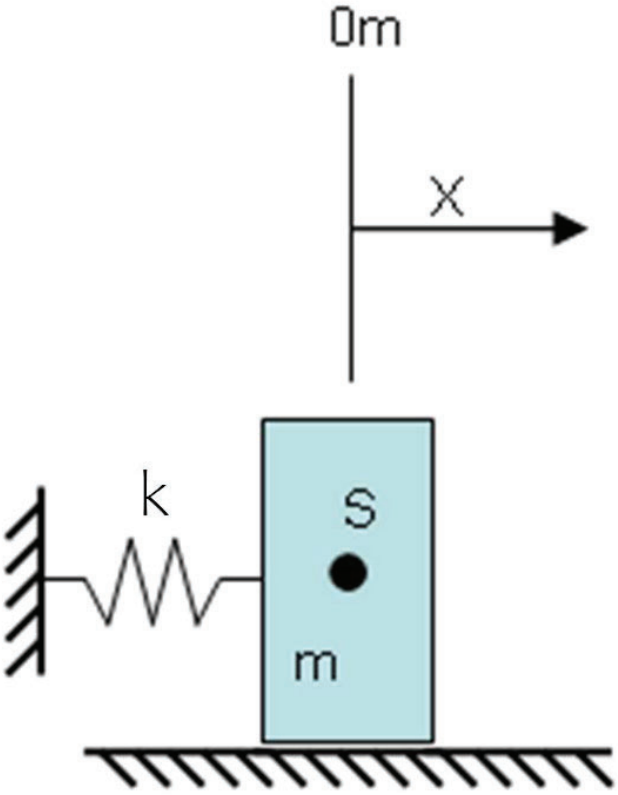


Federschwinger

Begonnen wurde mit einer Feder und der Schwingung in Form von $m\ddot{x}+kx=0$. Die Lösung ergibt die einfachste Eigenschwingung mit Eigenfrequenz ω_0. Uner weiterer Betrachtung einer zusätzlichen äußeren Schwingung wird die Gleichung zu $m\ddot{x}+kx=fe^{i\omega t}$. Die Lösung ist eine weitere Schwingung mit Amplitude $\frac{f}{m(\omega_0^2-\omega^2)}$ und brachte ein erstes Verständnis zur Resonanz: Wenn die äußere Schwingung zur Eigenfrequenz geht, geht die Amplitude gegen unendlich. Mithilfe der Matrixschreibweise von Gleichungssystemen konnte ein Mehrfedersystem in der Form $M\ddot{\mathbf{x}}+K\mathbf{x}=\mathbf{F}$, mit Massematrix M, Steifigkeitsmatrix K, einer äußeren Kraft $\mathbf{F}$ und den Ortsvektor der Massen $\mathbf{x}$. Unter Betrachtung eines solchen Systems ohne äußere Kraft und dem Ansatz $\mathbf{x}=\mathbf{u}e^{i\omega_0 t}$ erhält man das Eigenwertproble $K\mathbf{u}=m\omega_0^2\mathbf{u}$. Dabei ist ω_0 die Eigenfrequenz und $\mathbf{u}$ der Eigenvektor. Analytisch konnte so die Resonanz von allgemeinen Federsysteme berechnet werden.

Als nächstes wurden diese Schritte mithilfe des Programms Sagemath automatisiert. Zuerst wurde ein Zweifedersystem programmiert und so das Eigenwertproblem mit dem Computer gelöst. Gleiches wurde auch mit einem Dreifederschwinger getan. Daraufhin wurde der Dreifederschwinger numerisch berechnet und Näherungen für Eigenvektoren und Eigenfrequenz gefunden.

Um eine echte Schwingung darzustellen, wurde die Formel angepasst und um die Dämpfung ergänzt. Mit der Dämpfungsmatrix D und einer einfachen Zeitableitung erhält man den Zusatzterm $D\dot{\mathbf{x}}$. Es konnte die Amplitude in Abhängigkeit der Frequenz einer äußeren Schwingung als Graph dargestellt werden. So wurde gesehen, dass bei Erreichen der ersten Eigenfrequenz die Amplitude am höchsten wird und daraufhin nie mehr so stark ausschlägt. Aufgrund deieses Umstandes ist die erste Eigenfrequenz für die Statik am wichtigsten. Die höhe dieser Frequenz steht im direkten Verhältnis zur Steifigkeit eines Objekts. Als letzten Schritt wurden die errechneten Werte mithilfe einer Basistransformation ins physikalische Koordinatensystem zurückgewandelt.



2 Masse Feder Schwinger

Allgemeine Berechnung

In [2]:
```
var('c1 c2 c3 m Lambda x y')
```

Out[2]: (c1, c2, c3, m, Lambda, x, y)

Steifigkeitsmatrix K

In [3]:
```
K = Matrix([[c1+c2,-c2],[-c2,c2+c3]])
pretty_print(K)
```

$$\begin{pmatrix} c_1 + c_2 & -c_2 \\ -c_2 & c_2 + c_3 \end{pmatrix}$$

Masse M

In [4]:
```
I= Matrix([[1,0],[0,1]])
M=-I*m*Lambda**2
pretty_print(M)
```

$$\begin{pmatrix} -\Lambda^2 m & 0 \\ 0 & -\Lambda^2 m \end{pmatrix}$$

In [5]:
```
F = M + K
pretty_print(F)
```

$$\begin{pmatrix} -\Lambda^2 m + c_1 + c_2 & -c_2 \\ -c_2 & -\Lambda^2 m + c_2 + c_3 \end{pmatrix}$$

Determinante von F ausrechnen (Charakteristisches Polynom)

In [6]:
```
Fdet = det(F)
pretty_print(Fdet)
```

$$\left(\Lambda^2 m - c_1 - c_2\right)\left(\Lambda^2 m - c_2 - c_3\right) - c_2^2$$

Determinante 0 setzen und nach Lambda ausrechnen

In [7]:
```
sol = solve(Fdet==0, Lambda)
eigf= sol[1]

pretty_print(eigf)
```

$$\Lambda = \sqrt{\frac{c_1}{2\,m} + \frac{c_2}{m} + \frac{c_3}{2\,m} + \frac{\sqrt{c_1^2 + 4\,c_2^2 - 2\,c_1 c_3 + c_3^2}}{2\,m}}$$

Ausgerechneten Wert für Lambda in die Masse einsetzen und in F einsetzen

In [8]:
```
F2 = K - eigf.right_hand_side()**2*m*I
pretty_print(F2)
```

$$\begin{pmatrix} -\frac{1}{2}\,m\left(\frac{c_1}{m} + \frac{2\,c_2}{m} + \frac{c_3}{m} + \frac{\sqrt{c_1^2+4\,c_2^2-2\,c_1c_3+c_3^2}}{m}\right) + c_1 + c_2 & -c_2 \\ -c_2 & -\frac{1}{2}\,m\left(\frac{c_1}{m} + \frac{2\,c_2}{m} + \frac{c_3}{m} + \frac{\sqrt{c_1^2+4\,c_2^2-2\,c_1c_3+c_3^2}}{m}\right) + c_2 + c_3 \end{pmatrix}$$

Auslenkung in x und y Richtung

In [9]:
```
sol2 = solve(((F2[0,0]*x+F2[0,1]*y)==0, (F2[1,0]*x+F2[1,1])),x,y)
pretty_print(sol2[0])
```

$$\left[x = -\frac{c_1 - c_3 + \sqrt{c_1^2 + 4\,c_2^2 - 2\,c_1 c_3 + c_3^2}}{2\,c_2}, y = 1\right]$$

In [10]:
```
pretty_print((K/m).eigenvectors_right())
```

$$\left[\left(\frac{c_1 + 2\,c_2 + c_3 - \sqrt{c_1^2 + 4\,c_2^2 - 2\,c_1 c_3 + c_3^2}}{2\,m}, \left[\left(1, \frac{c_1 - c_3 + \sqrt{c_1^2 + 4\,c_2^2 - 2\,c_1 c_3 + c_3^2}}{2\,c_2}\right)\right], 1\right), \left(\frac{c_1 + 2\,c_2 + c_3 + \sqrt{c_1^2 + 4\,c_2^2 - 2\,c_1 c_3 + c_3^2}}{2\,m}, \left[\left(1, \frac{c_1 - c_3 - \sqrt{c_1^2 + 4\,c_2^2 - 2\,c_1 c_3 + c_3^2}}{2\,c_2}\right)\right], 1\right)\right]$$

In [11]:
```
eigv = (K/m).eigenvectors_right()[0][1][0]
eigw = (K/m).eigenvectors_right()[0][0]
```

Eigenvektor

In [12]:
```
pretty_print(eigv)
```

$$\left(1, \frac{c_1 - c_3 + \sqrt{c_1^2 + 4\,c_2^2 - 2\,c_1 c_3 + c_3^2}}{2\,c_2}\right)$$

Eigenwert

In [13]:
```
pretty_print(eigw)
```

$$\frac{c_1 + 2\,c_2 + c_3 - \sqrt{c_1^2 + 4\,c_2^2 - 2\,c_1 c_3 + c_3^2}}{2\,m}$$

Numerische Berechnung

In [14]:
```
c1 = 1
c2 = 2
c3 = 1.5
m = 5.0

import numpy as np
```

Die Werte werden in die Steifigkeitsmatrix eingetragen

In [15]:
```
K2 = np.array(K(c1=1,c2=2,c3=1.5),dtype=float)/m
```

Eigenvektor und Eigenwert berechnen

In [16]:
```
eigw = np.linalg.eig(K2)[0]
eigv = np.linalg.eig(K2)[1]
pretty_print(eigw)
pretty_print(eigv)
```

```
[0.24688711 1.05311289]

[[-0.74967818  0.66180256]
 [-0.66180256 -0.74967818]]
```

In [ ]:
```
```

Gedämpftes Federpendel

Eindimensionales Federpendel:

$m\ddot{x} + cx = f$

$x = ue^{i\omega t}$

$f = \underline{f}e^{i\omega t}$

$m * -\omega^2 e^{i\omega t}u + cue^{i\omega t} = \underline{f}e^{i\omega t}$

$= u(-m\omega^2 + c)$

$u = \frac{\underline{f}}{(-m\omega^2+c)}$

$m\ddot{x} + cx - b\dot{x} = f(t)$

Gedämpfte Schwingung eines Dreimassen-Federpendels

$$\begin{pmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{pmatrix} \ddot{x}(t) + \begin{pmatrix} c_1 + c_2 & -c_2 & 0 \\ -c_2 & c_2 + c_3 & -c_3 \\ 0 & -c_3 & c_3 + c_4 \end{pmatrix} x(t) - \begin{pmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{pmatrix} \dot{x}(t) = f(t)$$

$$\begin{pmatrix} c_1 + c_2 & -c_2 & 0 \\ -c_2 & c_2 + c_3 & -c_3 \\ 0 & -c_3 & c_3 + c_4 \end{pmatrix} = C$$

$x = ((ax_{\omega 0}) + (bx_{\omega 1}) + (cx_{\omega 2})e^{i\omega t}$

$f = (\alpha x_{\omega_0} + \beta x_{\omega_1} + \gamma x_{\omega_2})e^{i\omega t}$

$-m\omega^2 e^{i\omega t}(ax_{\omega 0}) + (bx_{\omega 1}) + (cx_{\omega 2}) + C(ax_{\omega 0}) + (bx_{\omega 1}) + (cx_{\omega 2})e^{i\omega t} - di\omega e^{i\omega t}(ax_{\omega 0}) + (bx_{\omega 1}) + (cx_{\omega 2}) = (\alpha x_{\omega_0} + \beta x_{\omega_1} + \gamma x_{\omega_2})e^{i\omega t}$

$ax_{\omega 0} * (-\omega^2 * m + \omega_0^2 * m - di\omega) + bx_{\omega 1} * (-\omega^2 * m + \omega_1^2 * m - di\omega) + cx_{\omega 2}(-\omega^2 * m + \omega_1^2 * m - di\omega) = (\alpha x_{\omega_0} + \beta x_{\omega_1} + \gamma x_{\omega_2})$

$\alpha x\omega_0 = ax\omega_0(-\omega^2 * m + \omega_0^2 * m - di\omega)$

$a = \frac{\alpha}{-\omega^2 * m + \omega_0^2 - m - di\omega}$

$b = \frac{B}{-\omega^2 * m + \omega_1^2 * m - di\omega}$

$c = \frac{\gamma}{-\omega^2 * m + \omega_1^2 * m - di\omega}$

Gedämpfte Schwingung mit festen Parametern numerisch berechnen:

In [2]:
```
var('c1,c2,c3,c4,omega, Lambda,m,y,z,d1, d2 ,d3,a,b,c')

c1=100
c2=120
c3=160
c4=180
m=0.5
d1=0.1
d2=0.15
d3=0.2
import numpy as np
```

$M\ddot{x} + Kx - D\dot{x} = f(t)$

$x = ((ax_{\omega 0}) + (bx_{\omega 1}) + (cx_{\omega 2})e^{i\omega t}$

Matrizen definieren:

In [3]:
```
f=np.array([1,0,0])
```

In [4]:
```
D= Matrix([[d1,0,0],[0,d2,0],[0,0,d3]])
pretty_print(D)
```

$$\begin{pmatrix} 0.100000000000000 & 0.000000000000000 & 0.000000000000000 \\ 0.000000000000000 & 0.150000000000000 & 0.000000000000000 \\ 0.000000000000000 & 0.000000000000000 & 0.200000000000000 \end{pmatrix}$$

In [5]:
```
K= Matrix([[c1+c2,-c2,0],[-c2,c2+c3,-c3],[0,-c3,c3+c4]])
pretty_print(K)
```

$$\begin{pmatrix} 220 & -120 & 0 \\ -120 & 280 & -160 \\ 0 & -160 & 340 \end{pmatrix}$$

In [6]:
```
I= Matrix([[1,0,0],[0,1,0],[0,0,1]])
M= -I*m*Lambda**2
pretty_print(M)
```

$$\begin{pmatrix} -0.500000000000000\,\Lambda^2 & 0.000000000000000 & 0.000000000000000 \\ 0.000000000000000 & -0.500000000000000\,\Lambda^2 & 0.000000000000000 \\ 0.000000000000000 & 0.000000000000000 & -0.500000000000000\,\Lambda^2 \end{pmatrix}$$

Eigenfrequenzen- und Eigenwerte berechnen:

In [25]:
```
K2= np.array(K,dtype=float)/m
```

In [8]:
```
K2
```

Out[8]:
```
array([[ 440., -240.,    0.],
       [-240.,  560., -320.],
       [   0., -320.,  680.]])
```

In [9]:
```
eig, Q = np.linalg.eig(K2)
```

In [10]:
```
eig
```

Out[10]:
```
array([158.74960128, 529.00355066, 992.24684806])
```

In [11]:
```
Q
```

Out[11]:
```
array([[ 0.58814706, -0.75480805,  0.290427  ],
       [ 0.68923582,  0.27991915, -0.66828082],
       [ 0.42312766,  0.5932201 ,  0.68487437]])
```

Eigenvektoren durch Basistransformation ins physikalische Koordinatensystem umwandeln:

In [12]:
```
Q.transpose()
```

Out[12]:
```
array([[ 0.58814706,  0.68923582,  0.42312766],
       [-0.75480805,  0.27991915,  0.5932201 ],
       [ 0.290427  , -0.66828082,  0.68487437]])
```

In [13]:
```
np.dot(Q,Q.transpose())
```

Out[13]:
```
array([[ 1.00000000e+00, -3.75815768e-16,  4.97492571e-17],
       [-3.75815768e-16,  1.00000000e+00,  8.19184217e-17],
       [ 4.97492571e-17,  8.19184217e-17,  1.00000000e+00]])
```

In [14]:
```
f2=np.dot(Q.transpose(),f);f2
```

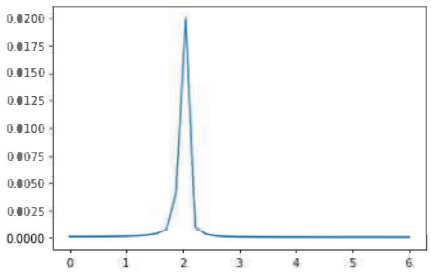
Out[14]:
```
array([ 0.58814706, -0.75480805,  0.290427  ])
```

$$f2 = \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}$$

Amplitude in Abhängigkeit von der Frequenz graphisch darstellen:

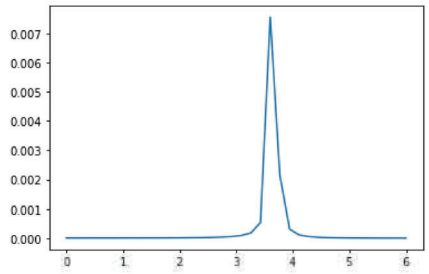
In [15]:
```
def amplitude(alpha,mass,damp,eig,freq):
    a = alpha/(mass*((2*np.pi*freq)**2 - eig**2) - damp*1j)
    return a

n = 6
freqs = np.linspace(0,n,6*n,endpoint=True)
y1 = abs(amplitude(f2[0],m,d1,sqrt(eig[0]),freqs))**2

import matplotlib.pyplot as plt
plt.plot(freqs, y1)
```

Out[15]:
```
[<matplotlib.lines.Line2D object at 0x6fff7ac5d6d0>]
```



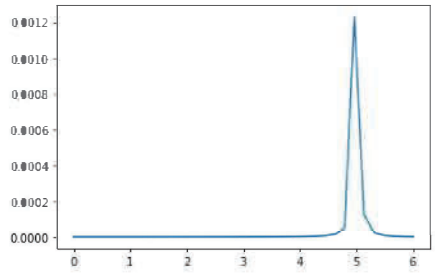
In [16]:
```
def amplitude(alpha,mass,damp,eig,freq):
    b = alpha/(mass*((2*np.pi*freq)**2 - eig**2) - damp*1j)
    return b

n = 6
freqs = np.linspace(0,n,6*n,endpoint=True)
y2 = abs(amplitude(f2[1],m,d2,sqrt(eig[1]),freqs))**2

import matplotlib.pyplot as plt
plt.plot(freqs, y2)
```

Out[16]:
```
[<matplotlib.lines.Line2D object at 0x6fff7ab644d0>]
```



In [17]:
```
def amplitude(alpha,mass,damp,eig,freq):
    c = alpha/(mass*((2*np.pi*freq)**2 - eig**2) - damp*1j)
    return c

n = 6
freqs = np.linspace(0,n,6*n,endpoint=True)
y3 = abs(amplitude(f2[2],m,d3,sqrt(eig[2]),freqs))**2

import matplotlib.pyplot as plt
plt.plot(freqs, y3)
```

Out[17]:
```
[<matplotlib.lines.Line2D object at 0x6fff7a9a1590>]
```



In [18]:
```
np.dot(Q,f2)
```

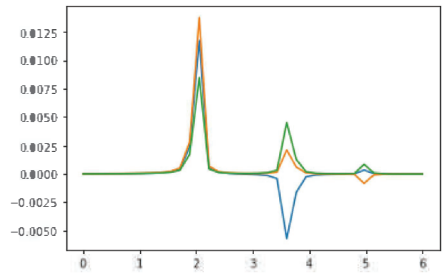
Out[18]:
```
array([ 1.00000000e+00, -3.48060192e-16,  4.97492571e-17])
```

In [19]:
```
x = []
for k in range(len(freqs)):
    x_eig = np.array([y1[k],y2[k],y3[k]])
    x_cart = np.dot(Q,x_eig)
    x.append(x_cart)

x = np.array(x)
```

Alle drei Graphen überlagert und untereinander darstellen:

In [20]:
```
plt.plot(freqs,x[:,0])
plt.plot(freqs,x[:,1])
plt.plot(freqs,x[:,2])
```

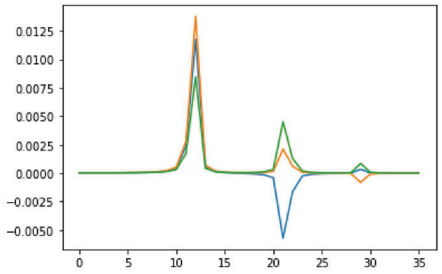
Out[20]:
```
[<matplotlib.lines.Line2D object at 0x6fff7a99b610>]
```



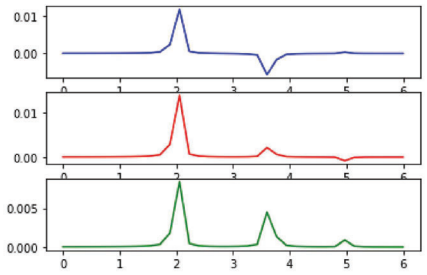
In [21]:
```
x2 = []
Y = np.array([y1,y2,y3])
```

In [22]:
```
x2 = np.dot(Q,Y)
```

In [23]:
```
for k in range(3):
    plt.plot(x2[k,:])
```



In [24]:
```
fig, axs = plt.subplots(3, 1)
colors = ['blue','red','green']
for k in range(3):
    axs[k].plot(freqs,x2[k,:],color=colors[k])
```



Schwingung eines Stabes mit finite Difference

In [15]:
```
var('rho, E, n, w')
```

Out[15]:
```
(rho, E, n, w)
```

Parameter definieren

In [28]:
```
rho = 1
E = 1
n = 2000
w = 0

import numpy as np
```

Schrittweite festlegen

In [29]:
```
h = np.pi/n
```

Steifigkeitsmatrix definieren

In [30]:
```
A_h = 2*np.eye(n)-np.diag(np.ones(n-1),-1)-np.diag(np.ones(n-1),1)
```

In [31]:
```
M = E/(2*h*rho) * A_h ; M
```

Out[31]:
```
array([[ 636.61977237, -318.30988618,    0.        , ...,    0.        ,
            0.        ,    0.        ],
       [-318.30988618,  636.61977237, -318.30988618, ...,    0.        ,
            0.        ,    0.        ],
       [   0.        , -318.30988618,  636.61977237, ...,    0.        ,
            0.        ,    0.        ],
       ...,
       [   0.        ,    0.        ,    0.        , ...,  636.61977237,
         -318.30988618,    0.        ],
       [   0.        ,    0.        ,    0.        , ..., -318.30988618,
          636.61977237, -318.30988618],
       [   0.        ,    0.        ,    0.        , ...,    0.        ,
         -318.30988618,  636.61977237]])
```

Eigenwerte und Eigenfrequenzen berechnen

In [32]:
```
eigw =np.linalg.eig(A_h) [w]
```

In [33]:
```
eigw
```

Out[33]:
```
array([3.97732613, 3.97779517, 3.97825933, ..., 0.36696611, 0.12567693,
       0.06673342])
```

In [34]:
```
Lambda = sqrt(eigw)
```

In [35]:
```
Lambda
```

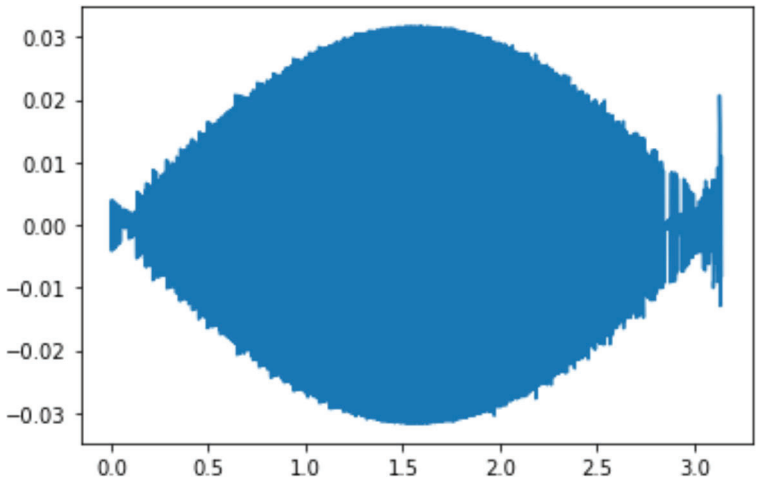
Out[35]:
```
array([1.99432348, 1.99444107, 1.99455743, ..., 0.60577728, 0.35450942,
       0.25832812])
```

Eigenvektoren berechnen und plotten

In [36]:
```
x = h*np.arange(1,n+1)
```

In [37]:
```
y = np.linalg.eig(M) [1][w]
```

In [38]:
```
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.show()
```



Verformung durch Biegemomente

M_b....Bewegungsmoment
dz....neutrale Faser
$d\phi$....Krümmungswinkel
ρ....Krümmungsradius
E....Elastizitätsmodul
I....Flächenträgheitsmoment
y....Abweichung von dz

$dz = |\rho| * d\phi$
EI....Biegesteifigkeit

Die Schwerpunktfaser (y = 0) wird bei der Biegung nicht gedehnt -> deshalb neutrale Faser
Der Krümmungsradius der neutraln Faser beträgt |ρ|
Alle Fasern, die um y von diesem Wert abweichen, verlängern sich um den Betrag von:

$$(|\rho| + y) * d\phi - dz = |\rho| * d\phi - dz = y * d\phi$$

Wobei für y der Abstand zur Schwerpunktfaser eingesetzt wird Die Dehnung wiederum wird beschrieben durch:

$$\epsilon = \frac{y*d\phi}{dz} = \frac{y*d\phi}{|\phi|*d\phi} = \frac{y}{|\rho|}$$

Durch das Hooksche Gesetz $(\sigma_b = E * \epsilon)$ kann auf die Spannung in dieser Faser geschlossen werden.
Jedoch kann auch aus dem Biegemoment auf die Biegespannung geschlossen werden:

$$\sigma_b = \frac{|M_b|}{I} * y = E * \epsilon = E * \frac{y}{|\rho|}$$

Hierbei kann die Querrschnittskordinate (y) herausgehoben werden, wodurch man eine Zusammenhang zwischen Biegemoment und Verformung bekommt:

$$|M_b| = EI * \frac{1}{|\varrho|}$$

-> wobei $\frac{1}{|\rho|}$ der Betrag der Krümmung einer Kurve ist

Ausgehend von dem unverformten Träger lässt sich die Funktion v(z) aufstellen.
Diese zeigt abhängig von der z-Achse den Grad der Verbiegung.
v ist positiv in gleicher Richtung wie y gemessen wird

Der mathematische Zusammenhang zwischen v(z) und der Krümmung der zugehörigen Kurve ist gegeben durch:

$$\frac{1}{\rho} = \frac{v''}{[1+(v')^2]^{\frac{3}{2}}}$$

Ein positives Biegemoment führt zu einer negativen Krümmung. Das Weglassen der Betragsstriche in der Verformungsgleichung führt zu einem Minuszeichen.

$$\frac{v''}{[1+(v')^2]^{\frac{3}{2}}} = -\frac{M_b}{EI}$$

In der Praxis ist jedoch der Anstieg v' sehr gering (der von v'' noch geringer). Dadurch kann der Nenner vernachlässigt werden (es wird duch 1 dividiert) und man erhält für kleine Verformungen die **Differenzialgleichung der Biegelinie 2. Ordnung**:

$$EI * v'' = -M_b$$

-> M_b und EI sind von z abhängig

Folgende Differenzialbeziehungen sind aus der Statik bekannt:

$$M_b' = F_Q$$
$$F_Q' = -q$$
$$M_b'' = F_Q\prime = -q$$

q....eine auf den Träger wirkende Linienlast
F_Q....Querkraft auf der Schnittebene

- *Einschub: durch zweimaliges Differenzieren der Differenzialgleichung der Bieglinie 2. Ordnung erhält man:*
$(EI * v'')'' = -M''$

b

Aus diesen Beziehungen ergiebt sich für kleine Verformungen die
Differenzialgleichung der Biegelinie 4. Ordnung:

$$(EI * v'')'' = q$$

Bei konstanter EI(Biegesteifigkeit) kann die Differenzialgleichung vereinfacht werden zur **Differenzialgleichung der Biegelinie 4. Ordnung bei konstanter Biegesteifigkeit**:

$$EI * v'''' = q$$

Statisch unbestimmte Systeme

Die statischen Gleichgewichtsbedingungen allein reichen bei statisch unbestimmten Problemen für die Berechnung von Lagerreaktionen und Schnittgrößen nicht aus.
Durch das formulieren weiterer Rand- oder Übergangsbedingungen für jedes zusätzliche Lager ist dies jedoch möglich.

Der Biegemomentenverlauf kann beim Rechnen mit der **Differentialgleichung der Biegelinie der 2. Ordnung** nur mit noch unbekannten Lagerreaktionen, wie der Lagerkraft F_B des Lagers B, aufgeschrieben werden, wobei die übrigen Lagerreaktionen mit Hilfe der statischen Gleichgewichtsbedingungen eliminiert werden müssen.
Durch das Integrieren stehen 3 Randbedingungen für die 3 Unbekannten (2 Integrationskonstanten, 1 Lagerkraft) zur Verfügung. Dadurch können folglich auch die noch unbekannte Lagerkraft F_B und die Schnittgrößen berechnet werden.

Da die Lagerreaktionen in der Differenzialgleichung nicht vorkommen, kommt man beim Rechnen mit der **Differentialgleichung der Biegelinie 4. Ordnung** mit 4 Randbedingungen aus. Sind die Integraitonsbekannten bestimm, können die Schnittgrößen mit

$$M_B = -EI * v''$$
$$F_Q = -(EI * v'')'$$

und die Lagerreaktionen aus den Gleichgewichtsbedingungen mit den Schnittgrößen an den Lagerstellen bestimmt werden.

Das Differenzenverfahren

Bei dem Differenzenverfahren wird anstatt mit dem Differenzialquotienten mit dem Differenzenquotienten gerechnet. Diese werden für ausgewählte Punkte (Stützstellen) aufgeschrieben, ein Gleichungssystem wird gebildet und somit die Funktionswerte der Stützstellen berechnet. Für bessere Genauigkeit der Ergebnisse sollten die Stützstellennah beieinander liegen.

Vorerst wird die x-Achse in äquidistante Abstände h unterteilt. Dann kann der Differentialquotient als Grenzwert des Differenzenquotienten an einer beliebigen Stützstelle i bei x_i ermittelt werden. Dies kann auf unterschiedliche Weisen erfolgen, wobei die Tangentensteigung jeweils durch die Sekantensteigung erstetzt wird:

$y_i' \approx (\frac{\Delta y}{\Delta x})_i = \frac{y_{i+1}+y_i}{h}$ -> vowärts genommene Differenzenformel

$y_i' \approx (\frac{\Delta y}{\Delta x})_i = \frac{y_i+y_{i-1}}{h}$ -> rückwärts genommene Differenzenformel

Durch die beidseitige Annäherung liefert eine Formel einen zu großen, die andere einen zu kleinen Wert. Zu einer besseren Näherung wird deshalb das arithmetische Mittel benötigt, woraus sich diese Formel ergiebt:

$y_i' \approx (\frac{\Delta y}{\Delta x})_i = \frac{1}{2}(\frac{y_{i+1}-y_i}{h} + \frac{y_i-y_{i-1}}{h}) = \frac{y_{i+1}-y_{i-1}}{2h}$ -> zentrale Differenzenformel

Für höhere Ableitungen ergeben sich diese **zentrale Differenzenformeln**:

$y_i' \approx \frac{1}{2h}(-y_{i-1} + y_{i+1})$
$y_i'' \approx \frac{1}{h^2}(y_{i-1} - 2y_i + y_{i+1})$
$y_i''' \approx \frac{1}{2h^3}(-y_{i-2} + 2y_{i-1} - 2y_{i+1} + y_{i+2})$
$y_i'''' \approx \frac{1}{h^4}(y_{i-2} - 4y_{i-1} + 6y_i - 4y_{i+1} + y_{i+2})$

Die 4. Ableitung aus $EI * v'''' = q(z)$ wird durch die entsprechende Differenzenformel angenähert und man erhält die
Differenzengleichung der Biegelinie bei konstanter Biegesteifigkeit:

$v_{i-2} - 4v_{i-1} + 6v_i - 4v_{i+1} + v_{i+2} = \frac{q_i * h^4}{EI}$

Da in den Randbedingungen die Differenzialquotienten durch die Differenzenformeln erstetzt werden, benötigt man die
Differenzengleichungen für Biegemoment und Querkraft bei konstanter Biegesteifigkeit:

$$M_{bi} = -\frac{EI}{h^2}(v_{i-1} - 2v_i + v_{i+1})$$
$$F_{Qi} = -\frac{EI}{2h^3}(-v_{i-2} + 2v_{i-1} - 2v_{i+1} + v_{i+2})$$

Bei der Formel für die Biegelinie mit veränderlicher Biegesteifigkeit $(EI * v'')'' = q$ werden die Ableitungen in zwei Schritten durch die Differenzengleichungen ersetzt, wobei beachtet werden muss, dass dies an drei verschiedenen Punkten geschieht und jeweils für diese Punkte die Nachbarspunkte in die Formel eingehen:

$$[EIv'']_{i-1} \approx EI_{i-1}\frac{1}{h^2}(v_{i-2} - 2v_{i-1} + v_i)$$
$$[EIv'']_{i} \approx EI_{i}\frac{1}{h^2}(v_{i-1} - 2v_{i} + v_{i+1})$$
$$[EIv'']_{i+1} \approx EI_{i}\frac{1}{h^2}(v_{i} - 2v_{i+1} + v_{i+2})$$

Daraus ergibt sich die
Differenzengleichung der Biegelinie bei veränderlicher Biegesteifigkeit:

$$I_{i-1}v_{i-2} - 2(I_{i-1} + I_i)v_{i-1} + (I_{i-1} + 4I_i + I_{i+1})v_i - 2(I_i + I_{i+1})v_{i+1} + I_{i+1}v_{i+2} = \frac{q_i h^4}{E}$$

Bei dem Biegemoment ergibt sich die gleiche Differenzenformel
$M_b(z) = -EI(z) * v''(z)$
wie bei konstanter Biegesteifigkeit, jedoch müssen bei der Querkraft die Differenzialquotienten ersetzt werden.

$$F_{Qi} \approx -\frac{1}{2h}\{-[EI'']_{i-1} + [EI'']_{i+1}\} = -\frac{1}{2h}\{-[EI_{i-1}\frac{1}{h^2}(v_{i-2} - 2v_{i-1} + v_i)] + [EI_{i+1}\frac{1}{h^2}(v_i - 2\imath$$

Diese Formel wird nach den Verschiebungen (v_i,v_{i+1},...) geordnet und man erhält die
Differentialgleichungen für die Schnittgrößen bei veränderlicher Biegesteifigkeit:

$$M_{bi} = -\frac{EI_i}{h^2}(v_{i-1} - 2v_i + 2_{i+1})$$
$$F_{Qi} = \frac{E}{2h^3}[I_{i-1}v_{i-2} - 2I_{i-1}v_{i-1} + (I_{i-1} - I_{i+1})v_i + 2I_{i+1}vi + 1 - I_{i+1}v_{i+2}]$$

Beim Einsetzen für die Ableitung von $I(z)$ gehen die Werte I_{i-1} und I_{i+1} in die Differenzengleichung ein, obwohl diese nicht innerhalb des Trägers liegen. Aufgrund dessen müssen stellvertretent jene Werte eingesetzt werden, die sich beim Erweitern des Definitionsbereichs der Funktion $I(z)$ ergeben würden.\ Für sprunghafte Änderungen der Biegesteifigkeit, welche dirket auf einem Punkt des Differenzenschemas (z.B.: i) liegen, muss das arithmetische Mittel der Biegesteifigkeit der zwei betroffenen Teilabschnitte genommen werden.

Zur Erleichterung des Rechnens kann man das Differenzenschema so anlegen, dass ein Gelenk genau auf einen Punkt i fällt und man somit die Biegesteifigkeit $EI_i = 0$ setzen kann.

Anwendung

```
In [35]: var('y,rho,dphi,dz,Mb,E,I,v,z,q,l,FQ')
Out[35]: (y, rho, dphi, dz, Mb, E, I, v, z, q, l, FQ)
```

In [78]:

```
def extention(y,dphi):
    ex = y*dphi
    return ex

def stretch(y,rho=False,dphi=False,dz=False):
    if abs(dphi) > 0  and abs(dz) > 0 and abs(y) > 0:
        eps = (y*dphi)/dz
    elif abs(y) > 0  and abs(rho) > 0:
        eps = y/abs(rho)
    return eps

def absBm(E,I,rho):
    a_Mb = E*I*(1/rho)
    return a_Mb

def FTension(y,a_Mb=False,I=False,E=False,eps=False,rho=False):
    if a_Mb > 0  and abs(I) > 0 and abs(y) > 0:
        f_tension = (a_Mb/I)*y
    elif abs(E) > 0 and abs(eps) > 0:
        f_tension = E*eps
    elif abs(E) > 0 and abs(y) > 0 and abs(rho) > 0:
        f_tension = E*(y/abs(rho))
    return f_tension

def Diffgl_2(EI,vd_2):
    m_Mb = EI*vd_2
    return m_Mb

def Diffgl_4(EI,vd_2):
    q = derivative(EI*vd_2,z,2)
    return q

def Diffgl_4k(EI,vd_2):
    q = EI*(derivative(vd_2,z,4))
    return q

#--------------------main-------------------------

z = 2

v(z) = z^7
vd_2 = derivative(v(z),z,2)

y = 3
dphi = 12
dz = 2
rho = 1.15
E = 2
I = 27
EI = E*I

eps = stretch(y,rho,dphi,dz)
a_Mb = absBm(E,I,rho)
f_tension = FTension(y,a_Mb,I,E,eps,rho)
diffgl_2 = Diffgl_2(EI,vd_2)
diffgl_4 = Diffgl_4(EI,vd_2)
diffgl_4k = Diffgl_4k(EI,vd_2)

print("Verlängerung der Faser: ", extention(y,dphi))
```

```
print("Dehnung der Faser: ", eps)
print("Spannung der Faser: ", f_tension)
print
("----------------------------------------------------------------------")
print("Betrag des Biegemoments: ", a_Mb)
print("Differentialgleichung der Biegelinie 2. Ordnung: ", diffgl_2)
print("Differentialgleichung der Biegelinie 4. Ordnung: ", diffgl_4)
print("Differentialgleichung der Biegelinie 4. Ordnung bei konstanter Bie
gsteifigkeit: ", diffgl_4k)
```

```
Verlängerung der Faser:  36
Dehnung der Faser:  18
Spannung der Faser:  5.21739130434783
----------------------------------------------------------------------
Betrag des Biegemoments:  46.9565217391304
Differentialgleichung der Biegelinie 2. Ordnung:  2268*z^5
Differentialgleichung der Biegelinie 4. Ordnung:  45360*z^3
Differentialgleichung der Biegelinie 4. Ordnung bei konstanter Biegsteif
igkeit:  272160*z
```

In [2]:
```
import numpy as np
n = 100
K = np.zeros((n,n))

stamp = [1,-4,6,-4,1]

for k in range(2,n-2):
    K[k,k-2:k-2+len(stamp)] = stamp


# Randbedingungen
K[0,2] = 1
K[1,1] = -1; K[1,3] = 1
K[-2,-1] = 1; K[-2,-2] = -2; K[-2,-4] = 2; K[-2,-5] = -1
K[-1,-2] = 1; K[-1,-3] = -2; K[-1,-4] = 1
```

In [15]:
```
rho = 1
EI = 1000
l = 1
h = l/(n-4)
```

In [16]:
```
K2 = (K/h/rho/EI)[2:-2,2:-2]
```

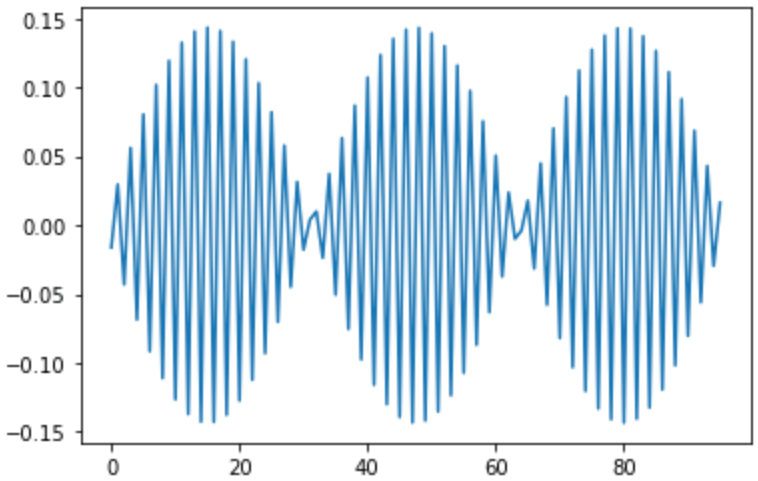
In [17]:
```
eig, Q = np.linalg.eig(K2)
```

In [18]:
```
eig
```

Out[18]:
```
array([1.53519942e+00, 1.53279977e+00, 1.52880731e+00, 1.52323246e+00,
       1.51608972e+00, 1.50739767e+00, 1.49717890e+00, 1.48545989e+00,
       1.47227094e+00, 1.45764608e+00, 1.44162291e+00, 1.42424252e+00,
       1.40554928e+00, 1.38559074e+00, 1.36441744e+00, 1.34208273e+00,
       1.31864259e+00, 1.29415544e+00, 1.26868190e+00, 1.24228466e+00,
       1.21502818e+00, 1.18697849e+00, 1.15820301e+00, 1.12877027e+00,
       1.09874969e+00, 1.06821136e+00, 1.03722577e+00, 1.00586362e+00,
       9.74195563e-01, 9.42291968e-01, 9.10222700e-01, 8.78056890e-01,
       8.45862714e-01, 8.13707180e-01, 7.81655917e-01, 7.49772972e-01,
       7.18120615e-01, 6.86759154e-01, 6.55746761e-01, 6.25139298e-01,
       5.94990170e-01, 5.65350172e-01, 5.36267360e-01, 5.07786928e-01,
       4.79951101e-01, 4.52799037e-01, 4.26366744e-01, 4.00687015e-01,
       3.75789368e-01, 3.51700007e-01, 3.28441792e-01, 3.06034228e-01,
       2.84493464e-01, 2.63832306e-01, 2.44060248e-01, 2.25183507e-01,
       2.07205085e-01, 1.90124830e-01, 1.73939518e-01, 1.58642946e-01,
       1.44226034e-01, 1.30676941e-01, 1.17981186e-01, 1.06121790e-01,
       9.50794123e-02, 8.48325103e-02, 7.53574957e-02, 6.66289038e-02,
       5.86195676e-02, 5.13007970e-02, 4.46425632e-02, 3.86136868e-02,
       3.31820284e-02, 2.83146823e-02, 2.39781702e-02, 2.01386358e-02,
       1.67620394e-02, 1.38143498e-02, 1.12617350e-02, 9.07074941e-03,
       7.20851689e-03, 5.64290922e-03, 4.34271898e-03, 3.27782608e-03,
       2.41935727e-03, 1.73983781e-03, 1.21333490e-03, 8.15591972e-04,
       5.24153485e-04, 3.18479572e-04, 1.80050140e-04, 9.24579619e-05,
       4.14903575e-05, 1.51994905e-05, 3.95771171e-06, 5.21043268e-07])
```

In [19]:
```
import matplotlib.pyplot as plt
plt.plot(Q[:,2])
```

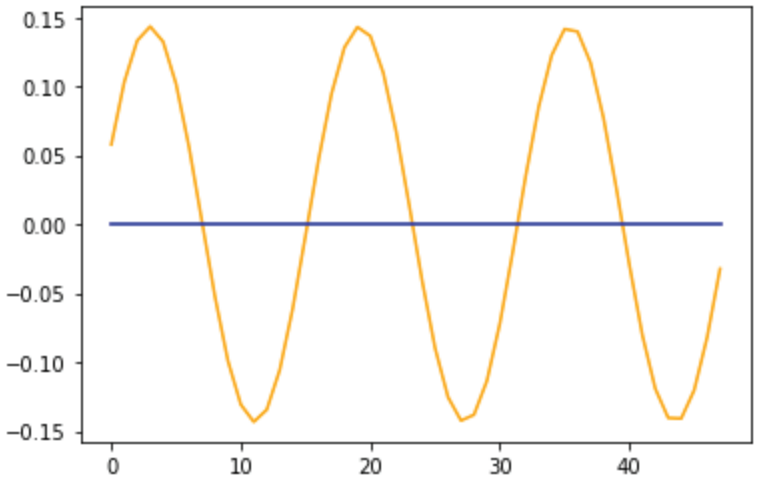
Out[19]:
```
[<matplotlib.lines.Line2D object at 0x6ffed4a5d450>]
```



In [20]:
```
import matplotlib.pyplot as plt
```

In [39]:
```
#plt.plot(np.zeros(n),color = "blue")
#plt.plot(np.arange(n),np.zeros(n),marker="o", color = "blue")
v=Q[1::2,5]
plt.plot(v,color="orange")
plt.plot(np.zeros_like(v),color = "blue")
```

Out[39]:
```
[<matplotlib.lines.Line2D object at 0x6ffed367e6d0>]
```



Optimierung eines Trägers in Schwingung

Ausgangssituation:

Gegeben ist ein waagrechter Träger mit einer Länge von 100 cm und einer rechteckigen Querschnittsfläche mit einer Höhe von 10 cm. Das eine Ende ist fixiert, auf dem anderen ist eine Masse, damit er in Schwingung versetzt wird. Auf dem fixierten Ende ist orthogonal ein Balken befestigt, welcher eine Länge von 10 cm hat.

Ziel:

Ziel ist es, eine Verbindung der beiden Bauteile zu finden, damit der waagrechte Träger möglichst wenig schwingt. Diese Verbindung soll oberhalb des Trägers verlaufen.

Problemlösung:

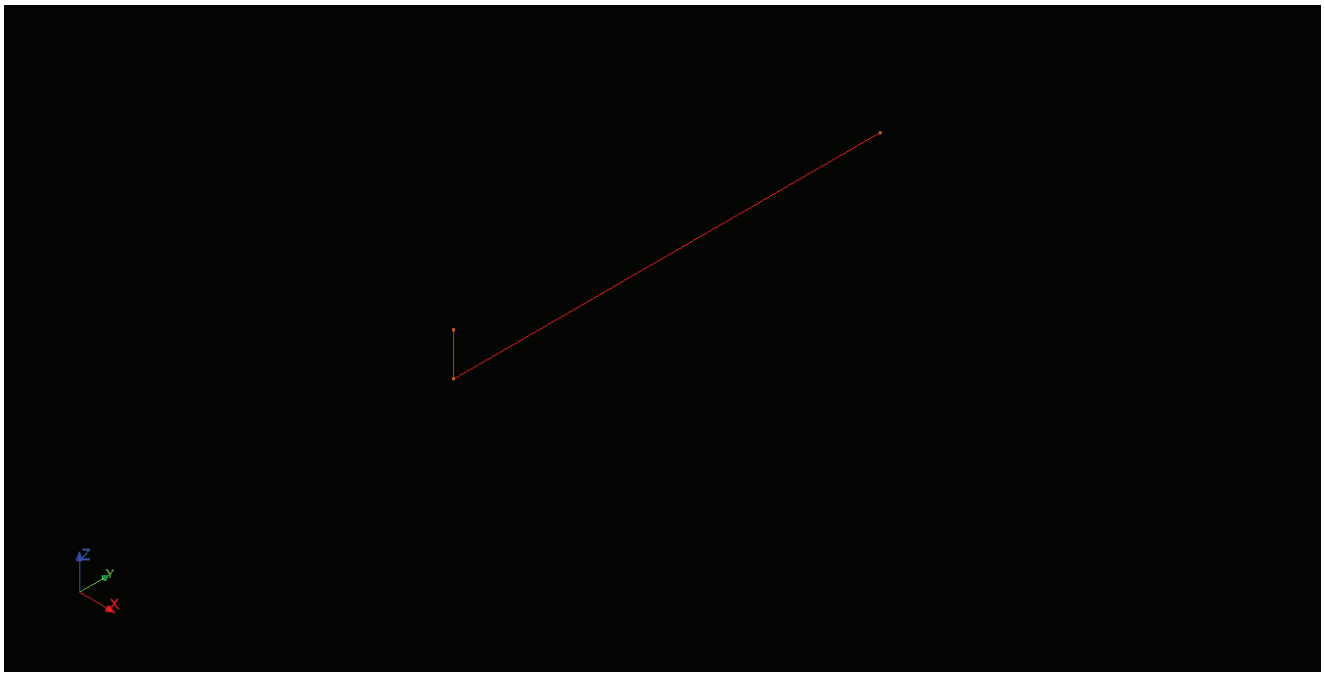
Um die größtmögliche Stabilität zu erreichen, ist eine hohe Eigenfrequenz von Nöten. Je höher die Eigenfrequenz ist, desto steifer ist das Objekt. Deshalb muss die Kurve so strukturiert werden, damit die Eigenfrequenz möglichst hoch ist.
Um das Maximum der Eigenfrequenz zu erlangen, sucht man das Minimum (denn:
$-max f(x) = min(-f(x))$

In [ ]:
```
WORK_DIR="C:\\Users\\oring\\Desktop\\OLIVIA\\Privat\\23_Modellierungswoch
e\\beam\\"
OUT_FILE = WORK_DIR + "out.txt"

def functional(x):
    with open(OUT_FILE,"a") as f:
        f.write(f"Current x: {x}\n")
    eig = eigenval(x)
    val = -np.log(eig) # Funktionalwert wird durch den negativen Logarith
mus berechnet
    if np.isnan(val):
        return MAX

    if np.any(x<=0):
        return MAX

    if np.any(x>H+2):
        return MAX
    print("Current x: ",x,"Current_val: ",val)

    return val


x = np.array([1,2,3,4,1.0])
```

Geometrie mit Salome:

Zuerst werden 3 Punkte (einer im Ursprung, die anderen am Ende der Balken) erstellt und mit zwei Linien verbunden.

In [ ]:
```
L = 100
H = 10

p0 = geompy.MakeVertex(0., 0., 0.)
p1 = geompy.MakeVertex(0., L, 0.)
p2 = geompy.MakeVertex(0., 0., H)

id_p0 = geompy.addToStudy(p0,"Vertex p0")
id_p1 = geompy.addToStudy(p1,"Vertex p1")
id_p2 = geompy.addToStudy(p2,"Vertex p2")

line_1 = geompy.MakePolyline([p0,p1])
line_2 = geompy.MakePolyline([p0,p2])

id_line_1 = geompy.addToStudy(line_1,"Line_1")
id_line_2 = geompy.addToStudy(line_2,"Line_2")
```



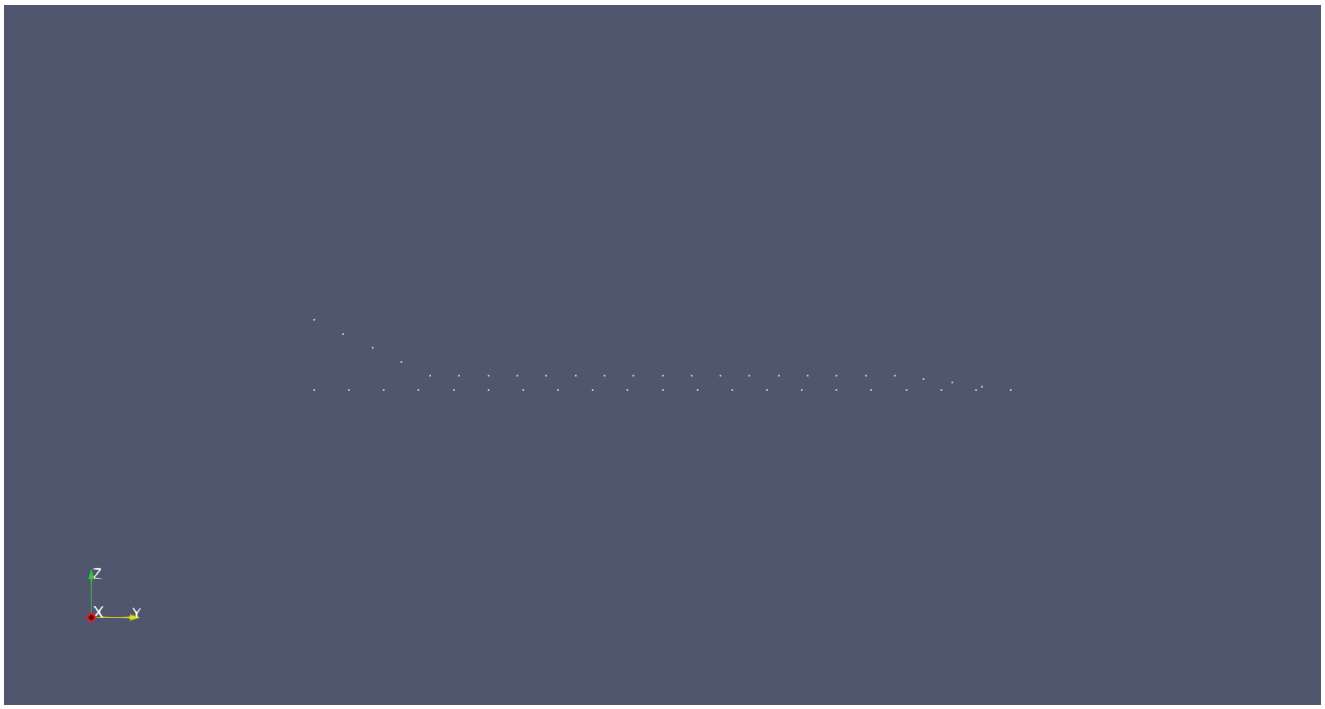
Dann werden 5 Punkte erstellt, welche auch (mit p2 und p1) verbunden werden und somit das Grundmodell bilden.
-> diese werden später beim Optimieren angepasst

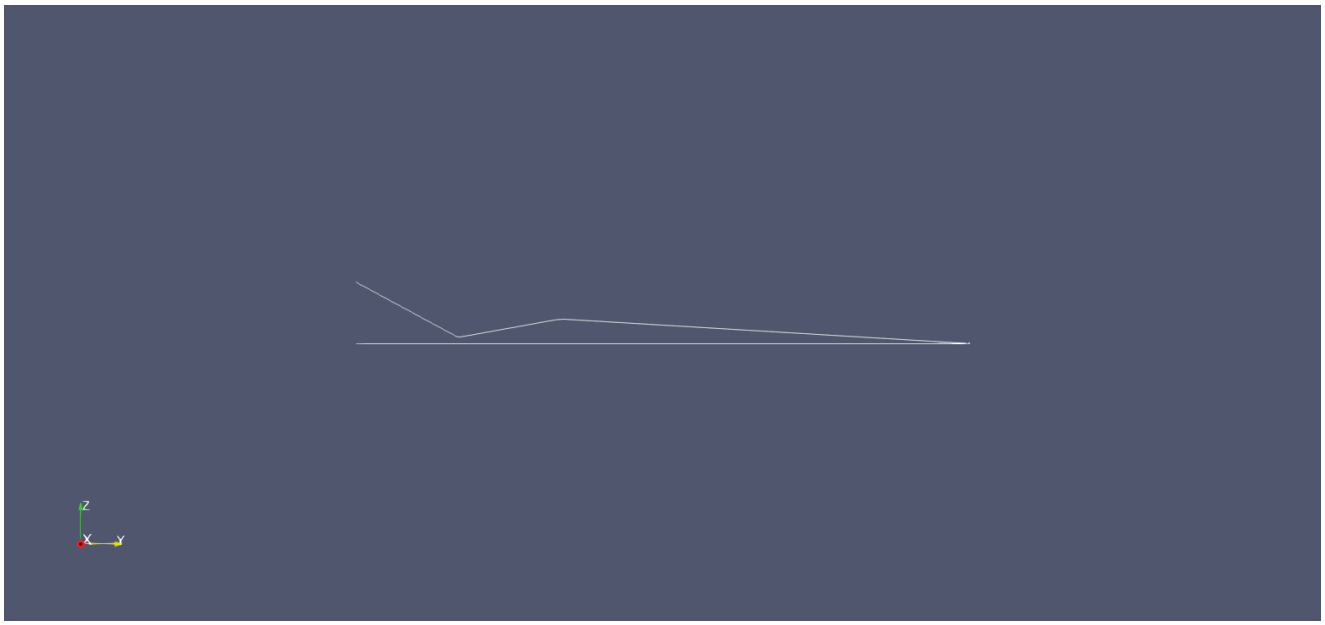
In [ ]:
```
def create_topline_points(x):
    points = []
    nr_points = len(x)
    h = L/(nr_points+1)
    for k in range(nr_points):
        p = geompy.MakeVertex(0., (nr_points - k)*h, x[k])
        points.append(p)
    return points

def create_topline(x):
    points = create_topline_points(x)
    points = [p2] + points[::-1] + [p1]
    polyline = geompy.MakePolyline(points)
    return polyline

polyline = create_topline(x)

id_line = geompy.addToStudy(polyline,"Line")
```

Aus den Punkten bzw. aus den Linien wird ein Mesh erstellt.

In [ ]:

```
PATH="C:\\Users\\oring\\Desktop\\OLIVIA\Privat\\23_Modellierungswoche\\beam"

def create_polymesh(x,fname=f"{PATH}\\structure.med",local_length=5.):
    polyline = create_topline(x)
    mesh1 = create_meshline(polyline,local_length=local_length)
    mesh1.GroupOnGeom(p2, "fix1")
    mesh1.GroupOnGeom(p1, "mass")
    mesh2 = create_meshline(line_1,local_length=local_length)
    mesh2.GroupOnGeom(p0, "fix2")

    mesh = smesh.Concatenate([mesh1, mesh2], 0, 1, 1e-01, name='mesh')

    edges = mesh.CreateEmptyGroup(SMESH.EDGE, "edges")
    edges.AddFrom(mesh.mesh)

    nodes = mesh.CreateEmptyGroup(SMESH.NODE, "edges")
    nodes.AddFrom(mesh.mesh)

    group=mesh.GetGroupByName("mass")[0]
    node_id=group.GetNodeIDs()[0]
    new_id = mesh.Add0DElement(node_id)
    el0d = mesh.CreateEmptyGroup(SMESH.ELEM0D, "mass")
    el0d.AddFrom(mesh.mesh)
    mesh.ExportMED(fname,40)
    return mesh

def create_meshline(line,local_length=5.):
    mesh = smesh.Mesh(line)
    regular1D = mesh.Segment()
    segLen10 = regular1D.LocalLength(local_length)
    mesh.Compute()
    #mesh.ExportMED(fname,40)
    return mesh


mesh=create_polymesh(x)
create_meshline(line_1)
```





Rechnen mit Code_Aster:

Materialeigenschaften und Randbestimmungen werden bestimmt.

In [ ]:

```
import numpy as np
WORK_DIR="C:\\Users\\oring\\Desktop\\OLIVIA\\Privat\\23_Modellierungswoch
e\\beam\\"
DEBUT(PAR_LOT="NON");

mesh=LIRE_MAILLAGE ( INFO=1 ,UNITE=20 ,FORMAT="MED");

#mesh=DEFI_GROUP(reuse =mesh, MAILLAGE=mesh,
#                 CREA_GROUP_NO = ( _F(TOUT_GROUP_MA='OUI',),)
#                  );



model=AFFE_MODELE(MAILLAGE=mesh,
                  AFFE=(_F(
                           GROUP_MA = ( 'edges',),
                           PHENOMENE='MECANIQUE',
                           MODELISATION='POU_D_T' ,
                          ),
                    _F(GROUP_MA = ( 'mass',),
                       PHENOMENE='MECANIQUE',
                       MODELISATION='DIS_T',
                       ),
                     ),);
steel=DEFI_MATERIAU(   ELAS=_F( E = 2.E+11,
                                NU = 0.3,
                                RHO = 8000.),);
material=AFFE_MATERIAU(MAILLAGE=mesh,
                       AFFE=_F(GROUP_MA=('edges',),
                       MATER=steel,),
                        );

a = 0.1#( 1.0 /8/10/2)**0.5*100;
elemcar=AFFE_CARA_ELEM (MODELE=model,
                        POUTRE =_F(
          GROUP_MA = ('edges',) ,
          SECTION='RECTANGLE',
          CARA = ( 'HY' , 'HZ' , ) ,
          VALE = ( a , 2*a ),
                  ),
                  DISCRET=_F(
                      GROUP_MA=( 'mass', ) ,
                      CARA='M_T_D_N',
                      VALE = 1.0,
                        ),
                    );
ground=AFFE_CHAR_MECA (
            MODELE=model,
            DDL_IMPO=(_F(
            GROUP_NO = ('fix1',"fix2"),
            DX=0 ,DY=0 ,DZ=0,
            DRX=0 ,DRY=0 ,DRZ=0,
            ),
            _F(GROUP_NO="edges",
            DY=0,DRY=0,
               ),
            )

            );
```

```
massin=POST_ELEM(
             MODELE=model ,CHAM_MATER=material , CARA_ELEM=elemcar ,
             MASS_INER=_F ( TOUT='OUI' , ) ,
             TITRE= 'massin' ,
              );
IMPR_TABLE(TABLE=massin,)

#modal analysis
ASSEMBLAGE(MODELE=model ,
           CHAM_MATER=material ,
           CARA_ELEM=elemcar ,
            CHARGE=ground ,
             NUME_DDL=CO ( 'numdof' ) ,
            MATR_ASSE = (
          _F( MATRICE=CO ( 'rigidity' ) , OPTION='RIGI_MECA' , ) ,
           _F ( MATRICE=CO ( 'masse' ) , OPTION='MASS_MECA' , ) ,
           ),
           );
modes=CALC_MODES(TYPE_RESU='DYNAMIQUE', #for a modal analysis
                 MATR_MASS=masse,
                 MATR_RIGI=rigidity,
                 OPTION='PLUS_PETITE' ,
                 CALC_FREQ=_F(NMAX_FREQ=8,),
                 #some of the lines in the following VERI_MODE
                 #may have to be uncommented
                 VERI_MODE=_F (
                 #PREC_SHIFT=5.0000000000000001E-3,
                 #STOP_ERREUR='NON',
                 #STURM='OUI',
                 #SEUIL=9.9999999999999995E-07,
                  ),
                 );

listeigenfreq = modes.LIST_VARI_ACCES()['FREQ']
lowest = listeigenfreq[0]
np.save(WORK_DIR+"eig.npy",[lowest])


#printing of the mode shapes in a med file
IMPR_RESU (
FORMAT='MED' ,UNITE=80 ,
RESU=_F(RESULTAT=modes,
        NOM_CHAM='DEPL',),
        );
#printing the values in the .resu file
#for the RESU "modes"
IMPR_RESU(MODELE=model,
          FORMAT='RESULTAT',
          RESU=_F(RESULTAT=modes,
          #this prints all the info relative to modal analysis
          TOUT_CHAM='NON', #with this we do not print DEPL
          TOUT_PARA='OUI', #prints all the parameter
#next lines print only the specified parameter
#NOM_PARA=(
#'FREQ','MASS_GENE',
#'MASS_EFFE_DX','MASS_EFFE_DY',
#),
#FORM_TABL='OUI', #optional
            ),
         );
```

```
FIN();
```

Optimierung mit Python:

Anfangs wird die Range durch "rranges" festgelegt und anschließend wird "optimize" von "scipy" importiert.
-> "resbrute" ist die Art der Optimierung
In diesem Fall wird jede mögliche Geometrie in den vorgegeben Begrenzungen berechnet und anschließend ausgegeben.

In [ ]:
```
res = functional(x)
rranges = (slice(0.1, 4.1, 0.5), slice(0.1, 4.1, 0.5),slice(0.1, 4.1, 0.
5),slice(0.1, 4.1, 0.5),slice(0.1, 4.1, 0.5))

from scipy import optimize

resbrute = optimize.brute(functional, rranges, full_output=True,
                          finish=optimize.fmin)

global minimum
print("res: ", resbrute)
```

Ergebnis und Interpretation:

Der schwingende Träger ist auf der Seite, welche näher zur Masse liegt, verstärkt. Der andere Teil erlangt durch eine kurvenähnliche Konstruktion Stabilität.

- Orginiale Eigenfrequenzen:

numéro d'erreur	fréquence (HZ)	norme
1	7.96958E-03	2.07381E-11
2	3.53535E-02	1.54264E-10
3	5.01181E-02	1.20589E-11
4	5.11483E-02	2.09591E-11
5	1.14214E-01	6.05406E-12
6	1.40097E-01	5.43980E-12
7	1.41642E-01	4.63577E-12
8	2.35005E-01	4.98900E-12

Norme d'erreur moyenne : 2.86423E-11

$7,96958e^{-3}$ -> kleinste Eigenfrequenz der Startkonfiguration

- Optimierte Eigenfrequenzen:

numéro d'erreur	fréquence (HZ)	norme
1	8.03073E-03	3.29458E-12
2	5.03976E-02	7.44513E-12
3	1.41052E-01	5.37974E-12
4	1.68324E-01	4.29377E-12
5	2.03599E-01	1.60410E-12
6	2.76400E-01	3.47062E-12
7	3.94013E-01	4.88880E-12
8	4.56691E-01	4.63096E-12

Norme d'erreur moyenne : 4.37596E-12

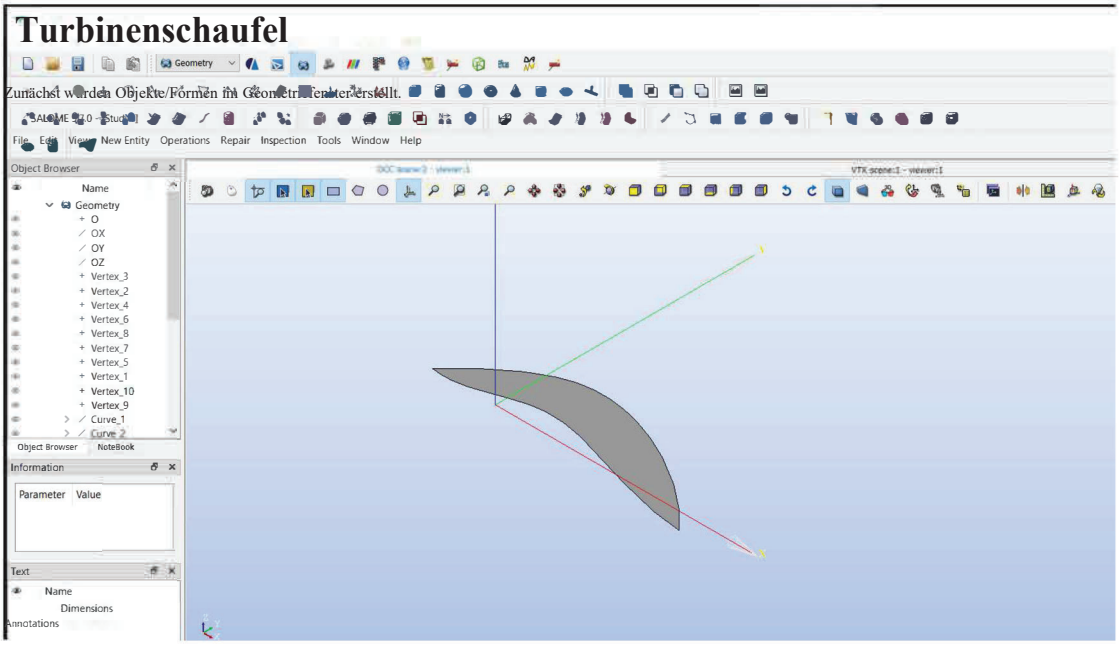
$8,03073e^{-3}$ -> kleinste Eigenfrequenz der optimierten Konfiguration

Das Modellieren von Objekten

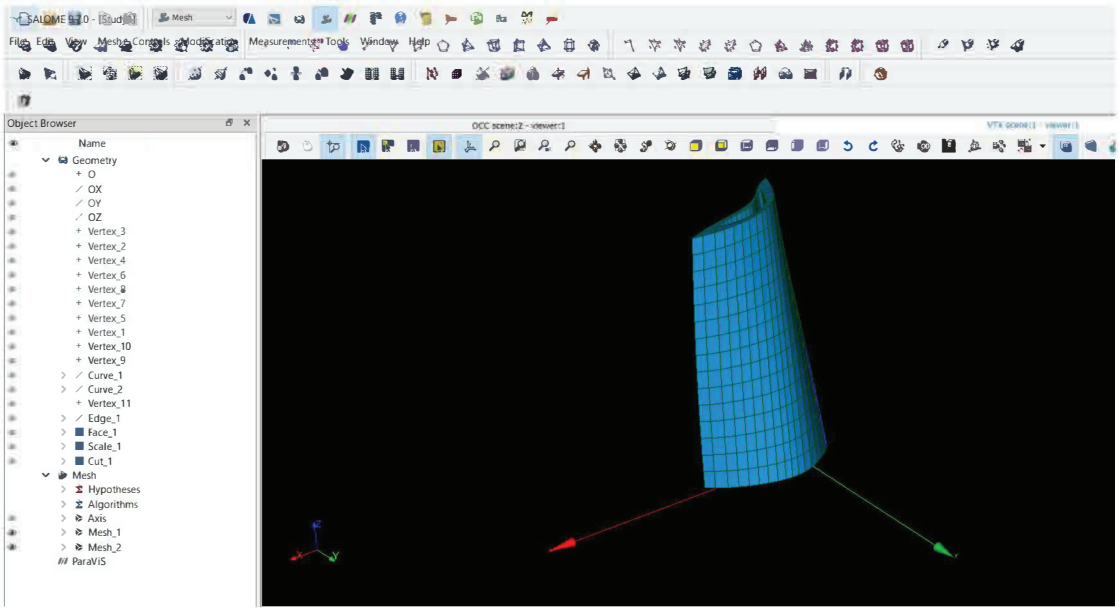
Um die Modelle zu erzeugen und im Post-Processing-Programm auszuwerten, wurde das Programm Salome verwendet. Um die Daten für das Post-Processing-F wurden die Modelle von dem Programm Aster durchgerechnet.

Turbinenschaufel

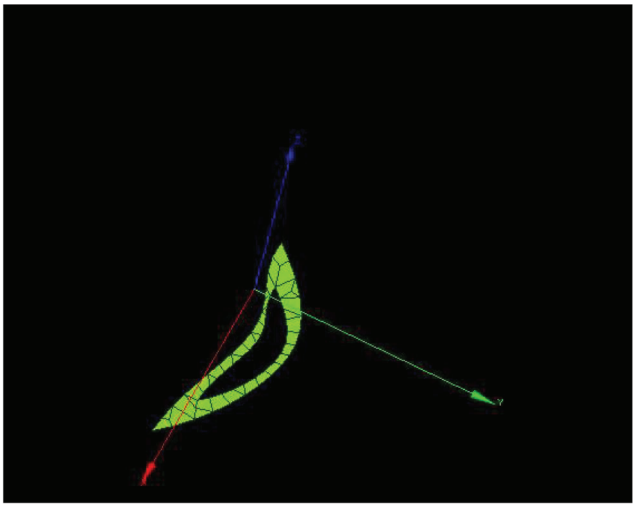
Zunächst wurden Objekte/Formen im Geometrie-Fenster erstellt.

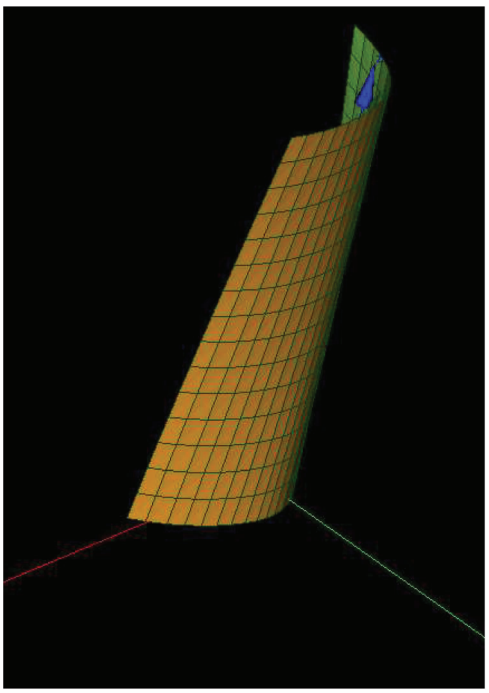


Daraufhin wurde diese Form an einer Linie im Mesh-Fenster extrudiert oder dirket in ein Mesh umgewandelt.



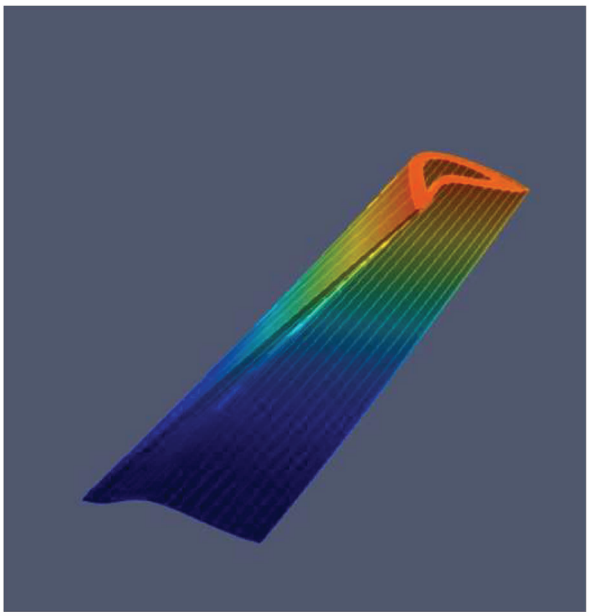
Anschließend wurden die fixierten Flächen und die durch eine Kraft belasteten Flächen festgelegt.



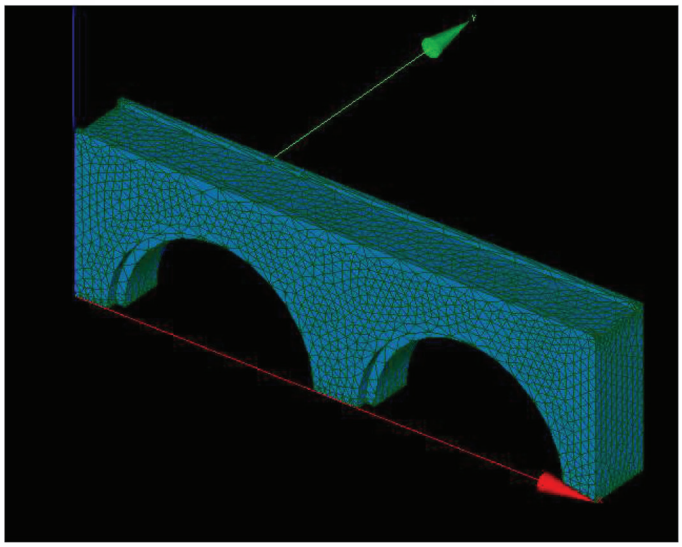


Diese Fläche stellt die Front des Turbinenblättchens dar und wird mit den jeweiligen frequenzen belastet.

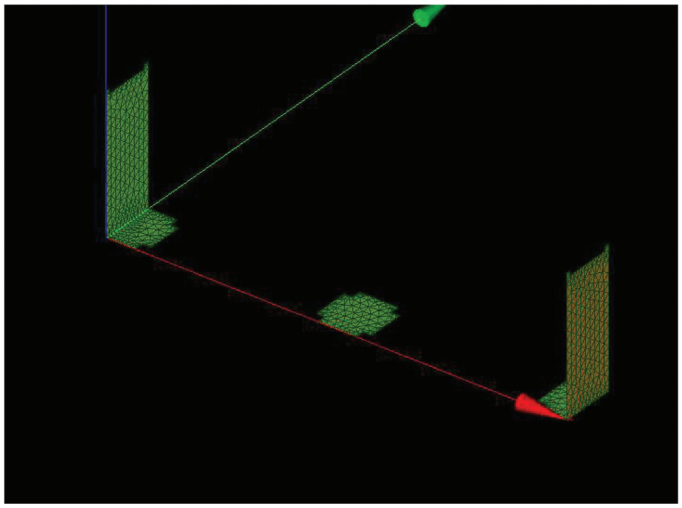
Das Volumen und ein beliebiger Punkt des Objekts müssen zusätzlich ausgewählt und richtig benannt werden. Als nächstes wurde die Mesh-Datei von dem Aster **eine Datei zu erzeugen, die die notwendigen Daten für das Post-Processing-Programm besitzt. Diese wurde wiederrum wieder in das ParaVis-Fenster von Salom** Animationen, in denen das jeweilige Objekt mit verschiedenen Frequenzen an der gewählten Stelle belastet wird, werden die Schwingungen des Objekts in Farbe des Objekts dargestellt.



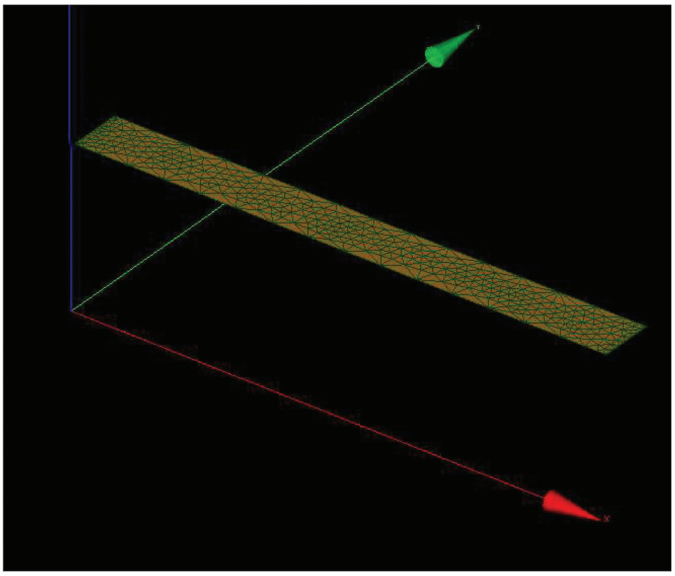
Brücke



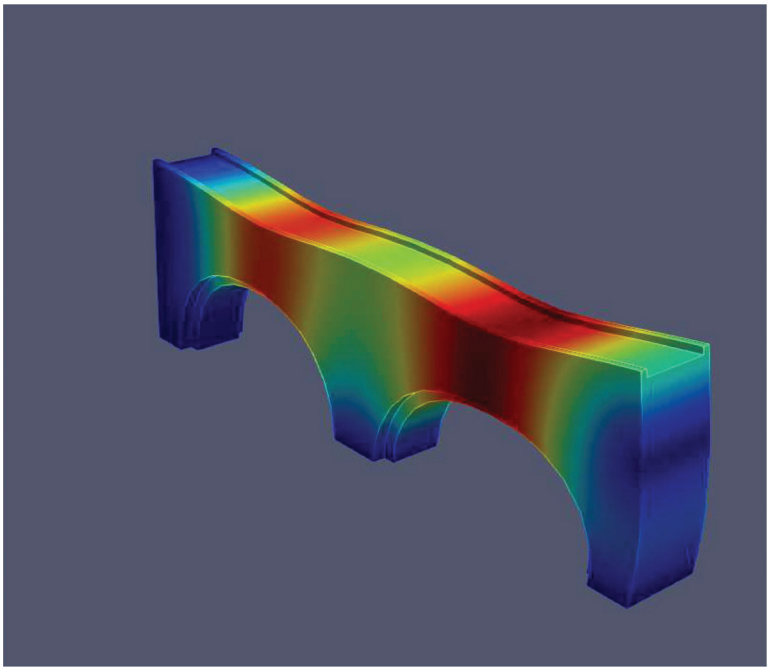
Die Brücke ist an den folgenden Flächen fixiert wobei sie sich in die x-Richtung bewegen kann, wie es auch in der Realität der Fall ist:



Auf die folgende Fläche wird Druck ausgeübt, um ein Veiwenden der Brücke zu simulieren:

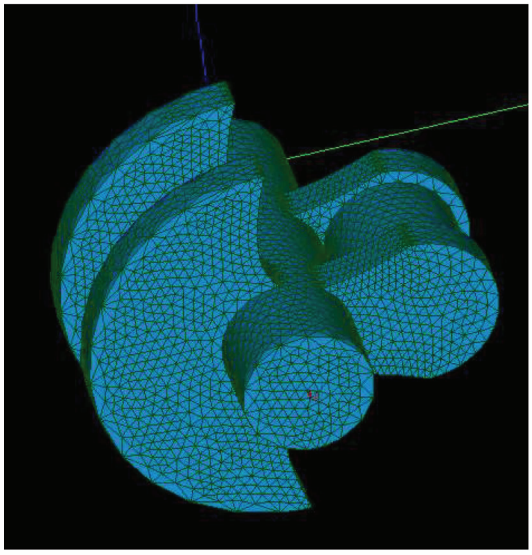


Im Post-Processing-Programm erhält man anschließend folgendes Bild(eine exemplarische Modeshape/ Eigenshape):

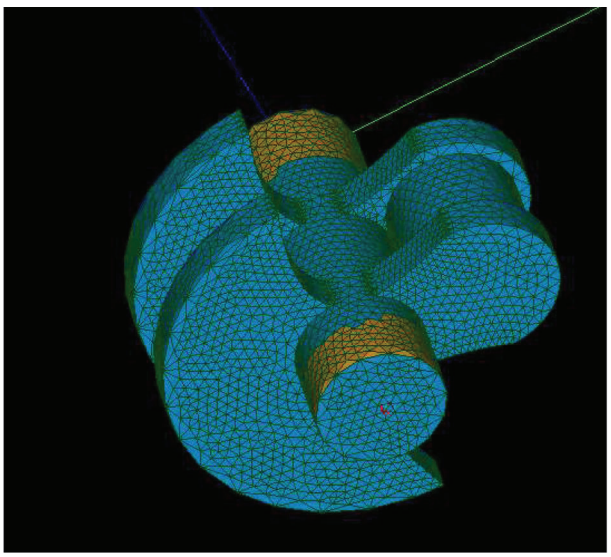


Kurbelwelle

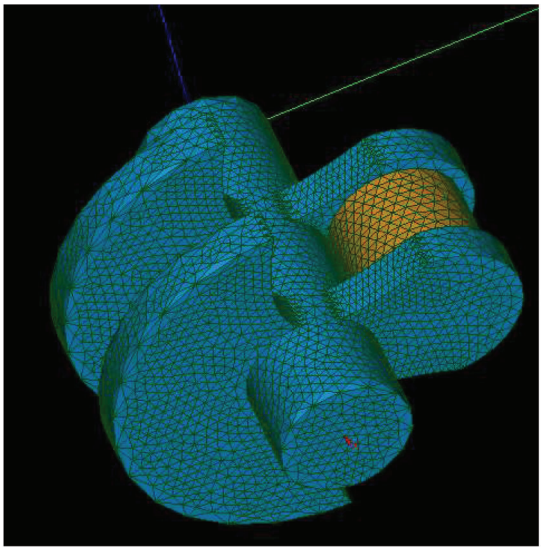
Im laufe des Arbeitsprozess wurde ein Modell einer Kurbelwelle erstellt. Dieses Bauteil wird in Verbrennungsmotoren benötigt und steht unter der Belastung von Schwingungen.



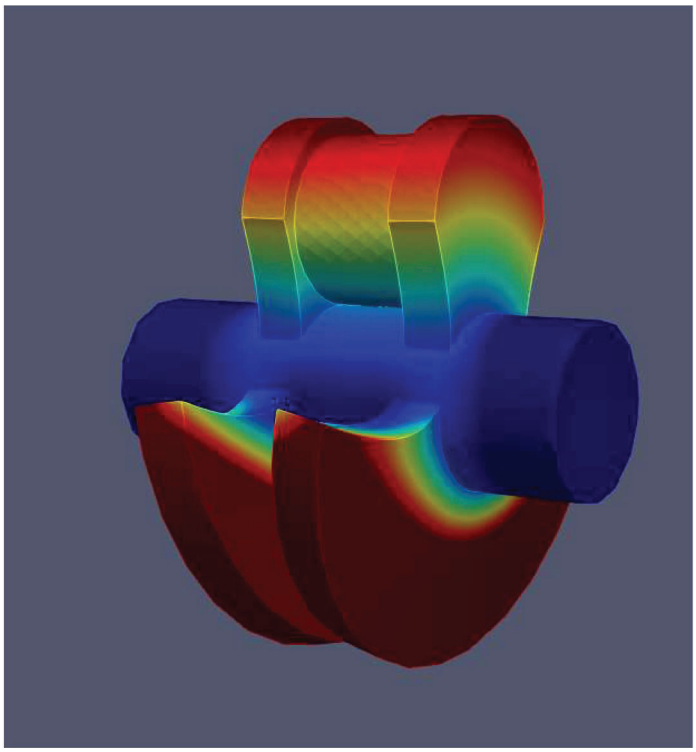
An den eingefärbten Flächen, also an den Rändern der Welle, ist das Objekt fixiert.



An der zylindrischen Fläche die von dem Wellenmittelpunkt versetzt ist wird der Druck angesetzt, wie es in der Realität der Fall ist.

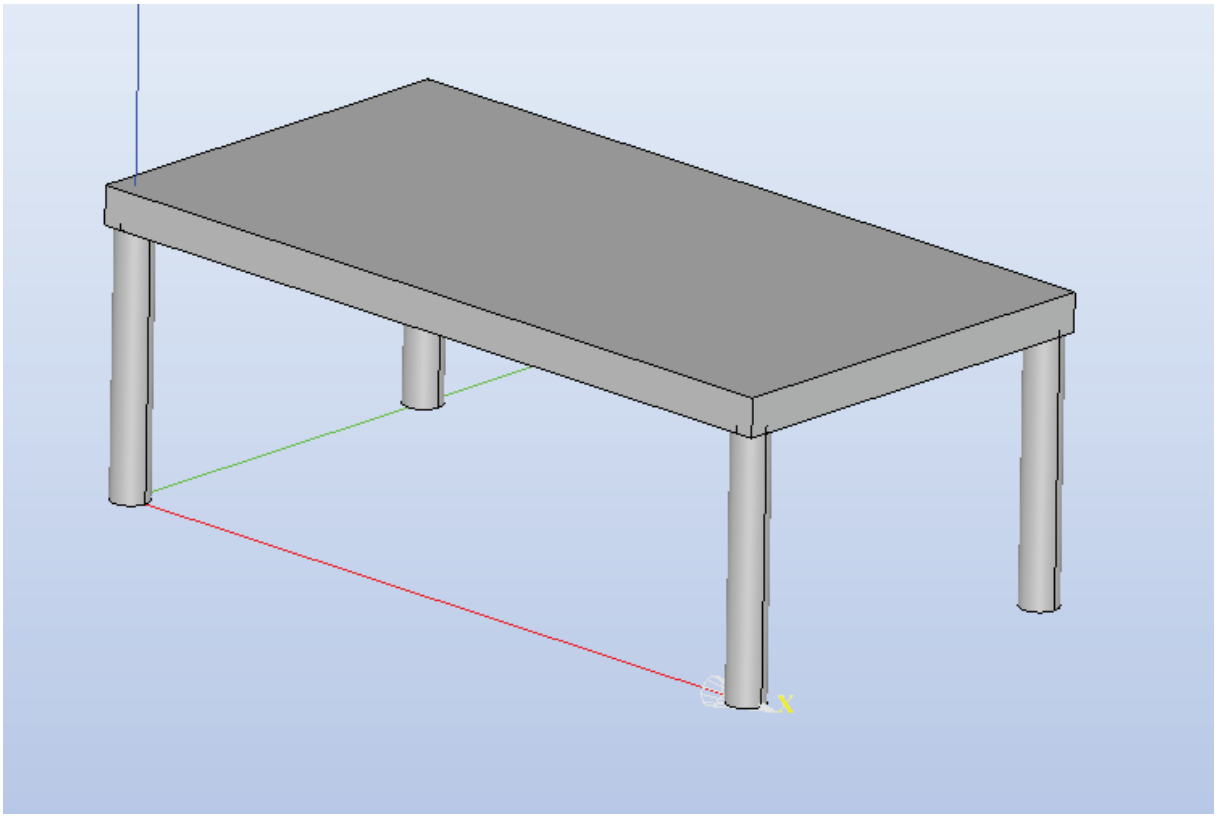


Im Post-Processing-Programm erhält man anschließend folgendes Bild(Schwingungsamplituden für eine Exemplarische Frequenz):

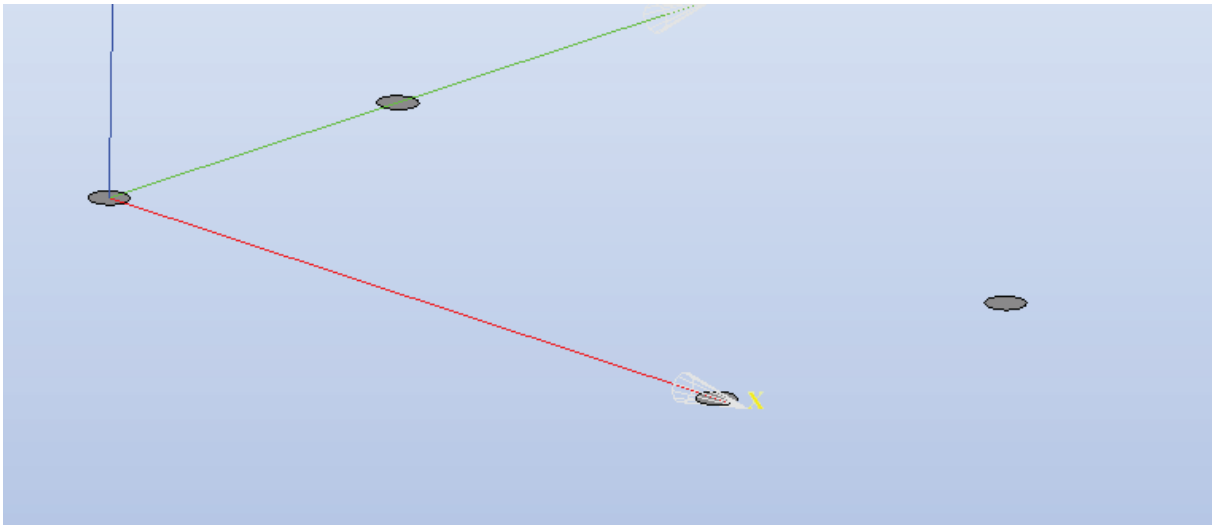


Bewegung eines Tisches unter Anregung mit Eigenfrequenz

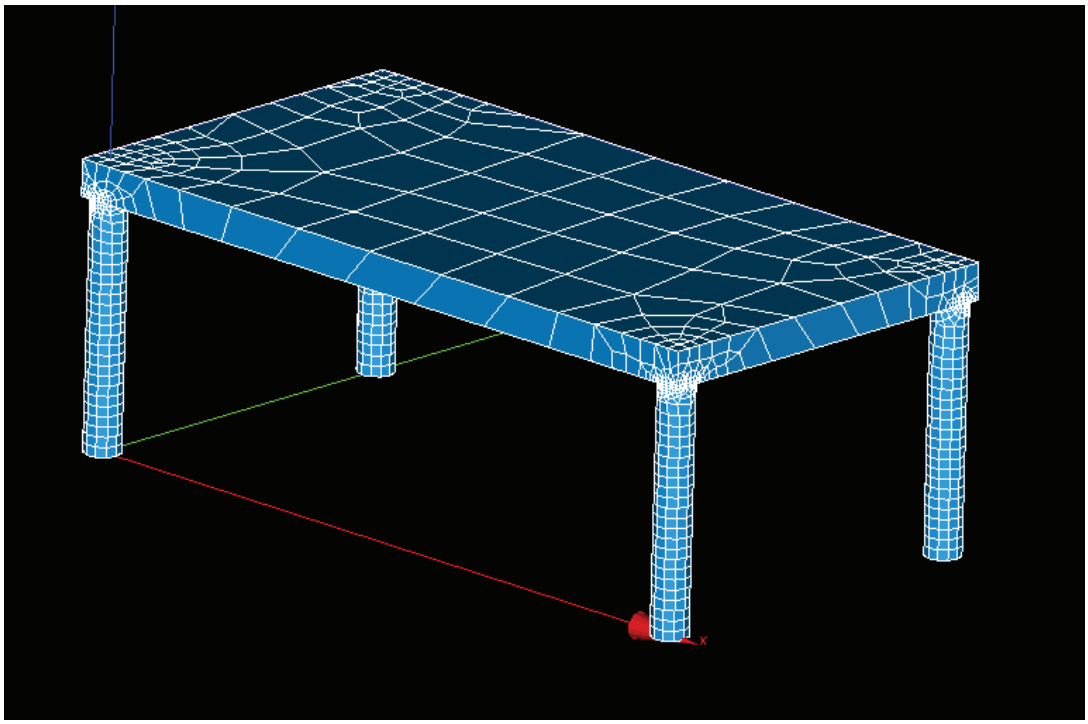
Zuerst wurde der Tisch 3D modelliert. Das Modell besteht aus vier Zylindern und einer Box



Danach wurden die Bodenplatten der Füße als Fixpunkte definiert.

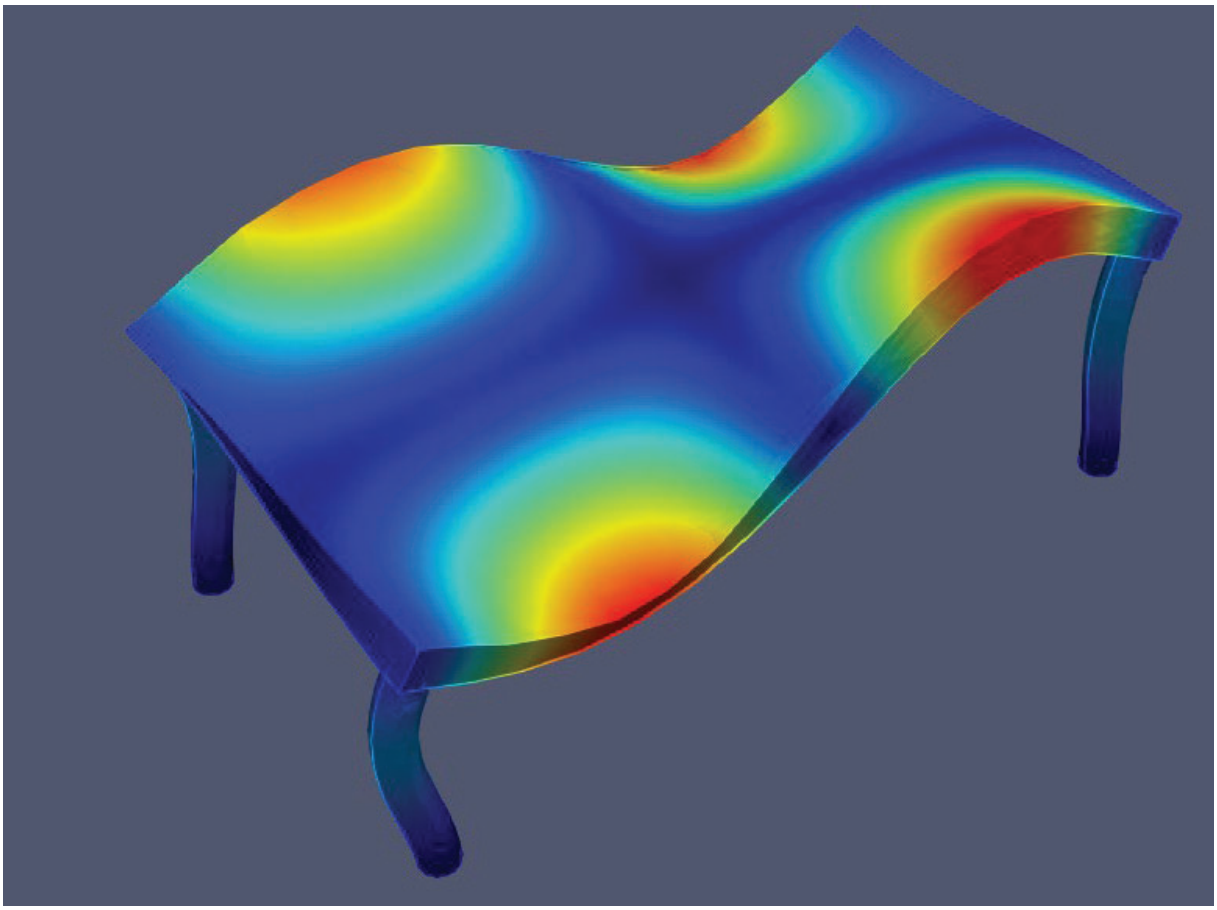


Das zugehörige Mesh sieht so aus:



Die Eigenfrequenzen werden nun mit dem Programm *Code_Aster* berechnet.

Nach der Berechnung dieser kann der Tisch mit einer davon angeregt werden. Unter der Anregung mit einer der Eigenfrequenzen schwingt der Tisch stark verstärkt wie folgt:

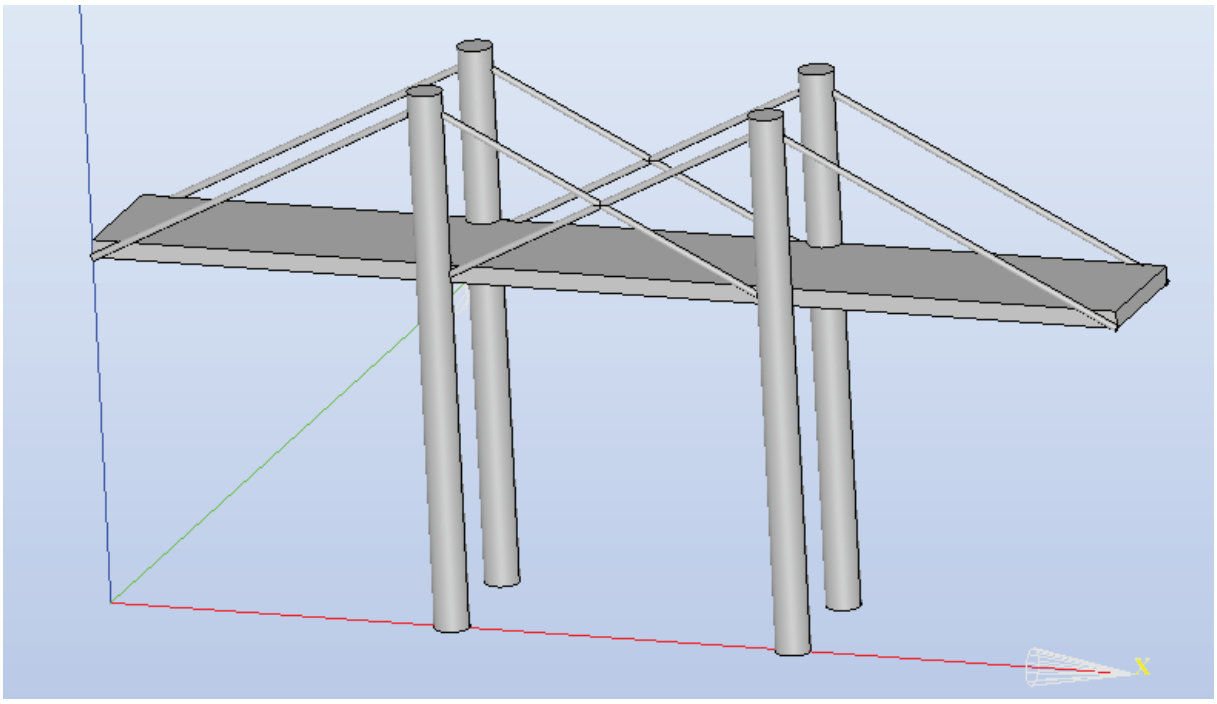


Anregung einer Brücke durch Wind mit Eigenfrequenz der Brücke

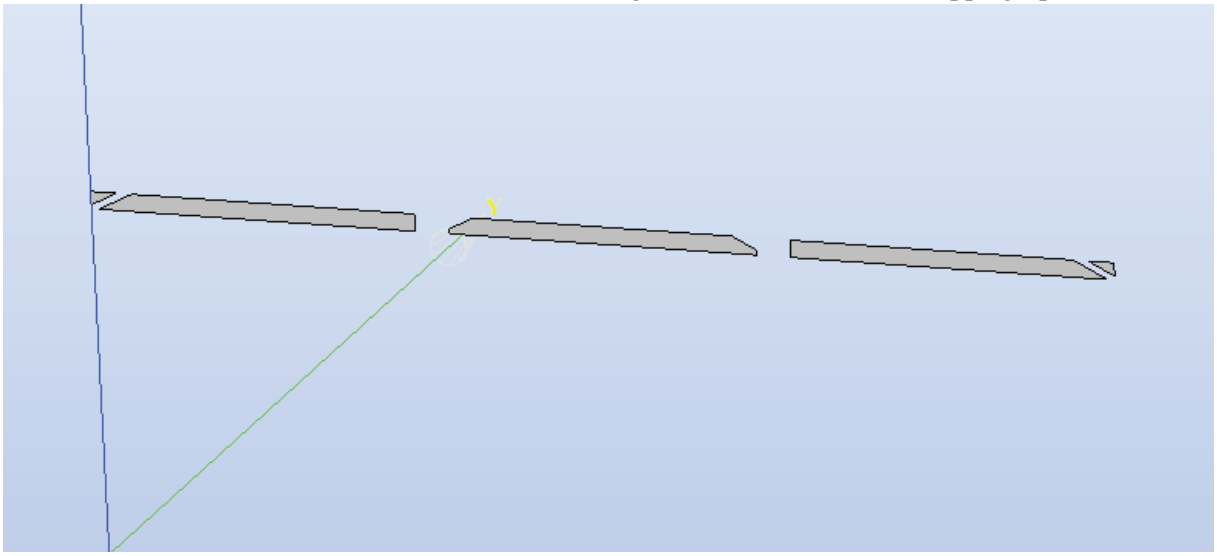
Zuerst wurde mit Hilfe von Python die Brücke erstellt

```
import salome
salome.salome_init()
import GEOM
from salome.geom import geomBuilder
geompy = geomBuilder.New()
gg = salome.ImportComponentGUI("GEOM")

#Zylinder für Stützen wird erstellt
stütze1 = geompy.MakeCylinderRH(1, 31)

#Die Zylinder werden kopiert und verschoben
translation1 = geompy.MakeTranslation(stütze1, 20, 0, 0)
translation2 = geompy.MakeTranslation(stütze1, 20, 8, 0)
translation3 = geompy.MakeTranslation(stütze1, 40, 0, 0)
translation4 = geompy.MakeTranslation(stütze1, 40, 8, 0)

#Die Straße wird erstellt und an die richtige Position geschoben
straße = geompy.MakeBox(0,0,20,60,8,21)

#Ein Vektor entlang der y-Achse für die Rotation der Seile
p1 = geompy.MakeVertex(0,0,0)
p2 = geompy.MakeVertex(0,1,0)
yAxis = geompy.MakeVector(p1,p2)

#Die Seile werden als Zylinder erstellt und an die vorgesehenen
#Positionen geschoben und gedreht
Cyl1 = geompy.MakeCylinderRH(0.2, 23)
wireRotat1 = geompy.MakeRotation(Cyl1, yAxis, 1.1)
wire1 = geompy.MakeTranslation(wireRotat1, 0,0,20)

wire2 = geompy.MakeTranslation(wireRotat1, 0,8,20)

wire3 = geompy.MakeTranslation(wireRotat1, 20,0,20)

wire4 = geompy.MakeTranslation(wireRotat1, 20,8,20)

wireRotat5 = geompy.MakeRotation(Cyl1, yAxis, -1.1)
wire5 = geompy.MakeTranslation(wireRotat5, 40,0,20)

wire6 = geompy.MakeTranslation(wireRotat5, 40,8,20)

wire7 = geompy.MakeTranslation(wireRotat5, 60,0,20)

wire8 = geompy.MakeTranslation(wireRotat5, 60,8,20)

#Die einzelnen Objekte der Brücke werden in ein Objekt zusammengesteckt
Brücke = geompy.MakeFuseList([translation1, translation2,
translation3, translation4, straße, wire1, wire2, wire3,
wire4, wire5, wire6, wire7, wire8])
Brücke1 = geompy.addToStudy(Brücke, "Brücke")
gg.createAndDisplayGO(Brücke1)
```

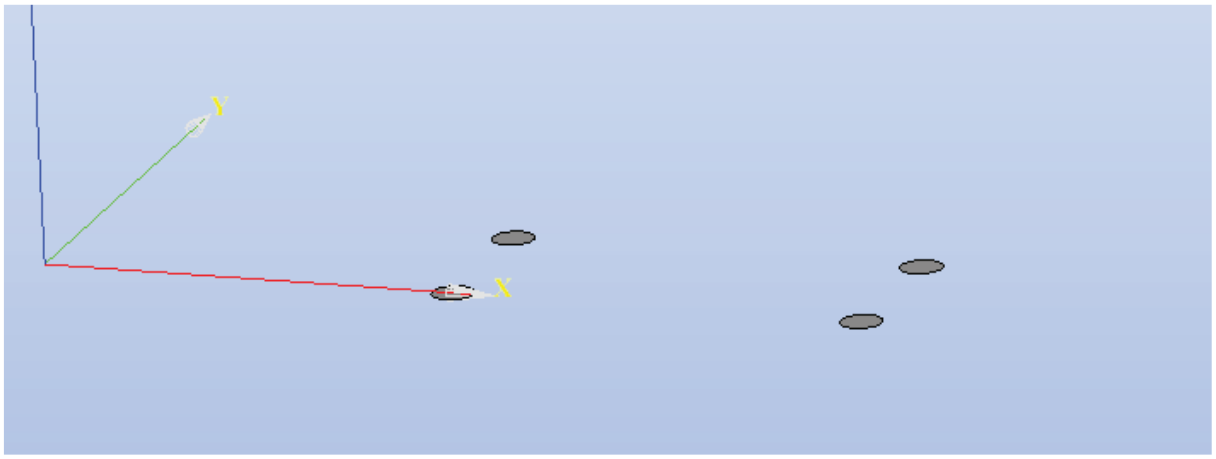
Geometrisch dargestellt im Programm *Salome* sieht der Code so aus:



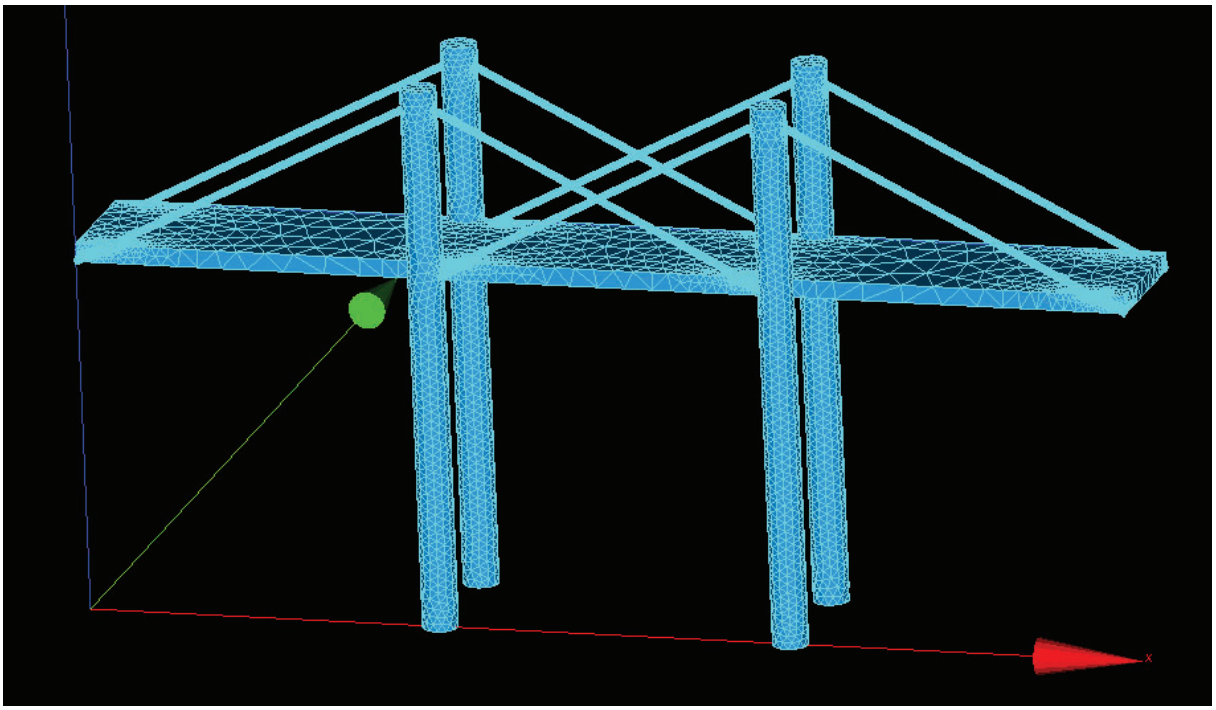
Danach wurden diese Flächen für die Krafteinwirkung definiert und in einer Gruppe gespeichert:



Nun wurden die Böden der Säulen als nicht bewegbare Fixpunkte festgelegt:

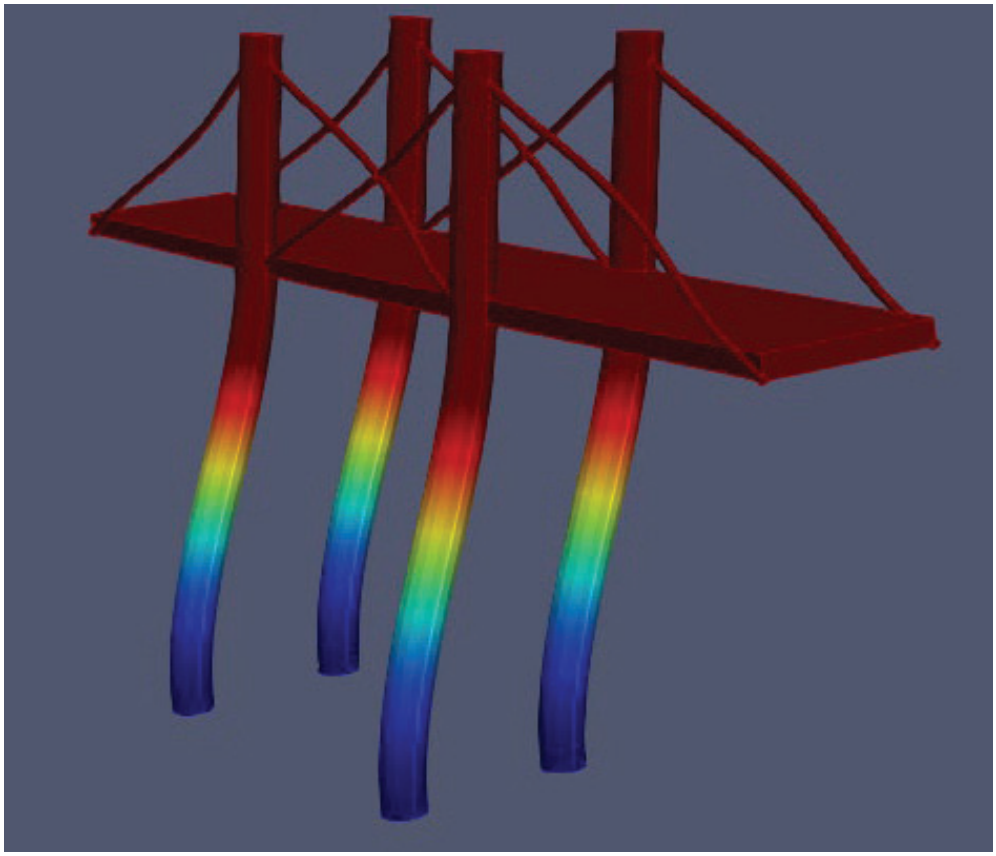


Das zugehörige Mesh zu der Geometrie sieht so aus:



Die Eigenfrequenzen werden mit dem Programm *Code_Aster* ausgerechnet.

Die Einwirkung der Kraft des Windes mit diesen Eigenfrequenzen lässt sich ebenfalls graphisch darstellen:

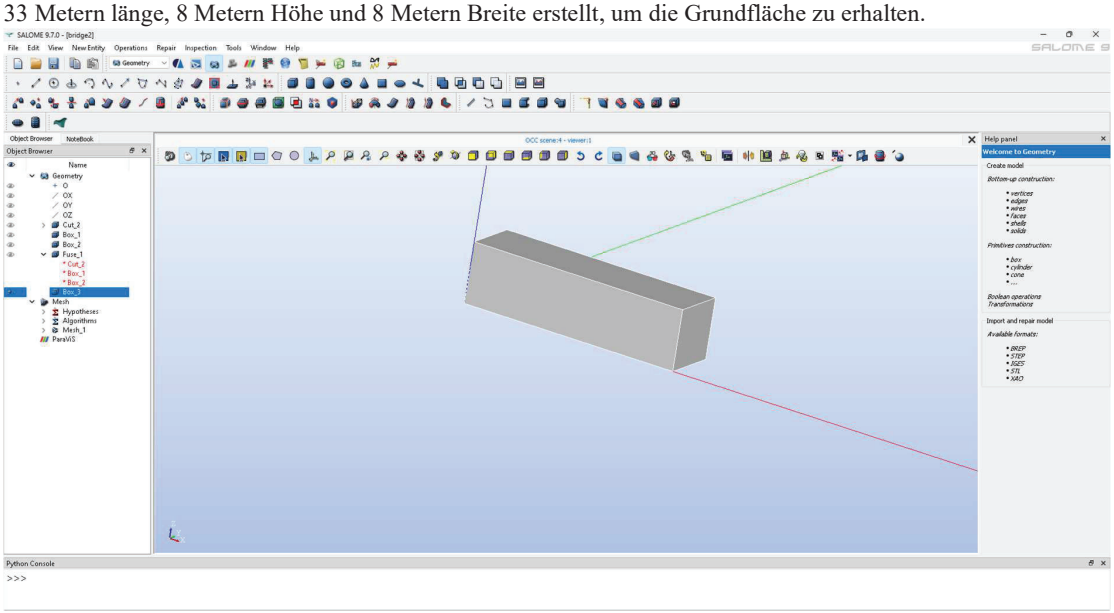


Brücke II

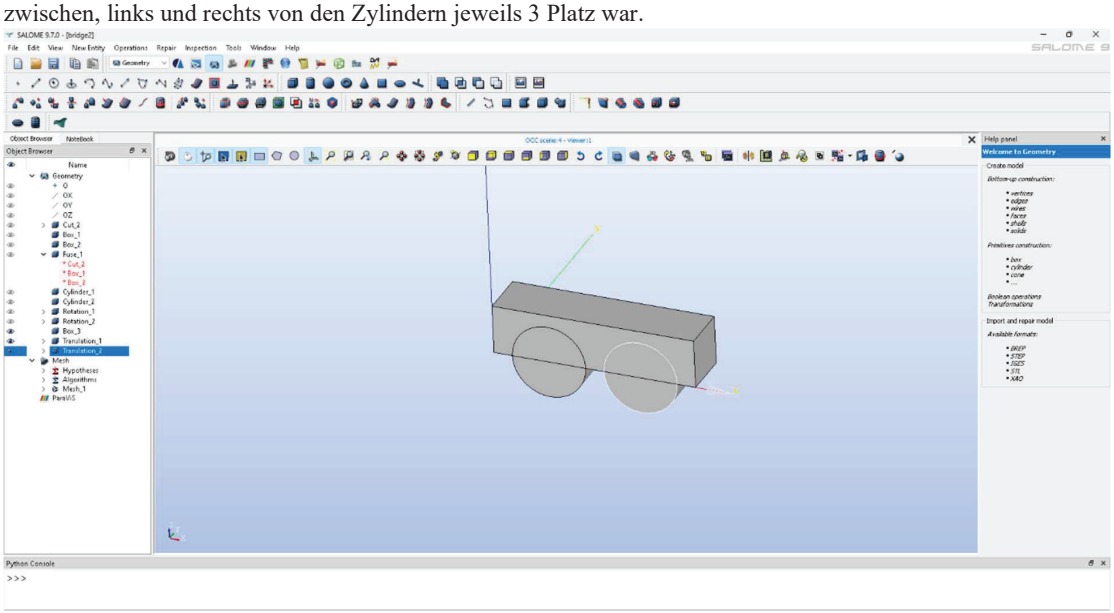
Um die 3D-Objekte zu erstellen haben wir das Programm „Salome“ benutzt und um die Eigenfrequenzen zu berechnen wurde das Programm „Aster“ verwendet. Das erste Objekt, das wir erstellten war eine simple Platte, die wir in Form einer Box erstellten. Die Eigenfrequenzen von der Platte haben wir dann im „Aster“ berechnet.

Als nächstes bekamen wir den Auftrag eine Brücke zu modellieren. Diese Brücke war 60 Meter lang und 31 Meter hoch. Sie hatte 4 Träger und die Platte war mit 8 Seilen an den Trägern befestigt. Danach haben wir wieder mit Hilfe von „Aster“ die Eigenfrequenzen berechnet.

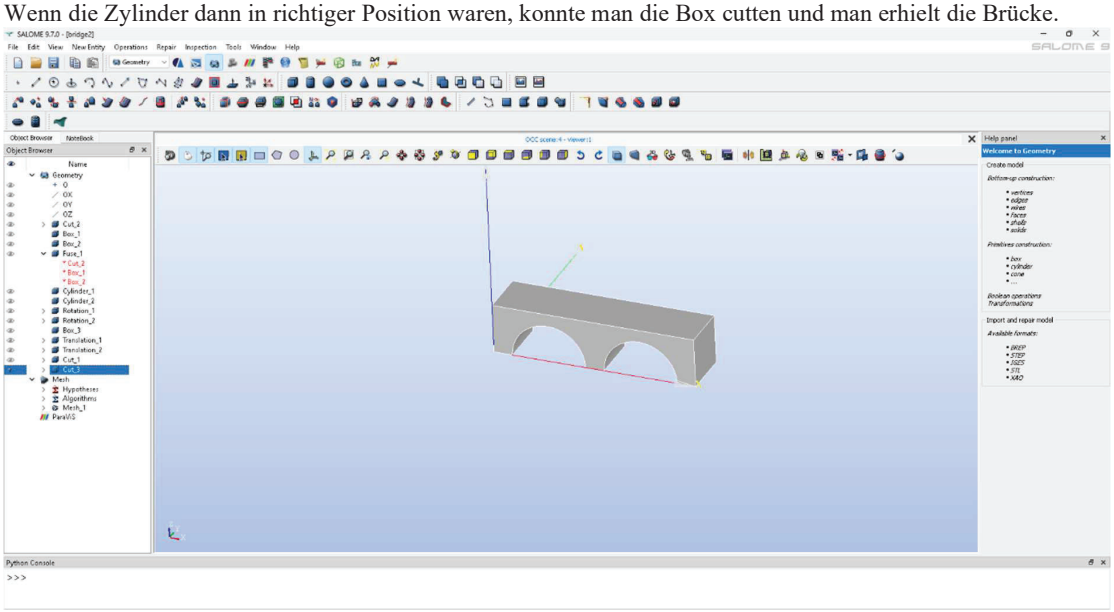
Beim Konstruieren einer anderen Brücke mit 2 Trägern im Stil von Aquädukten haben wir zuerst eine Box mit 33 Metern länge, 8 Metern Höhe und 8 Metern Breite erstellt, um die Grundfläche zu erhalten.



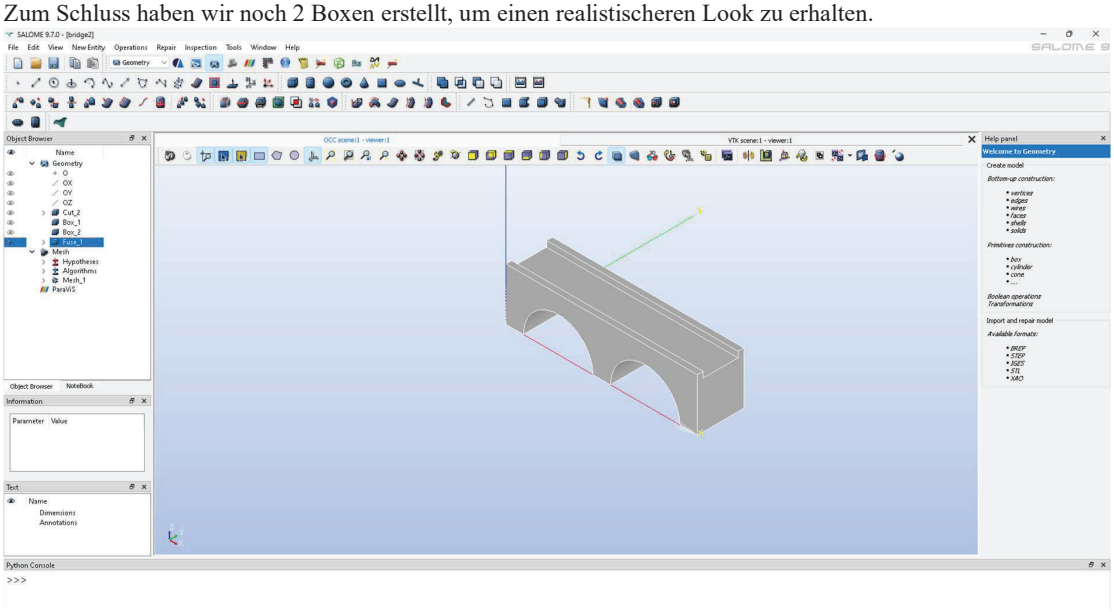
Danach nahmen wir 2 Zylinder mit einem Radius von 6 und einer Höhe von 8. Diese haben wir dann um 90 grad in y Richtung gedreht und in die Richtige Position gebracht, sodass die Halbkreise in den Boxen waren und zwischen, links und rechts von den Zylindern jeweils 3 Platz war.



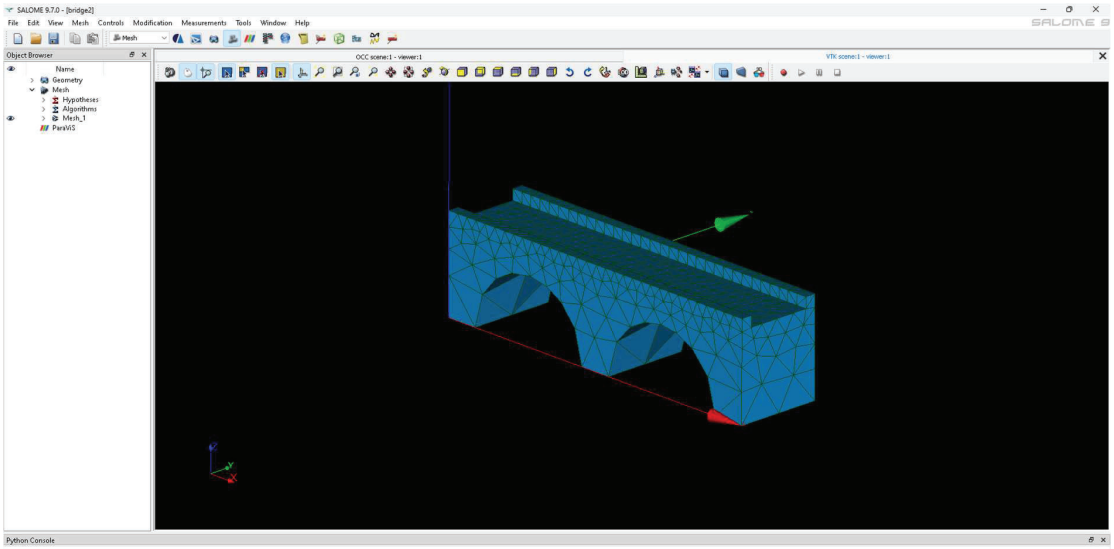
Wenn die Zylinder dann in richtiger Position waren, konnte man die Box cutten und man erhielt die Brücke.



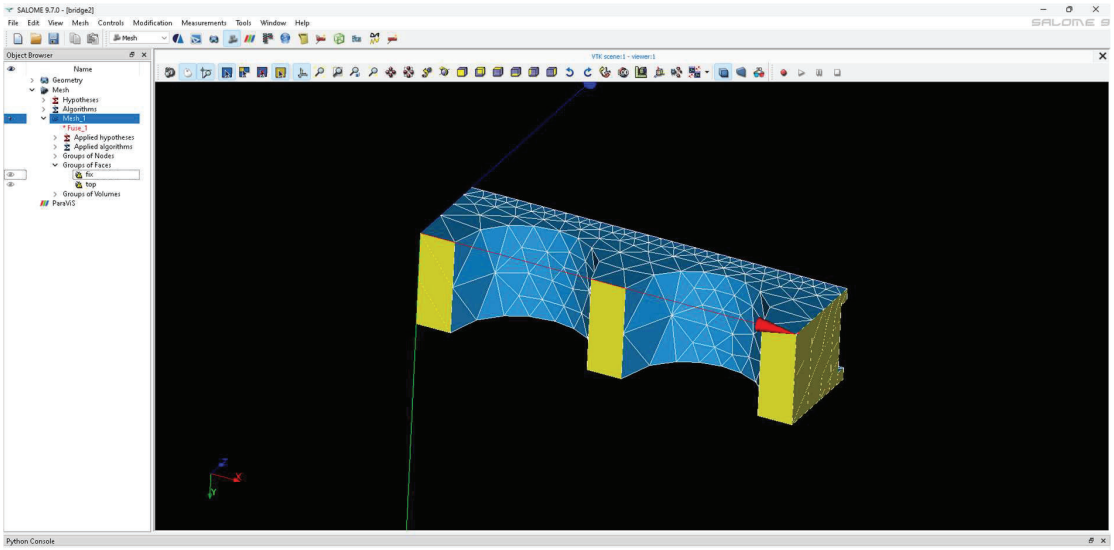
Zum Schluss haben wir noch 2 Boxen erstellt, um einen realistischeren Look zu erhalten.



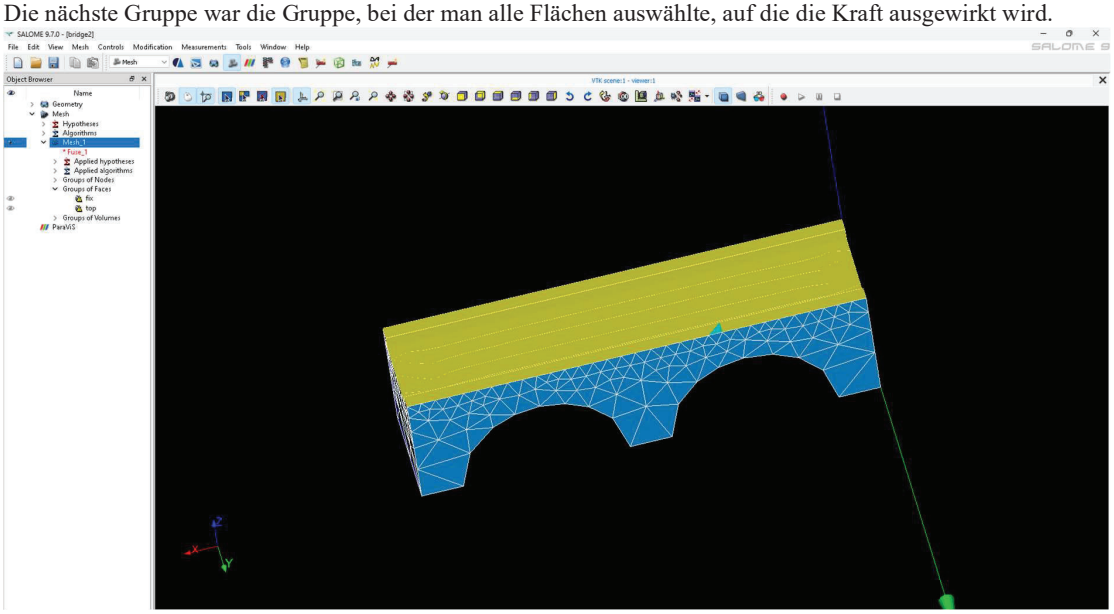
Aus diesem Objekt musste man dann noch ein Mesh machen, um das Objekt in Flächen und Knoten zu unterteilen.



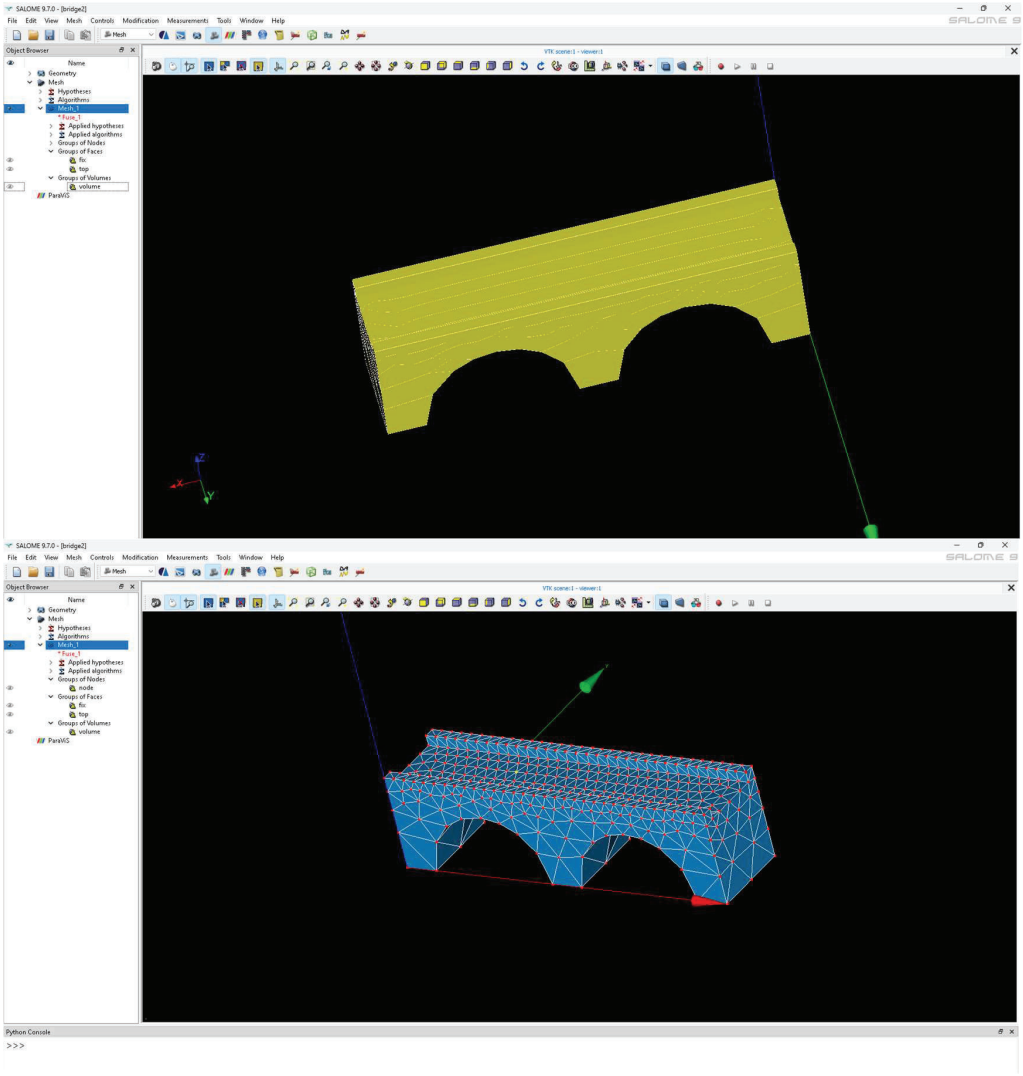
Der nächste Schritt war es 4 Gruppen zu erstellen, um die Eigenfrequenzen dann im „Aster“ auszurechnen. Die erste Gruppe war die Fixpunkt-Gruppe, bei der man alle Flächen auswählte, die von der Brücke fest fixiert waren.



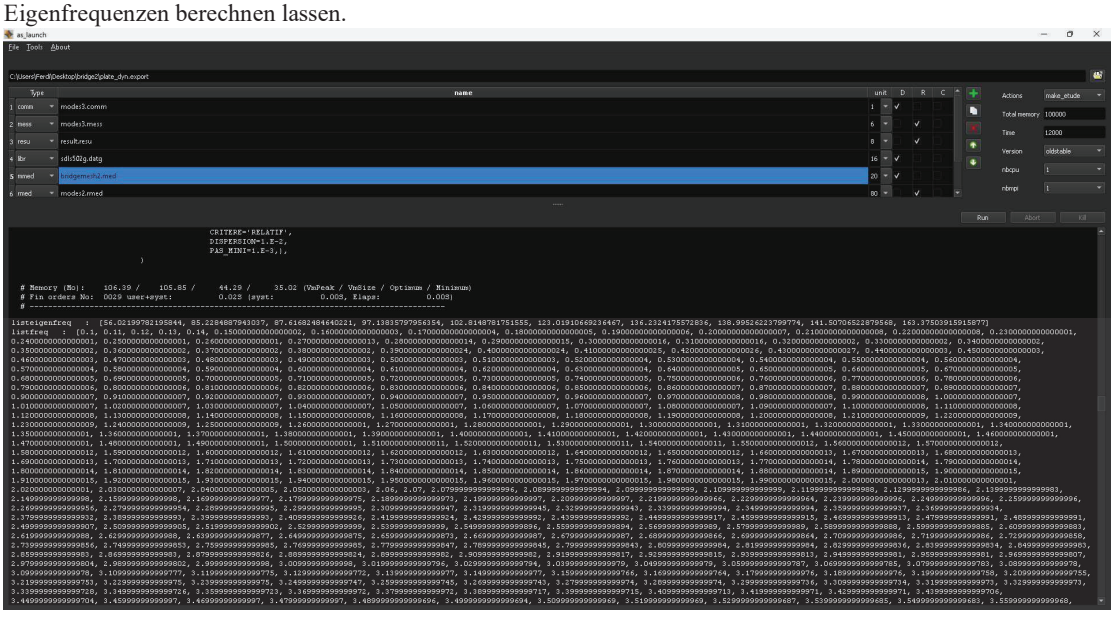
Die nächste Gruppe war die Gruppe, bei der man alle Flächen auswählte, auf die die Kraft ausgewirkt wird.



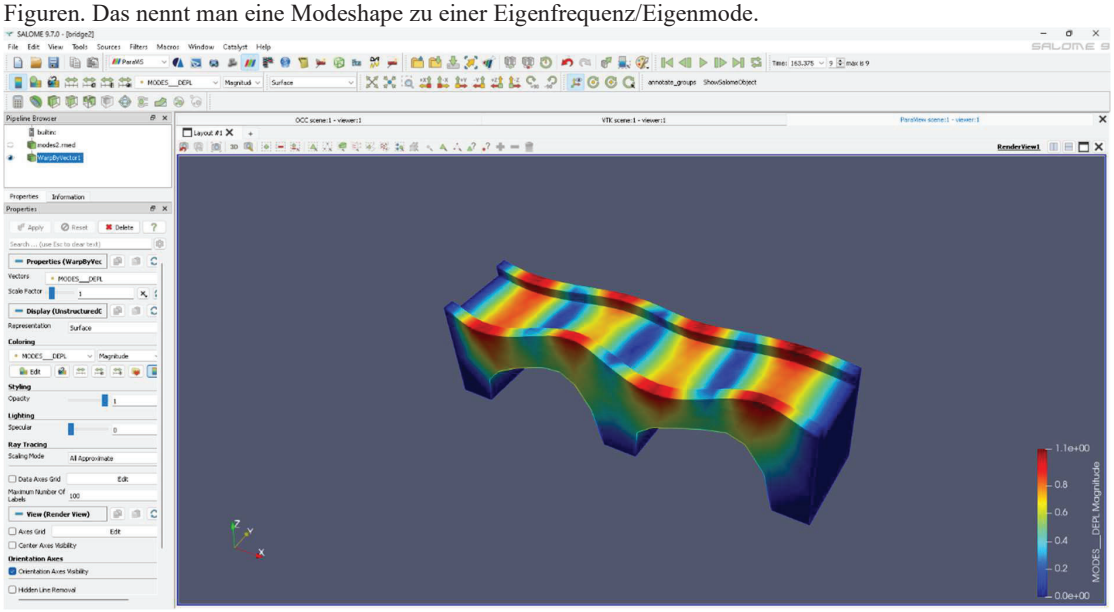
Dann gab es noch eine Gruppe, bei der man das Ganze Volumen auswählen musste und eine Gruppe, bei der nur ein einzelner Knotenpunkt hinzugefügt werden musste.



Der nächste Schritt bei der Brücke war es, die exportierte MED-Datei in „Aster“ zu werfen und die Eigenfrequenzen berechnen lassen.

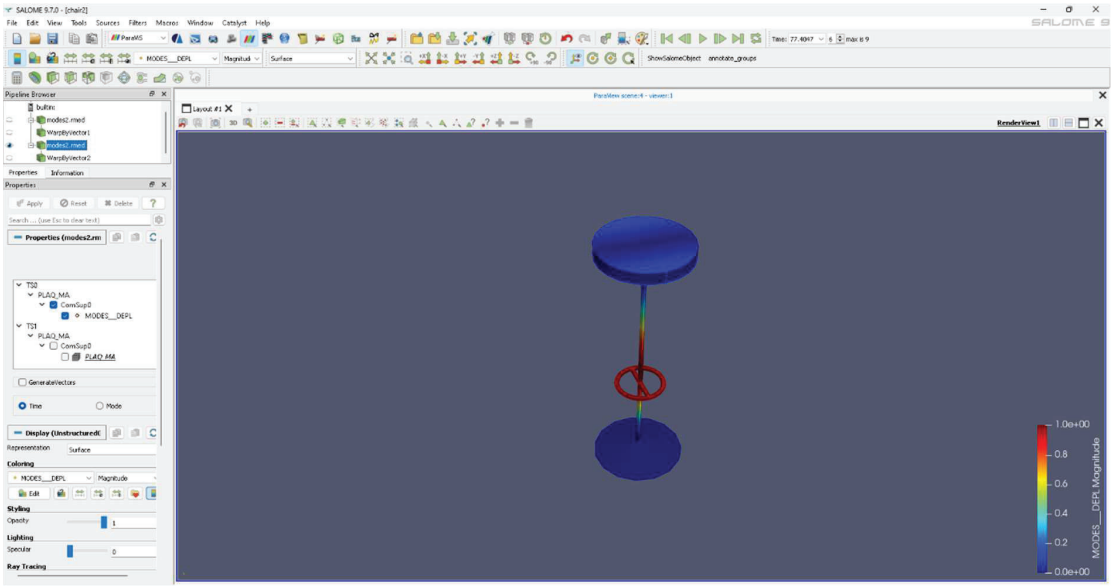


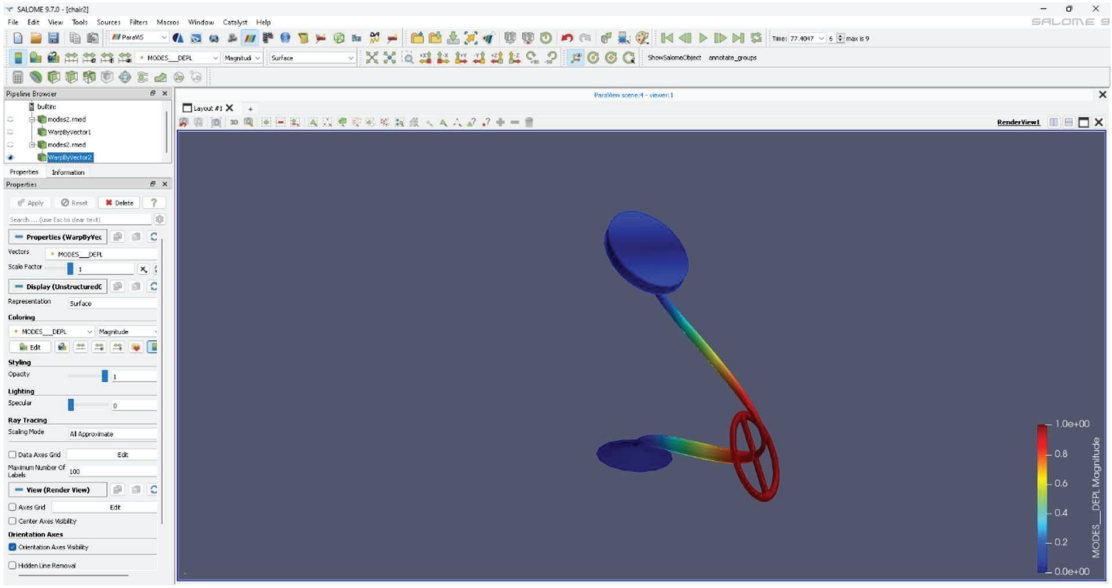
Als letztes musste man die Datei wieder in Salome einfügen und man erhält mehr oder weniger abstrakte Figuren. Das nennt man eine Modeshape zu einer Eigenfrequenz/Eigenmode.



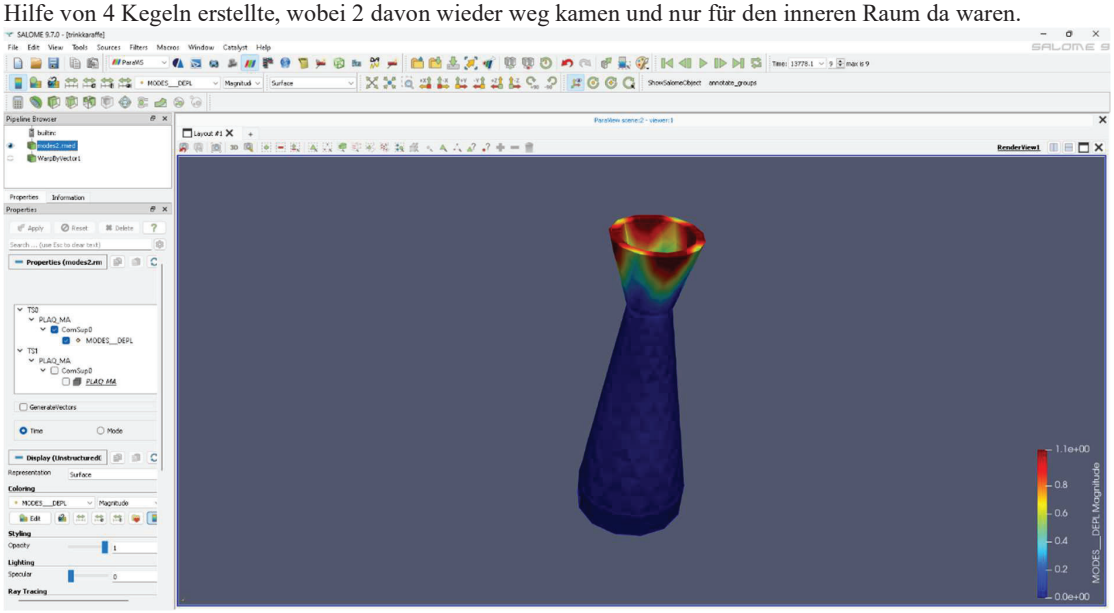
Barhocker

Das nächste Projekt war ein Barhocker mit Fußablage, wobei die Fußablage mit Abstand der schwierigste Teil war, weil es in „Salome“ nicht möglich ist ein Mesh zu erstellen, wenn 1. Objekte überlappen und 2. Objekte sich gerade so nicht berühren. Dennoch wurde er modelliert und die Eigenfrequenz berechnet .

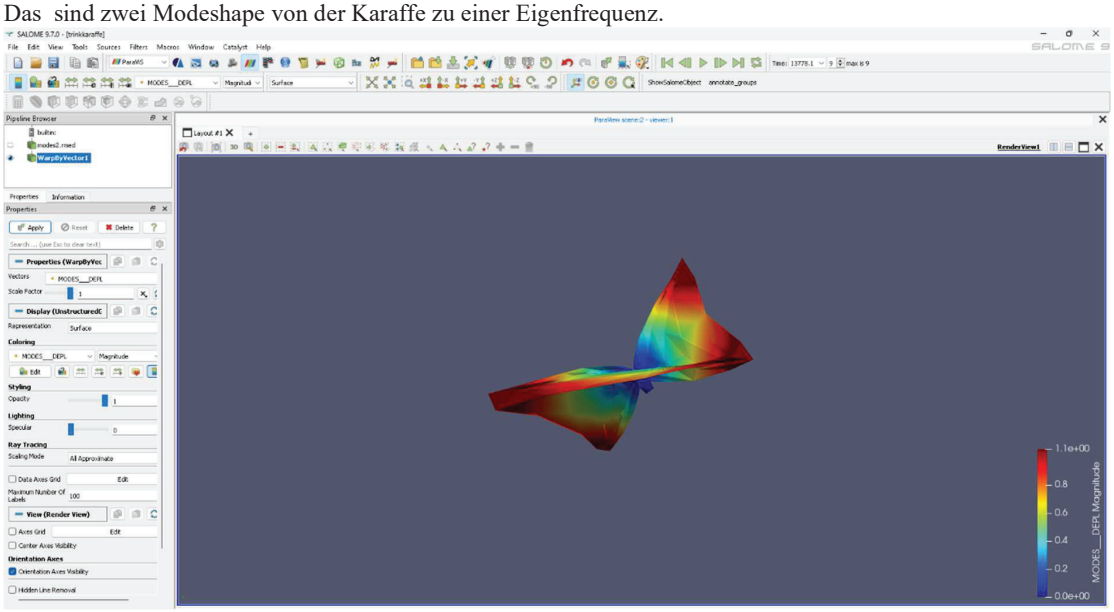


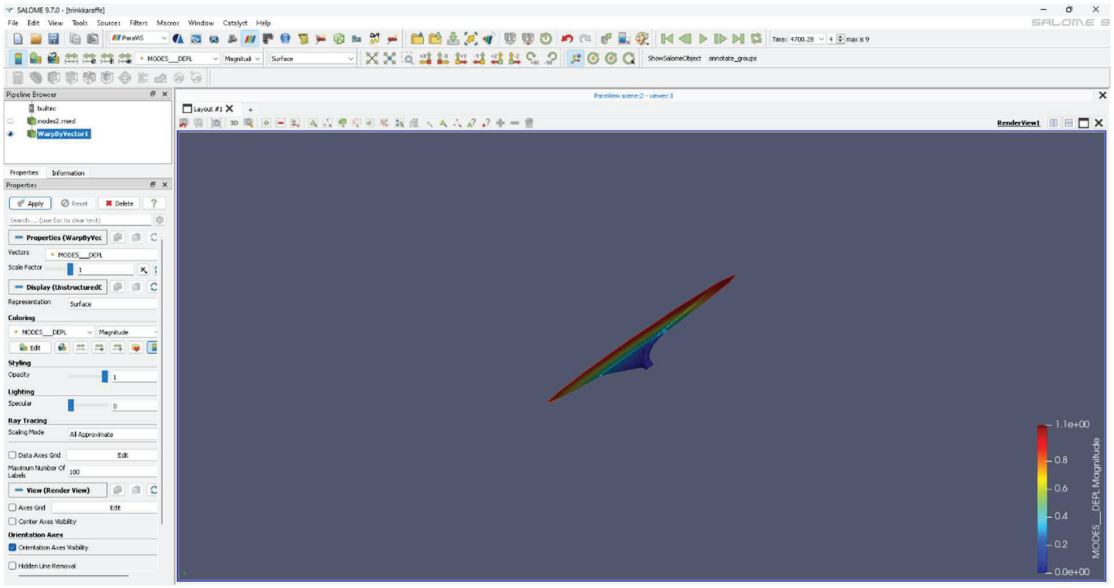


Als nächstes wurde eine Wasserkaraffe modelliert. Das war eines der einfacheren Objekte, weil man sie nur mit Hilfe von 4 Kegeln erstellte, wobei 2 davon wieder weg kamen und nur für den inneren Raum da waren.



Das sind zwei Modeshape von der Karaffe zu einer Eigenfrequenz.





Aster-Code

```
PLAQ_MA=LIRE_MAILLAGE(VERI_MAIL=_F(VERIF='OUI'),
                      FORMAT='MED', );
```

.med-Datei wird gelesen

```
MODEL=AFFE_MODELE(   MAILLAGE=PLAQ_MA,
                     AFFE=_F( TOUT = 'OUI',
                          PHENOMENE = 'MECANIQUE',
                          MODELISATION = '3D',),);
```

Modell wird aufgestellt

```
PLAQ_MA=DEFI_GROUP(reuse=PLAQ_MA,
                MAILLAGE=PLAQ_MA,
                CREA_GROUP_NO=_F(GROUP_MA='fix',NOM='fix',)
                ,);
```

Gruppe für Modell wird festgelegt

```
#-------------------------------------------------------------------
#                    CARACTERISTIQUES MATERIAUX
#-------------------------------------------------------------------
ACIER=DEFI_MATERIAU(   ELAS=_F( E = 2.7E+10,
                           NU = 0.2,
                           RHO = 2000.),);



MAT=AFFE_MATERIAU(   MAILLAGE=PLAQ_MA,
                     AFFE=_F( TOUT = 'OUI',
                     MATER = ACIER)
```

Material wird bestimmt

```
#-------------------------------------------------------------------
#                    CHARGEMENTS
#-------------------------------------------------------------------
CHAR0=AFFE_CHAR_MECA(     MODELE=MODEL,
                      FACE_IMPO=(_F( GROUP_MA = 'fix',
```

```
                    DZ = 0.,
                    DX = 0.,
                    DY = 0.),
                                                        _F( GROUP_MA = 'fixx',
                    DZ = 0.,
                    DY = 0.),
                                                        ),
                );


PRES = AFFE_CHAR_MECA(MODELE=MODEL,
                PRES_REP=_F(GROUP_MA="front",
                        PRES=1000000,
                        ),
                )
```

Belastung und Bewegungsrichtungen werden festgelegt

```
#-----------------------------------------------------------------
#                 RESOLUTION
#-----------------------------------------------------------------
ASSEMBLAGE(MODELE=MODEL,
        CHAM_MATER=MAT,
        #CARA_ELEM=carac,
        CHARGE=CHAR0,
        NUME_DDL=CO('NUM'),
        MATR_ASSE=(_F(MATRICE=CO('Stiffn'),
                 OPTION='RIGI_MECA',),
                _F(MATRICE=CO('Mass'),
                 OPTION='MASS_MECA',),),
        VECT_ASSE=_F(VECTEUR=CO('ForcAss'),
                 OPTION='CHAR_MECA',
                 CHARGE=(PRES,),),);
```

Matrizen, Nummerierung und Belastungsvektor werden erstellt

```
MODES=CALC_MODES(MATR_RIGI=Stiffn,


                 MATR_MASS=Mass,
                 #OPTION='BANDE',
                 #CALC_FREQ=_F(FREQ=(10.,3500000.),),
                 OPTION="PLUS_PETITE",
                 SOLVEUR_MODAL=_F(METHODE='SORENSEN',
                                  ),
```

Eigenwerte werden berechnet

```
#printing of the mode shapes in a med file
IMPR_RESU (
FORMAT='MED' ,UNITE=80 ,
RESU=_F(RESULTAT=MODES ,
      NOM_CHAM='DEPL',),
      );
```

Med(Mesh-) Datei wird geschrieben

```
#printing the values in the .resu file
#for the RESU "modes"
IMPR_RESU(MODELE=MODEL,
       FORMAT='RESULTAT',
       RESU=_F(RESULTAT=MODES,
       #this prints all the info relative to modal analysis
       TOUT_CHAM='NON', #with this we do not print DEPL
       TOUT_PARA='OUI', #prints all the parameter
```

Zusätzliche Informationen werden in .resu Datei geschrieben

```
#------------------------------------------------------------
# Dynamique
#------------------------------------------------------------
n=10
```

```
PROJ_BASE(BASE=MODES,
          NB_VECT=n,
          MATR_ASSE_GENE=(_F(MATRICE=CO('StifGen'),
                             MATR_ASSE=Stiffn,),
                          _F(MATRICE=CO('MassGen'),
                             MATR_ASSE=Mass,),),
          VECT_ASSE_GENE=_F(VECTEUR=CO('ForcGen'),
                            TYPE_VECT='FORC',
                            VECT_ASSE=ForcAss,),);
```

Basis wird zurücktransformiert

```
#-------------------------------------------------
# MODAL HARMONIC ANALYSIS
#-------------------------------------------------

# create a damping matrix corresponding to given modal damping degrees
dmodal = 0.00003;

Lamort=DEFI_LIST_REEL(VALE=(dmodal, dmodal, dmodal, dmodal, dmodal,),);

DampGen=COMB_MATR_ASSE(CALC_AMOR_GENE=_F(RIGI_GENE=StifGen,
                                         MASS_GENE=MassGen,
                                         LIST_AMOR=Lamort,),);
```

Dämpfungsmatrix wird erstellt

```
listeigenfreq = MODES.LIST_VARI_ACCES()['FREQ']
print( 'listeigenfreq   : ', listeigenfreq)

ListFreq=DEFI_LIST_FREQ(DEBUT=0.1,
                        INTERVALLE=_F(JUSQU_A=45.,
                                      PAS=0.25,),
                        RAFFINEMENT=_F(LIST_RAFFINE=listeigenfreq,
                                       CRITERE='RELATIF',
```

```
                    NB_POINTS=20,),
                INFO=2,);
```

Intervall und Schrittweite der Modeshapes wird festgelegt

```
listfreq= ListFreq.Valeurs()
print('listfreq   : ', listfreq)


DYNPROD=DYNA_VIBRA(TYPE_CALCUL='HARM',BASE_CALCUL='GENE',
                    MATR_MASS=MassGen,
                    MATR_RIGI=StifGen,
                    MATR_AMOR=DampGen,
                    SOLVEUR=_F( METHODE='LDLT', ),
                    LIST_FREQ=ListFreq,
                    TOUT_CHAM='OUI',
                    EXCIT=_F( VECT_ASSE_GENE = ForcGen,
                        COEF_MULT = 1.))
```

Eigenwertproblem wird gelöst

```
DYNHARD=REST_GENE_PHYS(   RESU_GENE=DYNPROD,
                    GROUP_NO = 'node',
                    LIST_FREQ=ListFreq,
                    MODE_MECA=MODES,
                    NOM_CHAM = 'DEPL',)
```

In physikalische Koordinaten umgewandelt

```
dispRI=RECU_FONCTION(RESU_GENE=DYNPROD,
                NOM_CHAM='DEPL',
                NOM_CMP='DZ',
                GROUP_NO='node',
                );


IMPR_FONCTION(FORMAT='TABLEAU',
        COURBE=(_F(FONCTION=dispRI,
                LEGENDE='dispRI', ),),
                TITRE='Real- and Complex part of dispRI',
```

```
            );
```

Resultat wird ausgewertet

```
Harm=REST_GENE_PHYS(RESU_GENE=DYNPROD, TOUT_CHAM='OUI',);


harmRI=RECU_FONCTION(RESULTAT=Harm,
                NOM_CHAM='DEPL',
                NOM_CMP='DZ',
                GROUP_NO='node',
                );


IMPR_FONCTION(FORMAT='TABLEAU',
       COURBE=(_F(FONCTION=harmRI,
                LEGENDE='harmRI', ),…
#printing the values in the .resu file
#for the RESU "modes"
IMPR_RESU(#MODELE=MODEL,
        FORMAT='MED',
        VERSION_MED='4.0.0',
        UNITE=81,
        RESU=_F(RESULTAT=Harm,
              PARTIE='REEL',
              NOM_CHAM='DEPL',
              TOUT_CMP='OUI',
        #this prints all the info relative to modal analysis
        #TOUT_CHAM='NON', #with this we do not print DEPL
        #TOUT_PARA='OUI', #prints all the parameter
#next lines print only the specified parameter
#NOM_PARA=(
#'FREQ','MASS_GENE',
#'MASS_EFFE_DX','MASS_EFFE_DY',
#),
#FORM_TABL='OUI', #optional
         ),
```

);

FIN()

Daten werden in .resu Datei übertragen

Appendix

Verwendete Literatur:

Dankert, Dankert - Technische Mechanik, Springer Verlag: https://link.springer.com/book/10.1007]/978-3-8348-2235-2 (https://link.springer.com/book/10.1007]/978-3-8348-2235-2)

Aubry J.P. - Beginning with Code_Aster: https://archives.framabook.org/beginning-with-code_aster/ (https://archives.framabook.org/beginning-with-code_aster/)

Weiterführende Links über Diskutierte Mathematik:

Eigenwertproblem: https://de.wikipedia.org/wiki/Eigenwertproblem (https://de.wikipedia.org/wiki/Eigenwertproblem)

Charakteristische Polynom: https://de.wikipedia.org/wiki/Charakteristisches_Polynom (https://de.wikipedia.org/wiki/Charakteristisches_Polynom)

Verallgemeinertes Eigenwertproblem: https://de.wikipedia.org/wiki/Verallgemeinertes_Eigenwertproblem (https://de.wikipedia.org/wiki/Verallgemeinertes_Eigenwertproblem)

Schur Zerlegung: https://de.wikipedia.org/wiki/Schur-Zerlegung (https://de.wikipedia.org/wiki/Schur-Zerlegung)

Spektralsatz: https://de.wikipedia.org/wiki/Spektralsatz (https://de.wikipedia.org/wiki/Spektralsatz)

Dynamik im Frequenzbereich: http://www.cae-wiki.info/wikiplus/index.php/Frequenzbereich (http://www.cae-wiki.info/wikiplus/index.php/Frequenzbereich)

Gruppenfotos

Mathematische Modellierungswoche 2023

Gruppe:

Signalverarbeitung

Unsere Gruppe:

Paul Aldrian
Alexander List
Kilian Bauer
Theresa Gschiel
Christian Grandtner
Lukas Leber
Betreuer Julius Baumhakel

Inhaltsverzeichnis

1. Einführung

Die Signalverarbeitung umfasst verschiedene Bearbeitungsschritte, welche sich mit den Informationen eines Signals beschäftigen. Dabei ist es zielführend, all diese Informationen des Signals zu extrahieren oder die Informationen für eine Übertragung vorzubereiten. Diese Übertragung verläuft im Austausch einer Informationsquelle und einem Informationsverbraucher.
Informationsgewinnung spielt dabei aber noch in mehreren Formen eine Rolle. Dabei ist das Verhalten von Prozessen, das Reduzieren von Daten und das Vorbereiten Daten visualisieren zu können, relevant.

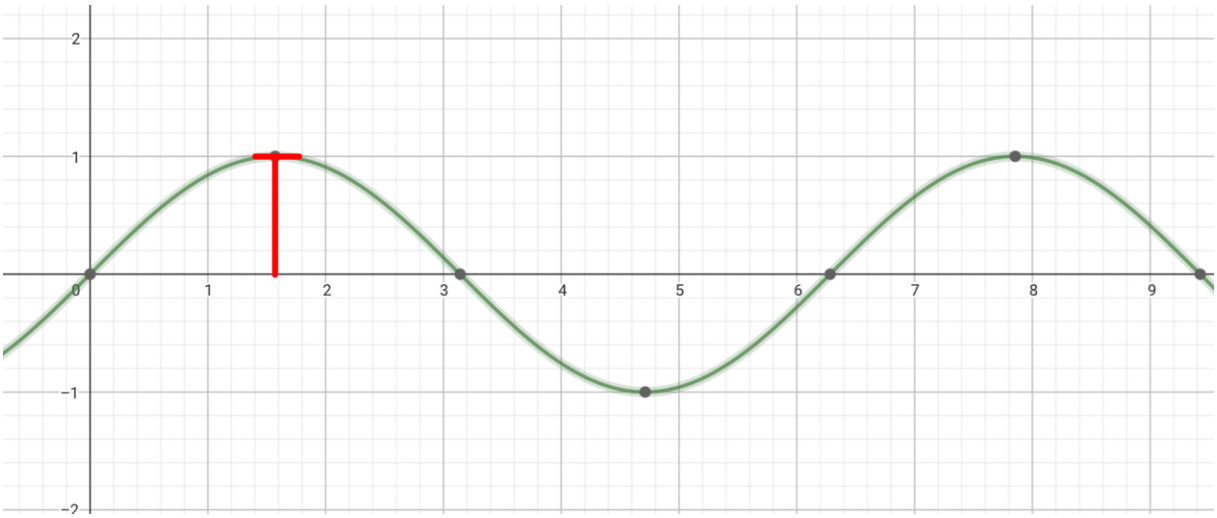
Um den Prozess der Signalverarbeitung deutlich zu machen, ist es ausschlaggebend, das zu verarbeitende Signal zu verstehen. Als Signale werden optische oder akustische Zeichen mit einer bestimmten Bedeutung beschrieben. Auf diese Weise können durch Signale Informationen übertragen werden. Die Übertragung dieser Information geschieht zwischen einem Sender, einer Quelle, welche das Signal aussendet und einem Empfänger, welcher das Signal, also die Information, erhält.

Dabei kann man Signale als Wellen oder Funktionen darstellen. Um diese Wellen zu beschreiben, gibt es verschiedene Faktoren. Diese Faktoren kann man mit einer normalen Sinusschwingung erklären.

y(x)= **A***sin(f*x-d)

1.1 Die Amplitude

Die maximale Auslenkung einer Schwingung aus ihrer Ruhelage nennt man Amplitude. Im physikalischen Sinn beschreibt die Amplitude die Größe der Druckschwankung. In Bezug auf Signalverarbeitung spielt die Amplitude so bei der Lautstärke eine Rolle. Eine höhere Amplitude heißt im Umkehrschluss auch eine größere Lautstärke. Durch Veränderung dieser kann direkt die Lautstärke

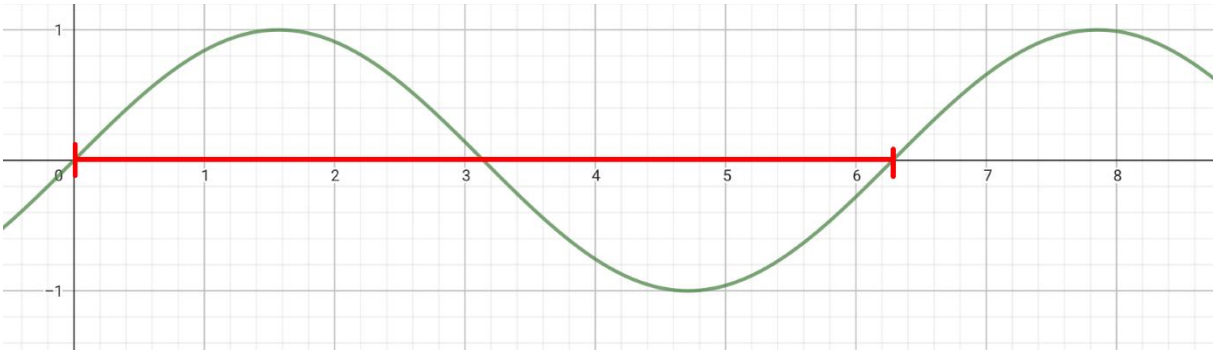


beeinflusst werden.

y(x)= A*sin(**f***x-d)

1.2 Die Frequenz

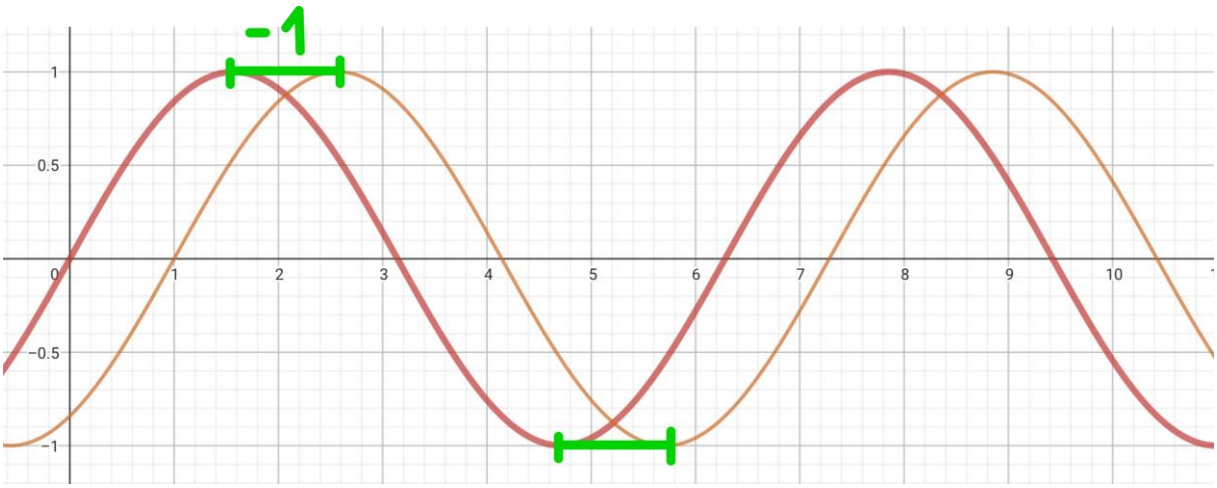
Die Anzahl der Schwingungen, die innerhalb einer Sekunde passieren, bezeichnet man als Frequenz. Die Frequenz ist ausschlaggeben für die Tonhöhe, die wir wahrnehmen. Schwingt ein Signal schneller, wird der Ton höher. Im Gegensatz schwingt die Welle langsamer, ertönt sie tiefer. Der Mensch kann Frequenzen im Raum von 20Hz bis 20.000Hz wahrnehmen. So kann es passieren, dass Frequenzen so schnell oder langsam schwingen, dass man sie nicht mehr hören kann.

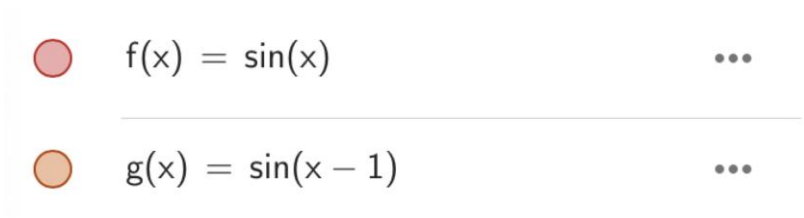


Der Durchlauf einer Periode, also einer Schwingung, liegt bei der Sinus-Funktion bei **2*π**
y(x)= A*sin(f*x-**d**)

1.3 Die Phase

Die Verschiebung der Null- und Extremstellen kann durch die Phase, einen dritten Parameter in der Sinusfunktion, erzielt werden. Obwohl sich genannte Stellen ändern, bleibt der Wertebereich der Funktion gleich. Je nachdem, ob dieser Parameter größer oder kleiner als 0 ist, verschiebt sich die Funktion nach rechts oder links.

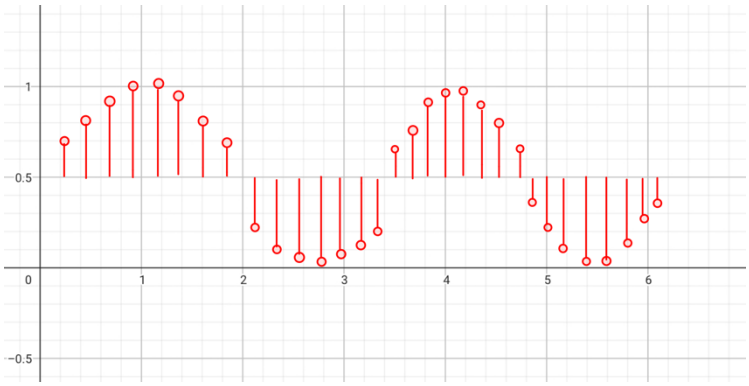




2. Signale im Computer

Anders als sie in der Natur vorkommen, speichert ein Computer Signale als viele, einzelne Punkte, die auf der Funktion liegen, ein. Weil ein Rechner an einem Signal eine numerische Untersuchung, also eine Untersuchung, die im physikalischen Sinn den Vorgang beschreibt, durch welchen Prozesse der Natur gefasst und geordnet werden, anwendet, diskretisiert er die Signale und entnimmt Proben.

Dieser Vorgang passiert durch sogenanntes Abtasten. Dabei werden analoge Signale in diskrete Signale umgewandelt. Als analoges Signal bezeichnet man ein unverändertes Signal, von diskreten Signalen hingegen spricht man, wenn Signale bereits „abgetastet“ wurden. Durch diese diskreten Werte können die Digitalsignale repräsentiert werden. Der Unterschied von den anfangs unverarbeiteten analogen Signalen und den verarbeiteten Digitalsignalen liegt in der Zeit. Das Analogsignal verläuft zeitkontinuierlich, stufenlose und ohne Unterbrechung, wohingegen Digitalsignale zeitdiskret sind und das Signal nur durch Punkte beschreibt.



3. Soundeffekte

Wie vorhin beschrieben, haben mehrere Faktoren einen Einfluss darauf, wie wir die Signale im Endeffekt wahrnehmen. Werden Parameter richtig verändert, kann man damit Eigenschaften eines Signals steuern. Weiters können die angeführten Aspekte auch dynamisch verändert werden

3.1 Geschwindigkeit

Durch das Verändern der Durchlaufgeschwindigkeit wird die Frequenz also auch die Tonhöhe beeinflusst.

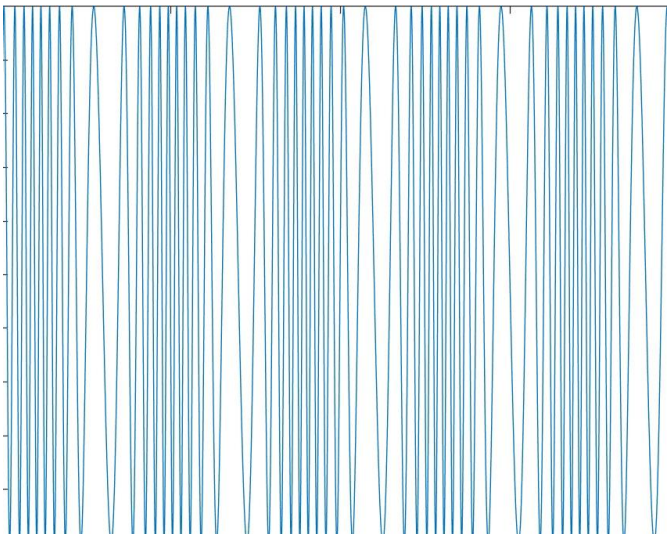
```
N = length(y);
for n=2:N;
 x=n/2;
y1 = y(floor(x));
y2 = y(ceil(x));

x1 = (floor(x));
x2 = (ceil(x));

z = y1 * (1-(x-x1)) + y2*(x-x1);
a(n)= z;
end
```

Verändert man die Geschwindigkeit dynamisch spricht man von einem Vibrato. Die Frequenz wird in bestimmten Abständen vergrößert oder verkleinert.
Floor bedeutet, dass der x/y-Wert auf den nächsten vorhandenen Wert aufgerundet wird.
Ceil rundet die x/y-Werte auf den nächsten vorhandenen Wert ab.

```
N = length(y);
for n=2:N/4;
 x=n*f(n);
y1 = y(floor(x));
y2 = y(ceil(x));

x1 = (floor(x));
x2 = (ceil(x));

z = y1 * (1-(x-x1)) + y2*(x-x1);
a(n)= z;
end
```

Mit Sinus:

```
N = length(y);
for n=2:N-4000;
x=n+(2*408*sin(2*pi*f(n)*2));
y1 = y(floor(x));
y2 = y(ceil(x));

x1 = (floor(x));
x2 = (ceil(x));

z = y1 * (1-(x-x1)) + y2*(x-x1);
a(n)= z;
end
```

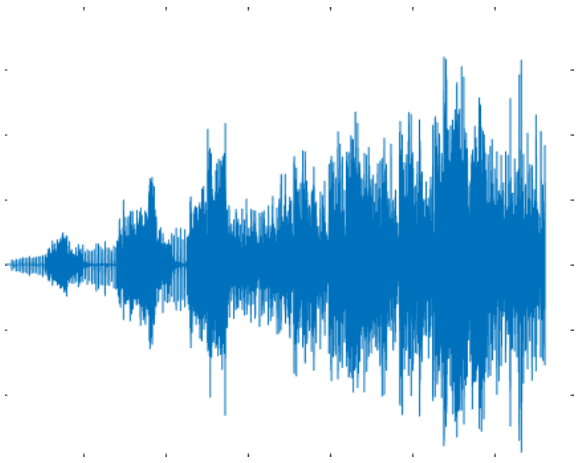


3.2 Lautstärke

Wie erwähnt verändert sich die Lautstärke durch die Amplitude. Je höher diese ist, desto lauter wird das Lied oder der Ton. Die ausgesuchte Lautstärke ist dabei konstant. Durch das Anfügen von einer Sinusfunktion kann auch zum Beispiel Tremolo, ein berühmter Effekt in der Musik, erzeugt werden.

Lineare Veränderung:
Die Lautstärke wird hier kontinuierlich lauter.

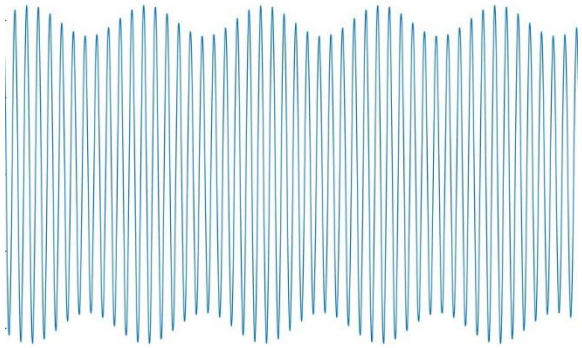
```
x = linspace(0, 50, length(y)).';

y = y.*(2*x+2);
```



Sinusveränderung:

In diesem Fall entsteht, wie bereits erwähnt, das Tremolo. Ein fortlaufendes auf - und abschwingen der Lautstärke.

```
x = linspace(0, 50, length(y)).';

y = y.*(60*sin(15*x)+70);
```



3.3 Echo

Das Echo beschreibt die Reflexion einer Schallwelle, welche mit einer so großen Verzögerung auftritt, dass sie als eigenständiges Hörereignis wahrgenommen werden kann. Im Vergleich zum Original hat das Echo zwar genau dieselbe Tonhöhe, jedoch einen leiseren Ton.
-> Formel + Formel Dynamisch

3.3.1 Chorus

Durch Verspäten mehrerer Tonspuren kann beim Chorus der Effekt erzielt werden, aus dem Ton einer Violine den Klang von vielen Violinen zu erzeugen.
Die Tonspur wird erzeugt durch die Formel:

$$D_k(n) = F_k + \frac{D_k}{2}\left(1 - \cos\left(2\pi \frac{f_k}{f_s} n\right)\right)$$

Dabei verwendet man mehrere verschiedene Werte von Verschiebungen(shifts), Frequenzen (freqs) und Amplituden (amp).

```
[x,fs]=audioread("356134__mtg__violin-c4.wav");
N = length(x);
v= linspace(0,3.2343, length(x));

amp = [0.01 0.014 0.02 0.018 0.017 0.013];
freqs = [0.2 -0.2 0.1 -0.1 0.15 -0.15];
shifts = [0.01 -0.01 0.05 -0.05 0.008 -0.008];
y = x;

for k = 1 : length(freqs)

  D = amp(k) * fs * sin(2*pi*v.*freqs(k)) + shifts(k)*fs;

for n = 10*amp*fs+1 : N - 10*amp*fs-1

   y(n) = y(n) +x(round(n + D(n)));

end
end
y = y / length(freqs);
  plot(y)
 audiowrite("violineChorus.wav", y, fs);
```

3.3.2 Räumliches Hören

Beim räumlichen Hören wird der Ton von einem Ort später wahrgenommen, wodurch vom Hirn die Richtung, aus der der Ton kommt, bestimmt werden kann.
Durch eine zweite Tonspur mit einem um die richtige Zeit verschobenen Ton wirkt es, als würde der Ton von einer Richtung kommen.

0,00058309 ist die Zeit in Sekunden, die der Schall braucht, um von einem (20cm entfernten) Ohr zum anderen zu wandern, wenn er orthogonal zu den beiden Ohren verläuft.

```
N=length(round(w));
for n = round(5.8309e-04*fs+1) . N
    v(n) = w(round(n - 5.8309e-04*fs));
end
y=[w(1:length(v)),v.'];
```

3.3.3 Dynamisch Räumliches Hören

Der Ton ändert durch den Sinus alle 2 Sekunden die Richtung, kommt allerdings nur von zwei verschiedenen Punkten.

```
[w,fs]=audioread("BabyElephantWalk60.wav");
N=length(round(w));
t=linspace(0,60,N);
D =  5.8309e-04 * sin(2*pi*t*0.5);

for n = round(D*fs+1) : N-(round(D*fs))

v(n) = w(round(n + D(n)*fs));

end
y=[w(1:length(v)),v.'];
audiowrite("BabyElephantWalk60RaumDyn.wav", y, fs);
```

3.3.4 360° Effekt

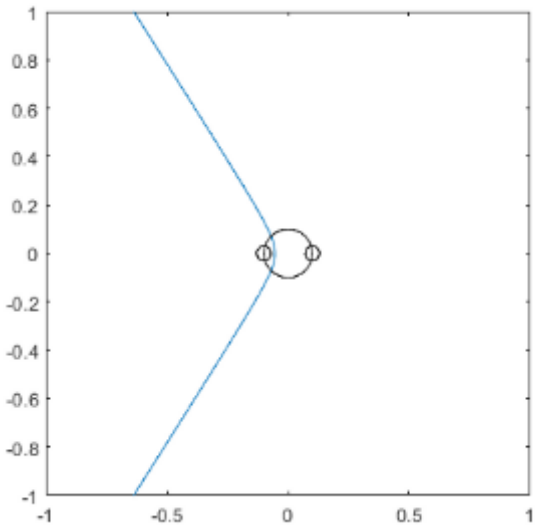
Durch Parametrisieren eines Kreises kann man für jeden Zeitpunkt (n) den Unterschied der Abstände vom Punkt (P), zu den beiden Ohren (O1 und O2) berechnen. Durch Dividieren dieses Abstandes durch die Schallgeschwindigkeit ergibt sich die zeitliche Verschiebung (t) für den jeweiligen Punkt.
So wirkt es, als würde sich der Ton um einen herumbewegen.

```
O1 = [0.1,0];
O2 = [-0.1,0];
alpha = linspace(1,16*pi,length(w));

for n = round(abs(0.1)*fs+1) : N - round(0.1*fs)

    P = [cos(alpha(n)),sin(alpha(n))];

    v1 = O1 - P;
    v2 = O2 - P;

    dif = norm(v1) - norm(v2);
    t = (dif)/343;
    v(n) = w(round(n - t*fs));
end
y=[w(1:length(v)),v.'];
```

3.3.5 Funktion zur Berechnung eines Punktes (Delay gegeben)

Mithilfe des unten abgebildeten Codes ist es möglich Punkte zu berechnen, aus dem die jeweiligen Geräusche kommen. Weiters kann man im Plot (GIF) erkennen, dass sich der Punkt auf einem Kreis bewegt, welcher logischerweise einen größeren Radius als unser Kopf besitzt, da das Geräusch nicht aus unserem Kopf kommt. In weiterer Folge ist zu erwähnen, dass fast alle Richtungen (Strecken, welche unten im Bild blau abgebildet sind) eine Gegenrichtung haben. Somit gibt es fast immer zwei Richtungen, die sich für das menschliche Ohr gleich anhören, da sie denselben Delay besitzen. Durch die erstellte Funktion kann man, wie bereits erwähnt, Punkte entlang diese Richtungen berechnen.
Die Punkte Q entlang der Linie ergeben sich als Lösung der Gleichung:

$$d = |\vec{QO}_1| - |\vec{QO}_2|$$



Die ersten Zeilen des Codes beinhalten, die wichtigsten Funktionen, da sie für die Berechnung des Punktes zuständig sind. Die weiteren Zeilen skizzieren einen menschlichen Kopf mit Ohren. In weiterer Folge sind sie auch dafür zuständig dem Plot das gewünschte Aussehen zu verleihen, da man ansonsten nicht erkennen könnte, wo sich der berechnete Punkt befindet.

```
O1=[0.1,0];
O2=[-0.1,0];
d=(0.000358*300);
f=@(Px,Py) sqrt((Px-0.1)^2+(Py-0)^2) -
sqrt((Px-(-0.1))^2+(Py-0)^2)-d;
fimplicit(f,[-1,1,-1,1]);


x = 0;
y = 0;
r = 0.1;


d = r*2;
px = x-r;
py = y-r;
h = rectangle('Position',[px py d
d],'Curvature',[1,1]);
daspect([1,1,1])



x = -0.1;
y = 0;
r = 0.03;


d = r*2;
px = x-r;
py = y-r;
h = rectangle('Position',[px py d
d],'Curvature',[1,1]);
daspect([1,1,1])



x = 0.1;
y = 0;
r = 0.03;


d = r*2;
px = x-r;
py = y-r;
h = rectangle('Position',[px py d
d],'Curvature',[1,1]);
daspect([1,1,1])
```

4 Diskrete Fourier-Transformation

Der Diskreten Fourier-Transformation hat in der Signalverarbeitung einen großen Stellenwert.
Sie bietet die Möglichkeit, ein Signal in seine Bestandteile zerlegt. Als Ergebnis erhält man die einzelnen Frequenzen, mit der jeweiligen Intensität. Wie später gezeigt wird, kann sie dadurch für die Manipulation von Signalen bestens genutzt werden, da es weitaus einfacher ist, einzelne Werte zu verändern, anstelle eines ganzen Signals.
Die Formel zur Berechnung lautet wie folgt:

$$w_k = \frac{1}{N}\left[\sum_{j=0}^{N-1} e^{-2\pi i \frac{jk}{N}} x_j\right]$$

4.1 Beweise der diskreten Fourier-Transformation

Während des Projekts wurden einige Beweise aufgestellt, die die für uns hilfreichen Eigenschaften der diskreten Fourier-Transformation erklären, was zu einem tieferen Verständnis der Gegebenheiten führte.

4.1.1 Linearität

Der erste Beweis soll die diskrete Fourier-Transformation hinsichtlich ihrer Linearität überprüfen:
Zu zeigen:

$$\mathcal{F}(x+y) = \mathcal{F}(x) + \mathcal{F}(y)$$

Zeige also:

$$\mathcal{F}(x+y)_k = \mathcal{F}(x)_k + \mathcal{F}(y)_k$$

Für alle k von 0 bis N-1:

$$\begin{aligned}
\mathcal{F}(x)_k + \mathcal{F}(y)_k &= \frac{1}{N}\left[\sum_{j=0}^{N-1} e^{-2\pi i \frac{jk}{N}} x_j\right] + \frac{1}{N}\left[\sum_{j=0}^{N-1} e^{-2\pi i \frac{jk}{N}} y_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} e^{-2\pi i \frac{jk}{N}} x_j + e^{-2\pi i \frac{jk}{N}} y_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} e^{-2\pi i \frac{jk}{N}} (x_j + y_j)\right] \\
&= \mathcal{F}(x+y)_k
\end{aligned}$$

Weites soll gelten:

$$a\mathcal{F}(x) = \mathcal{F}(ax)$$

Der Beweis hierfür ist dem ersten ähnlich, weswegen er als trivial wahrgenommen wird.

Dieser Beweis erklärt eine grundlegende Eigenschaft der diskreten Fourier-Transformation.
Er besagt, dass eine Addition der Ausgangswellen zum selben Ergebnis führen würde, wie das Zusammenrechnen der Ergebnisse. Dadurch ist es nicht von Relevanz wie viele Frequenzen in einem Signal stecken und es ist möglich, dass die diskrete Fourier-Transformation komplexere Signale zuverlässig in ihre Einzelteile zerlegt.

4.1.2 Periodizität

Dieser Beweis soll eine Periodizität der erhaltenen Funktion beweisen.

Zu zeigen:

$$w_k = w_{k+N}$$

Für alle k von 0 bis N-1:

$$\begin{aligned}
w_{k+N} &= \frac{1}{N}\left[\sum_{j=0}^{N-1} e^{-2\pi i \frac{j(k+N)}{N}} x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} (\cos(-2\pi \frac{j(k+N)}{N}) + i\sin(-2\pi \frac{j(k+N)}{N}))x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} (\cos(-2\pi j(\frac{k}{N} + \frac{N}{N})) + i\sin(-2\pi j(\frac{k}{N} + \frac{N}{N})))x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} (\cos(-2\pi j\frac{k}{N} - 2\pi j\frac{N}{N}) + i\sin(-2\pi j\frac{k}{N} - 2\pi j\frac{N}{N}))x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} (\cos(-2\pi j\frac{k}{N}) + i\sin(-2\pi j\frac{k}{N}))x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} e^{-2\pi i \frac{jk}{N}} x_j\right] \\
&= w_k
\end{aligned}$$

Dieser Beweis erklärt die Sinnlosigkeit weiterer Berechnungen, um eventuell mehr Werte zu erhalten, denn durch die Periodizität würde dies lediglich zum erneuten Erhalt der bereits berechneten Werte führen.

4.1.3 Symmetrie

Ein Beweis der Symmetrie würde die symmetrische Anordnung der diskreten Fourier-Transformation von reellen Signalen erklären

Zu zeigen:

$$w_k = \overline{w_{k-N}}$$

Für alle k von 0 bis N-1:

$$\begin{aligned}
\overline{w_{k-N}} &= \overline{\frac{1}{N}\left[\sum_{j=0}^{N-1} e^{-2\pi i \frac{j(k-N)}{N}} x_j\right]} \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} \overline{(\cos(-2\pi\frac{j(k-N)}{N}) + i\sin(-2\pi\frac{j(k-N)}{N}))x_j}\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} \cos(-2\pi\frac{j(k-N)}{N})x_j - i\sin(-2\pi\frac{j(k-N)}{N})x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} \cos(2\pi j\frac{k}{N} - 2\pi j\frac{N}{N})x_j - i\sin(2\pi j\frac{k}{N} - 2\pi j\frac{N}{N})x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} (\cos(2\pi j\frac{k}{N}) - i\sin(2\pi j\frac{k}{N}))x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} (\cos(-2\pi j\frac{k}{N}) + i\sin(-2\pi j\frac{k}{N}))x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} e^{-2\pi i \frac{jk}{N}} x_j\right] \\
&= w_k
\end{aligned}$$

4.1.4 Shift

Dieser Beweis erklärt einen entstehenden Shift nach der Multiplikation einer weiteren Welle

Zu zeigen:

$$w_{k-m} = \mathcal{F}(x_n e^{-2\pi i \frac{n\pi}{N}})$$

Für alle k von 0 bis N-1:

$$
\begin{aligned}
\mathcal{F}(x_n e^{-2\pi i \frac{nm}{N}}) &= \frac{1}{N}\left[\sum_{j=0}^{N-1} e^{-2\pi i \frac{jk}{N}} e^{2\pi i \frac{nm}{N}} x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} e^{2\pi i \frac{nm-jk}{N}} x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} (\cos(2\pi \frac{nm-jk}{N}) + i\sin(2\pi \frac{nm-jk}{N}))x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} \cos(2\pi \frac{nm}{N} - 2\pi \frac{jk}{N})x_j - i\sin(2\pi \frac{nm}{N} - 2\pi \frac{jk}{N})x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} \cos(2\pi \frac{m}{N} - 2\pi \frac{k}{N})x_j - i\sin(2\pi \frac{m}{N} - 2\pi \frac{k}{N})x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} (\cos(2\pi \frac{m-k}{N}) + i\sin(2\pi \frac{m-k}{N}))x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} (\cos(2\pi j\frac{m-k}{N}) + i\sin(2\pi j\frac{m-k}{N}))x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} (\cos(-2\pi j\frac{k-m}{N}) + i\sin(-2\pi j\frac{k-m}{N}))x_j\right] \\
&= \frac{1}{N}\left[\sum_{j=0}^{N-1} e^{2\pi i \frac{j(k-m)}{N}} x_j\right] \\
&= w_{k-m}
\end{aligned}
$$

Dieser Beweis erklärt einen entstehenden Shift bei der Einführung einer zweiten Welle, die für eine Verschiebung entlang der x-Achse verantwortlich ist.

4.2 Short-Term Fouriertransformation

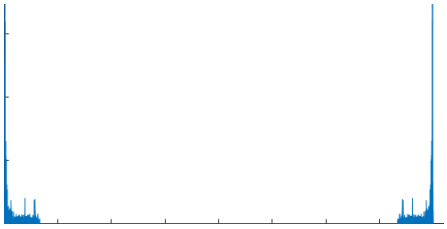
Mit der Short Term Fouriertransformation (stft) kann man die diskrete Fouriertransformation in einzelne, von der Zeit abhängige Abschnitte (Fenster) unterteilen. Damit kann man .wav-Dateien in .mp3-Dateien komprimieren oder auch eine Software für Musikerkennung, wie Shazam schreiben

4.3 Anwendung von Fouriertransformation

4.3.1 Lowpass

Mit der Funktion „lowpass“ kann man die Frequenzen eines mit der diskreten Fouriertransformation bearbeiteten Audiosignals so verändern, dass man die höheren Frequenz auf null setzt. Dazu legt man einen bestimmten „cutoff“ mit einer genau Hz Anzahl fest und setzt so die Frequenzen, die diese Herz-Anzahl überschreiten, auf null.
Bei dem Beispiel werden alle Frequenzen ab 3000Hz gelöscht. Also bleiben nur niedrige Töne übrig.

```
[y, fs]=audioread("CantinaBand60.wav");
s = 60;

cutoff = 3000;

y = lowpass(y, cutoff  , fs);

audioread("CantinaBand60LowPass.wav", fs);
```



Auch ohne den Befehl “lowpass” kann man einen Tiefpassfilter programmieren:

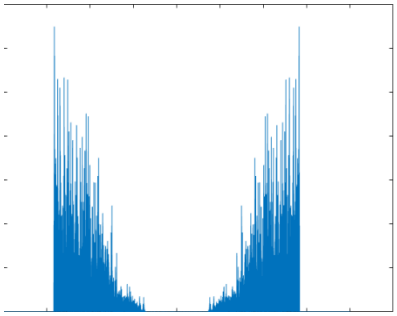
```
[y,fs] = audioread("rammstein-style-metal-10726.wav");
y = y(:,1);
s = length(y)/fs;
freq = (fft(y));

cutoff = 1200*s;
cutoff2= length(freq)-cutoff;

freq(cutoff:cutoff2)= 0;

y = abs(ifft(freq));

audiowrite("rammstein-style-metal-10726Tiefpass.wav", y,fs);
```

Wegen der Parallelität muss ein zweiter “cutoff” symmetrisch zum anderen gesetzt werden. Alles zwischen diesen “cutoff” und “cutoff2” wird herausgeschnitten.

4.3.2 Highpass

Die Funktion „highpass“ auf der anderen Seite lässt bei einem mit der diskreten Fouriertransformation Signal nur die Frequenzen durch welche über der mit dem „cutoff“ festgelegten Hz – Anzahl sind. So bleiben nur hohe Töne erhalten.

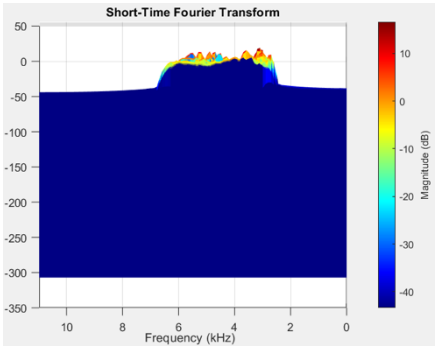
```
[y,fs]=audioread("PinkPanther60.wav");

s=60 ;

cutoff= 367.5;

y= highpass(y, cutoff, fs);

audiowrite("PinkPanther_Highpass.wav", y, fs);
```



4.3.3 Bandpass

Der „Bandpass“ ist eine Kombination von High- und Lowpass bei dem sowohl die unteren als auch die oberen Frequenzen gelöscht werden, sodass nur mehr mittlere Frequenzen abgespielt werden.
Bei diesem Beispiel werden nur Frequenzen zwischen 3000Hz und 6000Hz abgespielt. Durch den “Bandpass” kann man mit Songs einem “Retro-Effekt” belegen.

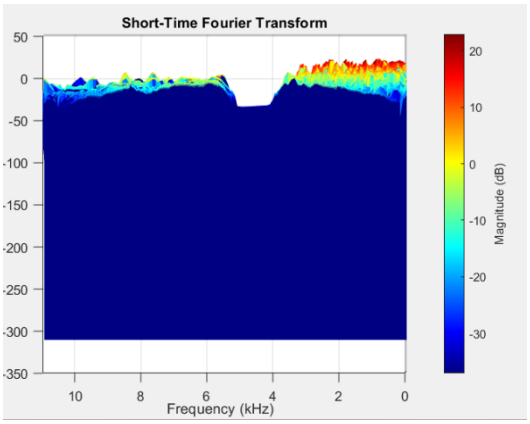
```
[y , fs] = audioread ("BabyElephantWalk60.wav") ;

cutoff1= 6000;
cutoff2=3000;

y = lowpass(y,cutoff1,fs);
y = highpass(y,cutoff2,fs);

audiowrite ("BabyElephantWalk60Bandpass.wav",y,fs)
```



4.3.4 Stopband

Der “Stopband” ist eine Kombination von High- und Lowpass die alle Frequenzen außerhalb eines gewissen Frequenzbereiches abspielt.
Beim Beispiel werden alle Frequenzen außer die zwischen 3000Hz und 6000Hz abgespielt.
Durch das “Stopband” macht man beim Song-Mastering das Frequenzband frei.

```
[y,fs] = audioread ("Violine.wav") ;

cutoff1= 3000;
cutoff2=6000;

x1= lowpass(y,cutoff1,fs);
x2= highpass(y,cutoff2,fs);

y = x1 + x2;

audiowrite("Violine_SB.wav", y, fs);
```

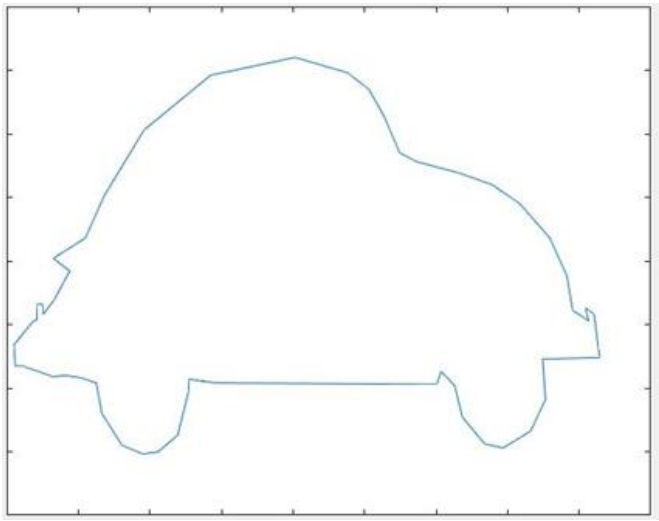


4.4 Visualisierung der Auswirkungen von Highpass und Lowpass auf Signale

Obwohl man die Auswirkungen von Highpass und Lowpass zwar hört, sind sie visuell in einer Audiodatei schwer zu erkennen, deshalb: Visualisierung der Auswirkungen anhand einer Zeichnung mit 2 Signalen

Wenn man die, für die Zeichnung notwendigen, Punktkoordinaten in einen x – Vektor (Signal 1) und einen y – Vektor (Signal 2) aufgeteilt hat, kann man durch die Funktion „plot“ in einem Koordinatensystem darstellen.

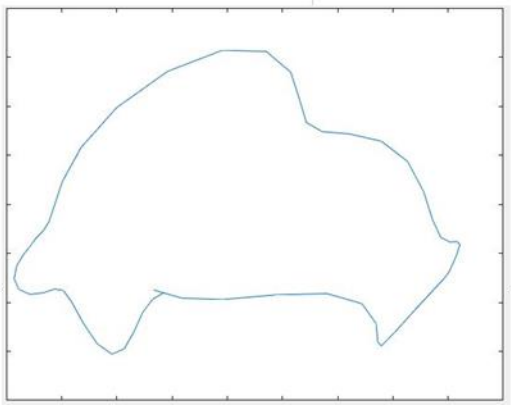
Ursprüngliche Zeichnung:



4.4.1 Veränderung durch den Lowpass (löschen von hohen Frequenzen)

Wenn man auf diese Koordinaten noch einen Lowpass anwendet, so verändert sich die Zeichnung in dem Sinn, dass die Ecken etwas abgerundet werden die allgemeine Form jedoch erhalten bleibt.

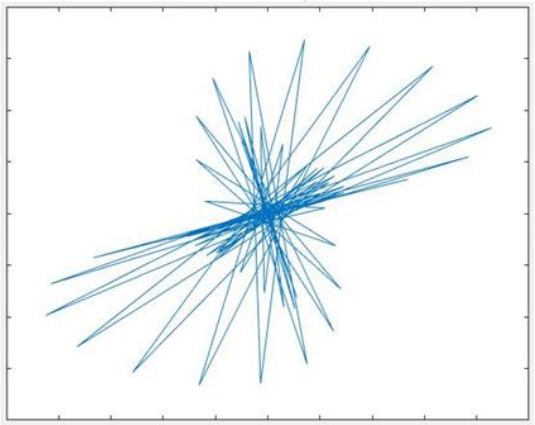
```
p = [276,395,255,393,254,403,239,437,212,450,191,452,161,445,133,420,125,396,104,392,82,390,65,391,44,387,24
x = p(1:2:length(p));
y = p(2:2:length(p))*-1;


fs= length(x)
fs1=length(y)
cutoff= 27;
x =  lowpass(x, cutoff , fs);

y = lowpass(y, cutoff , fs1);

plot(x,y)
```



4.4.2 Veränderung durch den Highpass (löschen von niedrigen Frequenzen):

Wird auf die Zeichnung jedoch ein Highpass angewendet so bleiben die Kanten der Zeichnung übrig und die vorherige Form rückt eher in den Hintergrund

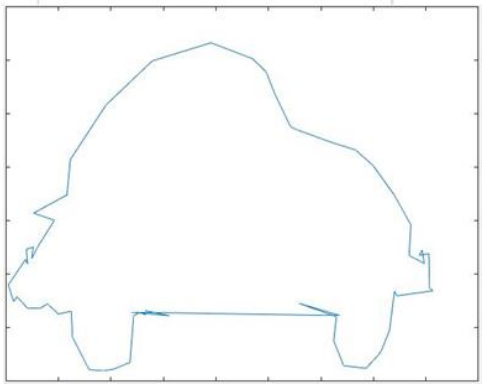
```
p = [276,395,255,393,254,403,239,437,212,450,191,452,161,445,133,420,125,396,104,392,82,390,65,391,44,387,24
x = p(1:2:length(p));
y = p(2:2:length(p))*-1;


fs= length(x)
fs1=length(y)
cutoff= 27;
x =  highpass(x, cutoff , fs);

y = highpass(y, cutoff , fs1);

plot(x,y)
```



4.4.3 Kantenschärfung (Verstärkung der hohen Frequenzen)

Mithilfe dieser beiden Manipulationen kann man bei einer erstellten Zeichnung auch einen Effekt mit dem Namen „Kantenschärfung“ erreichen. Bei der Kantenschärfung werden Lowpass mit 2*Highpass addiert, was zu einer Verstärkung der hohen Frequenzen, in diesem Fall der Kanten, führt.

```
p = [276,395,255,393,254,403,239,437,212,450,191,452,161,445,133,420,125,396,104,392,82,390,65,391,44,387,24
x=p(1:2:length(p));
y=p(2:2:length(p))*-1;


cutoff=length(x)/3


x = lowpass(x, cutoff, length(x))+2*highpass(x, cutoff+1, length(x));

y = lowpass(y, cutoff, length(y))+2*highpass(y, cutoff+1, length(y));

plot(x,y);
```



4.5 High- und Lowpass in der Bildverarbeitung

High- und Lowpass sind nicht nur für die Audioverarbeitung wichtige Filter, sondern auch in der Bildbearbeitung essenziell. Mithilfe dieser Filter ist es möglich, einerseits die Farbkanten zu erkennen und zu verdeutlichen, andererseits die Kanten verschwimmen zu lassen. Die unten angefügten Bilder verdeutlichen die oben genannten Fakten.

Ausgangsbild

Mit Highpass- Filter

Praktisch kann der Highpass-Filter zum Nachschärfen genutzt werden. Der Filter sucht Kanten mit großen Helligkeitsunterschieden – so erhält man ein Ergebnis. Durch Übereinanderlegen des Ausgangsbildes und dem Ergebnis des Highpass-Filters kann beispielsweise ein Bild, welches unscharf wirkt, wie etwa durch mangelnden Kontrast, scharf wirken lassen.

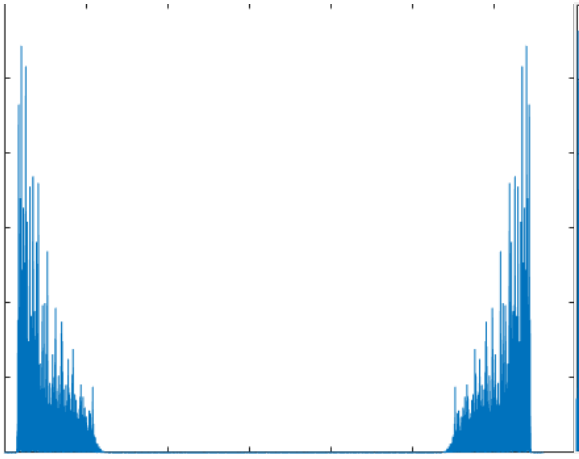
Mit Lowpass- Filter

Kanten geschärft

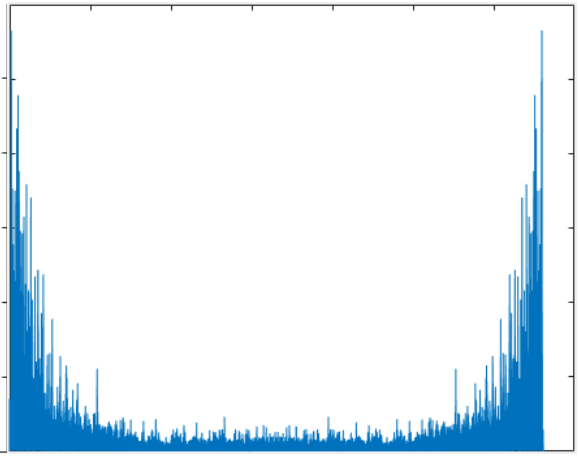
5. MP3

Beim Konvertieren von ".wav" zu ".mp3" Dateien werden durch die Fouriertransformation kleine Frequenzen hinausgefiltert, was zwar zu Datenverlusten führt, aber richtig angewandt nicht bemerkbar ist.
Beim Abspielen wird das gefilterte Signal wieder in eine Welle übersetzt. So kann Speicher gespart werden.

Mit MP3:



Ohne MP3:



Eine einfache Version von MP3 lässt sich umsetzen, indem man die insgesamt lauteste Frequenz der Audiodatei ausliest und alle Frequenzen, die weniger als 0,1-Mal der eben gemessenen Frequenz entsprechen, werden gleich Null gesetzt.

```
N=length(y);
for n= 2:N

if   abs(fft(y(n)))<0.1*max(abs(fft(y)));
     abs(fft(y(n)))=0;

end
end
y=abs(ifft(y));
```

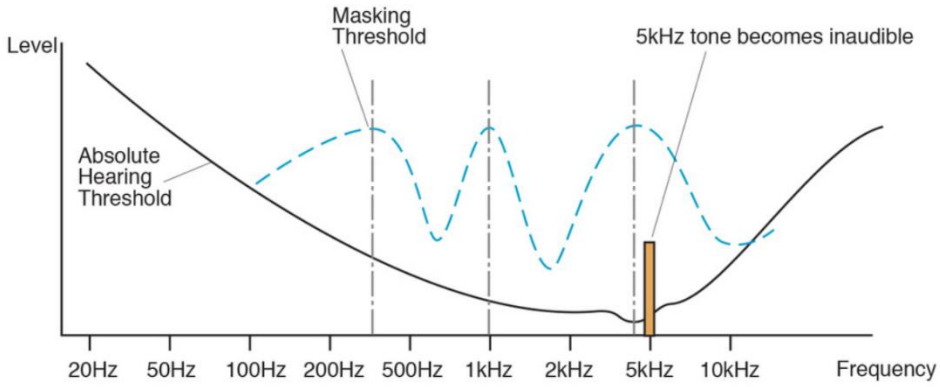
Diese Version hört allerdings kaum wie das Original an.

Um eine verbesserte Version dieser Kompression zu erzielen, muss man die Audiospur mithilfe der Kurzzeit-Fourier-Transformation (stft) in mehrere Fenster aufgeteilt werden.
Nun wird je Fenster die lauteste Frequenz ausgelesen und die, im Fenster zu leisen Frequenzen, gelöscht.

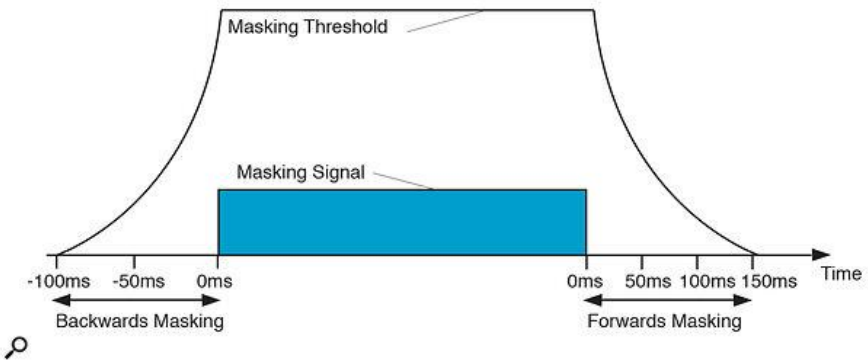
```
[szeil, sspal] = size(s);

for k = 1 : sspal
    ab = s(:,k);
    m = max(abs(ab));
    for n = 1 : szeil

        if(abs(ab(n)) <= 0.02 * m)

            ab(n) = 0;
        end
    end

    s(:,k) = ab;
end
```

Um die Frequenzlöschung zu optimieren und nicht hörbare Frequenzen zu löschen, benutzt man bei echten MP3 noch zusätzlich einen PAC (perceptual audio coder). Dabei sind drei Faktoren wichtig, um zu bestimmen welche Frequenzen gelöscht werden können:

- Welche Anteile zu niedrig sind hängt von der Frequenz ab (am Rand des Hörbereichs sind lautere nötig)
- Auf ein lautes Signal folgend können leise nicht wahrgenommen werden



- Am Frequenzband: Frequenzen in der Nähe einer lauten können auch nicht gut wahrgenommen werden



6. Shazam

Shazam ist eine Software, welche es ermöglicht, innerhalb von wenigen Sekunden oder sogar in unter einer Sekunde jegliche Musikstücke zu erkennen. Aufgenommene Audiosequenzen werden analysiert und aufgrund zeitpunktspezifischer Fingerprints, verschiedenen Liedern zugeteilt. Am Ende wird angegeben, welches Lied wie viele Treffer erzielt hat. Zusätzlich kann noch analysiert werden, bei welchem Lied die Treffer in Sequenz und nicht quer durchs Lied verteilt aufgetreten sind, um die Trefferquote zu erhöhen.

6.1 Fingerprints

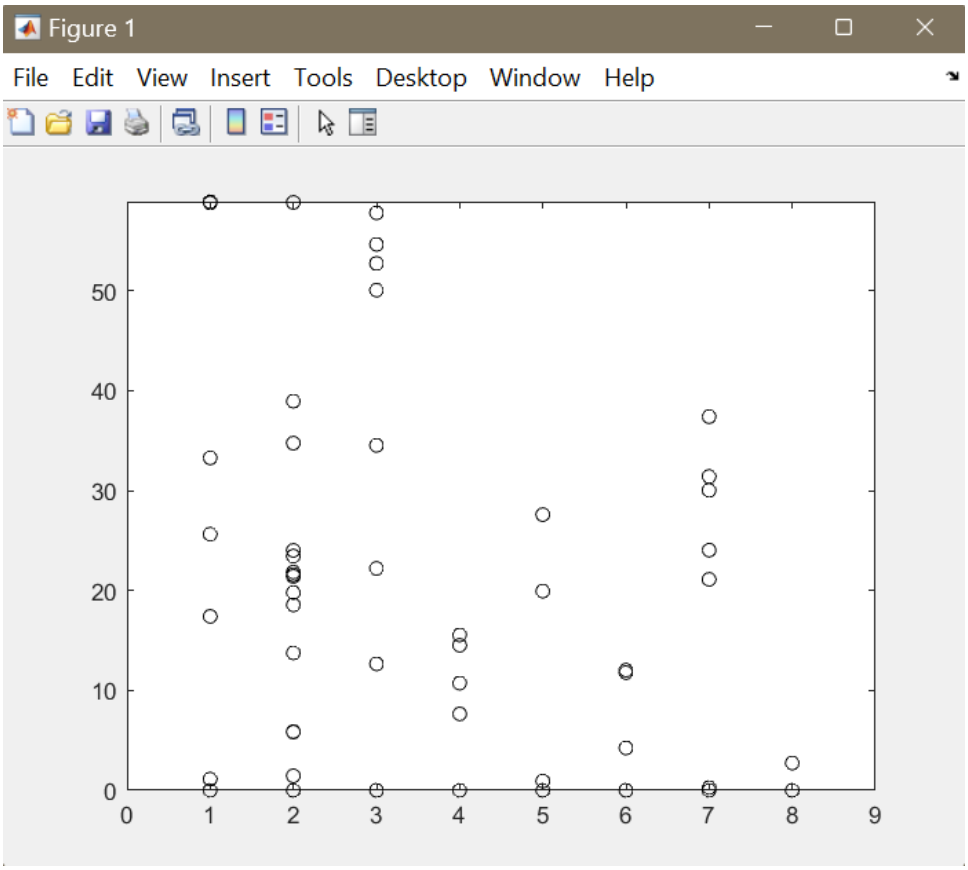
Um Frequenzen in einem Lied zu entziffern, muss man die Dateien des Liedes umwandeln. Vom Phasenraum in den Frequenzraum. Durch die Fouriertransformation kann dies erzielt werden, jedoch gibt es einen Nebeneffekt, der die Auslesung der Daten beeinträchtigen. Das Timing der Frequenzen ist nicht mehr zu erkennen. Daher werden Windows (Datenstücke), welche nur diesen Teil der Information transformieren, erstellt. Die Größe dieser Windows können variabel angefertigt werden. In diesem Fall werden 20 Windows pro Sekunde erzeugt. Weiters unterteilt man diese Windows auch in Bänder (im angeführten Code: 5). Aus diesen Bändern werden die jeweils lautesten Frequenzen herausgefiltert und dann zu einer gemeinsamen Zahl zusammengefasst. Dies ist der Fingerprint welcher anschließend in einer Hashmap gespeichert werden.

6.2 Hashmap

Als Hashmap bezeichnet man eine Datentabelle, welche durch ihre spezielle Indexstruktur das Aufspüren von verschiedensten Datenobjekten ermöglicht. Genauer funktioniert dies, weil die Position jedes Daten Objekts über eine bestimmte Hashfunktion definiert ist. Die Hashfunktionen bestehen aus einem Schlüssel-Wert-Paar. So kann über den Schlüssel der Wert abgerufen und so gefunden werden. Die Besonderheit einer Hashmap definiert sich im Zeitaufwand, welcher für das Auffinden draufgeht. Verglichen mit anderen Indexstrukturen bleibt konstant und ist nicht davon abhängig, wie groß die Tabelle im Endeffekt ist.

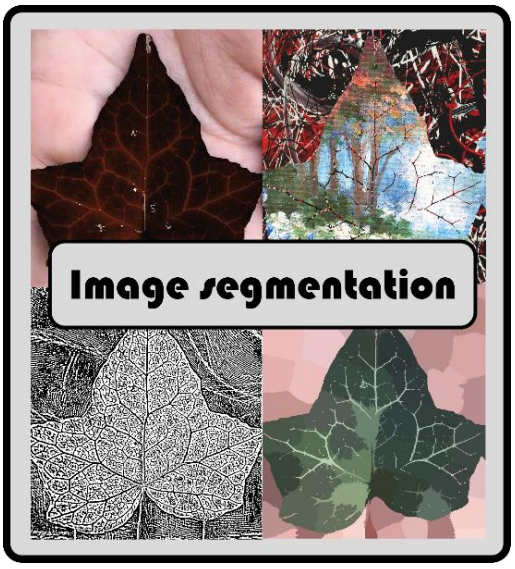
6.3 Stärken und Schwächen

Der vereinfachte Code hat weder Probleme mit veränderter Geschwindigkeit noch mit einer anderen Lautstärke. Noise, also Nebengeräusche wie zum Beispiel Stimmen oder Autogeräusche, beeinträchtigen die Ergebnisse des Outputs auch nicht. Jedoch kann bei veränderter Tonart kein Ergebnis gefunden werden, da die Frequenzen nicht mehr mit den Fingerprints übereinstimmen.




```
matches:  19 , song:  Fanfare60.wav , deviation 11.2405
matches:  18 , song:  BabyElephantWalk60.wav , deviation 19.6873
matches:  18 , song:  StarWars60.wav , deviation 13.505
matches:  15 , song:  PinkPanther60.wav , deviation 8.1638
matches:  14 , song:  CantinaBand60.wav , deviation 14.781
```

Das richtige Lied (2) wurde anhand von Aufnahme mit 2s Länge erkannt. Im Plot sind die gematchten Zeitpunkte ersichtlich



Die Mathematik hinter Photoshop & Co.

Datum: 17. 02. 2023

Betreuer: Gabriel Pichlbauer

Antonia Grabner, Farin Schreiner, Lorena Beraus, Nikolaus Skrabitz,
Julia Schrötter, Johannes J. Reisinger

Inhaltsverzeichnis

Einleitung

Image Segmentation hat das Ziel, Bilder bestmöglich in unterschiedliche, sinnvolle Bereiche zu unterteilen (segmentieren). Dafür wird anhand der Eigenschaften der Bildpunkte versucht zu erkennen welche Pixel zu einer gemeinsamen Gruppe gehören. Wichtig ist, dass eine sinnvolle Zuordnung von Bildpunkten zu sinnvollen Objekten stattfindet. Beispielweise können Bildbereiche in Vordergrund & Hintergrund, Hell & Dunkel oder generell nach Farbe & Form aufgeteilt werden.

Für die Umsetzung der einzelnen Projekte wurde das Programm *Octave* verwendet.

1 Mathematische Beschreibung von Bildern

Um mit Bildern arbeiten zu können, müssen wir erst einmal verstehen was ein Bild überhaupt ist: Grundsätzlich kann man sich ein Bild als ein Rechteck aus Zahlen (Matrix) vorstellen, die im Fall des Graustufenbildes Werte zwischen 0 und 255 annehmen. Dabei steht der Wert 0 für Schwarz und der Wert 255 für Weiß. Alle Werte, die dazwischen liegen, geben unterschiedliche Grautöne aus.

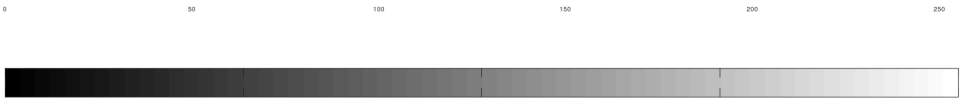


Abbildung 1: Graustufenskala

Um auch Farben darstellen zu können wird ein Pixel nicht nur durch einen Wert, sondern durch ein Tripel von Werten charakterisiert, wobei es dazu mehrere Möglichkeiten gibt. Der am häufigsten verwendete Farbraum ist der RGB Farbraum, bei dem jedem Pixel drei integer Werte zwischen 0 und 255 zugewiesen werden. Der erste steht für den „Rotwert “ der zweite für den „Grünwert “und der dritte für den „Blauwert “. Bei jeder Farbe steht 0 jeweils für die dunkelste und 255 für die hellste Farbabstufung. So können bei Verwendung von 3 gleichen Zahlen auch schwarz, weiß und alle Grautöne gebildet werden. Doch es gibt auch noch andere Modelle um die Farben der Pixel zu bestimmen. Der zweite Farbraum, den wir im Laufe der Woche angewendet haben, ist der HSV Farbraum. Hier beschreibt das „H “ die Farbe mithilfe eines Farbkreises (engl.: hue), während das „S “ für die Farbsättigung steht (engl.: saturation). Das „V “ gibt die Helligkeitsstufe eines Pixels an (engl.: value). Der dritte verwendete Farbraum war der Lab Farbraum. Hier beschreibt das „L “ den Helligkeitswert (engl.: Lightness) der Farbe, während die a-Koordinate einen Farbton in unterschiedlicher Intensität zwischen Grün und Rot angibt, und die b-Koordinate einen zwischen Blau und Gelb.

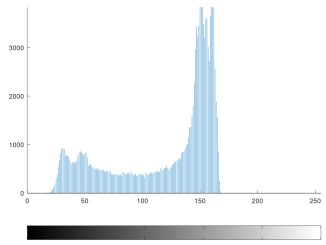
Octave bietet Möglichkeiten Bilder von jedem dieser Farbräume in jeden anderen zu konvertieren. Mit den Befehlen „rgb2lab() “ und „rgb2hsv() “ kann man Bilder in andere Farbräume verschieben und umgekehrt mit den Befehlen „lab2rgb “ und „hsv2rgb “ auch wieder zurück in den RGB-Farbraum konvertieren. Diese Farbräume haben wir benutzt, um bei ganzen Bildern sowohl die Farbe, als auch die Farbintensität und Helligkeit zu verändern. Analog kann auch lediglich die Farbe eines Bildsegmentes verändert werden.

2 Segmentierung von Graustufenbildern

Die Segmentierung bei Graustufenbildern ist am Beginn einfacher, weil man nur einen Wert beachten muss. Mit der Analyse und Veränderung der einzelnen Pixelwerte können Bilder in verschiedene Bereiche wie etwa Hell & Dunkel unterteilt werden.

2.1 Schwarz-Weiß Segmentierung

Das erste Projekt, mit dem wir uns beschäftigt haben, war die Segmentierung des Graustufenbildes eines Barcodes, bei dem wir händisch einen Schwellwert (Threshold) ausgesucht haben und allen Pixeln, die einen höheren Wert als den Threshold hatten, den Wert 255 (Weiß) und allen, die einen niedrigeren Wert hatten, den Wert 0 (Schwarz) zugewiesen haben. Um einen passenden Schwellwert auszuwählen, haben wir das Histogramm analysiert, welches die Verteilung der Farbwerte zwischen 0 und 255 anzeigt. Denn nicht bei jedem Graustufenbild ist der niedrigste Wert 0 und der höchste 255. Bei der Betrachtung des Histogramms kam die Idee auf, als Schwellwert den Mittelwert der Farbwerte zu nehmen. Damit haben wir auch einen Ansatzpunkt für den Schwellwert bei anderen Bilder gefunden. Nach einigem Herumprobieren bei verschiedenen Bildern wurde aber klar, dass diese Methode nicht bei allen Bildern ausreichend gut funktioniert. Vor allem wenn die Belichtung des Bildes nicht konstant ist, gibt es Probleme bei der Segmentierung. Für eine alternative Approximation kann auch der Median als Schwellwert gewählt werden.



(a) Histogramm des Ursprungsbildes

(b) oben: Ursprungsbild
unten: Schwarz-Weiß-Bild

Abbildung 2: Über einen Threshold segmentierter Barcode

2.1.1 Otsu-Thresholding

Das Otsu-Thresholding ist nach Nobuyuki Otsu benannt und eine Methode zum Ermitteln eines oder mehrerer Thresholds. Dabei werden sowohl die Varianz als Streuungsmaß der Werte als auch die Größe der Gruppen beim Finden des Schwellwertes miteinbezogen. Der Algorithmus versucht die Intra-Klassen Varianz σ_w^2 zu minimieren bzw. die Inter-Klassen Varianz σ_b^2 zu maximieren.

Wenn $x_1, x_2, ..., x_n \in \mathbb{R}$ gegeben und nach Größe sortiert sind und ein Schwellenwert $t \in \mathbb{R}$ gegeben ist, entstehen zwei Mengen:

$$M_1 = \{x_1, x_2, ..., x_m\} \text{ mit } x_m < t$$

$$M_2 = \{x_{m+1}, x_{m+2}, ..., x_n\} \text{ mit } x_{m+1} \geq t$$

Wir bezeichnen σ_1^2 und σ_2^2 die jeweiligen Varianzen der Gruppen M_1 und M_2 und mit σ^2 die Varianz des gesamten Bildes. Der Zusammenhang zwischen diesen lautet:

$$\sigma^2 = \sigma_1^2 + \sigma_2^2$$

Es bezeichne w_1 und w_2 weiters die relative Gruppengröße von M_1 und M_2, also

$$w_1 = \frac{m}{n} \qquad w_2 = 1 - w_1.$$

Dann ist die Intra-Klassen Varianz σ_w^2 beziehungsweise die Inter-Klassen Varianz σ_b^2 definiert durch

$$\sigma_w^2(t) = w_1(t) \cdot \sigma_1^2(t) + w_2(t) \cdot \sigma_2^2(t)$$

$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t)$$

$$\iff \quad \sigma_b^2(t) = w_1(t) \cdot (\overline{x} - \overline{x}_1(t))^2 + w_2(t) \cdot (\overline{x} - \overline{x}_2(t))^2$$

$$\iff \quad \sigma_b^2(t) = w_1(t) \cdot w_2(t) \cdot (\overline{x}_1(t) - \overline{x}_2(t))^2$$

.

Hier bezeichnet $\overline{x}_1(t)$ und $\overline{x}_2(t)$ den Mittelwert der Gruppen M_1 und M_2, und $\overline{x}$ den Mittelwert des Gesamtbildes. Im Programm haben wir diesen Algorithmus so umgesetzt, dass wir mit einer for-Schleife jedes für ein Graubild mögliche „t“ durchgehen. Falls ein „t“ einen besseres Ergebnis gibt als das bisher Beste, wird $\sigma_b(t)^2$ zum neuen Maximum von σ_b^2 und das aktuelle „t“ als neuer bester Threshold definiert.

```
function threshold = otsuthreshold(image)
  image = double(image(:));

  image_mean = mean(image);
  max = 0;
  opt_t = 0;

  for t = 1:255

      image_low = image(image<t);
      image_high = image(image>=t);
      image_low_mean = mean(image_low);
      image_high_mean = mean(image_high);
      m = size(image_low);
      n = size(image);
      w_1 = m/n;
      w_2 = 1 - w_1;

      sigma_b = w_1 * (image_mean - image_low_mean).^2 + w_2 *
          (image_mean - image_high_mean).^2;

      if sigma_b > max
          max = sigma_b;
          opt_t = t;
      endif
  endfor

  threshold = opt_t;

endfunction
```

2.1.2 Erkennen von verschiedenen Bildbereichen

Die oben genannten Methoden haben wir dann auf das Bild eines Barcodes (Abb.2) angewandt. Daraus erhielten wir ein Schwarz-Weiß-Bild, auf dem die Striche und Ziffern des Barcodes zu erkennen waren. Doch was für einen Barcode jetzt eigentlich interessant wäre, ist die Differenzierung zwischen Ziffern und Zahlen. Also haben wir uns überlegt, wie wir die Segmente des Bildes als einzelne Teile definieren können und diese im weiteren Verlauf dann entweder den Barcode-Strichen oder den Ziffern

zuordnen können. Der Befehl „bwlabeln“ von *Octave* schreibt zusammenhängenden Bereichen denselben Wert zu und unterteilt das Bild so in i Elemente. Da die Striche des Barcodes eindeutig länger sind als die Hälfte der Bildhöhe, haben wir mit Hilfe einer for-Schleife für jedes Element i überprüfen lassen, ob die Differenz des maximalen und des minimalen y-Wertes größer als die halbe Bildhöhe ist. Wenn dieser Fall eintrifft wurde das Segment den Strichen zugewiesen, falls nicht den Ziffern. Am Ende erhielten wir so zwei Bilder. Eines nur mit Strichen, das Andere ausschließlich mit Ziffern.

```
clear;clc;
pkg load image;

image = imread("D:\\Schule\\Modellierungswoche_Bildbearbeitung\\Images
                \\barcode.png");

th = otsuthreshold(image);
segment = (255*(image<th));
imshow(segment);

n = size(image)(1);
m = size(image)(2);

matrix = zeros(n*m,3);

image_x = ones(n,1) * (1:m);
image_y = (1:n)' * ones(1,m);

matrix(:,:) = [image(:),image_x(:),image_y(:)];

labels = bwlabeln(segment,4);

  image_line = ones(n,m)*255;
  image_line_aligned = ones(n,m)*255;
  image_numbers = ones(n,m)*255;

for i = 1:max(labels(:))

  temp_label = (labels==i);
  x_value = image_x(temp_label);
  y_value = image_y(temp_label);
```

```
31    diff =  max(y_value)-(min(y_value));
32
33    if diff > n/2
34
35      image_line_aligned(y_value,x_value) = 0;
36      image_line(y_value,x_value) = !segment(y_value,x_value);
37
38    else
39
40      image_numbers(y_value,x_value) = !segment(y_value,x_value);
41
42    endif
43
44  endfor
45  figure(1)
46  subplot(2,1,1);
47  imshow(image_line);
48  subplot(2,1,2);
49  imshow(image_line_aligned);
50
51  figure(2)
52  imshow(image_numbers);
```

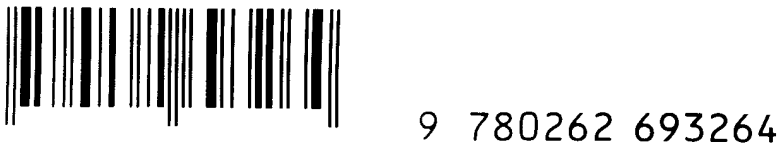


Abbildung 3: segmentierte Ziffern und Striche aus Barcode

2.2 Graustufen-Segmentierung

Nach der Aufteilung eines Bildes in zwei Bereiche (Schwarz und Weiß), war der nächste Schritt eine Aufteilung in mehrere verschiedene Graubereiche. Das Ziel war vorerst ein Bild in vier Bereiche zu unterteilen. Das bedeutet, dass drei Schwellwerte benötigt werden. Den Ansatz, den wir verfolgten, war, die bereits angewendete Methode (Otsu-Thresholding) für den mittleren Schwellwert zu nehmen, um das Bild erneut in zwei Bereiche zu teilen. Auf diese zwei Bereiche wendeten wir erneut Otsu-Thresholding an und erhalten so vier verschiedene Segmente. Für die optische Darstellung wird je der mittlere Grauwert der Segmente herangezogen und als Wert

eingesetzt. So haben wir ein Graustufenbild mit vier Farbbereichen erhalten.

Abbildung 4: Segmentierung in vier Graustufen

2.3 lokales Thresholding

Bei den oben genannten Techniken wurde globaes Thresholding eingesetzt. Das heißt, dass der Schwellwert durch Analyse des gesamten Bildes gewählt wird und für alle Pixel gleich ist. Um aber bei Bildern spezielle Feinheiten hervorherben zu können, macht es oft Sinn, lokales Thresholding anzuwenden. Hierbei wird für jeden Pixel der unmittelbare Bereich darum angeschaut und ein passender lokaler Threshold festgelegt. Liegt der Wert des Pixel über diesem lokalem Schwellwert, wird er weiß (Wert wird 255), liegt der darunter wird der Wert 0 gesetzt, und ist also schwarz. Mit dieser Methode können die Details eine Bildes sehr gut hervorgehoben werden. Im folgenden Beispiel (Abb. 5) wurde für den lokalen Threshold stets der Mittelwert einer 15x15 Umgebung gebildet.

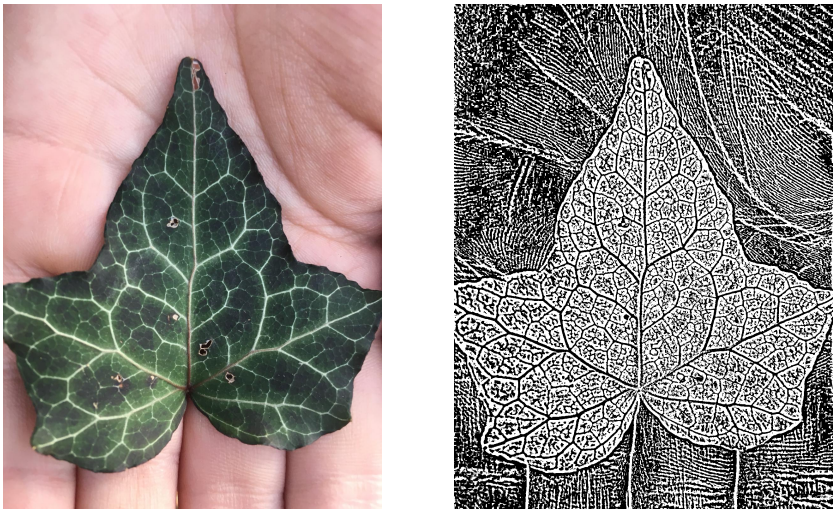


Abbildung 5: lokale Segmentierung

2.4 Konvertierung von Farbbildern zu Graustufenbildern

Nachdem wir uns mit der Segmentierung von Graustufenbildern beschäftigt haben, wollten wir dieselben Techniken auch auf Farbbilder anwenden. Diese müssen allerdings zuvor in Graustufenbilder konvertiert werden. Um das zu erreichen, gibt es verschiedene Ansätze: Der erste, der uns in den Sinn kam, war von den drei Farbwerten von jedem Pixel (rot, grün, blau) den Mittelwert zu bestimmen, der dann in unserem Graustufenbild einen Wert zwischen 0 und 255 annahm. Das erlaubte uns, die vorherigen Möglichkeiten zur Segmentierung anzuwenden.

```
1 function image_gray = mittelgray(image)
2
3 image = double(image);
4
5 image_gray = ((image(:,:,1)+image(:,:,2)+image(:,:,3))/3);
6 image_gray = uint8(round(image_gray));
7
8 end
```

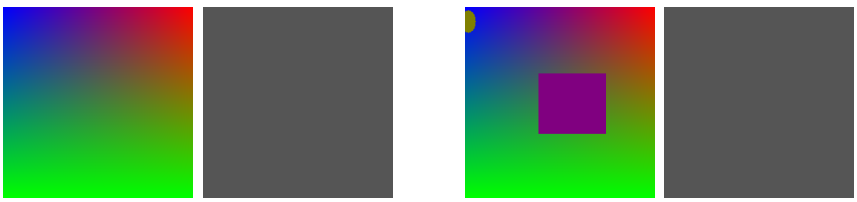


Abbildung 6: Konvertierung in Graustufenbilder mit Hilfe des Mittelwerts

Bei dieser Art der Konvertierung kann es aber zu Problemen kommen. Das Programm *Octave* kann ebenfalls Farbbilder in Graustufenbilder konvertieren (rgb2gray). Hierfür verwendet es jedoch nicht den Mittelwert, sondern gewichtet die drei Farbkanäle basierend auf empirische Datenlagen unterschiedlich. Das Graustufenbild entsteht durch eine Multiplikation der Rot-Werte mit 0.298936, der Grün-Werte mit 0.587043 und der Blau-werte mit 0.114021.

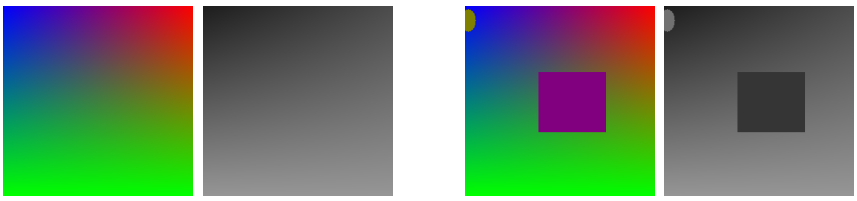


Abbildung 7: Konvertierung in Graustufenbilder von *Octave*

Um zu verstehen, warum sowohl der Mittelwert, als auch die Variante von *Octave* in bestimmten Fällen nicht sinnvoll funktioniert, beschäftigten wir uns mit Eigenvektoren sowie Eigenwerten und haben im Zuge dessen auch eine Einführung in die Matrizenrechnung bekommen. Mit diesen Informationen haben wir eine eigene Funktion für das Konvertieren von Farb- zu Graustufenbildern geschrieben. Diese berechnet auf Basis der Kovarianzmatrix die passende Gewichtung mit Hilfe des zum größten Eigenwert gehörigen Eigenvektors. Die Einträge des Eigenvektors repräsentieren dann die Gewichte für die passende Linearkombination.

```
1  function image_gray = myrgb2gray(image)
2
3  rvalue = image(:,:,1);
4  rvalue = rvalue(:);
5  gvalue = image(:,:,2);
6  gvalue = gvalue(:);
7  bvalue = image(:,:,3);
8  bvalue = bvalue(:);
9
10 size = length(image)*width(image);
11
12
13 m_values = zeros (size,3);
14
15 m_values(:,:) = ([rvalue, gvalue, bvalue]);
16
```

```
17
18
19 C = cov(m_values);
20
21 [eigenvec, eigenval] = eig(C);
22
23 [maxel, index] = max((diag(eigenval)));
24
25 myvec = eigenvec(:,index);
26
27
28 image_grey = m_values * myvec;
29
30
31   img_0 = (image_grey) - (min(image_grey(:)));
32
33   x = 255/(max(image_grey(:)) - min(image_grey(:)));
34
35   img_gray = uint8(img_0 * x);
36
37
38 image_gray = reshape(img_gray, length(image(:,:,1)), width(image(:,:,1)
                    ));
39
40
41 end
```

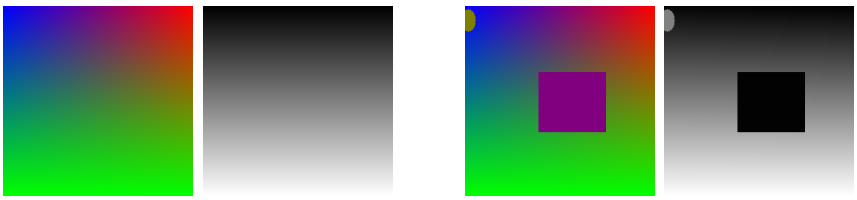


Abbildung 8: Konvertierung in Graustufenbilder mit Hilfe von Eigenvektoren

3 Segmentierung von Farbbildern

Um Farbbilder vor der Segmentierung nicht immer in Graustufenbilder konvertieren zu müssen, war das nächste Thema das wir erarbeitet haben das Segmentieren von Farbbildern. Im Unterschied zu Graustufenbildern, die pro Pixel nur einen Farb-

wert als Eintrag haben, besitzen Farbbilder pro Pixel drei Einträge. Das macht die Segmentierung von Farbbildern deutlich aufwendiger.

3.1 Der k-means Algorithmus

Um die Idee hinter der k-means segmentierung bei Farbbildern zu verstehen, kann man sich die einzelnen Pixel als Punkte im dreidimensionalen Raum vorstellen. Der Rot-Kanal ist dann auf der x-Achse, der Grün-Kanal auf der y-Achse und der Blau-Kanal auf der z-Achse. Um die Pixel in Segmente zu teilen sucht man sogenannte Cluster, die Pixeln mit ähnlicher Farbe zusammensetzen sollen. Das Ziel ist, dass bei k Clustern die Summe der Abstände der einzelnen Pixel zum Mittelpunkt des Clusters, dem sie zugeordnet wurden, möglichst gering ist. Es bezeichne $S_1, S_2, ..., S_k$ die jeweiligen Cluster, dann soll der Ausdruck

$$\sum_{i=1}^{k} \sum_{x \in S_i} ||x - \overline{x}_i||^2$$

minimiert werden. Würde man jetzt allerdings versuchen, diese Theorie über eine for-Schleife umzusetzen, würde das eine enorme Rechenleistung und Rechenzeit in Anspruch nehmen. Eine gute Approximation dessen schafft der Algorithmus k-means. Bei diesem werden die Pixelwerte am Anfang zufällig in k Clustern unterteilt. Danach wird für jeden Cluster k der Clustermittelpunkt ermittelt. Im Anschluss wird der Abstand jedes Pixels zu den Clustermittelpunkten berechnet und die Punkte werden den Clustern neu zugeordnet. Danach werden die Clustermittelpunkte erneut ausgerechnet. Diese Schritte werden so oft wiederholt, bis sich die Cluster nicht mehr verändern. Dieser Algorithmus ist bereits im Programm *Octave* vorhanden. Für die Segmentierung von Farbbildern haben wir ihn benutzt. Mithilfe des Befehls „kmeans“ haben wir die Pixel in verschiedene Bereiche unterteilt und anschließend jedem Pixel innerhalb eines Clusters die Farbwerte des Clustermittelpunkts zugewiesen.

Abbildung 9: k-means mit verschieden großen Clustern

3.2 Superpixel

Die Idee bei Superpixel ist eine sinnvolle Einteilung des Bildes in unterschiedlich große Bildbereiche, die sozusagen die darin liegenden Pixel in einem Superpixel vereinen. Der Algorithmus „SLIC“ nimmt als Grundlage für die Umsetzung von Superpixeln den Algorithmus k-means. Beim k-means-Algorithmus werden die Farbwerte der Pixel in einer Matrix mit drei Spalten, je eine pro Farbkanal, eingetragen und darauf basierend dann die Cluster gewählt. Bei „SLIC“ werden dieser Matrix dann zwei Spalten, eine für den x-Wert, eine für den y-Wert der Pixel hinzugefügt. Diese werden jeweils mit einem Parameter multipliziert, welcher deren Gewichtung beeinflusst. Die verschiedenen Cluster werden dann also nicht nur durch ähnliche Farben, sondern auch durch ihre Position im Bild unterteilt. Für ein besseres Ergebnis haben

wir die Bilder zuerst noch vom RGB-Farbraum in den LAB-Farbraum umgewandelt und am Ende nach Anwendung des Algorithmus wieder zurückkonvertiert. Um jeden Superpixel zu umranden, kann man den Parameter „outlines“ 1 setzten. Um die Linien hinzuzufügen, wurde in der Funktion „gradient“ notiert, wo sich im Bild Kanten befinden. Aus diesem Bild wurde dann ein schwarzes Bild gemacht, in dem die Kanten weiß erscheinen. Um sie noch deutlicher hervorzuheben, verwendeten wir die Funktion „imdilate“ , welche die Linien deutlicher macht. Dieses Bild ziehen wir dann vom Superpixelbild ab, um die schwarzen Umrandungen zu erhalten.

```
function image_finished = myslic(image, clusternumber, positionvalue,
                    outlines);

lab_image = rgb2lab(image);


l_value = lab_image(:,:,1)(:);
a_value = lab_image(:,:,2)(:);
b_value = lab_image(:,:,3)(:);


n = size(lab_image)(1);
m = size(lab_image)(2);


image_x = ones(n,1) * (1:m);
image_y = (1:n)' * ones(1,m);

x_value = image_x(:);
y_value = image_y(:);

x_value = (x_value*positionvalue);
y_value = (y_value*positionvalue);


m_values = zeros (n*m,5);

m_values(:,:) = ([l_value, a_value, b_value, x_value, y_value]);


[idx, centers] = kmeans(m_values, clusternumber);
```

```
31
32 segment = reshape (idx, n, m);
33
34 image_finished = (zeros(size(lab_image)));
35 image_finished(:,:,1) = (centers(:,1)(segment));
36 image_finished(:,:,2) = (centers(:,2)(segment));
37 image_finished(:,:,3) = (centers(:,3)(segment));
38
39 image_finished = lab2rgb(image_finished);
40
41 if outlines == 1
42       image_finished_gray = rgbtogray(image_finished);
43       [gx,gy] = gradient(double(image_finished_gray));
44       image_lines = sqrt(gx.^2 + gy.^2);
45       image_lines = uint8((image_lines > 0.01) * 255);
46       image_lines = imdilate(image_lines,strel("disk",2,0));
47       image_finished = image_finished - image_lines;
48   endif
49
50   endfunction
```

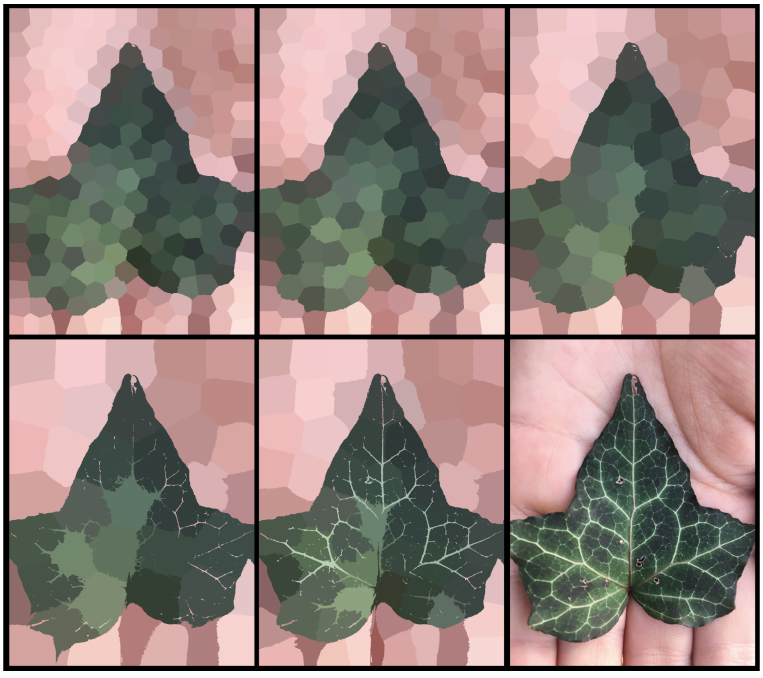


Abbildung 10: SLIC mit unterschiedlicher Anzahl an Superpixeln

3.3 Einfärben einzelner Bildbereiche

Mithilfe der Segmente, die wir mit dem Befehl k-means erstellt haben, konnten wir außerdem einzelne Teile eines Bildes einfärben. Dazu haben wir die einzelnen Segmente per Hand in Bereiche eingeteilt und anschließend zusammengefügt. Diesem neu entstandenen Bild haben wir dann mit dem Farbraum HSV eine neue Farbe zugeteilt und zuletzt wieder in das Originalbild eingefügt. Im Falle des Blattes bedeutet das, dass wir die Segmente in Hand und Blatt unterteilt haben und anschließend mit der Farbe des Blattes experimentiert haben.

Abbildung 11: eingefärbtes Blatt

4 Anwendung der Image Segmentation

Das Verfahren der Bildsegmentierung kommt in vielen technischen Bereichen zur Anwendung. In der Medizin wird es zur Erkennung von Gewebeanomalien, wie z.B. Tumoren eingesetzt. Auch für Röntgenbilder und das Computertomographie-Verfahren erweist sich die Segmentierung als wichtig. Doch nicht nur in der Medizin wird das Verfahren der Bildsegmentierung angewendet, auch in der Geographie werden beispielsweise Luftaufnahmen oder Satellitenaufnahmen sinnvoll segmentiert.

4.1 Bildbearbeitung

Will man verschiedene Bilder kombinieren, ist es am einfachsten, wenn sie dieselben Dimensionen haben. Das Objekt der Interesse wird segmentiert und daraus eine Maske erstellt, die die Werte 1 und 0 enthält. Als nächstes werden die Bilder mit der Maske multipliziert. So entsteht an den Stellen, wo eine 1 steht, wieder der Farbwert des Originalbildes. An den Stellen wo dagegen eine 0 steht, wird der Wert durch die Multiplikation mit 0 zu 0 und das Bild wird an diesen Stellen schwarz. Bei dem zweiten Bild wird die Maske umgedreht und ein gegenteiliges Bild entsteht. Dadurch werden die Positionen der Einsen und Nullen vertauscht und es entsteht ein Farbbild, das an den gegenteiligen Stellen des ersten Bildes schwarz ist.

```
clc; clear all;
pkg load image;

image1 = imread("Bilder\\mask.png");
image2_gr = imread("Bilder\\backgroun.png");
image3_gr = imread("Bilder\\foreground.png");

image2 = image2_gr(427:1791,1:1200,:);
image3 = image3_gr(1:1365,1:1200,:);

image0_grey = image1(:,:,1);

mask0_255 = uint8(255*(image1_grey>otsu(image1)));

mask0_1 = (mask0_255)./255;

image2_mask = image2;
image2_mask(:,:,1) = (image2(:,:,1).*(!mask0_1));
image2_mask(:,:,2) = (image2(:,:,2).*(!mask0_1));
image2_mask(:,:,3) = (image2(:,:,3).*(!mask0_1));

image3_mask = image3;
image3_mask(:,:,1) = (image3(:,:,1).*(mask0_1));
image3_mask(:,:,2) = (image3(:,:,2).*(mask0_1));
image3_mask(:,:,3) = (image3(:,:,3).*(mask0_1));

image_finished = (image2_mask) + (image3_mask)
```

Abbildung 12: "Jackson Pollock", Landschaftsbild, Maske von PoisonIvy, zusammengesetztes Bild

4.2 Anwendung im medizinischen Bereich

Mit den eigens erstellten Techniken haben wir uns dem Bild einer Retina gewidmet. Ziel war es die Blutgefäße der Retina gut sichtbar zu machen. Durch die inhomogene Belichtung des Bildes hat sich eine genaue Segmentierung als schwierig erwiesen. Wir haben zuerst das Farbbild in ein Graustufenbild konvertiert, da das Hervorheben der Blutgefäße über Helligkeitsunterschiede läuft. Anschließend wurde die lokale Thresholding-Technik aus Abschnitt 2.4 benutzt, um lokale Unterschiede im Bild hervorzuheben. Dadurch entstanden bei unserem Bild der Retina allerdings vor allem im weniger gut belichtetem Bereich Störelemente. Um diese zu entfernen, ist der Befehl „bwareaopen“ sehr hilfreich, da dieser im Bild zusammenhängende Bereiche erkennt und Flächen, die kleiner als eine bestimmte Größe sind, den Wert 0 zuweist. Doch auch mithilfe dieses Befehls konnte nicht alle Störelemente entfernt werden, da diese sonst kleine Blutgefäße entfernt hätten und größere Störelemente wären

erhalten geblieben. Als Versuch diesen Effekt auszugleichen, wurden die Befehle „imerode“ sowie „imdilate“ angewandt. Danach wurde noch einmal mit „bwareaopen„ versucht möglichst viele Störelemente zu entfernen. Das Ergebnis enthält immer noch Störelemente im Bild und es konnten nicht alle Blutgefäße segmentiert werden. Es hängen nicht alle Blutgefäßelemente zusammen, jedoch sind sie insgesamnt gut erkennbar

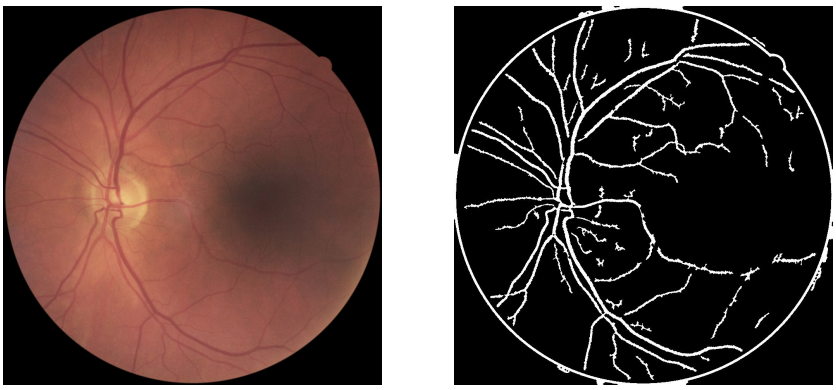
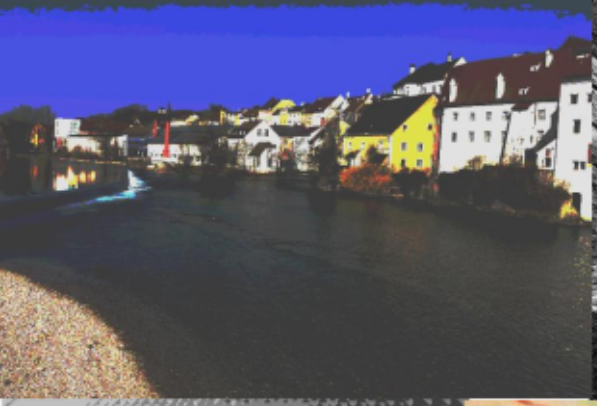
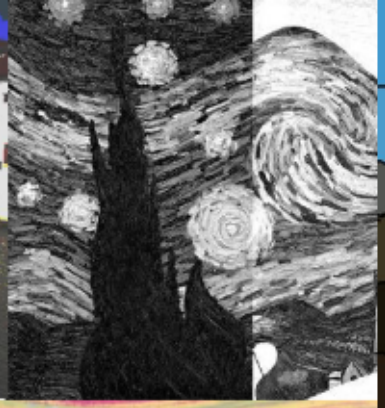
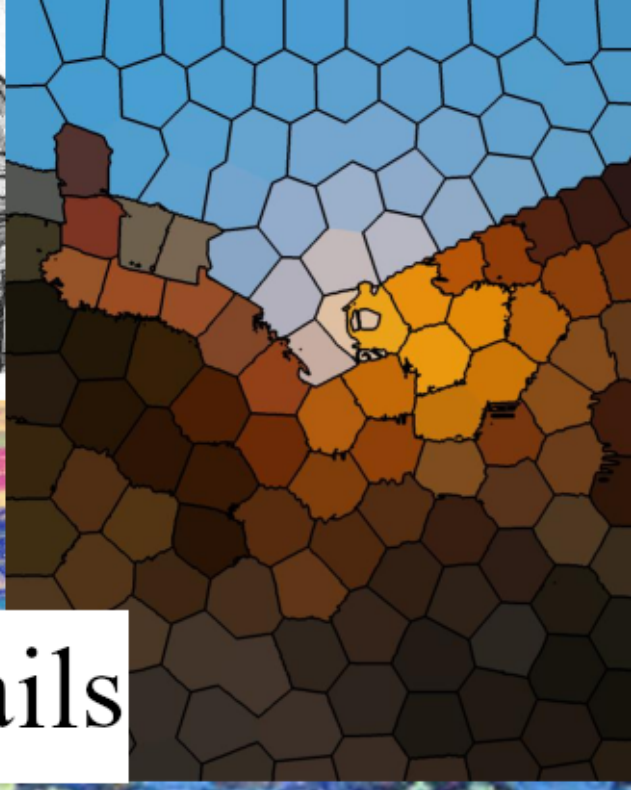
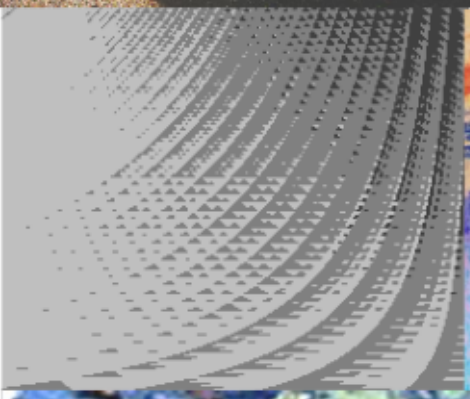
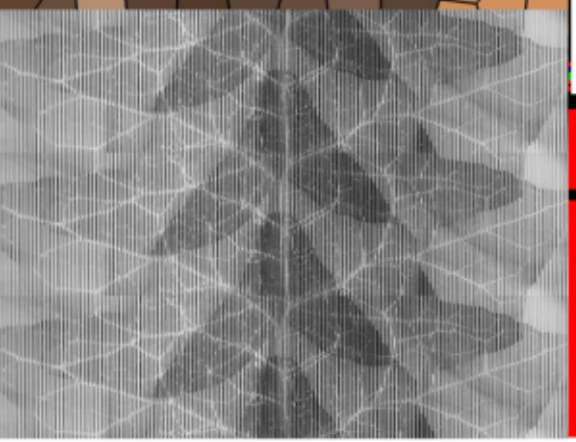
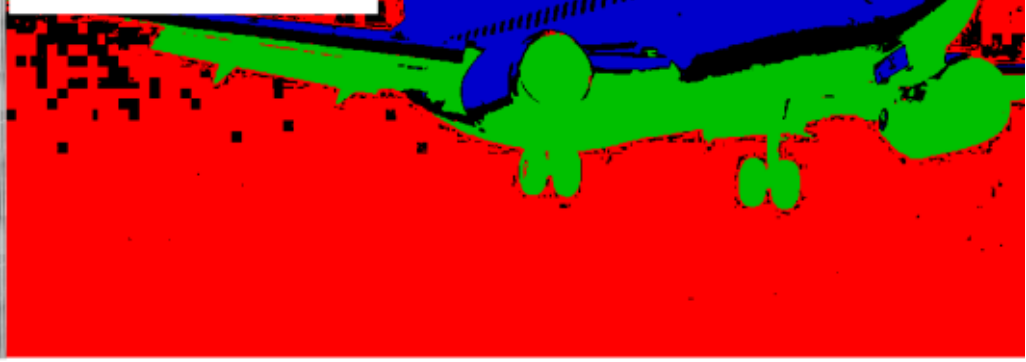


Abbildung 13: Bild einer Retina mit segmentierten Blutgefäßen


Funny Fails



Mathematische Spieltheorie

Teilnehmer: *Elodie Korsatko, Tabea Solhdju (Gibs), Laura König, Sianna Zinterl, Eileen Schmieger und Hannah Grabner (B(R)G Leibnitz)* unter der Leitung von *Florian Thaler*

Inhaltsverzeichnis

Gender Erklärung:

Um die Lesbarkeit zu erleichtern, wird in dieser Arbeit ausschließlich die weibliche Sprachform verwendet. Die Verwendung der weiblichen Form soll als geschlechtsunabhängig verstanden werden.

1.Einleitung

Egal ob bei “Schere, Stein, Papier”, Geschwistern, Schach, Spielen, Elfmeterschießen im Fußball, “Gefangenendilemmas”, Kriegen, Wirtschaft, Unternehmen, Gehaltsverhandlungen oder Nachbarschaftsproblemen - überall kann man die Spieltheorie anwenden. Doch was ist diese eigentlich? Damit haben wir uns diese Woche beschäftigt: Der Erfolg einzelner Personen oder Gruppen hängt von den Aktionen anderer ab. Die Handlungen von mindestens zwei Personen und deren Auswirkungen auf alle Beteiligten können modelliert werden und wir können herausfinden, welche Handlungen für die sogenannten “Player” vorteilhaft und ungünstig sind. Dargestellt werden kann ein Spiel in Form von einer Tabelle, in der alle möglichen Spielzüge und Ausgänge für jede Spielerin ersichtlich sind. Somit werden komplexe Entscheidungssituationen, die abhängig vom Verhalten anderer Spielerinnen sind, vereinfacht.

Es kann unterschieden werden in Konkurrenz (Handlungen aus eigenem Interesse, aktions- und strategieorientiert) und Kooperation, kooperieren stellt sich meistens als am erfolgsbringendsten heraus, da es auszahlungsorientiert ist und miteinander immer besser als gegeneinander arbeiten ist.

1.1.Geschichte

Im 20. Jahrhundert hat die Spieltheorie in unserer Gesellschaft große Bedeutung erlangt - 1928 legte der jüdische Ungar John von Neumann die Grundlage für die Spieltheorie. Er war bereits als Kind überdurchschnittlich intelligent, beeindruckte sogar Nobelpreisträger und beschäftigte sich mit Quantenmechanik und der Spieltheorie. 1944 veröffentlichte er mit dem Österreicher Oskar Morgenstern das Buch „Spieltheorie und wirtschaftliches Verhalten“, in dem sie mathematische Theorie und wissenschaftliche Anwendung verknüpften; als Lösung wurde das Nullsummenspiel (Minimax-Theorem) angewendet. Das “Nashgleichgewicht” nach dem US-Amerikaner John Nash galt ab 1950 als erste allgemeine Lösungsmöglichkeit. Dieses beschreibt die Strategiekombination, die in bestimmten Situationen für alle Player am besten ist: Keiner kann die eigene Strategie ändern, ohne einen Nachteil einzukassieren. John Nash interessierte sich sehr für Mathematik und Physik. Er arbeitete mit anderen Forschern an Anwendungen der Spieltheorie auf strategische Situationen im Kalten Krieg, jedoch litt er unter Schizophrenie, weshalb er von der Gesellschaft abgeschnitten wurde.

Wichtige Ausgangspunkte sind, dass wir davon ausgehen, dass die Entscheidungen unabhängig voneinander zur gleichen Zeit passieren, jede Spielerin aus reinem Egoismus handelt und die Spielerinnen nicht miteinander reden.

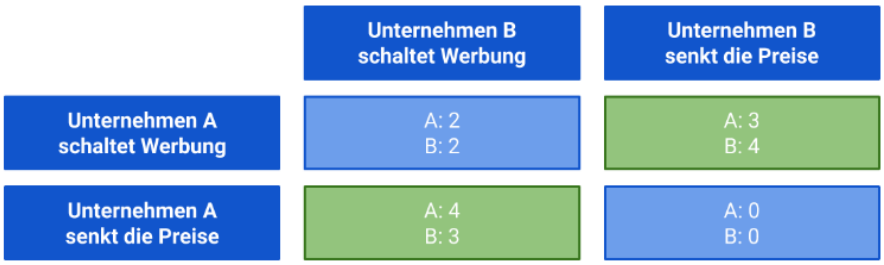
Beispiel: Schere-Stein-Papier

Spielerin 1	Spielerin 2			
		Schere	Stein	Papier
	Schere	(0, 0)	(-1, 1)	(1, -1)
	Stein	(1, -1)	(0, 0)	(-1, 1)
	Papier	(-1, 1)	(1, -1)	(0, 0)

Strategien können dominant, die rein oder auch gemischt sein.

Besonders interessant sind dominante Strategien. Bei der ersten kann man immer besser als der andere Player aussteigen (zum Beispiel das Gefangenendilemma); wichtig sind auch Nash-Gleichgewichte. Diese helfen eine optimale Entscheidung bei Kenntnis der Strategie der anderen Spielerin zu treffen.

Beispiel: Werbung vs. Preise senken



Quelle: https://www.bwl-lexikon.de/wiki/nash-gleichgewicht/, 16.02.2023

Nash Gleichgewichte existieren unter gewissen Voraussetzungen – diese können dann aber gesmischte Strategien sein. Siehe Bericht weiter unten.

Eines der bekanntesten Beispiele für die Spieltheorie ist der Kalte Krieg. Massive Aufrüstung der USA und der UdSSR, um die Vormachtstellung für sich zu beanspruchen. Dies gipfelte sich jedoch immer weiter auf, denn keiner der beiden Staaten konnte aufhören, ohne schwächer als der andere zu werden und dadurch den Krieg zu verlieren. Anhand der Atomkrise erkannte man, dass Kommunikation sinnvoll ist, da das gegenseitige Angreifen zu großen Verlusten geführt hätte.

Beispiel: Kalter Krieg.

		Sowjetunion	
		Angreifen	Nicht angreifen
USA	Angreifen	(-3, -3)	(5, -5)
	Nicht angreifen	(-5, 5)	(0, 0)

Quelle: https://www.oemg.ac.at/Mathe-Brief/fba2014/Spieltheorie_Einsiedler.pdf, 16.02.2023

2.Mathematische Definition eines Spiels

Ein mathematisches Spiel besteht aus der Menge P an Spielerinnen:

$P = \{1, 2, 3, \ldots n\}$

einer Menge an Strategien S:

Sei S_1 die Aktionsmenge von Spieler 1

Sei S_2 die Aktionsmenge von Spieler 2,

…

und einer Menge an Payoff-Funktionen π:

$\pi_1: S \rightarrow \mathbb{R}$

$\pi_2: S \rightarrow \mathbb{R}$

…

Dabei bezeichnet $S = S_1 \times S_2 \times \ldots \times S_n$ die Menge aller Aktionskombinationen von $S_1, \ldots S_n$. Häufig wird ein Spiel durch das Tripel (P, S, π_1, π_n) gekennzeichnet.

Einige Erklärungen und Bemerkungen:

Strategie: Eine Strategie bezeichnet eine Aktion oder Handlung welche von einer Spielerin ausgeführt wird.

Payoff-Funktion: Die Payoff-Funktion zeigt den Nutzen von Spielstrategien aus Sicht einer bestimmten Spielerin an.

Strategisches Spiel: Eine langfristige Planung des Vorgehens ist im Spiel entscheidend. Die meiste Zeit enthalten Strategiespiele keine, beziehungsweise sehr geringe Glücks- und Zufallselemente.

Rationale Spielerin: Eine rationale Spielerin maximiert ihren individuellen Nutzen und nimmt dabei keine Rücksicht auf die Auswirkungen auf andere Spielerinnen. Sie erreicht somit ihren eigenen maximalen Payoff. Wir betrachten nur Spiele mit rationalen Spielerinnen, welche ein sehr egoistisches Verhalten aufweisen. Zwischen den Spielerinnen selbst folgt dabei keine Kooperation.

2.1.Prisoner's dilemma

Das typische Beispiel für die mathematische Spieltheorie ist das „Prisoner's dilemma". In diesem Beispiel gibt es zwei Verdächtige, die beschuldigt werden, ein Verbrechen gemeinsam begangen zu haben. Die beiden dürfen nicht miteinander kommunizieren und werden daher in zwei verschiedene Räume gebracht. Sie bekommen beide den gleichen Vorschlag:

- Wenn beide reden und Aussagen, müssen sie nur für ein Jahr in das Gefängnis.
- Wenn nur sie redet und die andere schweigt, kommt diejenige die redet frei und diejenige die schweigt kommt für 15 Jahre ins Gefängnis.
- Wenn beide reden, kommen sie für zehn Jahre in das Gefängnis.

	Verdächtiger 2		
Verdächtiger 1		schweigen	Reden

	Schweigen	(-1,-1)	(-15,0)
	Reden	(0,-15)	(-10,-10)

In der ersten Zeile und in der ersten Spalte werden immer die jeweiligen Spielerinnen angegeben. Hier sind die Spieler die „Verdächtige 1“ und die „Verdächtige 2“. Die darauffolgende Zeile und Spalte geben die verschiedenen Aktionen an. In diesem Beispiel sind die Aktionen „schweigen“ und „reden“ und daher ist $S_1 = S_2 = \{$„reden“, „schweigen“$\}$ In den Feldern, wo die Aktionen zusammenlaufen, wird der Payoff eingetragen. Die erste Zahl (in grün) in der Klammer, steht für den Payoff der Verdächtigen 1 und die zweite Zahl (in rot) steht für die Verdächtige 2. Wenn man das Spiel von der Sicht der Verdächtigen 1 betrachtet, ist es für sie immer im Vorteil zu reden. Weil:

- Wenn die zweite Verdächtige schweigt, sollte sie reden, da sie in diesem Fall freikommt.
- Wenn die zweite Verdächtige aber redet, sollte sie auch reden, da sie sonst für 25 Jahre ins Gefängnis kommt und so nur für zehn Jahre eingesperrt wird.

Daher ist es immer besser zu reden, da der eigene Payoff des jeweiligen immer kleiner ist, als wenn sie schweigen würden. Aus diesem Grund müssen die Verdächtigen für zehn Jahre ins Gefängnis.

Würden beide Verdächtigen in diesem Beispiel miteinander reden können, würden sie sich wahrscheinlich für schweigen entscheiden, da es für beide am besten ist.

3.Das Nash-Gleichgewicht

Das Nash-Gleichgewicht ist ein wesentlicher Teil der Spieltheorie, um die optimale Strategie der beiden Player herauszufinden. Dieses wurde von John F. Nash 1950 entwickelt und erweiterte mit seiner Arbeit mit dem Titel „Non-cooperative Games“ die Spieltheorie von Morgenstern und von Neumann. Er bewies ebenfalls, dass das Nash-Gleichgewicht auch für Nicht-Nullsummenspiele und für mehrere Spieler gilt. Hierbei wird dieses durch ein Strategiepaar definiert, welches keinen der beiden Player durch einseitiges Abweichen

seiner Strategie individuell bevorzugt. Somit entsteht eine Situation, bei der es sich für keine der beiden Parteien lohnt, seine Strategie zu ändern

Ein Strategiepaar (s_1^* , s_2^*) bildet ein Nash-Gleichgewicht, falls

$\pi_1(s_1^*, s_2^*) \geq \pi_1(s_1, s_2^*)$ für alle s_1 aus der Aktionsmenge von Spielerin 1

$\pi_2(s_1^*, s_2^*) \geq \pi_2(s_1^*, s_2)$ für alle s_2 aus der Aktionsmenge von Spielerin 2

3.1Das Gefangenen-Dilemma

Player 1 ↓, Player 2→	Schweigen: „s“	Reden: „r“
schweigen	(-1, -1)	(-15, 0)
reden	(0, -15)	(-10, -10)

Überlegung: Bei welcher Strategie besteht ein Nash-Gleichgewicht?

Man betrachte die verschiedenen Payoffs für Player 1 und Player 2. Wenn beide schweigen, lohnt es sich für die Player zu reden, da sie, sofern ihr Gegenspieler schweigt, 0 Jahre ins Gefängnis müssen. Somit profitiert ein Player davon abzuweichen. Bei der Situation, dass Player 1 redet und Player 2 schweigt, wird Player 2 auf die Strategie reden wechseln, um nicht benachteiligt zu werden. In Folge ist die Strategie, bei der beide Player reden, das Nash-Gleichgewicht.

Überprüfung:

$$\pi_1(s_1^*, s_2^*) \geq \pi_1(s_1, r_2^*)$$

Wie oben beschrieben ist die Situation, wenn beide Parteien reden, das Nash-Gleichgewicht. Wenn aber Player 1 schweigt und Player 2 der optimalen Strategie folgt und redet, ist das Payoff für Player 1 in diesem Fall länger im Gefängnis zu sein.

$$\pi_1(s_1^*, s_2^*) = -10$$

$$\pi_1(s_1, s_2^*) = -15$$

Fügt man die Werte nun in die Formel ein, sieht man, dass die Strategie, bei der beide Player reden, ein Nash-Gleichgewicht ist.

$$-10 \geq -15$$

3.2 Der Kalte Krieg

Das Nash-Gleichgewicht lässt sich durch das Beispiel des Kalten Krieges gut veranschaulichen. Die USA und die UdSSR lieferten sich ein Duell im Wettrüsten. Um die Vormachtstellung für sich zu beanspruchen, hörte keine der beiden Parteien mit der massiven Aufrüstung auf, da sie sonst benachteiligt gewesen wären. Somit bildet die Situation, bei der beide Parteien weiterhin aufrüsten, das Nash-Gleichgewicht.

3.3 Matching Pennies

	Kopf	Zahl
Kopf	(2, 0)	(0, 2)
Zahl	(0, 2)	(2, 0)

Bei diesem Beispiel müssen die Player jeweils eine Münze entweder auf Kopf oder Zahl legen. Dabei sehen die Player nicht, was der jeweils andere gelegt hat. Wenn nun beide Münzen entweder auf Kopf oder beide auf Zahl liegen, gewinnt Player 1 und erhält beide Münzen. Wenn es jedoch unterschiedlich ist, gewinnt Player 2 und erhält dadurch beide Münzen.

<u>Überlegung:</u>

In diesem Spiel verliert immer eine der beiden Parteien und kann durch Umdrehen ihrer Münze siegen. Das heißt, dass durch Ändern seiner Strategie immer ein Player einseitig davon profitieren kann. Somit besteht bei diesem Beispiel kein Nash-Gleichgewicht.

<u>Überprüfung:</u>

$$\pi_1(s_1^*, s_2^*) \geq \pi_1(s_1, s_2^*)$$

Nun kann man die einzelnen Komponenten und Werte in die Formel einsetzen. Jedoch haben wir ab diesem Zeitpunkt einen Code auf Python geschrieben, um die Berechnung dem Computer zu überlassen.

3.4 Programmierung mit Python

- Durch die angegebenen Payoff-Matrizen überprüft diese Funktion, ob das Strategie-paar (i, j) ein Nash-Gleichgewicht ist:

Möglichkeit 1:

```
Import numpy as np


def checkIfNash_version_1(A1, A2, i, j):
    """

    Args:
        A1 (2d array): payoff matrix for player 1
        A2 (2d array): payoff matrix for player 2
        i (integer): index of strategy
        j (integer): index of strategy

    Antwort:
    # Boolescher Wert, der angibt, ob (i, j) ein Nash-
    Gleichgewicht darstellen
    """


    # Bestimmung der Anzahl der Aktionen bzw. Strategien von
     Spieler 1 und Spieler 2
    # can apply
    numActions_pl_1, numActions_pl_2 = A1.shape


    # Hilfslisten einführen:
    nash_test_list1 = [False for _ in range(0,numActions_pl_1)]
    nash_test_list2 = [False for _ in range(0,numActions_pl_2)]

    #  Benützung von zwei Loops, da Player 1 und Player 2 eine
    unterschiedliche Anzahl von Aktionen haben können
    for k in range(0, numActions_pl_1):
        if A1[i, j] >= A1[k, j]:
            nash_test_list1[k] = True
    for k in range (0,numActions_pl_2):
        if A2[i, j] >= A2[i, k]:
            nash_test_list2[k] = True
```

```
    # Die Strategie (i, j) ist nur ein Nash-Gleichgewicht, wenn
    alle obigen Abfragen „True“ ergeben

    test1 = np.all(nash_test_list1)
    test2 = np.all(nash_test_list2)
    retVal = None
    if np.all([test1, test2]):
        retVal = True
    else:
        retVal = False


    return retVal
```

Möglichkeit 2:

```
def checkIfNash_version_2(A1, A2, i, j):
    """
    Args:
        A1 (2d array): payoff matrix for player 1
        A2 (2d array): payoff matrix for player 2
        i (integer): index of strategy
        j (integer): index of strategy

    Antwort:
    # Boolescher Wert, der angibt, ob (i, j) ein Nash-
    Gleichgewicht darstellen
    """
    # Bestimmung der Anzahl der Aktionen bzw. Strategien von
    Spieler 1 und Spieler 2
    # can apply
    numActions_pl_1, numActions_pl_2 = A1.shape


    # auxiliary variables
    y1 = 1
    y2 = 1


    # Benützung der Payoff-Matrizen um zu bestimmen, ob (i, j) ein
    Nash-Gleichgewicht ist
    for k in range(0, numActions_pl_1):
        if A1[i, j] >= A1[k, j]:
            y1 = y1 * 1
        else:
            y1 = y1 * 0
    for k in range (0,numActions_pl_2):
```

```
        if A2[i, j] >= A2[i, k]:
            y2 = y2 * 1
        else:
            y2 = y2 * 0

    retVal = None
    if y1 * y2 == 1:
        retVal = True
    else:
        retVal = False


    return retVal
```

Angabe …:

```
from gameAnalysisFunctions import checkIfNash_version_1
from gameAnalysisFunctions import checkIfNash_version_2

# Angabe der Payoff-Matrizen


n1 = 2      # Anzahl der Aktionen von Player 1
n2 = 2      # Anzahl der Aktionen von Player 2


A1 = np.zeros((n1, n2))
A2 = np.zeros((n1, n2))

# Payoff-Matrix für das Spiel “Matching Pennies”
# A1[0, 0] = 2
# A1[0, 1] = 0
# A1[1, 0] = 0
# A1[1, 1] = 2

# A2[0, 0] = 0
# A2[0, 1] = 2
# A2[1, 0] = 2
# A2[1, 1] = 0
# Payoff-Matrix für das Gefangenen-Dilemma:
A1[0, 0] = -1
A1[0, 1] = -15
A1[1, 0] = 0
A1[1, 1] = -10
```

```
A2[0, 0] = -1
A2[0, 1] = 0
A2[1, 0] = -15
A2[1, 1] = -10


# ### part I


print("# ### pure nash equilibria ###")
print("################################")
print("")

# Alle Nash-Gleichgewichte finden: alle möglichen Aktionspaare
finden und prüfen:

for i in range(0, n1):
    for j in range(0, n2):
        # result = checkIfNash_version_1(A1, A2, i, j)
        result = checkIfNash_version_2(A1, A2, i, j)
# Je nachdem was gewünscht ist, das passende Resultat auswählen

        print("The strategy pair ({:d}, {:d}) forms a Nash
equilibrium: {:s}".format(i, j, str(result)))
```

4.Dominante Strategie

Eine Strategie ist dann für eine ausgewählte Spielerin dominant, wenn sie **unabhängig von der Strategie** der anderen Spielerinnen, zum **größten Payoff** führt. Sie trägt also immer den größten Nutzen für die Spielerin, die sie ausführt.

Am Beispiel des Gefangenen-Dilemmas, ist die dominante Strategie gut zu erkennen.

Für Spielerin 1 ist die Strategie „reden“ dominant da π_1(r,r) ≥ π_1(s,r) und π_1(r,s) ≥ π_1(s,s).

Dies bedeutet „reden“ ist unabhängig davon ob sich Spielerin 2 für reden oder schweigen entscheidet, die Strategie mit dem größeren Payoff.

Es kann wie folgt veranschaulicht werden:

- Spielerin 2 schweigt: Payoff Spielerin 1 schweigt: -1

  Payoff Spieler 1 **redet: 0**

- Spielerin 2 redet: Payoff Spieler 1 schweigt: -15

  Payoff Spieler 1 **redet: -10**

➔ Spielerin 1 erhält immer den **größeren Payoff** wenn sie redet.

Anhand dieses Beispiels konnten wir den mathematischen Ausdruck einer dominanten Strategie für zwei Spielerinnen mit jeweils 2 Strategien (a,b) definieren:

π_1(a,a) ≥ π_1(b,a) und π_1(a,b) ≥ π_1(b,b).

Falls die Bedingungen erfüllt werden können, ist a die dominante Strategie für Spielerin 1.

Generell kann außerdem die Aussage getätigt werden, dass jede Strategie, die das Nash-Equilibrium aufweist, auch die dominante Strategie ist. Umkehrend gilt diese Aussage jedoch nicht. Nicht jede dominante Strategie, ist ein Nash-Equilibrium.

Dieser Code bestimmt dominante Strategien eines gegebenen Spiels.

```
def findDominantStrategies(A1, A2):
    """
    this function determines all the dominant strategies for player 1 and player 2
    Args:
        A1 (2d array): payoff matrix of player 1
        A2 (2d array): payoff matrix of player 2
    Returns:
        lists containing the dominant strategies for player 1 and player 2 respectively
    """
    numActions_pl_1, numActions_pl_2 = A1.shape
```

```
dominant_strategies_list_pl_1 = []
dominant_strategies_list_pl_2 = []
# search for dominant strategies for player 1
for i in range(0, numActions_pl_1):
    # check if strategy i is dominant
    y = 1
    for j in range(0, numActions_pl_2):
        for k in range(0, numActions_pl_1):
            if A1[i, j] >= A1[k, j]:
                y = y * 1
            else:
                y = y * 0
    if y == 1:
        dominant_strategies_list_pl_1.append(i)

# search for dominant strategies for player 2
for j in range(0, numActions_pl_2):
    # check if strategy j is dominant
    y = 1
    for i in range(0, numActions_pl_1):
        for k in range(0, numActions_pl_2):
            if A2[i, j] >= A2[j, k]:
                y = y * 1
```

```
                else:
                    y = y * 0
        if y == 1:
            dominant_strategies_list_pl_2.append(j)

    return dominant_strategies_list_pl_1,
dominant_strategies_list_pl_2
```

5.Gemischte Strategien

Ein Spiel muss nicht nur aus reinen Strategien bestehen, die Strategien können auch mit Wahrscheinlichkeiten versehen werden, mit welchen diese gespielt werden. Wenn man das Beispiel „Schere-Stein-Papier“ hernimmt, ist es nicht sehr optimal immer die gleiche Aktion zu verwenden, sondern man sollte strategisch vorgehen und unberechenbar mischen. Im Beispiel Schere-Stein-Papier bezeichnet p_1 die Wahrscheinlichkeit Schere zu spielen, p_2 Stein und p_3 Papier. Die gemischte Strategie wird nun durch folgenden Vektor dargestellt:

$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$ die Einträge müssen ≥ 0 sein und in der Summe 1 ergeben.

5.1Nash Gleichgewicht bei gemischten Strategien

Bei gemischten Strategien treffen Spielerinnen nicht direkt eine Entscheidung, sondern wählen nur mit einer bestimmten Wahrscheinlichkeit eine Strategie. Mit dieser Information zeigt das Nash-Theorem, dass unter bestimmten Voraussetzungen stets ein Nash-Gleichgewicht besteht.

Um das Nash-Gleichgewicht in gemischten Strategien zu finden verwenden wir folgenden Code:

```
print("# ### mixed nash equilibria ###")

print("################################")

print("")
```

```
# use the nashpy package to find also mixed nash equilibria

game = nashpy.Game(A1, A2)

nash_equilibria = game.support_enumeration()


print("All Nash equilibria (including mixed ones): ")

print(list(nash_equilibria))
```

6.Beispiel: Schere-Stein-Papier

Anhand dieses Beispiels werden jetzt nochmal die verschiedenen Strategien demonstriert, welche oben beschrieben wurden. Bei dem Spiel, spielen zwei Spielerinnen gegeneinander und müssen ohne zu wissen was der andere nimmt sich entweder für Stein, Papier oder Schere entscheiden.

Spielerin 1	Spielerin 2			
		Schere	Stein	Papier
	Schere	(0, 0)	(-1, 1)	(1, -1)
	Stein	(1, -1)	(0, 0)	(-1, 1)
	Papier	(-1, 1)	(1, -1)	(0, 0)

Erklärung der Tabelle:

Wenn beide Spielerinnen Stein spielen, ist der Pay-off (0,0) und somit gibt es in der Runde keinen Gewinnerin. Spielt Spielerin1 dennoch Papier und Spielerin2 spielt Stein, dann gewinnt Spielerin1 und Spielerin2 verliert. Der Pay-off schaut dann wie folgt aus, (1,-1). Die dritte Möglichkeit ist, Spielerin1 wählt Schere und Spielerin2 wieder Stein. Der Pay-off ist dann (-1,1). Daraus kann man schließen das Spielerin1 verliert und Spielerin2 gewinnt. Das sind alle möglichen Pay-offs die ein Spieler erhalten kann. Spielerin2 kann offensichtlich auch zwischen den 3 Optionen (Schere, Stein, Papier) wählen kann und ist daher nicht gezwungen Stein zu spielen. Insgesamt entstehen so 4 Möglichkeiten wie das Spiel ausgehen kann. Diese sind oben in der Tabelle dargestellt.

Wenn man dieses Beispiel auf eine Nash-Gleichgewicht oder dominante Strategie prüft, zeigt sich, dass es keins von beidem gibt. Es stellt sich die Frage, ob sich dies verändert, wenn man eine weiter

Spielaktion, wie zum Beispiel den Brunnen, hinzufügt. Brunnen gewinnt gegen Stein und Schere und stört so das Gleichgewicht des Spiels. Trotzdem gibt es weder ein Nash-Gleichgewicht noch eine dominante Strategie. Mit unseren Codes konnten wir das überprüfen.

7. Matrix Spiele

Für ein zwei Spielerinnen Spiel gibt es wie gehabt zwei Aktionsmengen. Wir nehmen nun an, dass S_1, S_2 aus jeweils zwei Elementen besteht. Dann zeichnen sich Matrix-Spiele dadurch aus, dass die Payoff-Funktion folgendermaßen geschrieben werden kann:

Spielerin 1: $\pi_1(\sigma_1, \sigma_2) = \sigma_1^T$ A_1 σ_2; $\quad \sigma_1 = \begin{bmatrix} p_1 \\ 1-p_1 \end{bmatrix}$, $\quad \sigma_2 = \begin{bmatrix} p_2 \\ 1-p_2 \end{bmatrix}$

Spielerin 2: $\pi_2(\sigma_1, \sigma_2) = -\sigma_1^T$ A_2 σ_2; $\quad \sigma_1 = \begin{bmatrix} p_1 \\ 1-p_1 \end{bmatrix}$, $\quad \sigma_2 = \begin{bmatrix} p_2 \\ 1-p_2 \end{bmatrix}$

Dabei bezeichnen A_1, A_2 die Payoff-Matrizen.

<u>Beispiel:</u>

$\sigma_1 = \begin{bmatrix} p_1 \\ 1-p_1 \end{bmatrix}$, $\sigma_2 = \begin{bmatrix} p_2 \\ 1-p_2 \end{bmatrix}$, $A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}$

$$\sigma_1^T A \sigma_2 = \begin{bmatrix} p_1 & 1-p_1 \end{bmatrix} * \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} * \begin{bmatrix} p_2 \\ 1-p_2 \end{bmatrix} = \begin{bmatrix} p_1 & 1-p_1 \end{bmatrix} \begin{bmatrix} a_{1,1} * p_2 + & a_{1,2} * (1-p_2) \\ a_{2,1} * p_2 + & a_{2,2} * (1-p_2) \end{bmatrix}$$

$$= p_1 * \Big(a_{1,1} * p_2 + a_{1,2} * (1-p_2) \Big) + (1-p_1) * (a_{2,1} * p_2 + a_{2,2} * (1-p_2))$$

Das gleiche Prinzip kann man nun auch bei π_2 anwenden.

8. Nullsummenspiel

Das Nullsummenspiel hat ihren Ausgang tatsächlich im Spiel. Wenn die Summe und die Verluste aller Spielerinnen gleich Null sind, handelt es sich um ein Nullsummenspiel. Solche Spielsituationen sind in vielen Bereichen aufzufinden, wie beispielsweise am Markt: Wenn zwei etwa gleich große Firmen (hier Spar und Billa) gleich viel Werbung machen und somit gleich viele Kundinnen anlocken, handelt es sich für die beiden um ein Nullsummenspiel.

Mathematisch gesehen heißt ein Spiel ein Nullsummenspiel, wenn die Spiele die Eigenschaft: $\pi_1(\sigma_1, \sigma_2) + \pi_2(\sigma_1, \sigma_2) = 0$ für alle möglichen Strategien haben.

Im Falle eines Matrix Spieles bedeutet $A_1 = -A_2$ weil σ_1^T A σ_2 + σ_1^T A σ_2 = 0

8.1Nullsummenspiel: Beweisführung#

Das folgende Beispiel „Rock, Paper, Scissors“ stellt ein sogenanntes Nullsummenspiel dar.

π_1 drückt den Payoff für Spieler 1 aus, π_2 drückt den Payoff für Spieler 2 aus.

Zuerst wird π_1 aufgestellt: $A_1 = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$, $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, $y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$

$A_1 * x = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 * 0 & x_2 * (-1) & x_3 * 1 \\ x_1 * 1 & x_2 * 0 & x_3 * (-1) \\ x_1 * (-1) & x_2 * 1 & x_3 * 0 \end{bmatrix}$

$y^T A_1 * x = [y_1 \quad y_2 \quad y_3] \begin{bmatrix} x_1 * 0 & x_2 * (-1) & x_3 * 1 \\ x_1 * 1 & x_2 * 0 & x_3 * (-1) \\ x_1 * (-1) & x_2 * 1 & x_3 * 0 \end{bmatrix}$

$\pi_1(x, y) = y_1 * (x_1 * 0 + x_2 * (-1) + x_3 * 1) + y_2 * (x_1 * 1 + x_2 * 0 + x_3 * (-1)) + y_3 * (x_1 * (-1) + x_2 * 1 + x_3 * 0)$

Dann wird π_2 aufgestellt: $A_2 = \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}$, $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, $y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$

$A_2 * x = \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 * 0 & x_2 * 1 & x_3 * (-1) \\ x_1 * (-1) & x_2 * 0 & x_3 * 1 \\ x_1 * 1 & x_2 * (-1) & x_3 * 0 \end{bmatrix}$

$y^{*T} A_2 * x = [y_1 \quad y_2 \quad y_3] \begin{bmatrix} x_1 * 0 & x_2 * 1 & x_3 * (-1) \\ x_1 * (-1) & x_2 * 0 & x_3 * 1 \\ x_1 * 1 & x_2 * (-1) & x_3 * 0 \end{bmatrix}$

$\pi_2(x, y) = y_1 * (x_1 * 0 + x_2 * 1 + x_3 * (-1)) + y_2 * (x_1 * (-1) + x_2 * 0 + x_3 * 1) + y_3 * (x_1 * 1 + x_2 * (-1) + x_3 * 0)$

Daraus lässt sich schließen, dass $\boldsymbol{\pi_1(x, y) + \pi_2(x, y) = 0}$ ergibt. Daher handelt es sich um ein Nullsummenspiel. Das heißt, dass die Summe der Gewinne und Verluste im Spiel „Rock, Paper, Scissors“ für alle Spielerinnen gleich Null ist.

9. Maximin-Konzept

Dieses Konzept geht auf John von Neumann zurück. Es geht darum eine Strategie zu wählen, die den worst payoff maximiert. Daher der Name.

Wir definieren:

$\varphi(s_1) = \min\{\pi_1(s_1, s_2): s_2 \in S_2\}$ → schlechtestes Payoff von Spielerin 1, wenn Spielerin 1 die Aktion s_1 ausführt

Eine (konservative) Strategie für P_1:

Finde s_1 mit dem größten Wert für $\varphi(s_1)$.

↳ Maximin-Strategie

Genauso kann auch Spielerin 2 vorgehen.

Beispiel: Prisoner`s dilemma

Aufgabe: Finde Maximin-Strategie für P_1

φ ("s") = min {-1, -15} = -15

φ ("r) = min {0, -10} = -10

↳ Maximin-Strategie: "r"

Für gemischte Strategien lautet das Maximin-Problem für Spielerin 1:

$$\max_{\sigma_1 \in \Sigma_1} \{\min\{\sigma_1 A_1 \sigma_2 : \sigma_2 \in \Sigma_2\}\}$$

Hierbei bezeichnet Σ_1 die Menge aller **gemischten Strategien** für P_1; entsprechende bezeichnet Σ_2 die Menge aller **gemischten Strategien** für P_2.

Das Maximin Problem kann als lineares Programm geschrieben werden. Dazu schreiben wir den Maximin- Term um:

- Hilfsaussage:

  $$\min_{\sigma_2 \in \Sigma_2} \sigma_1^T A \sigma_2 = \min_{s_2 \in S_2} \sigma_1^T A s_2$$

  Das bedeutet: Das Minimum wird in den puren Strategien von Spielerin 2 angenommen.

Das Maximin lässt sich also folgendermaßen schreiben:

$$\max_{x_1, x_2 \in \mathbb{R}} \min_{s_2 \in S_2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T A s_2$$

unter der Nebenbedingung

$x_1, x_2 \geq 0$

$x_1 + x_2 > 1$

Eine weitere Umformung führt auf ein Problem der Gestalt:

$$\max_x c^T x$$

unter der Nebenbedingung

$Bx \leq b$

$x \geq 0$

Ein Problem von dieser Gestalt heißt ein lineares Programm.

10. Lineare Programme

Wir betrachten beispielhaft das lineare Programm

max $(-x_1 - 3x_2)$
$x_1, x_2 \in \mathbb{R}$ unter der Nebenbedingung

$2x_1 + 3x_2 \leq 6$
$x_1 - x_2 > -1$
$x_1, x_2 \geq 0$

In Normalform lautet das Problem

max $[-1, -3]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ unter den Nebenbedingungen
x_1, x_2

$$\begin{bmatrix} 2 & 3 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 6 \\ 1 \end{bmatrix}$$
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \geq 0$$

Die Menge

$$F = \{(x_1, x_2): \begin{bmatrix} 2 & 3 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 6 \\ 1 \end{bmatrix}, \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \geq 0\}$$
$$= \{(x_1, x_2): 2x_1 + 3x_2 \leq 6, -x_1 + x_2 \leq 1, x_1 \geq 0, x_2 \geq 0\}$$

heißt die zulässige Menge des Problems. Wir können das Problem rechnerisch oder zeichnerisch lösen. Laut einem Theorem befinden sich die gesuchten Stellen an den Eckpunkten von F, das einen Simplex darstellt.

Nun schreiben wir das Maximin-Problem als lineares Programm:

max min $[x_1, x_2]$ A s_1
x_1, x_2 $s_1 \in S_1$ unter der Nebenbedingung
$x_1 + x_2 = 1$
$x_1 \geq 0, x_2 \geq 0$ $[c_1, c_2]$

Wir müssen eine zusätzliche Variable einführen. Eine Liste von Zahlen kann minimiert werden, wenn wir eine Zahl z suchen, welche kleiner als jedes Element aus dieser Liste ist.

Das neue Problem in drei Unbekannten lautet

max z unter der Nebenbedingung
x_1, x_2, z

$z \leq [x_1, x_2]$ A $= [x_1, x_2]\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}$ => $1x_2, 2x_2 = [x_1 a_{1,1} + x_2 a_{2,1}, x_1 a_{1,2} + x_2 a_{2,2}]$

$x_1 + x_2 = 1$
$x_1 \geq 0, x_2 \geq 0$

Unser Problem als lineares Programm (aus Sicht von Spielerin 1)

max $[0, 0, 1]\begin{bmatrix} x_1 \\ x_2 \\ z \end{bmatrix}$

x_1, x_2, z unter der Nebenbedingung

$x_1 \geq 0, x_2 \geq 0$

$[1, 1, 0]\begin{bmatrix} x_1 \\ x_2 \\ z \end{bmatrix} = 1$

$$\begin{bmatrix} -a_{1,1} & -a_{2,1} & 1 \\ -a_{1,2} & -a_{2,2} & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ z \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Wir können das Problem für ein Spiel in Python lösen. Dazu verwenden wir das Paket scipy.

```
c = [0, 0, -1]

A_eq = [[1, 1, 0]]

b_ub = [0, 0]

b_eq = [1]


x0_bounds = (0, None)

x1_bounds = (0, None)

x2_bounds = (None, None)


print(" > player 1:")


# adapt matrix A_ub - the only term which is player dependent for

# finding a maximin strategy

A = np.ones((2, 3))

A[0 : n2, 0 : n1] = -A1.transpose()
```

```
A_ub = [A[0, :].tolist(), A[1, :].tolist()]

res = linprog(c, A_ub = A_ub, A_eq = A_eq, b_ub = b_ub, b_eq = b_eq,
bounds = [x0_bounds, x1_bounds, x2_bounds])

print(res)


print(" > player 2:")


# find maximin strategies


A = np.ones((2, 3))

A[0 : n2, 0 : n1] = -A2.transpose()


A_ub = [A[0, :].tolist(), A[1, :].tolist()]

res = linprog(c, A_ub = A_ub, A_eq = A_eq, b_ub = b_ub, b_eq = b_eq,
bounds = [x0_bounds, x1_bounds, x2_bounds])

print(res)
```

11. Konklusion:

Im Laufe der Woche haben wir mathematische Spieltheorien so gut kennengelernt, dass wir nun überall in unserem Alltag Situationen entdecken, die man mit der mathematischen Spieltheorie analysieren kann. Egal ob es darum geht zum Bus zu laufen, obwohl man lieber geht, Vokabeln für Schularbeiten zu lernen, obwohl sie vielleicht nicht abgefragt werden oder früher nach Hause zu kommen, um noch ein Stück Kuchen zu bekommen, die Spieltheorie kann uns helfen. Am ersten Tag der Woche ging es nur um Theorie, aber schon am zweiten Tag schrieben wir unsere ersten Codes mit Python. Für uns alle war dies eine ganz neue Erfahrung, da niemand von uns mit diesem Gebiet zuvor in Kontakt gekommen

sind. Deshalb ein ganz, ganz großes Dankeschön an unsere Gruppenleiter Florian, der uns unglaublich geduldig und liebenswert durch diese Woche begleitet hat.

Flugmechanische Modellierung: The Archer's Paradox

Marcel Dam, Jakob Erlacher, Niklas Hörtner, Benedikt Mautner
Georg Perz, Pia Sametz, Steve Keeling

Modellierungswoche 2023

Inhaltsverzeichnis

1 Einleitung

Motivation für unsere flugmechanische Modellierung eines Pfeiles ist das sogenannte Archer's Paradox, welches im Video *The Archer's Paradox in SLOW MOTION* von SmarterEveryDay thematisiert wird. Dieses Phänomen schildert die Beobachtung, dass ein Pfeil nach Abschuss, die Pfeilauflage quer verlässt, zu wobbeln beginnt und generell instabil erscheint. Obwohl all diese instabil wirkenden Zustände in einem Videoausschnitt veranschaulicht werden, trifft der Pfeil dennoch das Ziel. Wie kann das Ziel nun getroffen werden, wenn sich der Pfeil doch so unbeständig bewegt? Im Video werden als Lösung dieses Paradoxons die sogenannten Knotenpunkte angesprochen, die bei der Schwingung des Pfeiles entstehen.
Wir haben es uns nun zur Aufgabe gemacht, einen Pfeil, dessen Flugbahn und Schwingungen mechanisch, mathematisch naturgemäß und sinnvoll zu modellieren.

2 Masse-Feder-Systeme

Eine gute Näherung für das Verhalten eines Pfeiles kann mit Masse-Feder-Systemen getroffen werden. Aufgrund dessen liefern die folgenden Kapitel mathematische Beschreibungen von solche Systemen.

2.1 System einer Masse mit Schwerkraft

Wird eine Feder mit einem Gewicht belastet, dann wirkt eine elastische Kraft. Diese kann man durch ein Modell beschreiben, welches sich aus der aktuellen Position $w(t)$ und der Ruhelage $\overline{w}$ der Masse ausdrücken lässt.

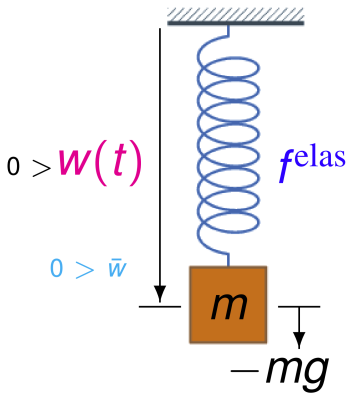


Abbildung 1: Einfaches Masse-Feder-System

$$f^{elas}(x) = -\kappa(w(t) - \overline{w}) \begin{cases} > 0, & |w(t)| > |\overline{w}| \\ < 0, & \text{sonst} \end{cases} \tag{1}$$

κ stellt in diesem Modell die Federkonstante dar. Sie gibt das Bestreben der Feder an, sich in die Ruhelage zurück zu begeben.

Laut Newton kann die Bewegung der Massen bezüglich deren Auslenkung so modelliert werden:

$$m \cdot w'' = m(w(t) - \overline{w})'' = f^{elas}(t) - mg = -\kappa(w(t) - \overline{w}) - mg \tag{2}$$

Ein Lösungsansatz dieser Differentialgleichung (DGL) liegt in der Formel der harmonischen Schwingung

$$w(t) = asin(\omega t) + bcos(\omega t) + c \tag{3}$$

$$a, b, \omega \in \mathbb{R}\backslash\{0\}$$
$$c \in \mathbb{R}$$

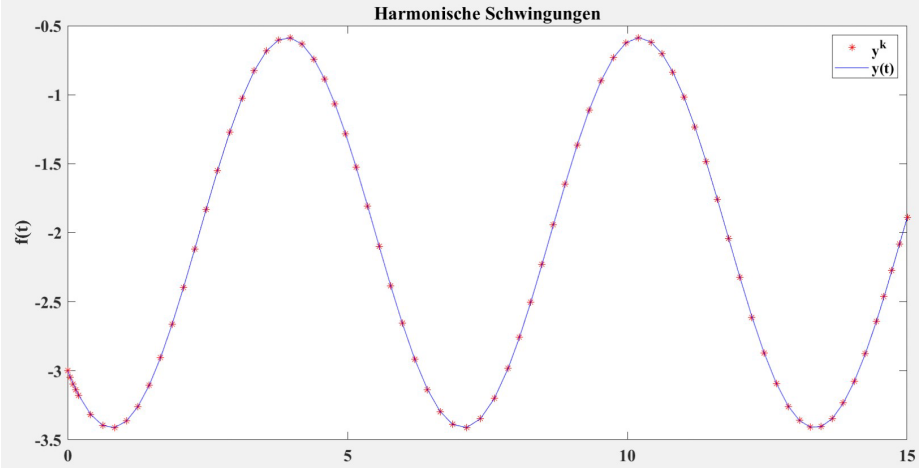


Abbildung 2: Schwingung ohne Dämpfung

Nichtsdestotrotz sind solche harmonischen Schwingungen nicht realistisch, da es eine innere Reibung in der Feder gibt, welche es nicht zu vernachlässigen gilt. Ein Modell, um diese Reibung darzustellen, hängt linear von der Geschwindigkeit ab, wobei $\mu > 0$ der Reibungskoeffizient ist.

$$f^{reib}(t) = -\mu(w(t) - \overline{w})' = -\mu w'(t) \tag{4}$$

Die Lösung dieser Differentialgleichung ist eine gedämpfte Schwingung mit einem charakteristischen Vorfaktor wie e^{-t}.

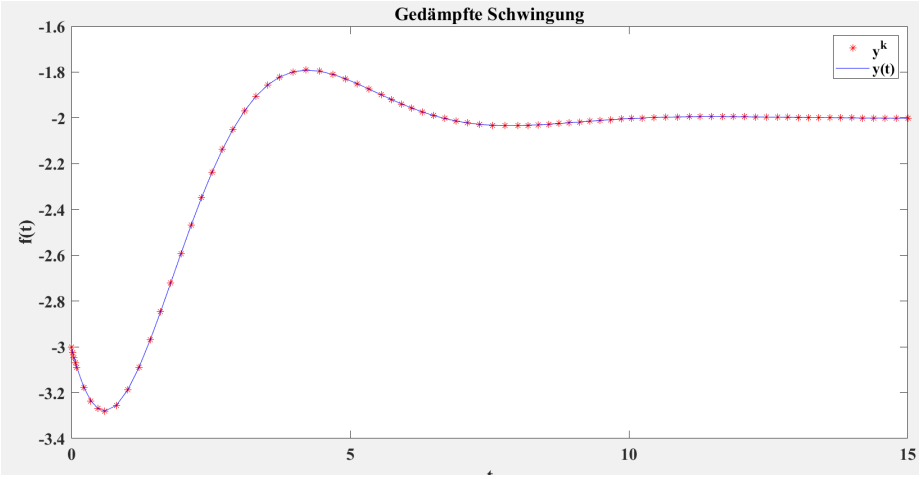


Abbildung 3: Schwingung mit Dämpfung

2.2 Lösungsansätze von DGLs zweiter Ordnung

Differentialgleichungen diskret zu lösen, ist in den meisten Fällen leider nicht möglich und eine Approximationsmethode muss verwendet werden. Als Grundlage solch numerischer Lösungen können Methoden wie Vorwärts- oder Rückwärts Euler verwendet werden. Diese approximieren den Differentialquotienten mithilfe des Differenzenquotienten in kleinen Abschnitten, sodass die Differenz zwischen jeder Evaluation vernachlässigbar klein ist. In MATLAB gibt es mehrere Funktionen, welche Differentialgleichungen numerisch lösen können. Wir verwenden großteils die beliebte Funktion *ode45()* und implementieren am Ende einen eigenen Löser, die Pheta Methode. Problematisch mit all diesen Lösungsbefehlen ist, dass sie nur Differentialgleichungen erster Ordnung lösen können und nicht Zweiter. Deswegen müssen Differentialgleichungen zweiter Ordnung in DGLs erster Ordnung umgeschrieben werden. Um dies zu bewerkstelligen, muss ein Trick angewandt werden. Man schreibt die Differentialgleichung wie folgt um und kann diese dann auch in eine Matrix einschreiben. Auch die DGLs, relevant für unsere Feder-Massen-Ketten, müssen von zweiter Ordnung in Erste umgeschrieben werden. Formel 2 mit den Bedingungen m, $\kappa = 1$, $p = -3$, $q = -1$, $\kappa\overline{w} - mg = -2$ wird betrachtet:

$$x(t) = w(t),\ y(t) = w'(t) \tag{5}$$

$$\begin{cases} x'(t) = w'(t) = y(t) \\ y'(t) = w''(t) = -w(t) - 2 = -x(t) - 2 \end{cases}$$

in Matrizen und Vektor Form:

$$X'(t) = AX(t) + b,\ X(0) = X_0 \tag{6}$$

$$X(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix},\ X_0 = \begin{bmatrix} -3 \\ -1 \end{bmatrix},\ A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix},\ b = \begin{bmatrix} 0 \\ -2 \end{bmatrix} \tag{7}$$

Gibt es bei Differentialgleichungen ein konstantes Glied, dann nennt man sie inhomogen. Um diesen Umstand bei der Umschreibung zu Matrizen zu berücksichtigen, muss dieses konstante Glied abgewandelt in einen Vektor mit der Matrix addiert werden.

2.3 Längsbewegung

In diesem Beispiel sieht man nun zwei Massen, welche mit drei Federn waagerecht verbunden sind. Sie beeinflussen sich somit auf der x-Achse gegenseitig.

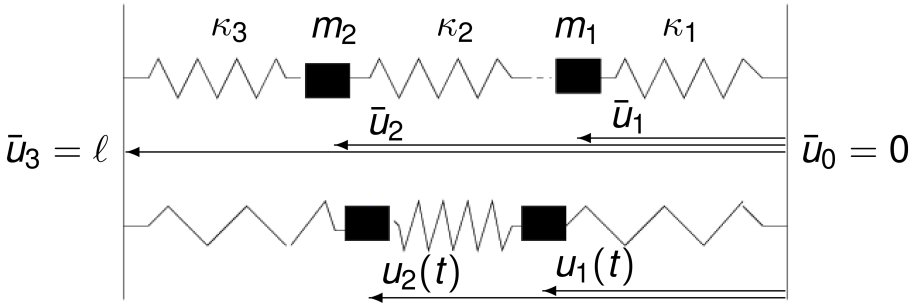


Abbildung 4: Masse-Feder-System mit zwei Massen

Die elastischen Kräfte, welche auf die Massen wirken, können folgendermaßen beschrieben werden: $\Delta u_i(t) = u_i(t) - \overline{u}_i$

$$m_1 u_1''(t) = \kappa_2[\Delta u_2(t) - \Delta u_1(t)] - \kappa_1 \Delta u_1(t) \tag{8}$$

$$m_2 u_2''(t) = \kappa_2[\Delta u_1(t) - \Delta u_2(t)] - \kappa_3 \Delta u_2(t) \tag{9}$$

In Matrizenform und mit den Konditionen $m_i = m, \kappa_i = \kappa, \overline{u}_i = i\ell/3$ kann der Sachverhalt mit Dämpfung $\mu \leq 0$ so dargestellt werden:

$$mU''(t) + \mu U'(t) = AU(t) + b,\ U(0) = U_0,\ U'(0) = U_1 \tag{10}$$

mit $U_0 = [u_1(0); u_2(0)], U_1 = [u_1'(0); u_2'(0)]$ gegeben und

$$U(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix},\ A = \kappa \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix},\ b = \kappa \begin{bmatrix} 0 \\ \ell \end{bmatrix} \tag{11}$$

Die Formel in erster Ordnung schaut folgendermaßen aus:

$$\mathcal{U}'(t) = \mathcal{A}\mathcal{U}(t) + \beta,\ \mathcal{U}(0) = \mathcal{U}_0 \tag{12}$$

Auch mit Block-Vektoren und Block-Matrizen kann man das System darstellen

$$\mathcal{U}(t) = \begin{bmatrix} U(t) \\ U'(t) \end{bmatrix},\ \mathcal{A} \begin{bmatrix} Z & I \\ \frac{1}{m}A & -\frac{\mu}{m}I \end{bmatrix},\ \beta = \begin{bmatrix} z \\ \frac{1}{m}b \end{bmatrix},\ \mathcal{U}_0 = \begin{bmatrix} U_0 \\ U_1 \end{bmatrix} \tag{13}$$

mit $Z = [0,0;0,0], I = [1,0;0,1]$ und $z = [0;0]$

Mit diesen Formeln kann man nun die x-Positionen der Massen berechnen und die Schwingungen dieser aufzeigen. Die Massen werden in unserer Animation als schwarzes Quadrat dargestellt und die Schwingung der Massen werden in einem Diagramm veranschaulicht. Auch hier ist es wichtig, eine Reibung einzubauen, sodass der Prozess so realistisch wie möglich dargestellt werden kann. Harmonische beziehungsweise gedämpfte Schwingungen können in den Diagrammen 2 und 3 erkannt werden.
In der zweiten Vorübung wird eine Masse-Feder-Kette in querer Auslenkung mit zwei Massen, drei Federn und ohne Schwerkraft behandelt.

2.4 Querbewegung

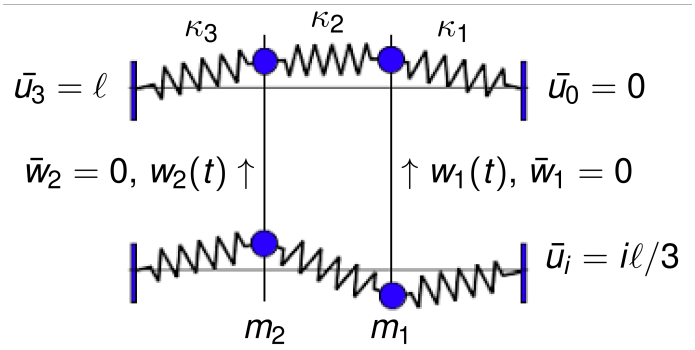


Abbildung 5: Quere Auslenkung der Massen

Sind die Federn so stark ausgedehnt, dass ihre Ruhelängen so gut wie Null sind, dann gilt zum Beispiel für die Federkraft $F_1 \approx -\kappa\sqrt{u_1^2 + w_1^2}$. Nach trigonometrischen Rechnungen im rechtwinkeligen Dreieck, welches sich aus Ruhelage, Ausschlag und Feder ergibt, können für die senkrecht wirkenden Kräfte auf die Massen Formeln entwickelt werden: $F_1 = -\kappa w_1, F_2 = -\kappa(w_2 - w_1)$ und $-\kappa_3 w_2$. Folgend kann ein System der Kräfte formuliert werden:

$$mw_1''(t) = \kappa_2[w_2(t) - w_1(t)] - \kappa_1 w_1(t) \tag{14}$$

$$mw_2''(t) = \kappa_2[w_1(t) - w_2(t)] - \kappa_3 w_2(t) \tag{15}$$

In Form von Vektoren und Matrizen und Dämpfung $\mu \geq 0$,

$$mW''(t) + \mu W'(t) = AW(t),\ W(0) = W_0,\ W'(0) = W_1 \tag{16}$$

mit $W_0 = [w_1(0); w_2(0)],\ W_1 = [w_1'(0); w_2'(0)]$ gegeben.

Die Formel in erster Ordnung schaut folgendermaßen aus:

$$\mathcal{W}'(t) = \mathcal{A}\mathcal{W}(t), \mathcal{W}(0) = \mathcal{W}_0 \tag{17}$$

Mit Block-Vektoren und Block-Matrizen kann man das System ebenfalls darstellen,

$$\mathcal{U}(t) = \begin{bmatrix} W(t) \\ W'(t) \end{bmatrix},\ \mathcal{A} \begin{bmatrix} Z & I \\ \frac{1}{m}A & -\frac{\mu}{m}I \end{bmatrix},\ \mathcal{W}_0 = \begin{bmatrix} W_0 \\ W_1 \end{bmatrix} \tag{18}$$

3 Modellierung eines Pfeils

Im Laufe der Woche entwickelten wir zwei verschiedene Modelle für die Bewegung eines Pfeils. Einerseits wurde ein strukturelles, andererseits ein empirisches Modell entwickelt.
Ein strukturelles Modell ist theoretisch fundiert, welches die zugrunde liegenden Beziehungen in einem System beschreibt. Dieses versucht Mechanismen zu verstehen und Phänomene zu erklären.
Im Gegensatz dazu versucht ein empirisches Modell mit Hilfe von beobachteten Daten ein System zu erkennen und durch statistische Methoden zu modellieren. Solche Modelle beruhen auf empirisch gesammelten Daten und nicht auf klar formulierten Theorien.

3.1 Empirisches Modell

Wir haben uns damit beschäftigt, ein empirisches Modell zur Veranschaulichung der Flugbahn eines Pfeils zu entwerfen. Für dieses Modell haben wir eine Vielzahl an Formeln verwendet. Als Erstes haben wir uns mit den wichtigsten Grundlagen der Kapitel Numerik und Analysis vertraut gemacht, indem wir mit Matrizen und Integralrechnung gearbeitet haben. Schließlich sahen wir uns das Video von SmarterEveryDay zum Archer's Paradox mehrmals an, aus welchem wir folgende Schlüsse zogen: Ein Pfeil ist während des Fluges elastisch und dreht sich dabei um die eigene Achse. Das Programm MATLAB nutzten wir zur grafischen Darstellung, mit welchem wir einen Code für die Simulation der Flugbahn des Pfeils entwarfen. Bei einer Einführung haben wir gelernt, harmonische Schwingungen, gedämpfte Schwingungen und einen exponentiellen Zerfall zu modellieren. Daraufhin überlegten wir uns eine Funktionsgleichung

mit den Eigenschaften $f''(0) = 0$ und $f''(1) = 0$ im Intervall [0;1]. Zuerst wollten wir es mit einer Polynomfunktion versuchen. Bald bemerkten wir, dass eine Sinusfunktion bereits schwingt und deshalb besser dafür geeignet ist. Daraus ergab sich der Funktionsterm $f(x) = sin(\pi \cdot x)$. Als nächstes haben wir einen Parameter α für den Term $f(x) = sin(\pi \cdot x) - \alpha$ ermittelt, damit der Schwerpunkt nicht nur bei $x = 0.5$ sondern auch bei $y = 0$ liegt. Dabei haben wir zuerst die Länge des Bogens der Funktion in $[0; 1]$ bestimmt. Dafür wurde die folgende Formel verwendet:

$$l = \int_0^1 \sqrt{1 + [v'(x)]^2} dx \tag{19}$$

danach wurde die x-Koordinate des Zentroids (welches dem Schwerpunkt entspricht) berechnet, sodass das Ergebnis 0,5 beträgt. Die folgende Formel bestätigte unsere Vermutung:

$$\overline{x} = \frac{1}{l} \int_0^1 x\sqrt{1 + [v'(x)]^2} dx = \frac{1}{2} \tag{20}$$

Für die Berechnung der y-Koordinate diente die nachstehende Formel: ?? daraufhin konnte der Parameter α bestimmt werden: $\alpha = \frac{y_{zentroid}}{l}$ Länge. Nachdem wir den vollständigen Funktionsterm der Funktion

$$f(x) = \frac{1}{10} sin(\pi x) - 0.0631634135 \tag{21}$$

definierten, haben wir den Abschnitt mit MATLAB modelliert. Durch die Elastizität und Rotation ist der Schwerpunkt zu jedem Zeitpunkt anders, obwohl wir mit α vorher das Zentroid bei $y = 0$ definiert haben. Zu einem späteren Zeitpunkt bemerkten wir jedoch, dass wir dies vernachlässigen konnten.

3.1.1 Funktionen in MATLAB

```
th = @(t) exp(-t/10).*sin(t);
```

Mit dieser Funktion haben wir einen Dämpfer als eine Sinusfunktion mit exponentiellem Abstieg dargestellt. Dies fügten wir hinzu, da wir einen Verlust der Schwingung des Pfeiles haben, z.B. ausgelöst durch entstehende Wärme innerhalb des Pfeiles. Jedoch haben wir den Luftwiderstand vernachlässigt.

```
u = @(x,t) x + q*t;
```

Da sich die Richtung von u nach vorne ändert, wurde eine lineare Funktion verwendet in Abhängigkeit von x und t.

```
v = @(x,t) cos(om*t)*th(t)*n0(x);
```

Die Funktion der v-Richtung beschreibt zwei Bewegungen des Pfeils. Einerseits die Rotation auf die wir später eingehen und andererseits die Biegung des Pfeils. Diese wird durch die Kombination der Funktion $n0(x)$ (zuständig für Streckung und Stauchung) und $th(t)$ (zuständig für Abnahme von Streckung und Stauchung) dargestellt.

```
w = @(x,t) sin(om*t)*th(t)*n0(x)+p+s*t-g*t^2/2+ (1/q)*(s- g*t)*x;
```

Die Funktion der w-Richtung beschreibt gleich wie die Funktion der v-Richtung die Rotation des Pfeils. Bei der Funktion w ist auch noch die Gravitation $\frac{g \cdot t^2}{2}$ und einige Parameter zur realitätsgetreueren Darstellung inkludiert. Die Rotation des Pfeils wird bei den Funktionen der v- und w-Richtung durch jeweils Cosinus und Sinus dargestellt.

3.1.2 Ergebnis

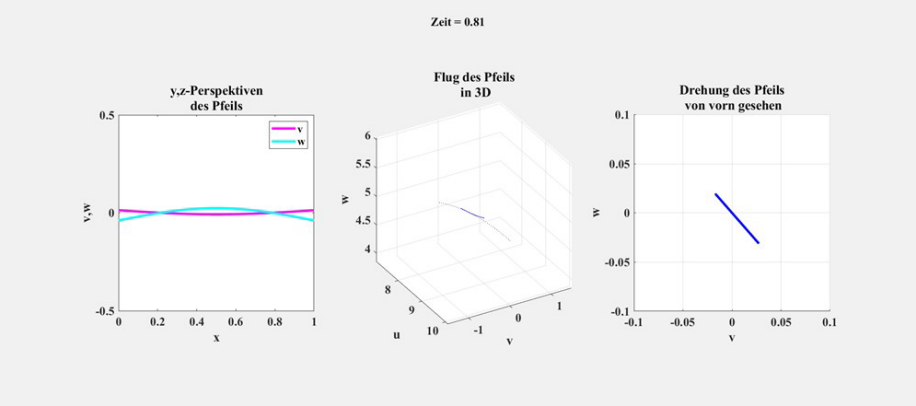


Abbildung 6: Ergebnis des empirischen Modells

Die Zeit t=0.81 gibt die vergangene Zeit seit Abschuss des Pfeiles an. Das erste Diagramm zeigt die Schwingung von der v- und w- Richtung des Pfeiles von der Seite. Die v-Richtung ist magenta und die w-Richtung ist hellblau dargestellt. Das Diagramm in der Mitte zeigt, wie der Pfeil in 3D fliegt. Der in blau dargestellte Pfeil liegt auf der Flugparabel, die durch schwarze Punkte präsentiert wird. Das dritte Diagramm bildet die Rotation und die Schwingung des Pfeiles von der Perspektive in der w- und v- Richtung.

3.2 Strukturelles Modell

3.2.1 Verbindung mit Federsystemen

Um einen realistischen Bezug zu einem Pfeil herzustellen, modellierten wir ein Feder-Masse-System mit n Massen. Das Endprodukt orientiert sich an dieser Grafik.

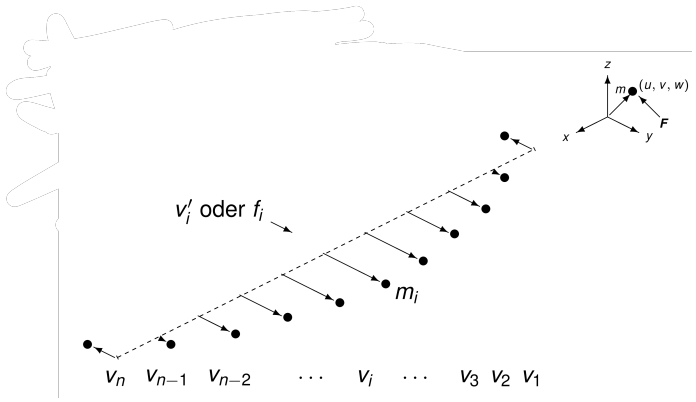


Abbildung 7: Pfeilmodell

Hierzu mussten wir unser Modell grundsätzlich umstrukturieren, wobei es auf Blockmatrizen angepasst wurde. Diese ändern ihre Größe basierend auf der Zahl der Massen dynamisch. Darüber hinaus haben wir in diesem strukturellen Modell auch die Steifigkeit Sigma der Federn eingebaut. Dies gelang durch Implementierung der folgenden biharmonischen Matrix.

$$B = \sigma \begin{bmatrix} -1 & 2 & -1 & 0 & 0 \\ 2 & -5 & 4 & -1 & 0 \\ -1 & 4 & -6 & 4 & -1 \\ 0 & -1 & 4 & -5 & 2 \\ 0 & 0 & -1 & 2 & -1 \end{bmatrix} \tag{22}$$

Weiters fügten wir Funktionen für die Bewegungen in x- und z-Richtung hinzu. Das Modell wurde linear in die x-Richtung und mit einem parabelähnlichen Abfall in die y-Richtung beschrieben. Letzteres wurde gewählt, um die Schwerkraft zu imitieren. Der Luftwiderstand wurde in diesem Modell vernachlässigt, da er für das Archer's Paradoxon irrelevant ist. Hierdurch lässt sich unser Modell in 3D veranschaulichen.
Um das Paradoxon besser visualisieren zu können, fügten wir noch einen Zylinder hinzu, der den Bogengriff symbolisch veranschaulicht.

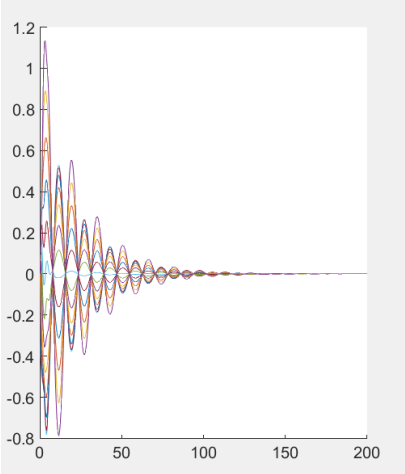


Abbildung 8: Bewegung aller Massen

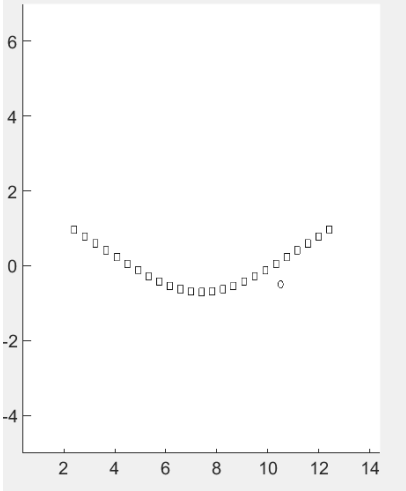


Abbildung 9: Darstellung des Archer's Paradox in 2D

3.2.2 Steifigkeit versus Elastizität

Um einen Pfeil realistisch modellieren zu können, muss der Unterschied zwischen Steifigkeit (σ) und Elastizität (κ) erarbeitet werden.

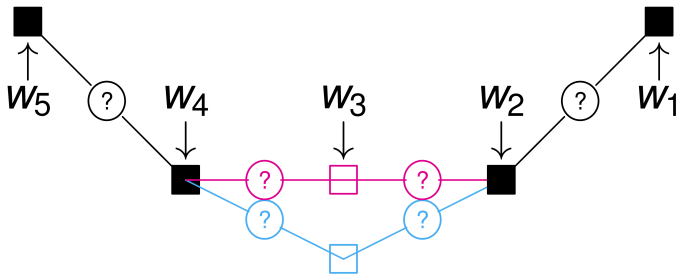


Abbildung 10: Sigma v. Kappa

Wesentlich ist, dass in der Abbildung die rote Linie die Elastizität beschreibt und die blaue die Steifigkeit. Die Unterschiede sind, dass die Veränderungen bei elastischen Modellen sich nur auf die direkten Nachbarn auswirken. Bei steifen Modellen wirken sich Auslenkungen auf die zwei nächsten Nachbarn, also auf die vier nächsten Glieder aus.

3.3 Auswirkungen verschiedener Faktoren

Hier wird der Code ausgeführt, der veranschaulicht, wie sich der Pfeil bewegt. Dazu werden die Größen der Variablen sg (σ, Steifigkeit), ka (κ, Elastizität) und mu (μ, Dämpfung) einzeln verändert, um herauszufinden, welche Wirkung jede auf die Schwankung des Pfeiles hat.

```
function Cord
N = 20; M = 500; T = 400; dt = T/(M-1); dx = 1/(N-1);
x = linspace(0,1,N)'; t = linspace(0,T,M);

sg = 1.0e-2;
ka = 1.0e-2;
mu = 1.0e-1;
th = 0.5;
e = ones(N,1);
m = ones(N,1); m(1)=20*m(1); M1 = spdiags(1./m,0,N,N);


B = spdiags([-e 4*e -6*e 4*e -e],-2:2,N,N);
B(1,1) =-1; B(1,2) = 2; B(2,1) = 2; B(2,2) =-5;
B(N-1,N)= 2; B(N-1,N-1)=-5; B(N,N) =-1; B(N,N-1)= 2; B=sg*B/dx^4;

L = spdiags([e -2*e e],-1:1,N,N); L(1,1)=-1; L(N,N)=-1; L=ka*L/dx^2;
Z = 0*speye(N); I = speye(N); AA = [Z,I;M1*(B+L),-mu*M1];

V0 = zeros(N,1); z = zeros(N,1); F = [z;z];

V1 = zeros(N,1);
V1(1)=1;
V1 = V1 - mean(m.*V1)/mean(m);

z = zeros(N,1); F = [z;z];

VV0 = [V0;V1]; VV = VV0; v = VV(1:N); vsav = v;
AL = speye(2*N) - (1-th)*dt*AA; AR = speye(2*N) + th*dt*AA;
subplot(1,2,1)
plot(x,v); xlim([0,1]); ylim([-1,1]); drawnow;
for k=2:M
    f = zeros(N,1); f(N)=-v(N); F = [z;M1*f];
VV = AL \ (AR*VV + dt*F);
v = VV(1:N); vsav=[vsav,v];
plot(x,v); xlim([0,1]); ylim([-1,1]); drawnow;
end
subplot(1,2,2)
surf(x,t,vsav')
end
```

3.3.1 Wirkung der Steifigkeit

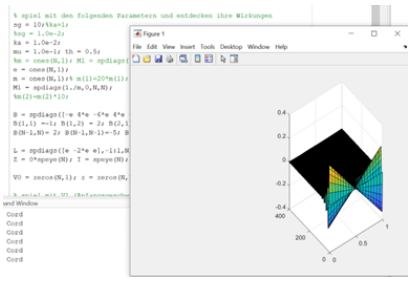


Abbildung 11: Caption

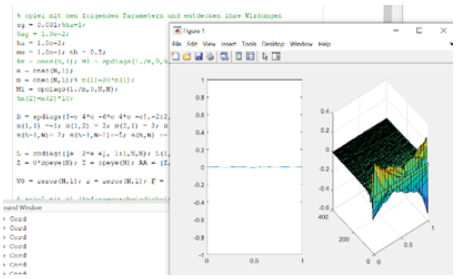


Abbildung 12: Caption

Je höher die Steifigkeit (in Abbildung 11 höher als in Abbildung 12), desto weniger wobbelt der Pfeil während seiner Fluglaufbahn.

3.3.2 Wirkung der Elastizität

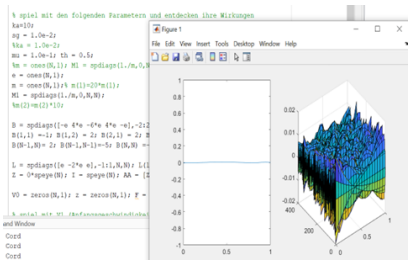


Abbildung 13: Caption

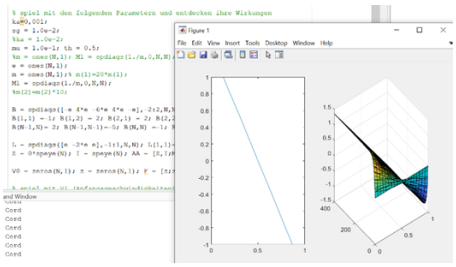


Abbildung 14: Caption

Je höher die Elastizität (in Abbildung 13 hoher als in Abbildung 14), desto mehr schwankt der Pfeil in seiner Bewegung. Im Gegensatz zur Elastizität bezieht sich die Steifigkeit unter der Wirkung einer Kraft auf vier Nachbarteilchen und bevorzugt deshalb Steifigkeit. Elastizität hingegen bezieht sich auf nur zwei. Wir haben uns gefragt, ob Kappa für die Simulation der Bewegung des Pfeiles in unserem Code überhaupt notwendig ist. Die Antwort darauf lautet: „Ja", allerdings hat die Elastizität keinen besonders großen Einfluss auf die Flugbahn des Pfeiles.

3.3.3 Wirkung der Dämpfung

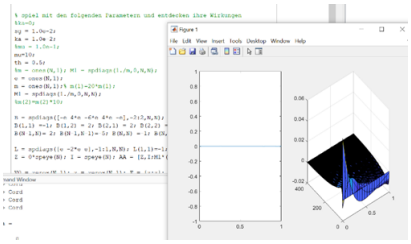


Abbildung 15: Caption

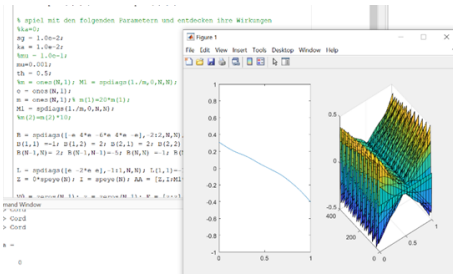


Abbildung 16: Caption

Je höher die Dämpfung (in Abbildung 15 höher als in Abbildung 16), desto schneller pendelt sich die Wobb-Bewegung wieder ein. Wäre die Dämpfung gleich Null, dann würde der Pfeil endlos hin und her wackeln, was wieder sehr realitätsfern wäre.

3.4 Wirkung der Anfangsgeschwindigkeit

Wenn die Anfangsgeschwindigkeit Null ist, fliegt der Pfeil nicht, daher gibt es auch keine Ausschwankungen.

Ziel war es, den Pfeil während der Ausschwankungen auf seiner ursprünglichen Achse zu halten, dafür mussten wir eine Formel finden. Schlussendlich konnten wir diese identifizieren, wodurch jede Simulation am Ende bei Null landete.

```
1     V1 = V1 - mean(m.*V1)/mean(m);
```

V1 ist ein Array der Anfangsgeschwindigkeiten, welcher angepasst wird. Konkret werden die Massen (m) mit den nicht angepassten Geschwindigkeiten multipliziert und dann der Durchschnitt dieser durch den Durchschnitt der Massen dividiert. Hierzu wird die MATLAB-Funktion *mean* verwendet, die den Durchschnitt eines Arrays ausrechnet.

Mithilfe dieser Formel landet der Pfeil im Durchschnitt immer auf der Ausgangsachse.

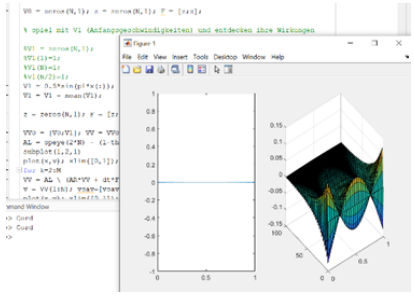


Abbildung 17: Mit Mittelwert

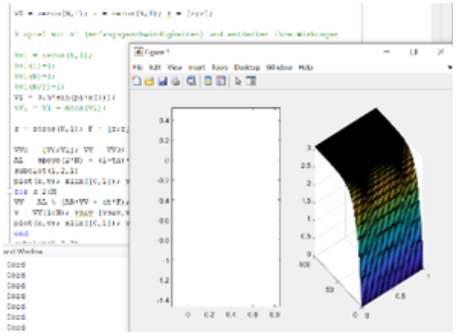


Abbildung 18: Ohne Mittelwert

4 Numerische Verfahren

4.1 ode45

ode45 ist eine MATLAB-interne Funktion, die das numerische Lösen von Differenzialgleichungen erster Ordnung sehr simplifiziert. Ein typischer Aufruf dieser Funktion sieht so aus:

```
[t,Y] = ode45(@RechteSeite, tspan, X0);
```

Man muss *ode*45 ein Handle zu einer solchen Funktion übergeben, welche den nächsten Zeitschritt mittels der ersten Ableitung beschreibt. Angeführt wird nun ein Beispiel für die Beschreibung eines Zwei-Massen-Schwingersystems in die Längsrichtung, wie in Abbildung 4 dargestellt.

```
    A = [0,           0,           1,   0;
         0,           0,           0,   1;
         (-k1-k2)/m1, k2/m1,      -my,  0;
         k2/m2,       (-k2-k3)/m2, 0,  -my];
    b = [0; 0; (k1*u1+k2*u1-k2*u2)/m1; (k2*u2-k2*u1+k3*u2)/m2];
    dX = A*X + b;
```

k_i stellen hier die Federkonstanten der Feder i dar, während m_i die Massen darstellen. Die Elastizität wird von my beschrieben, wobei hier $my > 0$ gilt.

4.2 Theta Methode

Die Theta Methode kann verwendet werden, um numerische Lösungen zu Differenzialgleichungen zu finden.

$$\frac{y^{k+1} - y^k}{\tau} = \theta \cdot f^{k+1} + (1 - \theta) \cdot f^k \tag{23}$$

wobei $0 \leq \theta < 1$, da bei $\theta = 1$ die Formel die Form für der Euler Methode annimmt. f ist hier die treibende Funktion für y'. Wird diese Methode angewendet,

kann die Differenzialgleichung mit einem System aus linearen Gleichungen gelöst werden. Dies wird in MATLAB mit einem internen Befehl gelöst, der durch ein \ dargestellt wird.

```
1 for k=2:M
2     VV = AL \ (AR*VV + dt*F);
3     v = cat(1, VV(1:n_elems), VV(n_elems*2+1:n_elems*3));
4     vsav=[vsav,v];
5 end
```

M beschreibt hier die insgesamte Anzahl der Iterationen, AL und AR repräsentieren hier die beiden Faktoren der Theta Methode $1-(1-\theta)h$ und $1+\theta h$. VV ist hier y^{k-1} vor und $VV = y^k$ nach der Berechnung auf Zeile 2. Zeile 3 wählt von den Ergebnissen nur die Positionen in x- und y-Richtung aus. Die anderen Werte (Geschwindigkeiten) sind irrelevant. vsav speichert alle Zeitschritte. $dt * F$ ist ein konstanter Faktor, der eine Kraft F wirken lassen kann, welche das System inhomogen macht.

5 Code

Eine vollständige Version des Codes kann unter folgendem Link gefunden werden: https://github.com/Benimautner/modellierungswoche23