Numerische Mathematik - Projektteil 2

Richard Weiss Florian Schager Christian Sallinger Fabian Zehetgruber
Paul Winkler Christian Göth

Random code snippet, damit alle checken, wie man code displayt:

```
print("Hello World!")
```

1 Eigenschwingungen

1.1 Problembeschreibung

Das Projekt beschäftigt sich mit den Eigenschwingeungen einer fest eingespannten Saite. Sei dazu $u(t,x)$ die vertikale Auslenkung der Saite an der Position $x \in [0,1]$ zur Zeit t. u wird näherungsweise durch die sogenannte Wellengleichung

$$\frac{\partial^2 u}{\partial x^2}(t,x) = \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2}(t,x) \tag{1}$$

für alle $x \in (0,1)$ und $t \in \mathbb{R}$ beschrieben, wobei c die Ausbreitungsgeschwindigkeit der Welle ist. Wenn die Saite an beiden Enden fest eingespannt ist, so gelten die Randbedingungen

$$u(t,0) = u(t,1) = 0$$

für alle $t \in \mathbb{R}$.

Zur Berechnung der Eigenschwingungen suchen wir nach Lösungen u, die in der Zeit harmonisch schwingen. Solche erfüllen folgenden Ansatz

$$u(x,t) = \Re(v(x)e^{-i\omega t})$$

mit einer festen, aber unbekannten Kreisfrequenz $\omega > 0$ und einer Funktion v, welche nur noch vom Ort x abhängt. Durch Einsetzen erhalten wir für v die sogenannte Helmholz-Gleichung

$$-v''(x) = \kappa^2 v(x), \qquad x \in (0,1), \tag{2}$$

mit der unbekannten Wellenzahl $\kappa := \frac{\omega}{c}$ und den Randbedingungen

$$v(0) = v(1) = 0. \tag{3}$$

1.2 Analytische Lösung

$$v_\kappa(x) = C_1 \cos(\kappa x) + C_2 \sin(\kappa x), \qquad x \in [0,1], \tag{4}$$

mit beliebigen Konstanten C_1, C_2 löst die Helmholz-Gleichung (2). Das erkennt man durch stumpfes Einsetzen.

$$\begin{aligned} -v_\kappa''(x) = -\frac{\partial^2}{\partial x^2}(C_1 \cos(\kappa x) + C_2 \sin(\kappa x)) &= -\frac{\partial}{\partial x}(-C_1\kappa \sin(\kappa x) + C_2\kappa \cos(\kappa x)) \\ &= -(-C_1\kappa^2 \cos(\kappa x) - C_2\kappa^2 \sin(\kappa x)) = \kappa(C_1 \cos(\kappa x) + C_2 \sin(\kappa x)) = \kappa^2 v_\kappa(x) \end{aligned}$$

Wir fragen uns, für welche $\kappa > 0$, Konstanten C_1 und C_2 existieren, sodass v_κ auch die Randbedingungen (3) erfüllt.

$$0 \overset{!}{=} \begin{cases} v_\kappa(0) = C_1 \cos 0 + C_2 \sin 0 = C_1 \\ v_\kappa(1) = C_1 \cos\kappa + C_2 \sin\kappa = C_2 \sin\kappa \end{cases}$$

Nachdem $\cos 0 = 1$ und $\sin 0 = 0$, erhält man, aus der oberen Gleichung, $C_1 = 0$. Mit der unteren Gleichung folgt aber auch $C_2 \sin\kappa = 0$. Wenn nun auch $C_2 = 0$, dann erhielte man die triviale Lösung $v_\kappa = 0$. Für eine realistischere Modellierung, d.h. $v_\kappa \neq 0$, müsste $\sin\kappa = 0$, also $\kappa \in \pi\mathbb{Z}$.

Das sind die gesuchten $\kappa > 0$. Sei nun eines dieser κ fest. Offensichtlich ist $C_1 = 0$ eindeutig, $C_2 \in \mathbb{R}$ jedoch beliebig.

1.3 Numerische Approximation

Häufig lassen sich solche Probleme nicht analytisch lösen, sodass auf numerische Verfahren zurückgegriffen wird, welche möglichst gute Näherungen an die exakten Lösungen berechnen sollen. Als einfachstes Mittel dienen sogenannte Differenzenverfahren. Sei dazu $x_j := jh$, $j = 0, \ldots, n$ eine Zerlegung des Intervalls $[0,1]$ mit äquidistanter Schrittweite $h = 1/n$. Die zweite Ableitung in (2) wird approximiert durch den Differenzenquotienten

$$v''(x_j) \approx D_h v(x_j) := \frac{1}{h^2}(v(x_{j-1}) - 2v(x_j) + v(x_{j+1})), \qquad j = 1, \ldots, n-1. \tag{5}$$

Es sei angemerkt, dass (**??**) tatsächlich einen Differenzenquotienten beschreibt. Um das einzusehen, verwenden wir den links- und rechts-seitigen Differenzenquotient erster Ordnung, sowie $x_{j-1} = x_j - h$, $x_{j+1} = x_j + h$. Wir erhalten $\forall\, j = 1, \ldots, n-1$:

$$\begin{aligned} v''(x_j) &= \lim_{h\to 0} \frac{1}{h}(v'(x_j + h) - v'(x_j)) \\ &= \lim_{h\to 0} \frac{1}{h}\left(\frac{1}{h}(v(x_j + h) - v(x_j)) - \frac{1}{h}(v(x_j) - v(x_j - h))\right) \\ &= \lim_{h\to 0} \frac{1}{h^2}(v(x_j + h) - 2v(x_j) + v(x_j - h)) \\ &= \lim_{h\to 0} D_h v(x_j) \end{aligned}$$

1.3.1 Approximationsfehler

Für hinreichend glatte Funktionen v mit einer geeigneten Konstanten $C > 0$ wird der Approximationsfehler quadratisch in h klein, d.h. dass

$$|v''(x_j) - D_h v(x_j)| \leq Ch^2. \tag{6}$$

Nachdem v hinreichend glatt ist, gilt nach dem Satz von Taylor, dass $\forall\, j = 1, \ldots, n-1$:

$$\begin{aligned} v(x_j + h) &= \sum_{\ell=0}^{n+2} \frac{h^\ell}{\ell!} v^{(\ell)}(x_j) + \mathcal{O}\big(h^{n+3}\big), \\ v(x_j - h) &= \sum_{\ell=0}^{n+2} \frac{(-h)^\ell}{\ell!} v^{(\ell)}(x_j) + \mathcal{O}\big(h^{n+3}\big). \end{aligned}$$

Man beachte, dass sich die ungeraden Summanden, der oberen Taylor-Polynome, gegenseitig aufheben. Damit erhalten wir für den Differenzenquotient $D_h v(x_j)$, $j = 1, \ldots, n-1$ eine asymptotische Entwicklung.

$$\begin{aligned} D_h v(x_j) &= \frac{1}{h^2}(v(x_j - h) + v(x_j + h) - 2v(x_j)) \\ &= \frac{1}{h^2}\left(2v(x_j) + h^2 v''(x_j) + \sum_{\substack{\ell=4 \\ \ell \in 2\mathbb{N}}}^{n+2} \frac{h^\ell}{\ell!} v^{(\ell)}(x_j)(1 + (-1)^\ell) - 2v(x_j)\right) + \mathcal{O}\big(h^{n+3}\big) \\ &= v''(x_j) + 2\sum_{\ell=1}^{\lfloor \frac{n}{2} \rfloor} \frac{h^{2\ell}}{(2\ell+2)!} v^{(2\ell)}(x_j) + \mathcal{O}\big(h^{n+1}\big) \end{aligned}$$

Daraus folgt unmittelbar die quadratische Konvergenz (6), d.h. $\forall\, j = 1, \ldots, n-1$:

$$D_h v(x_j) - v''(x_j) = \mathcal{O}\big(h^2\big), \qquad h \to 0.$$

1.3.2 Eigenwertproblem

Wir wollen nun den Differenzenquotienten $D_h v(x_j)$ verwenden, um ein Eigenwertproblem der Form $A\vec{v} = \lambda\vec{v}$ mit einer Matrix $A \in \mathbb{R}^{(n-1)\times(n-1)}$ zu dem Eigenvektor $\vec{v} := (v(x_1), \ldots, v(x_{n-1}))^T)$ und dem Eigenwert $\lambda := \kappa^2$ herzuleiten. Das soll die Helmholz-Gleichung (2), mit Tupeln und Eigenwerten für die Funktionen v_κ bzw. Vorfaktoren κ, approximieren.

Es wird eine Matrix $-A_n$, $n \geq 2$ gesucht, die den Differenzenquotienten $D_h v(x_j)$ auf den oberen Vektor $\vec{v}^{(n)}$ komponentenweise anwendet. Wir rufen in Erinnerung, dass $h = 1/n$ und definieren die naheliegende Matrix

$$-A_n := \frac{1}{h^2}\begin{pmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{pmatrix} \in \mathbb{R}^{(n-1)\times(n-1)}.$$

Wenn nun die Randbedingungen (3) gelten sollen, d.h. $v(x_0), v(x_n) = 0$, leistet diese Matrix A_n tatsächlich das Gewünschte. Sie approximiert die linke Seite der Helmholz-Gleichung (2).

$$A_n \vec{v}^{(n)} = -\frac{1}{h^2} \begin{pmatrix} v(x_0) - 2v(x_1) + v(x_2) \\ v(x_1) - 2v(x_2) + v(x_3) \\ \vdots \\ v(x_{n-3}) - 2v(x_{n-2}) + v(x_{n-1}) \\ v(x_{n-2}) - 2v(x_{n-1}) + v(x_{n-0}) \end{pmatrix} = - \begin{pmatrix} D_h v(x_1) \\ \vdots \\ D_h v(x_{n-1}) \end{pmatrix} \xrightarrow{n\to\infty} -v''$$

Die Matrix A_n wurde in der Funktion `my_numpy_matrix` implementiert.

```
def my_numpy_matrix(n):

    assert n >= 2

    h = 1/n

    a = -2 * np.ones(n-1)
    b = np.ones(n-2)

    A = np.diag(b, -1) + np.diag(a, 0) + np.diag(b, 1)
    A = -A/h**2

    return A
```

Das Eigenwertproblem wurde mit `np.linalg.eig`, für beliebige $n \geq 2$, gelöst. `np.linalg.eig` retourniert die Eigenwerte und Eigenvektoren nicht zwangsläufig, der größe der Eigenwerte nach, sortiert. Nachdem der Zusammenhang zwischen Eigenwert und Eigenvektor nicht verloren gehen soll, erfordert dies einigen logistischen Aufwand. Das ist aber nicht wesentlich und wird daher nicht weiter erklärt. Wir vergleichen lieber die Eigenwerte und Eigenvektoren mit den analytischen Ergebnissen.

In der folgenden Tabelle, sind die 3 Eigenwerte (sollten diese bereits existieren), der Matrizen $A_2, \ldots, A_{10}$, aufgelistet. Diese Tabelle ist analog zu (7) zu verstehen.

	2	3	4	5	6	7	8	9	10
1	8.0	9.0	9.372583	9.54915	9.646171	9.705051	9.743420	9.769795	9.788697
2	NaN	27.0	32.000000	34.54915	36.000000	36.897999	37.490332	37.900800	38.196601
3	NaN	NaN	54.627417	65.45085	72.000000	76.192948	79.016521	81.000000	82.442950

Diese Werte legen diese ein gewisses, laut (6) vielleicht sogar quadratisches, Konvergenzverhalten nahe.

```
n -> inf:
---------

(1 * pi)^2 = 9.869604401089358
(2 * pi)^2 = 39.47841760435743
(3 * pi)^2 = 88.82643960980423
```

Die Matrix A_n besitzt also scheinbar $n-1$ paarweise verschiedene Eigenwerte $\lambda_{1,n} < \cdots < \lambda_{n-1,n}$, welche jeweils gegen $\lambda_i := (i\pi)^2$, $i \in \mathbb{N}$ konvergieren.

$$\begin{array}{ccccccccc} \lambda_{1,2} & \to & \lambda_{1,3} & \to & \cdots & \to & \lambda_{1,n} & \xrightarrow{n\to\infty} & \lambda_1 = (1\pi)^2 \\ & & \lambda_{2,3} & \to & \cdots & \to & \lambda_{2,n} & \xrightarrow{n\to\infty} & \lambda_2 = (2\pi)^2 \\ & & & & \ddots & \ddots & \vdots & \vdots & \vdots \\ & & & & & & \lambda_{i,n} & \xrightarrow[n\to\infty]{} & \lambda_i = (i\pi)^2 \end{array} \tag{7}$$

Nachdem $\left\{\sqrt{\lambda_i} : i \in \mathbb{N}_0\right\} = \pi\mathbb{N}_0 \subsetneq \pi\mathbb{Z}$, könnte man sich nun fragen, wo die andere Hälfte der analytischen Ergebnisse steckt. Die Erklärung ist ganz unspektakulär $\lambda := (\pm\kappa)^2$.

Wir bezeichnen mit $\epsilon_i(n) := |\lambda_i - \lambda_{i,n}|,\ i = 1, \ldots, n-1$ den absoluten Konvergenz-Fehler des i-ten Eigenwertes. In der folgenden Abbildung 1 wurde ϵ_i mit der Vergleich-Geraden id^2, doppelt logarithmisch, geplottet.

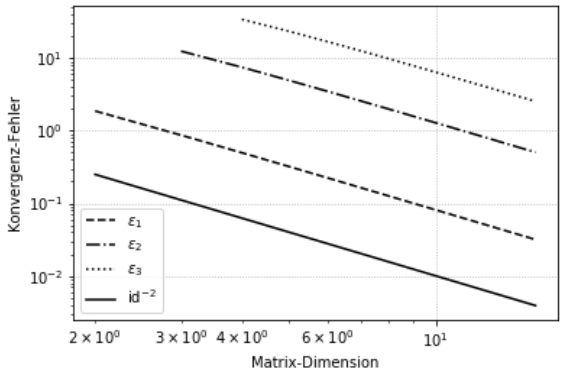
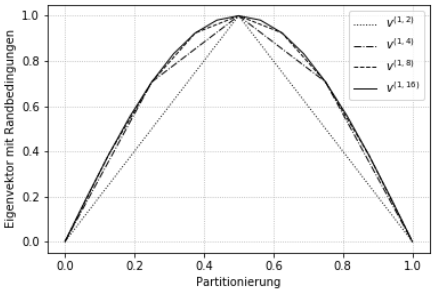


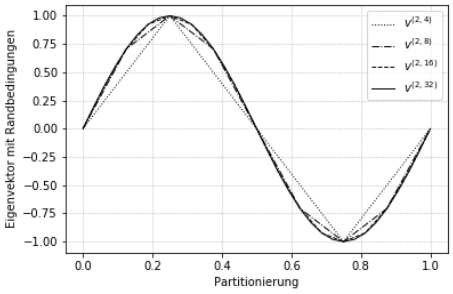
Abbildung 1: Konvergenz-Fehler der Eigenwerte von A_n

Allem Anschein nach, verschwindet ϵ_i quadratisch. Das korreliert mit dem Ergebnis (6). Man beachte, dass der i-te Eigenwert erst ab einer Matrix $A_n,\ n > i$ existiert. Daher fangen die plots von ϵ_i desto später an, je größer i ist. Für größeres i ist auch der intitiale Fehler größer. Obwohl dieser ebenfalls quadratisch konvergiert, werden mehr Rechenoperationen für ein genaues Ergebnis benötigt.

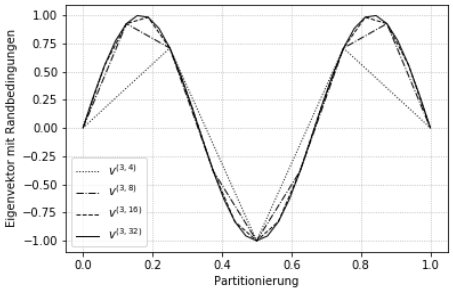
Seien $\vec{v}^{(1,n)}, \ldots, \vec{v}^{(n-1,n)}$ die Eigenvektoren (modulo Vielfache), zu den Eigenwerten $\lambda_{1,n} < \cdots < \lambda_{n-1,n}$, der Matrix A_n. Diese sollten nun gegen die Funktionen v_{κ_i}, $\kappa_i = \sqrt{\lambda_i}$, vielleicht sogar quadratisch, konvergieren. Folgende Abbildungen sollen dies veranschaulichen.



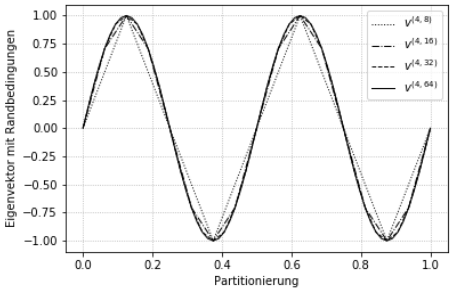
(a) Eigenvektoren $v^{(1,n)}$, $n = 2, 4, 8, 32$



(b) Eigenvektoren $v^{(2,n)}$, $n = 4, 8, 16, 64$



(c) Eigenvektoren $v^{(3,n)}$, $n = 8, 16, 32, 64$



(d) Eigenvektoren $v^{(4,n)}$, $n = 8, 16, 32, 64$

Abbildung 2: Eigenvektoren $v^{(i,n)}$, $i = 1, \ldots, 4$ der Matrizen A_n

Erstaunlicherweise, gibt es scheinbar keinen Konvergenz-Fehler, da die Eigenvektoren direkt an den Grenzfunktionen liegen. Mit anderen Worten, $\forall\, n \in \mathbb{N}, \forall\, i, j = 1, \ldots n-1$:

$$\vec{v}_j^{(i,n)} = v_{\kappa_i}(x_j).$$

Anschaulich, erhält man ein zu (7) analoges Schema.

$$\begin{array}{ccccccccc} \vec{v}^{(1,2)} & \to & \vec{v}^{(1,3)} & \to & \cdots & \to & \vec{v}^{(1,n)} & \xrightarrow[n\to\infty]{} & v_{\kappa_1} \\ & & \vec{v}^{(2,3)} & \to & \cdots & \to & \vec{v}^{(2,n)} & \xrightarrow[n\to\infty]{} & v_{\kappa_2} \\ & & & & \ddots & \ddots & \vdots & \vdots & \vdots \\ & & & & & & \vec{v}^{(i,n)} & \xrightarrow[n\to\infty]{} & v_{\kappa_i} \end{array} \tag{8}$$

1.4 Verallgemeinerte Problembeschreibung

Die Ausbreitungsgeschwindigkeit c in (1) hängt vom Material der Saite ab. Bisher haben wir sie als konstant angenommen, d.h. die Saite bestand aus einem Material. Sei nun für $c_0, c_1 \in \mathbb{R}$

$$c(x) := \begin{cases} c_0, x \in (0, 1/2) \\ c_1, x \in (1/2, 1) \end{cases} . \tag{9}$$

1.5 Verallgemeinerte Analytische Lösung

Zuerst leiten wir eine zur Helmholz-Gleichung (2) ähnliche Gleichung her und geben einen (4) entsprechenden Lösungsansatz an, wenn die Lösung v auf $(0,1)$ stetig differenzierbar sein soll. Dabei betrachten wir eine angepasste Version der Wellengleichung (1).

$$\frac{\partial^2 u}{\partial x^2}(t,x) = \frac{1}{c^2(x)}\frac{\partial^2 u}{\partial t^2}(t,x), \qquad x \in (0,1), \qquad t \in \mathbb{R}$$

Wir verwenden jedoch den selben Ansatz, wie Vorher. Das war $u(x,t) = \Re(v(x)e^{-i\omega t})$, mit einer festen, aber unbekannten Kreisfrequenz $\omega > 0$ und einer Funktion v, welche nur noch vom Ort x abhängt.

Einsetzen und analoges Nachrechnen, gibt, mit der unbekannten Wellenzahl $\kappa(x) := \frac{\omega}{c(x)}$, die Randbedingungen (3) und

$$-v''(x) = \kappa^2(x)v(x), \qquad x \in (0,1).$$

Um Probleme mit der Differenzierbarkeit von κ zu vermeiden, führen wir die Abkürzungen $\kappa_0 := \frac{\omega}{c_0}$, $\kappa_1 := \frac{\omega}{c_1}$ ein, und definieren den Lösungsansatz durch Fallunterscheidung und mit (vorerst) beliebigen Konstanten $C_{01}, C_{02}, C_{11}, C_{12}$.

$$v(x) := \begin{cases} C_{01}\cos(\kappa_0 x) + C_{02}\sin(\kappa_0 x), & x \in (0,1/2) \\ C_{11}\cos(\kappa_1 x) + C_{12}\sin(\kappa_1 x), & x \in (1/2,1) \end{cases}$$

Durch Berücksichtigung der Randbedingungen (3), erhält man (fast analog zu Vorher)

$$C_{01} = 0, \qquad C_{11}\cos\kappa_1 + C_{12}\sin\kappa_1 = 0.$$

Soll v auf $1/2$ stetig fortgesetzt werden, so müssen dessen links- und rechts-seitiger Grenzwert übereinstimmen.

$$C_{02}\sin(\kappa_0/2) = \lim_{x\to 1/2-} v(x) = \lim_{x\to 1/2+} v(x) = C_{11}\cos(\kappa_1/2) + C_{12}\sin(\kappa_1/2)$$

Um stetige Differenzierbarkeit zu erhalten, muss auch die Ableitung

$$v'(x) = \begin{cases} C_{02}\kappa_0\cos(\kappa_0 x), & x \in (0,1/2) \\ -C_{11}\kappa_1\sin(\kappa_1 x) + C_{12}\kappa_1\cos(\kappa_1 x), & x \in (1/2,1) \end{cases}$$

auf $1/2$ stetig fortgesetzt werden.

$$C_{02}\kappa_0\cos(\kappa_0/2) = \lim_{x\to 1/2-} v'(x) = \lim_{x\to 1/2+} v'(x) = -C_{11}\kappa_1\sin(\kappa_1/2) + C_{12}\kappa_1\cos(\kappa_1/2)$$

Aus den Randbedingungen und stetigen Fortsetzungen, ergibt sich also das homogene lineare Gleichungssystem $R\vec{C} = 0$, mit

$$R := \begin{pmatrix} \sin(\kappa_0/2) & -\cos(\kappa_1/2) & -\sin(\kappa_1/2) \\ \kappa_0\cos(\kappa_0/2) & \kappa_1\sin(\kappa_1/2) & -\kappa_1\cos(\kappa_1/2) \\ 0 & \cos\kappa_1 & \sin\kappa_1 \end{pmatrix} \in \mathbb{R}^{3\times 3}, \qquad \vec{C} := \begin{pmatrix} C_{02} \\ C_{11} \\ C_{12} \end{pmatrix} \in \mathbb{R}^{3\times 1}.$$

Sei $R \in \mathrm{GL}_3(\mathbb{R})$ regulär, so ist deren Kern trivial, d.h. $\ker R = \{0\}$, und somit auch die Lösung $\vec{C} = 0$. Dieser Trivialfall wurde jedoch vorhin bereits ausgeschlossen. Darum betrachten wir $\det R = 0$. Mit SymPy berechnet man

$$\begin{aligned}\det R &= \sin\left(\frac{\kappa_0}{2}\right)\cos\left(\frac{\kappa_1}{2}\right)\kappa_1 + \sin\left(\frac{\kappa_1}{2}\right)\cos\left(\frac{\kappa_0}{2}\right)\kappa_0 \\ &= \sin\left(\frac{\omega}{2c_0}\right)\cos\left(\frac{\omega}{2c_1}\right)\frac{\omega}{c_1} + \sin\left(\frac{\omega}{2c_1}\right)\cos\left(\frac{\omega}{2c_0}\right)\frac{\omega}{c_0} =: f_c(\omega) \end{aligned} \tag{10}$$

Für feste c_0, c_1, lässt sich das gewünschte ω, als (nicht eindeutige) Nullstelle dieser Funktion f_c, charakterisieren.

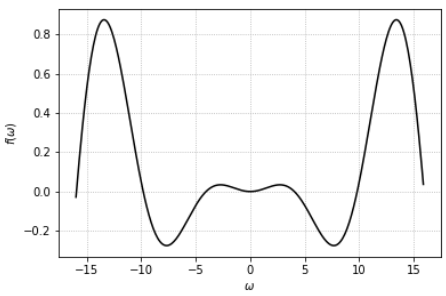
1.6 Verallgemeinerte Semi-Analytische Lösung

Das Ergebnis aus (10) wurde, in Form von folgender Funktion, implementiert.

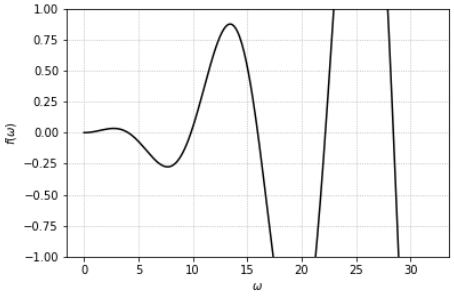
```
# c ... pair of propagation speeds

def get_zero_function(c):

    # allocate some sympy symbols:
    omega = sp.Symbol('\omega')
    kappa = sp.IndexedBase('\kappa')

    # implement the matrix R (properly):
    R = sp.Matrix([[ sp.sin(kappa[0]/2),  kappa[0]*sp.cos(kappa[0]/2), 0],
                   [-sp.cos(kappa[1]/2),  kappa[1]*sp.sin(kappa[1]/2), sp.cos(kappa[1])],
                   [-sp.sin(kappa[1]/2), -kappa[1]*sp.cos(kappa[1]/2), sp.sin(kappa[1])]])
    R = R.T

    # calculate R's determinant (properly):
    det = sp.det(R)
    det = sp.simplify(det)

    # substitute for kappa_0 and kappa_1:
    kappa_0 = omega/c[0]
    kappa_1 = omega/c[1]
    substitution = {kappa[0]: kappa_0, kappa[1]: kappa_1}
    det = det.subs(substitution)

    # transform expression det into proper numpy function:
    zero_function = sp.lambdify(omega, det, 'numpy')

    return zero_function
```

Nun können wir f_c für fixe c_0, c_1 plotten lassen. Dann bekommen wir ein besseres Verständnis dafür, welchen Startwert wir wählen sollen, um die Gleichung $f_c(\omega) = 0$, mit `scipy.optimize.fsolve` lösen zu lassen.

Für die folgenden Plots in Abbildung 3, wurden die arbiträren Werte $c_0 = 100$, $c_1 = 1$ gewählt. Die davon abhängige Funktion f_c ist scheinbar gerade. Das liegt an (10), sowie dass cos gerade und sin ungerade ist.



(a) auf dem Intervall $(-16, 16)$



(b) auf dem Intervall $(0, 64)$ und herangezoomt

Abbildung 3: Plots von f_c für $c_0 = 100$, $c_1 = 1$

Dementsprechend, können passende Startwerte $\tilde{\omega}$ für iterative Verfahren gewählt werden. Die jeweils ersten Ergebnisse ω vom, bereits erwähnten, `scipy.optimize.fsolve` sind in der folgenden Tabelle. Die Quadrate ω^2 dieser Ergebnisse sind Approximationen der Grenzwerte der Eigenwerte des nächsten Unterkapitels.

	1	2	3	4	5	6
$\tilde{\omega}$	5.000000	10.000000	15.000000	20.000000	25.000000	30.000000
ω	4.057425	9.826058	15.956815	22.170349	15.956815	28.413934
ω^2	16.462695	96.551423	254.619961	491.524375	254.619961	807.351663

1.7 Verallgemeinertes Eigenwertproblem

Wir wollen nun den Differenzenquotienten $D_h v(x_j)$ verwenden, um ein verallgemeintertes Eigenwertproblem der Form $A\vec{v} = \lambda B\vec{v}$ mit Matrizen $A, B \in \mathbb{R}^{(n-1)\times(n-1)}$ herzuleiten.

Sei abermals $x_j := jh$, $j = 0, \ldots, n$ unsere Zerlegung des Intervalls $[0, 1]$ mit äquidistanter Schrittweite $h = 1/n$. Die Matrix $-A_n$, für den Differenzenquotienten $D_h v(x_j)$, und der Vektor $\vec{v}^{(n)} := (v(x_1), \ldots, v(x_{n-1}))^T$, bleiben ebenfalls nach wie vor so, wie sie waren.

$B_n\lambda$ soll nun, analog zu Vorher, κ^2 repräsentieren. Diesmal, ist κ jedoch als (stückweise konstante) Funktion zu verstehen. Also wird die Matrix B_n deren Fallunterscheidungen übernehmen und λ konstant bleiben. Es läuft darauf hinaus, dass

$$B_n := \begin{cases} \operatorname{diag}^{-2}(c_0, \ldots, c_0, c_1, \ldots, c_1,), & n-1 \in 2\mathbb{N} \\ \operatorname{diag}^{-2}(c_0, \ldots, c_0, \frac{c_0+c_1}{2}, c_1, \ldots, c_1,), & n-1 \in 2\mathbb{N}+1 \end{cases}, \qquad \lambda := \omega^2,$$

wobei c_0, c_1 in $B_n \in \mathrm{GL}_{n-1}(\mathbb{R})$ jeweils $\left\lfloor \frac{n-1}{2} \right\rfloor$-mal vorkommen. Dabei sei vorausgesetzt, dass $c_0, c_1 \neq 0$ und $c_0 + c_1 \neq 0$. Die Wahl von B_n lässt sich wie folgt begründen.

Seien $\vec{a}, \vec{b}$ Vektoren mit gleich vielen Komponenten. Dann ist die Matrix-Vektor-Multiplikation $\cdot$, mit einer erzeugten Diagonalmatrix, äquivalent zur komponentenweisen Multiplikaiton $\odot$.

$$\operatorname{diag}(\vec{a}) \cdot \vec{b} = \vec{a} \odot \vec{b} = \vec{b} \odot \vec{a} = \operatorname{diag}(\vec{b}) \cdot \vec{a} \tag{11}$$

Bei dem vorherigen Eigenwertproblem wäre B_n als Einheits-Matrix I_n zu interpretieren. Der Eigenwert λ konnte gleich ganz κ^2 approximieren, weil dieser Wert konstant war. Man hätte aber freilich auch mit der Skalarmatrix $(I_n c)^{-2}$ und ω^2 anstelle von κ^2 arbeiten können. Da κ^2 nun aber, als Funktion, zwei unterschiedliche Werte

$$\left(\frac{\omega}{c_0}\right)^2, \left(\frac{\omega}{c_1}\right)^2$$

annehmen kann, müssen wir die obere Eigenschaft (11) von Diagonalmatrizen ausnutzen. Damit realisieren wir die Fallunterscheidung zwischen $x_j < 1/2$ und $x_j > 1/2$. Für $x_j = 1/2$, was genau bei $n - 1 \in 2\mathbb{N}$ auftritt, wird gemittelt.

Nachdem die Inverse einer Diagonalmatrix genau die Matrix selbst mit komponentenweise Kehrwerten ist, lassen sich gleich B_n und B_n^{-1} leicht implementieren. Die zuständigen Funktionen besitzen die kreativen Namen `my_other_numpy_matrix` bzw. `my_other_numpy_matrix_inverse`.

```
def my_other_numpy_matrix_inverse(n, c):

    times = np.floor((n-1)/2)
    times = int(times)

    lower = [c[0]]*times
    upper = [c[1]]*times

    middle = [(c[0] + c[1])/2]

    if n%2 != 0:
        B_inverse = np.diag(lower + upper)**2
        return B_inverse
    else:
        B_inverse = np.diag(lower + middle + upper)**2
        return B_inverse

def my_other_numpy_matrix(n, c):
    return 1/my_other_numpy_matrix_inverse(n, c)
```

Das verallgemeinerte Eigenwertproblem $A_n\vec{v} = \lambda B_n\vec{v}$ werden wir zunächst auf $B_n^{-1}A_n\vec{v} = \lambda\vec{v}$ umformulieren. Dieses kann nun ebenfalls mit `np.linalg.eig`, für beliebige $n \geq 2$, gelöst werden.

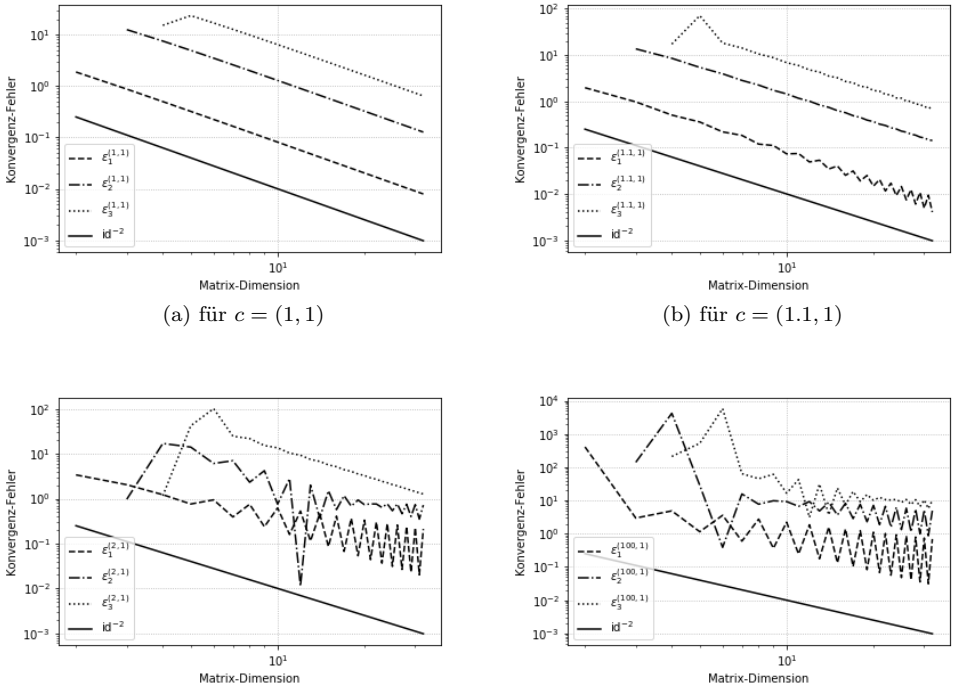
Die Matrix $B_n^{-1}A_n$ besitzt hoffentilch wieder $n - 1$ paarweise verschiedene Eigenwerte $\lambda_{1,n}^c < \cdots < \lambda_{n-1,n}^c$, die jeweils konvergieren.

Um einen ersten Eindruck des möglichen Konvergenz-Verhaltens zu bekommen, vergleichen wir die Eigenwerte mit den oberen semi-analytischen Ergebnissen mit $c_0 = 100$, $c_1 = 1$. Es folgt eine zur oberen analoge Tabelle mit Eigenwerten. Die Ergebnisse lassen zwar zu wünschen übrig, aber immerhin pendelt sich die Größenordnung rasch ein.

	2	3	4	5	6
1	20402.0	13.499662	21.329378	15.313793	20.048572
2	NaN	180004.500338	56813.374144	68.017688	96.940091
3	NaN	NaN	344805.296478	250012.501563	102552.962302
4	NaN	NaN	NaN	750004.166956	422576.319931

Wir bezeichnen (optimistischerweise) mit $\epsilon_i^c(n) := |\lambda_i^c - \lambda_{i,n}^c|$, $i = 1, \ldots, n-1$ den absoluten Konvergenz-Fehler des i-ten Eigenwertes. λ_i^c erhalten wir durch ω^2 von `scipy.optimize.fsolve`, wobei $\tilde{\omega} := \sqrt{\lambda_{i,n}^c}$

als Startwert für das iterative Verfahren gewählt wird. Theoretisch hängt λ_i^c also noch von n ab. In der folgenden Abbildung 4 wurde ϵ_i^c mit der Vergleich-Geraden id^2, doppelt logarithmisch, geplottet.



(a) für $c = (1, 1)$ (b) für $c = (1.1, 1)$

(c) für $c = (2, 1)$ (d) für $c = (100, 1)$

Abbildung 4: Konvergenz-Fehler der Eigenwerte von $B_n^{-1}A_n$

Es macht Sinn, dass Abbildung 4a mit Abbildung 1 korreliert. Diese „Bergsteiger-Konvergenz“ steigt anscheinend mit dem Verhältnis c_0/c_1 (no pun intended). Auch das macht einigermaßen Sinn, aber die Oszillation pendelt sich interessanterweise ein und wird nicht drastisch stärker. Die Abbildung 4 bleibt übrigens unverändert, wenn man die Komponenten von c vertauscht.

Seien $\vec{v}^{(1,n),c}, \dots, \vec{v}^{(n-1,n),c}$ die Eigenvektoren (modulo Vielfache), zu den Eigenwerten $\lambda_{1,n}^c < \cdots < \lambda_{n-1,n}^c$, der Matrix $B_n^{-1}A_n$. Wir antizipieren ein weniger schönes Konvergenz-Verhalten, als das der Eigenvektoren der Matrizen $(A_n)_{n\in\mathbb{N}}$. Nichts desto trotz, wurden die Eigenvektoren $\vec{v}^{(i,n_i),c}$, normiert bzgl. $\|\cdot\|_\infty$, geplottet, wobei

$$\begin{aligned} c_{\max} &= 4, & c &\in \{(c_0, c_1) : c_0, c_1 = 1, \dots, c_{\max}\}, \\ i &= 1, \dots, 2c_{\max}, & n_i &\in \left\{2^{p_{\min}+p_{\mathrm{add}}} : p_{\min} = \lceil \log_2(i+1) \rceil,\ p_{\mathrm{add}} = 0, \dots, 3\right\} =: N_i. \end{aligned}$$

Dabei wurden die Vektoren $\left\{\vec{v}^{(i,n),c} : n \in N_i\right\}$ jeweils zu einem Bild zusammengefasst. Nachdem wir dadurch auf 128 Bilder kommen, werden wir nicht alle herzeigen. Außerdem, sieht nur ein Bruchteil davon „schön“ aus.

To be continued!!!