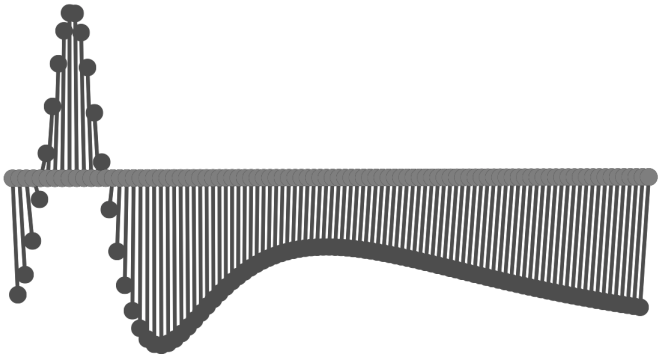
BERGISCHE UNIVERSITÄT WUPPERTAL

Fakultät für Mathematik und Naturwissenschaften
Fachgruppe Physik

Bachelorarbeit mit dem Thema

Gekoppelte Pendel und die Sine-Gordon-Gleichung



Lea Otterbeck (1422800)
Bachelor of Applied Science
Physik und Mathematik

Erstgutachter: Priv.-Doz. Dr. M. Karbach
Zweitgutachter: Prof. Dr. Francesco Knechtli
Datum: 19. April 2018

Inhaltsverzeichnis

1. Einleitung

In dieser Arbeit wird ein System einer gekoppelten Pendelkette mathematisch beschrieben und simuliert sowie der Zusammenhang mit der Sine-Gordon-Gleichung untersucht. Dazu finden sich im Internet zahlreiche Videos, in denen eine solche Kette experimentell realisiert wird (z.B. in [1] und [2]).

Zunächst wird das System endlich vieler gekoppelter Pendel vorgestellt und die Bewegungsgleichungen hergeleitet. Danach werden drei Methoden zur numerischen Lösung dieser vorgestellt, darunter eine in Python eingebundene Fortran-Routine. Diese wird zum Vergleich für eine zeitdiskretisierte Bewegungsgleichung und ein iteratives Verfahren verwendet.

Anschließend wird der Übergang zu unendlich vielen Pendeln betrachtet und Lösungen der Sine-Gordon-Gleichung vorgestellt. In das System werden verschiedene Anfangsbedingungen gegeben und die verschiedenen Methoden miteinander verglichen. Unter Anderem werden Solitonenlösungen als Anfangsbedingungen übergeben und beobachtet, wie sich diese mit der Zeit entwickeln. Dafür wird ein Python-Programm geschrieben, welches außerdem die Pendel in drei Dimensionen animiert.

2. System endlich vieler gekoppelter Pendel

Im Folgenden wird ein System einer harmonisch gekoppelten Pendelkette und Kreispendelkette betrachtet und der Lagrangian aufgestellt. Aus den Euler-Lagrange-Gleichungen resultiert die Bewegungsgleichung der Pendel zweiter Ordnung. Diese wird auf ein System erster Ordnung reduziert und einer Python-Routine zur Berechnung übergeben. Als Alternative wird die Bewegungsgleichung in der Zeit diskretisiert und die Winkel für die gegebenen Zeiten explizit berechnet. Zuletzt wird ein iteratives Verfahren mit Hilfe eines Ansatzes angewendet.

2.1. Bewegungsgleichung der endlichen Pendelkette

Endlich viele Pendel einer Pendelkette seien über eine Schraubenachse mit Federn an der Aufhängung $z = 0$ und $y = 0$ entlang der x-Achse harmonisch gekoppelt und gleichmäßig verteilt. Wenn d die Gesamtlänge der Pendelkette ist, dann ist der Abstand zwischen zwei Pendeln bei N Pendeln d/N und $x_n = nd/N \quad \forall n = 1, ..., N$ den x-Wert des Aufhängungspunktes des n-ten Pendels beschreibt. Die Pendelmassen seien alle gleich schwer, also $m_n = m \quad \forall n = 1, ..., N$. Die Parametrisierung der Pendelkette mit Pendeln der Länge $l > 0$ lautet:

$$\vec{r}_n = \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} \to \begin{pmatrix} x_n \\ l \sin \varphi_n \\ -l \cos \varphi_n \end{pmatrix}, \tag{2.1}$$

wobei φ der Winkel des Pendels zur Ruhelage ist. Für die kinetische Energie ergibt sich:

$$T = \sum_{n=1}^{N} \frac{m}{2} \dot{\vec{r}}_n^2 = \frac{m}{2} \sum_{n=1}^{N} (\dot{x}_n^2 + \dot{\varphi}_n^2 l^2). \tag{2.2}$$

Die Gravitation wirke entlang der z-Achse mit Ortsfaktor g. Für das Gravitationspotential gilt dann:

$$U_G = \sum_{n=1}^{N} m g z_n = -mgl \sum_{n=1}^{N} \cos \varphi_n. \tag{2.3}$$

Die Federkonstante sei $k > 0$ und die Ruhelänge L so gewählt, dass $L = d/N$. Dann folgt für das gesamte Federpotential aller Federn:

$$\begin{aligned} U_F =& \frac{k}{2} \sum_{n=1}^{N-1} (L - (x_{n+1} - x_n))^2, \\ =& \frac{k}{2} \sum_{n=1}^{N-1} (L - (\underbrace{(n+1)\frac{d}{N} - n\frac{d}{N}}_{=\frac{d}{N}=L} + \alpha\varphi_{n+1} - \alpha\varphi_n))^2, \\ =& \frac{k}{2} \sum_{n=1}^{N-1} \alpha^2 (\varphi_{n+1} - \varphi_n)^2, \end{aligned} \tag{2.4}$$

wobei durch die Drehung φ der Pendel und damit Streckung oder Stauchung der Feder eine Verschiebung in x-Richtung $x_n = \alpha\varphi_n + n\frac{d}{N}$, wie bei einer Schraube, mit Kopplungsfaktor $\alpha > 0$ stattfindet. Für den Lagrangian $\mathcal{L} = T - U$, wobei $U = U_G + U_F$ ist, ergibt sich $\forall\, n = 1, ..., N$

$$\begin{aligned}\mathcal{L} &= \frac{m}{2}\sum_{n=1}^{N}(\dot{x}_n^2 + \dot{\varphi}_n^2 l^2) + mgl\sum_{n=1}^{N}\cos\varphi_n - \frac{k}{2}\sum_{n=1}^{N-1}\alpha^2(\varphi_{n+1} - \varphi_n)^2,\\ &= \sum_{n=1}^{N}\left[\frac{m}{2}(\alpha^2 + l^2)\dot{\varphi_n}^2 + mgl\cos\varphi_n - \frac{k}{2}\alpha^2(\varphi_{n+1} - \varphi_n)^2\right]\\ &\quad + \frac{k}{2}\alpha^2(\varphi_{N+1} - \varphi_N)^2. \end{aligned} \tag{2.5}$$

Damit folgt für die Euler-Lagrange-Gleichungen $\forall n = 2, ..., N-1$:

$$\frac{\mathrm{d}}{\mathrm{dt}}\frac{\partial\mathcal{L}}{\partial\dot{\varphi}_n} = m(\alpha^2 + l^2)\ddot{\varphi}_n, \tag{2.6}$$

$$\frac{\partial\mathcal{L}}{\partial\varphi_n} = -mgl\sin\varphi_n + k\alpha^2(\varphi_{n+1} - 2\varphi_n + \varphi_{n-1}). \tag{2.7}$$

Mit Reibung lassen sich die Euler-Lagrange-Gleichungen folgendermaßen erweitern:

$$\frac{\mathrm{d}}{\mathrm{dt}}\frac{\partial\mathcal{L}}{\partial\dot{\varphi}_n} - \frac{\partial\mathcal{L}}{\partial\varphi_n} = -\gamma|\vec{\dot{r}}_n|^R\left|\frac{\partial\vec{\dot{r}}_n}{\partial\dot{\varphi}}\right|\frac{\dot{\varphi}_n}{|\dot{\varphi}_n|} = -\gamma(\alpha^2 + l^2)^{\frac{R+1}{2}}|\dot{\varphi}_n|^R\frac{\dot{\varphi}_n}{|\dot{\varphi}_n|}. \tag{2.8}$$

Daraus lässt sich die Bewegungsgleichung für φ_n $\forall n = 2, ..., N-1$ für endlich viele Pendel berechnen:

$$\begin{aligned}\ddot{\varphi}_n =& -\frac{g}{l}\frac{1}{(\frac{\alpha}{l})^2 + 1}\sin\varphi_n + \frac{k}{m}\frac{(\frac{\alpha}{l})^2}{(\frac{\alpha}{l})^2 + 1}(\varphi_{n+1} - 2\varphi_n + \varphi_{n-1})\\ &- \frac{\gamma}{m}(\alpha^2 + l^2)^{\frac{R-1}{2}}|\dot{\varphi}_n|^R\frac{\dot{\varphi}_n}{|\dot{\varphi}_n|},\\ =& -\Omega_g^2\sin\varphi_n + \Omega_k^2(\varphi_{n+1} - 2\varphi_n + \varphi_{n-1}) - \Gamma(\alpha^2 + l^2)^{\frac{R-1}{2}}|\dot{\varphi}_n|^R\frac{\dot{\varphi}_n}{|\dot{\varphi}_n|}, \end{aligned} \tag{2.9}$$

mit $\sigma = \alpha/l$, $\Gamma = \gamma/m$ und

$$\begin{aligned}\Omega_g^2 =& \omega_g^2\frac{1}{\sigma^2 + 1},\\ \Omega_k^2 =& \omega_k^2\frac{\sigma^2}{\sigma^2 + 1},\end{aligned}$$

wobei $\omega_g = \sqrt{g/l}$ und $\omega_k = \sqrt{k/m}$ die Schwingungsfrequenzen sind.Für die Ränder ergibt sich aus den erweiterten Euler-Lagrange-Gleichungen:

$$\ddot{\varphi}_1 = \Omega_g^2\sin\varphi_1 + \Omega_k^2(\varphi_2 - \varphi_1) - \Gamma(\alpha^2 + l^2)^{\frac{R-1}{2}}|\dot{\varphi}_1|^R\frac{\dot{\varphi}_1}{|\dot{\varphi}_1|}, \tag{2.10a}$$

$$\ddot{\varphi}_N = \Omega_g^2\sin\varphi_N + \Omega_k^2(\varphi_{N-1} - \varphi_N) - \Gamma(\alpha^2 + l^2)^{\frac{R-1}{2}}|\dot{\varphi}_N|^R\frac{\dot{\varphi}_N}{|\dot{\varphi}_N|}. \tag{2.10b}$$

Zur besseren Kontrolle der Parameter wird die Zeit skaliert. Dazu wird eine neue Zeitvariable

$$\tau = \Omega_g t \tag{2.10c}$$

eingeführt. Eine Zeiteinheit von τ hängt durch die Relation

$$\tau = \Omega_g\, t = \omega_g \sqrt{\frac{1}{\sigma^2+1}}\, t = \sqrt{\frac{1}{\sigma^2+1}} \frac{2\pi}{T}\, t \leq \frac{2\pi}{T}\, t \quad \text{mit} \quad \sqrt{\frac{1}{\sigma^2+1}} \leq 1 \tag{2.10d}$$

von der Schwingungsdauer $T = 2\pi/\omega_g$ ab. Für kleine σ ist τ somit die Zeit in Einheiten der Schwingungsdauer T durch

$$\tau \simeq \frac{2\pi}{T}\, t \tag{2.10e}$$

gegeben. Gleichung (2.9) kann damit zu

$$\ddot{\varphi}_n = -\eta_{nk}\varphi_k - \sin\varphi_n - \Lambda\dot{\varphi}_n \tag{2.11}$$

transformiert werden, wobei η eine symmetrische tridiagonale Matrix der Dimension $(N \times N)$ ist mit

$$\eta = -\left(\frac{\Omega_k}{\Omega_g}\right)^2 \begin{pmatrix} -1 & 1 & 0 & & & & \dots & & 0 \\ 1 & -2 & 1 & 0 & & & \dots & & 0 \\ 0 & 1 & -2 & 1 & 0 & & \dots & & 0 \\ & & & & & & & & \\ \vdots & & & \ddots & \ddots & \ddots & & & \vdots \\ & & & & & & & & \\ 0 & & \dots & & 0 & 1 & -2 & 1 & 0 \\ 0 & & \dots & & & 0 & 1 & -2 & 1 \\ 0 & & \dots & & & & 0 & 1 & -1 \end{pmatrix} \in (N \times N). \tag{2.12}$$

Mit

$$\Lambda = \frac{\Gamma}{\Omega_g^{2-R}} (\alpha^2 + l^2)^{\frac{R-1}{2}} |\dot{\varphi}_n|^{R-1} \tag{2.13}$$

wird der Reibungskoeffizient bezeichnet. Er hängt von der Reibungsart R ab und verschwindet im reibungsfreien Fall. Für Stokesche Reibung beträgt $R = 1$ und für Newtonsche Reibung $R = 2$.

2.2. Bewegungsgleichung der Kreispendelkette

Zur Simulierung einer Kreispendelkette müssen die bisherigen Berechnungen geringfügig angepasst werden. Die Pendel seien entlang eines Rings mit Radius r und Länge $S = 2\pi r$ in der xy-Ebene gleichmäßig verteilt. Der Winkel β_n gibt hierbei den Ort auf der Kreiskette des n-ten Pendels in der xyEbene an. Dann lautet die Parametrisierung:

$$\vec{r}_n = \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} \rightarrow \begin{pmatrix} (r + l\sin\varphi_n)\sin(\beta_n) \\ -(r + l\sin\varphi_n)\cos(\beta_n) \\ -l\cos(\varphi_n) \end{pmatrix}, \tag{2.14}$$

wobei der Winkel in Abhängigkeit der Kreisbogenlänge bis zum n-ten Pendel s_n dargestellt werden kann als:

$$\beta_n = \frac{s_n}{r}. \tag{2.15}$$

Die Pendel seien in je einem Abstand von $\frac{2\pi r}{N}$ auf dem Kreis verteilt, also $\beta_n = n\frac{2\pi}{N}$. Auch hier beschreibt α die Kopplung zwischen dem Winkel und der Verschiebung in s-Richtung. Aus $s_n = \alpha\varphi_n + n\frac{2\pi r}{N}$ und $s_n = r\beta_n$ folgt für den Winkel β unter Berücksichtigung der Kopplung α:

$$\beta_n = \frac{\alpha}{r}\varphi_n + n\frac{2\pi}{N}. \tag{2.16}$$

Durch die periodischen Randbedingungen gilt:

$$\varphi_{N+1} = \varphi_1. \tag{2.17}$$

Womit sich das Gesamtfederpotential bei Wahl der Ruhelänge $L = \frac{2\pi r}{N}$ zu

$$\begin{aligned} U_F =& \frac{k}{2}\sum_{n=1}^{N}(L - (s_{n+1} - s_n))^2, \\ =& \frac{k}{2}\sum_{n=1}^{N}(L - (\alpha(\varphi_{n+1} - \varphi_n) + \underbrace{(n+1)\frac{2\pi r}{N} - n\frac{2\pi r}{N}}_{=\frac{2\pi r}{N}=L}))^2, \\ =& \frac{k}{2}\sum_{n=1}^{N}\alpha^2(\varphi_{n+1} - \varphi_n)^2 \end{aligned} \tag{2.18}$$

verändert. Mit der kinetischen Energie T (für große Radien $r > l$)

$$\begin{aligned} T =& \frac{m}{2}\sum_{n=1}^{N}\dot{\vec{r}}_n^2, \\ =& \frac{m}{2}\sum_{n=1}^{N}(l^2\dot{\varphi}_n^2\cos^2\varphi_n + (r + l\sin\varphi_n)^2\cos^2\beta_n\dot{\beta}_n^2 \\ &+ (r + l\sin^2\varphi_n)^2\sin^2\beta_n\dot{\beta}_n^2 + l^2\dot{\varphi}_n^2\sin^2\varphi_n), \\ =& \frac{m}{2}\sum_{n=1}^{N}\dot{\varphi}_n^2(l^2 + \alpha^2(1 + \frac{l}{r}\sin\varphi_n)^2) \doteq \frac{m}{2}\sum_{n=1}^{N}\dot{\varphi}_n^2(l^2 + \alpha^2) + \mathcal{O}(\alpha^2\frac{l}{r}), \end{aligned} \tag{2.19}$$

und dem Gravitationspotential U_G aus Gleichung (2.3) ergibt sich der Lagrangian

$$\mathcal{L} = \frac{m}{2}\sum_{n=1}^{N}(\alpha^2 + l^2)\dot{\varphi}_n^2 + mgl\sum_{n=1}^{N}\cos\varphi_n - \frac{k}{2}\sum_{n=1}^{N}\alpha^2(\varphi_{n+1} - \varphi_n)^2, \tag{2.20}$$

welcher, bis auf den zusätzlichen Summanden außerhalb der Summe, gleich dem Lagrangian aus Gleichung (2.5) ist. Die Bewegungsgleichung für die Kreispendelkette entspricht also $\forall\, n = 2, ..., N-1$, der der Pendelkette. Lediglich die Bewegungsgleichung der Pendel

verändert sich minimal durch die periodischen Randbedingungen. Für die Ränder wird Gleichung (2.10a) ergänzt:

$$\ddot{\varphi}_1 = \Omega_g^2 \sin\varphi_1 + \Omega_k^2(\varphi_2 - 2\varphi_1 + \varphi_N) - \Gamma(\alpha^2 + l^2)^{\frac{R-1}{2}} |\dot{\varphi}_1|^R \frac{\dot{\varphi}_1}{|\dot{\varphi}_1|}, \tag{2.21}$$

$$\ddot{\varphi}_N = \Omega_g^2 \sin\varphi_N + \Omega_k^2(\varphi_{N-1} - 2\varphi_N + \varphi_1) - \Gamma(\alpha^2 + l^2)^{\frac{R-1}{2}} |\dot{\varphi}_N|^R \frac{\dot{\varphi}_N}{|\dot{\varphi}_N|}. \tag{2.22}$$

Wieder wird die Zeit gemäß Gleichung (2.10c) skaliert und analog zu Gleichung (2.11) lautet die Bewegungsgleichung:

$$\ddot{\varphi}_n = -\eta^K_{nk}\varphi_k - \sin\varphi_n - \Lambda\dot{\varphi}_n, \tag{2.23}$$

wobei η^K genauso wie η eine symmetrische tridiagonale Matrix der Dimension $(N \times N)$ ist

$$\eta^K = -\left(\frac{\Omega_k}{\Omega_g}\right)^2 \begin{pmatrix} -2 & 1 & 0 & & & & \dots & 0 & 1 \\ 1 & -2 & 1 & 0 & & & \dots & & 0 \\ 0 & 1 & -2 & 1 & 0 & & \dots & & 0 \\ & & & & & & & & \\ \vdots & & & \ddots & \ddots & \ddots & & & \vdots \\ & & & & & & & & \\ 0 & & \dots & & 0 & 1 & -2 & 1 & 0 \\ 0 & & \dots & & & 0 & 1 & -2 & 1 \\ 1 & & \dots & & & & 0 & 1 & -2 \end{pmatrix} \in (N \times N) \tag{2.24}$$

In η^K werden die periodischen Randbedingungen berücksichtigt und ausgehend von der Bewegungsgleichung der Pendelkette ergibt sich mit $\eta \to \eta^K$ die Bewegungsgleichung der Kreispendelkette.

3. Numerische Methoden zur Lösung der Bewegunsgleichungen

Im Folgenden werden drei verschiedene numerische Methoden zur Lösung der Bewegungsgleichung vorgestellt. In jeder numerischen Berechnung ist eine Diskretisierung der Zeit $\tau \in [0, \tau_{end}]$ in M äquidistante Zeitintervalle $\Delta\tau = \tau_{end}/M$, $M \in \mathbb{N}$ nötig. Die diskretisierte Zeit ist dann gegeben durch

$$\tau_i = i\Delta\tau, \quad i = 0, ..., M. \tag{3.1}$$

Der Winkel zur Zeit τ_i wird im Folgenden durch einen oberen Index (i) gekennzeichnet $\varphi(\tau_i) \to \varphi^{(i)}$.

3.1. Reduzierung der Bewegungsgleichung endlicher Pendel zu einer Differentialgleichung erster Ordnung

Die Bewegungsgleichung (2.9) der endlichen Pendel ist ein System inhomogener Differentialgleichungen zweiter Ordnung. Für eine numerische Berechnung von φ mit Hilfe einer vorgefertigten Routine muss diese zunächst auf ein Differentialgleichungssystem erster Ordnung reduziert werden. Dazu führen wir neue Variablen φ_{N+1} bis φ_{2N} ein:

$$\begin{array}{rcl} \varphi_{N+1} &=& \dot{\varphi}_1 \\ &\vdots& \\ \varphi_{2N} &=& \dot{\varphi}_N \end{array} \quad \Rightarrow \quad \begin{pmatrix} \dot{\varphi}_1 \\ \vdots \\ \dot{\varphi}_N \\ \dot{\varphi}_{N+1} \\ \vdots \\ \dot{\varphi}_{2N} \end{pmatrix} = \begin{pmatrix} \varphi_{N+1} \\ \vdots \\ \varphi_{2N} \\ \ddot{\varphi}_1 \\ \vdots \\ \ddot{\varphi}_N \end{pmatrix}.$$

Für die ersten N Ableitungen von φ ergeben sich also die neuen Variablen φ_{N+1} bis φ_{2N}, deren Ableitungen wiederum von den Bewegungsgleichungen von φ_1 bis φ_N aus Gleichung (2.11) abhängen. Damit ergibt sich $\forall n = 1, ..., N$

$$\dot{\varphi}_n = \varphi_{n+N} \tag{3.2a}$$

$$\begin{aligned} \dot{\varphi}_{n+N} &= -\sin\varphi_n - \eta_{nk}\varphi_k - \Lambda\dot{\varphi}_n, \\ &= -\sin\varphi_n - \eta_{nk}\varphi_k - \Lambda\varphi_{n+N}. \end{aligned} \tag{3.2b}$$

Das reduzierte Differentialgleichungssystem ist nichtlinear und gekoppelt. Zur Lösung wird eine Python-Routine vorgestellt.

3.1.1. Python-Routine

Die verwendete Python-Routine `integrate.ode()` aus der Bibliothek `scipy` greift auf das odepack aus der Fortran Bibliothek zurück. Odepack ist eine Sammlung aus Fortran Lösungsmethoden mit Anfangswertproblemen für gewöhnliche Differentialgleichungssysteme, welche in [5] beschrieben werden. Python nutzt daraus die Methode Lsoda zur Lösung von Differentialgleichungssystemen erster Ordnung. Lsoda löst Systeme der Form $\frac{\mathrm{d}y}{\mathrm{d}t} = f(y, t)$ und wechselt dabei, abhängig von dem Verhalten des Problems, automatisch zwischen steifen (BDF) und nicht-steifen (Adams) Lösungsmethoden. BDF steht für Backward Differentiation Formulas (eng.) und ist ein implizites Mehrschrittverfahren.

3.2. Zeitdiskretisierung der Bewegungsgleichung

Eine Möglichkeit zur numerischen Lösung der Differentialgleichung aus Gleichung (2.11) besteht darin, diese in der Zeit zu diskretisieren. Um die Bewegungsgleichung zu diskretisieren wird für die zweite Ableitung zur Zeit τ_i der Differenzenquotient zweiter Ordnung

$$\ddot{\varphi}_{n,l}^{(i)} = \frac{\varphi_{n,l}^{(i+1)} - 2\varphi_{n,l}^{(i)} + \varphi_{n,l}^{(i-1)}}{(\Delta\tau)^2} + \mathcal{O}((\Delta\tau)^2) \tag{3.3}$$

und für die erste Ableitung von φ zur Zeit τ_i der symmetrische Differenzenquotient

$$\dot{\varphi}_{n}^{(i)} = \frac{\varphi_{n}^{(i+1)} - \varphi_{n}^{(i-1)}}{2\Delta\tau} + \mathcal{O}((\Delta\tau)^2) \tag{3.4}$$

eingesetzt. Mit dieser Näherung folgt eine einzige Berechnung für jeden Zeitschritt und $\forall\, i \geq 1$ gilt

$$\varphi_n^{(i+1)} = 2\varphi_n^{(i)} - \varphi_n^{(i-1)} - \Delta\tau^2 \left[\eta_{nk}\varphi_k^{(i)} + \sin\varphi_n^{(i)} + \Lambda \frac{\varphi_n^{(i+1)} - \varphi_n^{(i-1)}}{2\Delta\tau} \right]. \tag{3.5}$$

Zur Berechnung von $\varphi_n^{(1)}$ wird φ_n zur virtuellen Zeit $\tau_{-1} = -\Delta\tau$ benötigt. Mit Hilfe des symmetrischen Differenzenquotienten 1. Ordnung

$$\dot{\varphi}_{n}^{0} = \frac{\varphi_{n}^{(1)} - \varphi_{n}^{(-1)}}{2\Delta\tau} + \mathcal{O}((\Delta\tau)^2) \tag{3.6}$$

und $\dot{\varphi}_n^0$ als Anfangsbedingung gilt

$$\varphi_n^{(1)} = 2\varphi_n^0 + \Delta\tau\dot{\varphi}_n^0 - \varphi_n^{(1)} - \Delta\tau^2 \left[\eta_{nk}\varphi_k^{(i)} + \sin\varphi_n^0 + \Lambda\dot{\varphi}_n^0 \right]. \tag{3.7}$$

Umstellen nach $\varphi_n^{(1)}$ liefert φ_n zur Zeit τ_1. Zusammen mit der Anfangsauslenkung φ_n^0 als Anfangsbedingung berrechnen sich die Winkel φ für alle Zeiten mit

$$\varphi_n^{(i+1)} = \begin{cases} \varphi_n^0, & i = -1, \\ \varphi_n^0 + \Delta\tau\dot{\varphi}_n^0 - \dfrac{\Delta\tau^2}{2}\left[\eta_{nk}\varphi_k^{(0)} + \sin\varphi_n^{(0)} + \Lambda\dot{\varphi}_n^0 \right], & i = 0, \\ 2\varphi_n^{(i)} - \varphi_n^{(i-1)} - \Delta\tau^2 \left[\eta_{nk}\varphi_k^{(i)} + \sin\varphi_n^{(i)} + \Lambda \dfrac{\varphi_n^{(i+1)} - \varphi_n^{(i-1)}}{2\Delta\tau} \right], & \forall i \geq 1\ . \end{cases}$$

Da auf der rechten Seite durch die Reibung $\forall\ i \geq 1$ ein weiteres φ zur Zeit τ_{i+1} steht hängt die Auflösung der Bewegungsgleichung von der Reibung ab. Die Methode wird im Folgenden für den Fall ohne und mit Stokescher Reibung $\forall\ i \geq 1$ betrachtet. Für die Zeiten τ_0 und τ_1 verändert sich nur der Faktor Λ, siehe Gleichung (2.13).

Reibungsfreier Fall

Im reibungsfreien Fall ist $\Gamma = 0$ und damit $\Lambda = 0$. Daraus resultiert die Bewegungsgleichung

$$\varphi_n^{(i+1)} = 2\varphi_n^{(i)} - \varphi_n^{(i-1)} + (\Delta\tau)^2(-\eta_{nk}\varphi_k^{(i)} - \sin\varphi_n^{(i)}) \tag{3.9}$$

ohne Reibung $\forall\ i \geq 1$.

Stokesche Reibung

Für die Stokesche Reibung $R = 1$ ist der Reibungsfaktor Λ

$$\Lambda\,|_{R=1} = \frac{\Gamma}{\Omega_g}$$

und damit gilt

$$\varphi_n^{(i+1)} = 2\varphi_n^{(i)} - \varphi_n^{(i-1)} + \Delta\tau^2 \left[-\eta_{nk}\varphi_k^{(i)} - \sin\varphi_n^{(i)} - \Lambda|_{R=1} \left(\frac{\varphi_n^{(i+1)} - \varphi_n^{(i-1)}}{2\Delta\tau} \right) \right].$$

Umstellen nach $\varphi^{(i+1)}$ liefert dann

$$\varphi_n^{(i+1)} = \frac{1}{1+\frac{\Delta\tau\Lambda|_{R=1}}{2}} \left[2\varphi_n^{(i)} - \left(1 - \frac{\Delta\tau\Lambda|_{R=1}}{2} \right) \varphi_n^{(i-1)} - (\Delta\tau)^2(\eta_{nk}\varphi_k^{(i)} + \sin\varphi_n^{(i)}) \right]. \quad (3.10)$$

Im Vergleich zum reibungsfreien Fall in Gleichung (3.9) taucht neben dem Reibungsterm noch ein Skalierungsfaktor auf.

3.3. Iterative Methode zur Lösung des nichtlinearen gewöhnlichen DGL-Systems

Alternativ zur Nutzung einer vorgefertigten Python-Routine oder einer Zeitdiskretisierung der Bewegungsgleichung wird nun ein iteratives Verfahren zur Lösung nichtlinearer Differentialgleichungssysteme vorgestellt. Dieses wurde 2015 in einem Paper mit dem Titel „A computational iterative method for solving nonlinear ordinary differential equations“ von den Autoren H. Temimi und A. Ansari vorgestellt und wird im Folgenden angewandt. Die Methode selber wird in dieser Arbeit nicht vorgestellt und kann in [4] nachgelesen werden. Im nächsten Abschnitt wird diese Methode auf die Differentialgleichung für endlich viele Pendel (Gleichung (2.11)) angewendet.

3.3.1. Aufteilung in linearen und nichtlinearen Anteil

Zunächst muss das Differentialgleichungssystem, bestehend aus n Differentialgleichungen, in einen linearen und nichtlinearen Anteil zerlegt werden. Dabei beschreibt L den linearen und N den nichtlinearen Anteil. Als $L_n\{\varphi_n\}$ und $N_n\{\varphi_n\}\ \forall\ n = 1, ..., N$ wird der lineare und nichtlineare Anteil der Differentialgleichung des n-ten Pendels bezeichnet. Als Ansatz wird

$$L_n\{\varphi_n\} = 0 \quad (3.11)$$

gesetzt und angenommen, dass die Anfangsbedingungen $\varphi_n(t = 0) =: \varphi_n^0$ und $\dot{\varphi}_n(\ t\ =\ 0)$ $=:\ \dot{\varphi}_n^0$ diese löst. Daraus folgt eine Funktion $\varphi_{n,0}$ mit der die Iteration gestartet wird. Die Startfunktion $\varphi_{n,0}$ wird in den nichtlinearen Anteil N eingesetzt und die Lösung des folgenden Gleichungssystems liefert $\varphi_{n,1}$, indem eben diese in L_n eingesetzt wird:

$$L_n\{\varphi_{n,1}\} = -N_n\{\varphi_{n,0}\} \quad (3.12)$$

Diese Vorgehensweise wird iterativ fortgeführt, sodass für den $(\nu + 1)$-ten Schritt gilt:

$$L_n\{\varphi_{n,\nu+1}\} = -N_n\{\varphi_{n,\nu}\} \quad (3.13)$$

Der Index ν in $\varphi_{n,\nu}$ bezeichnet die Funktion φ_n des ν-ten Iterationsschrittes. Die Wahl des linearen Anteils hängt von den Anforderungen an die Iteration ab. Sollen die Rechnungen in jeder Iteration möglichst einfach gehalten werden, kann der lineare Anteil $L_n = \ddot{\varphi}_n$ gewählt werden und der Rest der Differentialgleichung in N_n untergebracht werden. Damit beinhaltet der nichtlineare Anteil auch lineare Funktionen. Für die Differentialgleichung der gekoppelten Pendel sind damit allerdings mehr Iterationsschritte nötig und je nach Anfangsbedingungen und gewähltem $\Delta\tau$ konvergiert das Verfahren gegebenenfalls nicht. Deswegen werden für Gleichung (2.11) L_n und N_n wie folgt gewählt:

$$L_n = \ddot{\varphi}_n + \eta_{nk}\varphi_k, \tag{3.14a}$$

$$N_n = \sin\varphi_n + \Lambda\dot{\varphi}_n. \tag{3.14b}$$

Damit erhöht sich zwar der Rechenaufwand in jeder Iteration minimal, allerdings konvergiert das Verfahren schneller. Aus Gleichung (3.11) folgt

$$\ddot{\varphi}_{n,0} = -\eta_{nk}\varphi_{k,0}, \tag{3.15}$$

und somit der Startwert $\varphi_{n,0}$ für die Iteration. Aus dem ersten Iterationsschritt ergibt sich $\varphi_{n,1}$ mit

$$\ddot{\varphi}_{n,1} + \eta_{nk}\varphi_{k,1} = -\sin\varphi_{n,0} - \Lambda\dot{\varphi}_{n,0}. \tag{3.16}$$

Analog gilt für den $(\nu+1)$-ten Schritt

$$\ddot{\varphi}_{n,\nu+1} + \eta_{nk}\varphi_{k,\nu+1} = -\sin\varphi_{n,\nu} - \Lambda\dot{\varphi}_{n,\nu}. \tag{3.17}$$

Konvergiert das Verfahren, das heißt existiert der Grenzwert

$$\lim_{\nu\to\infty} \varphi_{n,\nu}(t) = \varphi_{n,\infty}(t), \quad n = 1, ..., N, \tag{3.18}$$

so ist $\varphi_n(t) = \varphi_{n,\infty}(t), n = 1, ..., N$ eine Lösung von Gleichung (2.11). Aufgrund der nichtlinearen Sinus-Funktion sind diese Differentialgleichungen nur numerisch lösbar. Die Genauigkeit der Lösung ist durch die Diskretisierung der Zeit beschränkt und es wird eine untere Schwelle ϵ gewählt, sodass für

$$\max_{k=1,...,N} \max_{i=1,...,M} |\varphi_{k,\nu+1}(\tau_i) - \varphi_{k,\nu}(\tau_i)| < \epsilon, \tag{3.19}$$

die Iteration abgebrochen wird. Die untere Schwelle ist als $\epsilon = 1 \cdot 10^{-5}$ gewählt.

3.3.2. Schwingungsansatz mit Variation der Konstanten

Eine Möglichkeit zur numerischen Lösung besteht darin, das Gleichungssystem $L_n = 0$ $\forall\, n = 1, ..., N$ in Gleichung (3.15) analytisch per Entkopplung zu lösen und anschließend die Konstanten zu variieren. Dazu wird die Matrix η diagonalisiert und mit Hilfe der Diagonalmatrix $\eta^D = \mathrm{diag}(\omega_1^2, ..., \omega_N^2)$ kann η umgeformt werden zu

$$\eta = A^t \eta^D A.$$

Auf der Diagonalen von η^D stehen die Eigenwerte $\omega_1^2, ..., \omega_N^2$ und in den Spalten von A^t die zugehörigen Eigenvektoren $\mathrm{v}_1, ..., \mathrm{v}_N$ von η. Es gilt $A^{-1} = A^t$ und damit wird Gleichung (3.15) entkoppelt. Es folgt die bekannte Schwingungsgleichung

$$\ddot{\Phi} = -\eta^D \Phi, \tag{3.20}$$

mit $\Phi = A\varphi$. Die Eigenwerte $\omega_1^2, ..., \omega_N^2$ entsprechen den quadrierten Eigenfrequenzen der Φ's und damit gilt

$$\Phi_k = \alpha_k \cos(\omega_k \tau) + \beta_k \sin(\omega_k \tau), \tag{3.21}$$

wobei α und β eindeutig durch die Anfangsbedingungen φ^0 und $\dot{\varphi}^0$ bestimmt sind zu

$$\alpha_k = A\varphi_k^0 \quad \wedge \quad \beta_k = \frac{1}{\omega_k} A w_k^0.$$

Für den ersten Iterationsschritt wird nun

$$\mathrm{A}L\{\varphi_{n,\nu+1}\} = -\mathrm{A}N\{\varphi_{n,\nu}\}$$

betrachtet. Für festes k folgt

$$\ddot{\Phi}_{k,\nu+1} + \eta_{kk}^D \Phi_{k,\nu+1} = -A_{kn} N_{n,\nu} \Leftrightarrow \ddot{\Phi}_{k,\nu+1} + \omega_k^2 \Phi_{k,\nu+1} = -\hat{N}_{k,\nu}, \tag{3.22}$$

mit $\hat{N}_k = A_{kn} N_n$. Als Ansatz für Φ nehmen wir die Lösung aus Gleichung (3.21) und variieren die Konstanten α und β. Damit folgt für Φ und erste sowie zweite Ableitung:

$$\Phi_k(\tau) = \alpha_k(\tau) \cos(\omega_k \tau) + \beta_k(\tau) \sin(\omega_k \tau), \tag{3.23a}$$

$$\dot{\Phi}_k(\tau) = \underbrace{\dot{\alpha}_k \cos(\omega_k \tau) + \dot{\beta}_k \sin(\omega_k \tau)}_{=0 \text{ (Ansatz)}} + \omega_k [\beta_k(\tau) \cos(\omega_k \tau) - \alpha_k(\tau) \sin(\omega_k \tau)], \tag{3.23b}$$

$$\ddot{\Phi}_k(\tau) = \omega_k [\dot{\beta}_k \cos(\omega_k \tau) - \dot{\alpha}_k \sin(\omega_k \tau)] - \omega_k^2 \underbrace{(\alpha_k(\tau) \cos(\omega_k \tau) + \beta_k(\tau) \sin(\omega_k \tau))}_{=\Phi_k(\tau)}. \tag{3.23c}$$

Einsetzen in Gleichung (3.22) liefert

$$\ddot{\Phi}_k + \omega_k^2 \Phi_k = \omega_k [\dot{\beta}_k \cos(\omega_k \tau) - \dot{\alpha}_k \sin(\omega_k \tau)] =^! -\hat{N}_k. \tag{3.24}$$

Damit ergeben sich zusammen mit dem Ansatz aus Gleichung (3.23b) zwei Gleichungen zur Bestimmung von α und β:

$$\dot{\alpha}_k \cos(\omega_k \tau) + \dot{\beta}_k \sin(\omega_k \tau) = 0, \tag{3.25a}$$

$$\dot{\beta}_k \cos(\omega_k \tau) - \dot{\alpha}_k \sin(\omega_k \tau) = -\frac{1}{\omega_k} \hat{N}_k. \tag{3.25b}$$

Um β zu eliminieren und α zu erhalten wird Gleichung (3.25a) mit $\cos(\omega_k \tau)$ und (3.25b) mit $\sin(\omega_k \tau)$ multipliziert und die zweite Gleichung von der ersten subtrahiert. Analog können Gleichung (3.25a) mit $\sin(\omega_k \tau)$ und (3.25b) mit $\cos(\omega_k \tau)$ multipliziert und beide Gleichungen aufaddiert werden um α zu eliminieren:

$$\dot{\alpha}_k = \frac{1}{\omega_k} \sin(\omega_k \tau) \hat{N}_k,$$
$$\dot{\beta}_k = -\frac{1}{\omega_k} \cos(\omega_k \tau) \hat{N}_k.$$

Integrieren liefert:

$$\alpha_k(\tau) = \frac{1}{\omega_k} \int_0^\tau dt \sin(\omega_k t) \hat{N}_k(t) + \alpha_k(0) \quad \wedge \quad \beta_k(\tau) = -\frac{1}{\omega_k} \int_0^\tau \mathrm{d}t \cos(\omega_k t) \hat{N}_k(t) + \beta_k(0).$$

Eingesetzt in den Ansatz aus Gleichung (3.22) ergibt sich

$$\Phi_k(\tau) = A_{kn}\varphi_n^0 c_k(\tau) + A_{kn}\dot{\varphi}_n^0 s_k(\tau) + \frac{c_k(\tau)}{\omega_k}\int_0^\tau \mathrm{d}t s_k(t)\hat{N}_k(t) - \frac{s_k(\tau)}{\omega_k}\int_0^\tau \mathrm{d}t c_k(t)\hat{N}_k(t),$$

mit

$$c_k(\tau) := \cos(\omega_k\tau) \quad \wedge \quad s_k(\tau) := \sin(\omega_k\tau).$$

Für den $(\nu+1)$-ten Iterationsschritt wird also

$$\Phi_{k,\nu+1}(\tau) = \Phi_k^0 c_k(\tau) + \dot{\Phi}_k^0 s_k(\tau) + \frac{c_k(\tau)}{\omega_k}\int_0^\tau \mathrm{d}t s_k(t)\hat{N}_{k,\nu}(t) - \frac{s_k(\tau)}{\omega_k}\int_0^\tau \mathrm{d}t c_k(t)\hat{N}_{k,\nu}(t),$$

gelöst, wobei in $\hat{N}$, mit

$$\hat{N}_{k,\nu}(t) = A_{kn}N_{n,\nu} = A_{kn}[\sin(\varphi_{n,\nu}(t)) + \Lambda\dot{\varphi}_{n,\nu}(t)],$$

das $\varphi_\nu = A^t\Phi_\nu$ aus dem vorherigem Schritt ν eingesetzt wird. Da die Werte in den Integralen nur diskret vorliegen, wird eine numerische Integration über diskrete Werte benötigt. Für das Integral

$$I_k(\tau) = \int_0^\tau \mathrm{d}t f_k(t),$$

gilt für für die diskretisierten Zeitschritte $\tau_i = i\Delta\tau$, $i = 1, ..., M$:

$$I_k^{(i)} := \int_0^{\tau_i} \mathrm{d}t f_k(t) = \frac{\Delta\tau}{2}\sum_{j=0}^{i-1}\left(f_k^{(j+1)} + f_k^{(j)}\right) + \mathcal{O}((\Delta\tau)^2).$$

Konvergiert nun das Verfahren für

$$\lim_{\nu\to\infty}\Phi_{k,\nu} = \Phi_{k,\infty}, \quad k = 1, ..., N,$$

so ist $\Phi_k(\tau) = \Phi_{k,\infty}(\tau)$ eine Lösung von Gleichung (3.22) und es folgt $\varphi_n(\tau) = A_{nk}\Phi_{k,\infty}(\tau)$ als Lösungsfunktion für die Winkel der Pendel.

4. Die Sine-Gordon-Gleichung

Die Sine-Gordon Gleichung ist eine nichtlineare Wellengleichung. In einer Dimension lautet sie

$$\varphi_{\tau\tau} - \varphi_{xx} = \sin\varphi,$$

wobei $\varphi = \varphi(x,\tau)$ mit

$$\varphi_{\tau\tau} = \frac{\partial^2\varphi}{\partial\tau^2} \quad \wedge \quad \varphi_{xx} = \frac{\partial^2\varphi}{\partial x^2}.$$

Sie ist eine nichtlineare partielle Differentialgleichung zweiter Ordnung und besitzt sogenannte Solitonen und Multisolitonen als Lösungen. Sie beschreibt unter anderem ein harmonisch gekoppeltes System von N Pendeln für $N \to \infty$.

4.1. Übergang zu unendlich vielen Pendeln

Für jedes Pendel n sollen weitere Pendel so der Pendelkette hinzugefügt werden, sodass die Dichte pro Länge konstant bleibt. Sei d der Abstand zwischen dem ersten und letzten Pendel. Dann ist die Dichte $\rho = m/d$. Mit $\Delta m = m/N$ und $\Delta x = d/N$ gilt

$$\rho = \frac{m}{d} = \frac{m/N}{d/N} = \frac{\Delta m}{\Delta x}. \tag{4.1}$$

Die Kraft der Feder κ pro Längeneinheit soll ebenso konstant sein:

$$\kappa = kd = \Delta k \Delta x, \tag{4.2}$$

mit $\Delta k = kN$. Auch der Reibungskoeffizient pro Masse Γ wird konstant gehalten.

$$\Gamma = \frac{\gamma}{m} = \frac{\gamma/N}{m/N} = \frac{\Delta\gamma}{\Delta m} \tag{4.3}$$

mit $\Delta\gamma = \gamma/N$. Nun werden Masse m, Federkonstante k und Reibungskoeffizient γ aus Gleichung (2.11) entsprechend $m \to \Delta m$, $k \to \Delta k$ und $\gamma \to \Delta\gamma$ umskaliert. Da φ_n das n-te Pendel zur Zeit τ bezeichnet, können wir den Winkel als orts- und zeitabhängige Funktion auffassen, also $\varphi_n \to \varphi(x,\tau)$ und $\varphi_{n+1} \to \varphi(x+\Delta x,\tau)$. Damit folgt für die Bewegungsgleichung:

$$\begin{aligned}
\ddot{\varphi}(x,\tau) = &-\sin\varphi(x,\tau) + \frac{\Delta k}{\Delta m}\frac{\sigma^2}{\omega_g^2}\left[\varphi(x+\Delta x,\tau) - 2\varphi(x,\tau) + \varphi(x-\Delta x,\tau)\right] \\
&- \frac{\Delta\gamma}{\Delta m}\frac{(\alpha^2+l^2)^{\frac{R-1}{2}}}{\Omega_g^{2-R}}|\dot{\varphi}(x,\tau)|^R\frac{\dot{\varphi}(x,\tau)}{|\dot{\varphi}(x,\tau)|}, \qquad (4.4) \\
= &-\sin\varphi + \frac{\kappa}{\rho}\frac{\sigma^2}{\omega_g^2}\left(\frac{\varphi(x+\Delta x,\tau) - 2\varphi(x,\tau) + \varphi(x-\Delta x,\tau)}{(\Delta x)^2}\right) \\
&- \frac{\Gamma}{\Omega_g^{2-R}}(\alpha^2+l^2)^{\frac{R-1}{2}}|\dot{\varphi}(x,\tau)|^R\frac{\dot{\varphi}(x,\tau)}{|\dot{\varphi}(x,\tau)|}. \qquad (4.5)
\end{aligned}$$

Der zweite Differenzenquotient läuft im Limes von unendlich vielen Pendeln $N \to \infty$, also $\Delta x \to 0$, gegen die zweite Ableitung. Für unendlich viele Pendel gilt dann

$$\varphi_{\tau\tau} = -\sin\varphi + \left(\frac{\Omega_\kappa}{\Omega_g}\right)^2 \varphi_{xx} - \Lambda\varphi_\tau,$$

mit $\varphi_\tau = \partial\varphi/\partial\tau$. $\Omega_g = -\omega_g^2/(\sigma^2+1)$ und analog zu Ω_k^2 wird

$$\Omega_\kappa^2 = \frac{\kappa}{\rho}\frac{\sigma^2}{\sigma^2+1}$$

neu eingeführt. Λ wird in Gleichung (2.13) definiert. Für den reibungsfreien Fall $\Lambda = 0$ ergibt sich die bekannte Sine-Gordon-Gleichung in einer Dimension:

$$\varphi_{\tau\tau} = -\sin\varphi + \nu^2\varphi_{xx}, \tag{4.6}$$

wobei $\nu = \Omega_\kappa/\Omega_g$.

4.2. Solitonen und Multisolitonen

Die Sine-Gordon-Gleichung besitzt bekannte Solitonen- und Multisolitonenlösungen [3]. Eine davon ist die Soliton (+) beziehungsweise Antisoliton (-) Lösung

$$\varphi(x,t) = 4\arctan\left(\exp\left(\pm\frac{k(x-x_0)-ct}{\sqrt{k^2\nu^2-c^2}}\right)\right), \tag{4.7a}$$

wobei $\varphi(x,t) = \varphi(kx-ct)$ eine sich fortbewegende Welle beschreibt und $|c| < |k|\,|\nu|$ gelten muss. Eine weitere Lösung der Sine-Gordon-Gleichung ist durch die Soliton-Soliton-Kollision

$$\varphi(x,t) = 4\arctan\left(\frac{c\sinh(\frac{kx}{\sqrt{k^2\nu^2-c^2}})}{\cosh(\frac{ct}{\sqrt{k^2\nu^2-c^2}})}\right) \tag{4.7b}$$

und der Soliton-Antisoliton-Kollision

$$\varphi(x,t) = 4\arctan\left(\frac{\sinh(\frac{ct}{\sqrt{k^2\nu^2-c^2}})}{c\cosh(\frac{kx}{\sqrt{k^2\nu^2-c^2}})}\right) \tag{4.7c}$$

gegeben. Zuletzt sei noch die Breather Soliton Lösung

$$\varphi(x,t) = 4\arctan\left(\frac{\sqrt{1-\omega^2}\sin(\omega t)}{\omega\cosh(\sqrt{1-\omega^2}x)}\right) \tag{4.7d}$$

erwähnt. Auch hier gilt $|\omega| < 1$.

5. Computersimulation der gekoppelten Pendel

Die Simulation der Pendelkette wird in Python mit Hilfe der Funktion `animation` von der Klasse `FuncAnimation` aus der Python Bibliothek `matplotlib` visualisiert. Aus Performancegründen werden die Winkel φ_n für alle n Pendel zunächst mit einer zuvor gewählten Schrittweite $\Delta\tau$ und Anzahl an Schritten berechnet. Anschließend öffnet sich die Animation und stellt die Bewegung der Pendel dar. Die Funktion erlaubt außerdem ein Pausieren der Animation durch Mausklick und Drehen des Blickwinkels mit gedrückter Maustaste. Die Hauptklasse `PendelNumerisch`, welche die Animation startet, kann im Anhang in Auflistung 3 eingesehen werden. Die Klasse `StartFunc` liefert die Anfangsbedingungen der Pendel. In der Hauptklasse wird eine Routine zur numerischen Berechnung, wie in Abbildung 5.1 dargestellt, ausgewählt und die Ergebnisse an die Hauptklasse zurückgegeben.

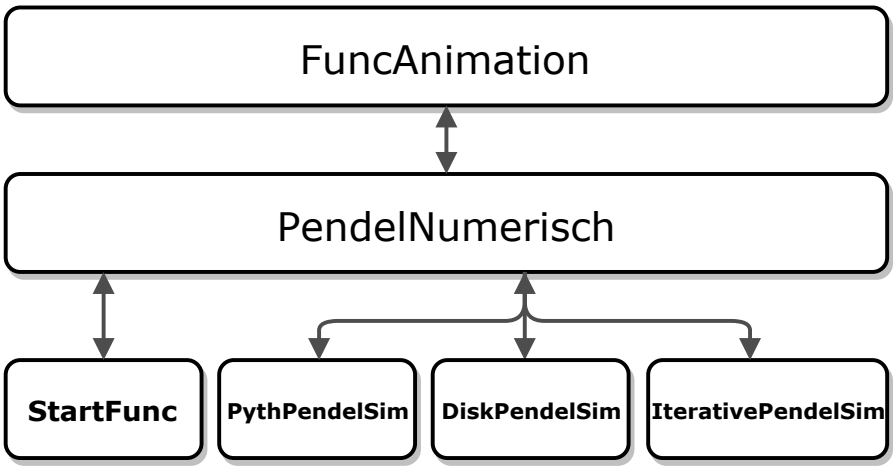


Abbildung 5.1: Skizziert ist der Zusammenhang der Klassen des Programms.

In der Klasse `PythPendelSim` wird die Bewegungsgleichung auf ein System erster Ordnung reduziert und der zuvor beschriebenen Python-Routine übergeben. In der Klasse `DiskPendelSim` wird die Bewegungsgleichung zeitdiskretisiert und explizit für alle Zeiten berechnet. Dagegen liefert die Klasse `IterativePendelSim` die Lösung mittels des Iterationsverfahrens mit Schwingungsansatz und Variation der Konstanten.

Die Pendel werden sowohl mit der oben beschriebenen Python-Routine als auch mit der iterativen Methode und der Diskretisierung der Differentialgleichung selber simuliert. Als Anfangsbedingungen werden zum einen Solitonenlösungen gewählt, zum anderen auch einzelne Pendel ausgelenkt. Im Folgenden werden einige Ausschnitte aus den Animationen der Simulierungen gezeigt und miteinander verglichen. Dabei sind die Parameter der Pendel zunächst konstant gehalten. Lediglich die Zeitschritte $\Delta\tau$, Anzahl der Zeitschritte, Reibung und Anfangsauslenkung und -geschwindigkeit werden variiert.

Die verbleibenden Parameter werden wie folgt gewählt:

Länge der Pendelkette	d	=	1
Masse	m	=	0.0001
Ortsfaktor	g	=	9.81
Federkonstante	k	=	5
Pendellänge	l	=	0.5
Kopplungsfaktor	α	=	0.05l
Reibungsfaktor	Λ	=	0

Lediglich für die Kreispendelkette wird die Länge $d = 5$ gewählt.

5.1. Pendel als Solitonen

Da die Bewegungsgleichung für $N \to \infty$ der Sine-Gordon-Gleichung entspricht, sollte die Simulation für viele Pendel bei entsprechend gewählten Anfangsbedingungen denen der Sine-Gordon-Lösung entsprechen. Die Bewegung der Pendelkette $x \in [0, 1]$ wird für verschiedene Soliton Anfangsbedingungen und 100 Pendel simuliert. Die Zeitentwicklung der Auslenkungen in y- und z-Richtung entlang der x-Achse wird geplottet. Dabei sind die Pendel entlang der x-Achse bei $z = 0$ aufgehangen und die y- sowie z-Achse ist auf die Länge der Pendel normiert.

Zunächst wird Gleichung (4.7a) angepasst:

$$\varphi_S(x,t) = 4 \arctan\left(\exp\left(\pm\frac{50x - ct}{\sqrt{\nu^2 - c^2}}\right)\right),$$

mit $c = 0.85\nu$ und als Anfangsbedingung für die Pendel gewählt. Das fortlaufende Soliton ist in der y-Auslenkung an zwei Peaks in Abbildung 5.3 zu erkennen. Der eine zeigt in positive y-Richtung und der andere ist um π um die z-Achse gedreht und zeigt damit in Richtung negativer y-Achse.

Mit der Zeit τ wandert das Soliton von der linken Seite der Pendelkette ($x = 0$) zur rechten bei $x = 1.0$. Dort wird er am offenen Ende bei $\tau \simeq 58$ reflektiert und läuft wieder zurück Richtung $x = 0$. Die Reflexion am offenen Ende ist in Abbildung 5.3 an der y-Auslenkung deutlich zu beobachten. In z-Richtung wird ein mit der Zeit laufender Peak beobachtet. Dort wo das Soliton sich gerade befindet, werden die Pendel einmal um die x-Achse gedreht. In Abbildung 5.2 können Ausschnitte aus der Animation betrachtet werden.

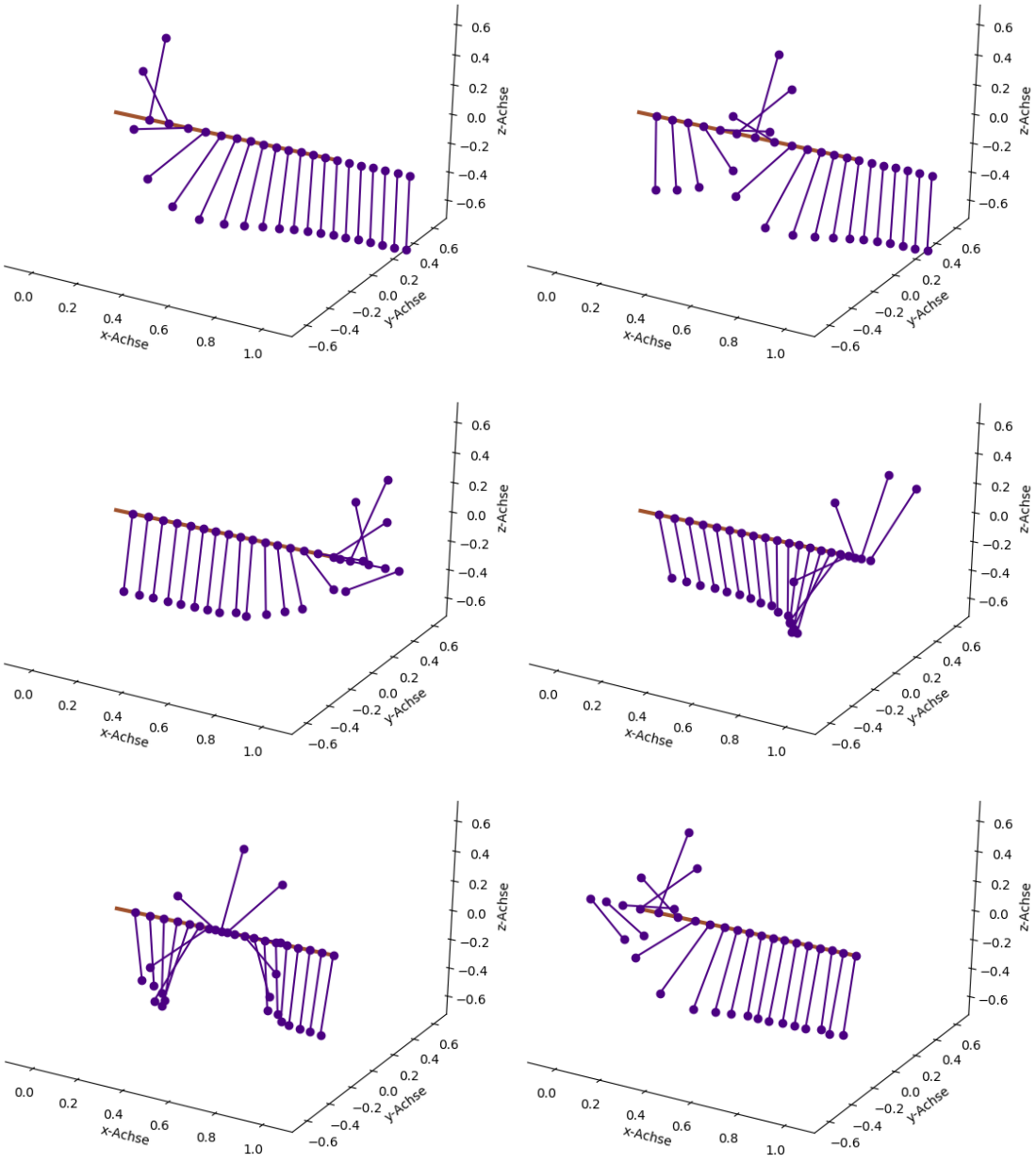


Abbildung 5.2: Ausschnitt aus der Animation eines Solitons $\varphi_S(x,t) = 4\arctan(\exp((50(x-1)-ct)/\sqrt{\nu^2-c^2}))$ auf der linearen Pendelkette von 20 Pendeln mit $\nu^2 \simeq 6.371$, $c = 0.85\nu$ und $\Delta\tau = 0.001$.

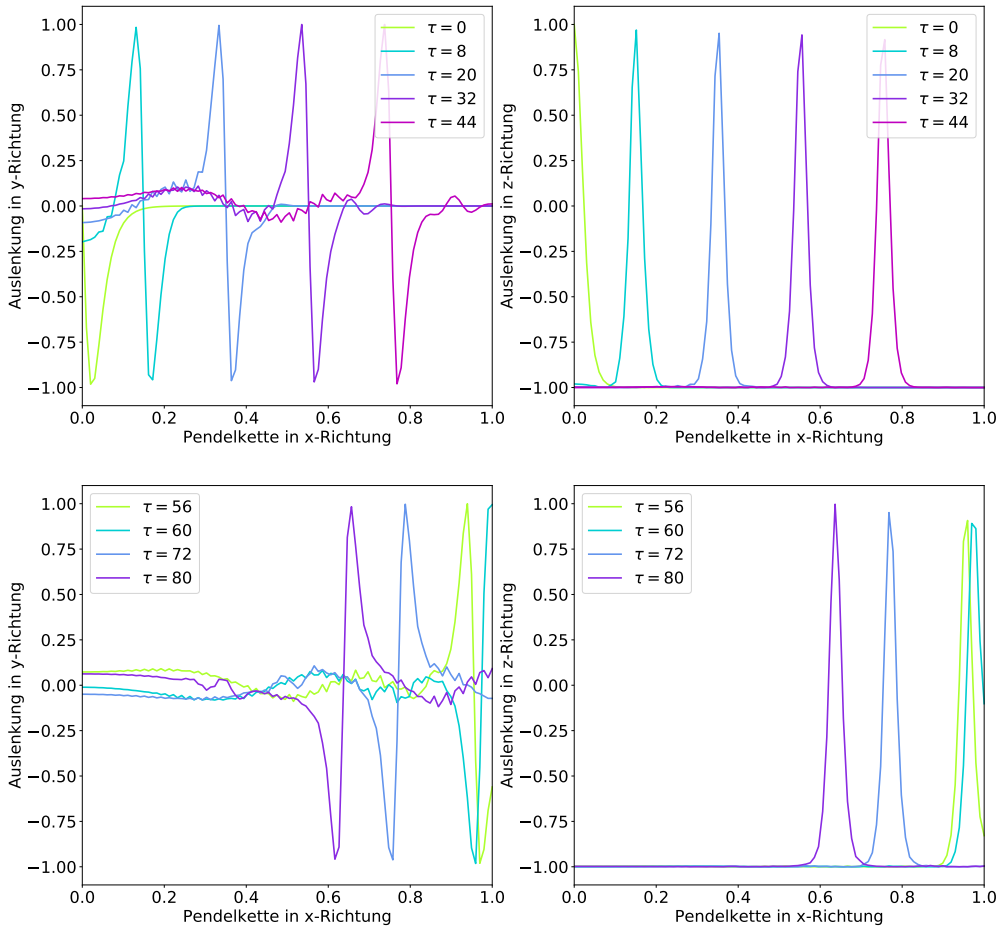


Abbildung 5.3: Zeitentwicklung der Pendelauslenkung in y- und z-Richtung mit einem Soliton $\varphi_S(x,t) = 4\arctan(\exp((50(x-1)-ct)/\sqrt{\nu^2-c^2}))$ als Anfangsbedingung ohne Reibung mit $\nu^2 \simeq 6.371$, $c = 0.85\nu$ und $\Delta\tau = 0.001$.

Da die Bewegungsgleichung für eine endliche Anzahl an Pendeln geplottet wird, ist eine Abweichung der Bewegung von der eines Solitons mit zunehmender Zeit zu erwarten. Neben kleineren Schwingungen in y-Richtung, die sich mit der Bewegung des Solitons überlagern, beschreibt die Anfangsbedingung das System auch für große Zeiten und den gewählten Parametern gut.

Für die Antisolitonlösung wird die folgende Funktion verwendet:

$$\varphi_A(x,t) = 4\arctan\left(\exp\left(-\frac{50(x-1)-ct}{\sqrt{\nu^2-c^2}}\right)\right),$$

mit $c = 0.85\nu$. Im Gegensatz zum Soliton wird die Funktion um die Länge der Pendelkette nach rechts verschoben, da die Welle in die entgegengesetzte Richtung läuft.

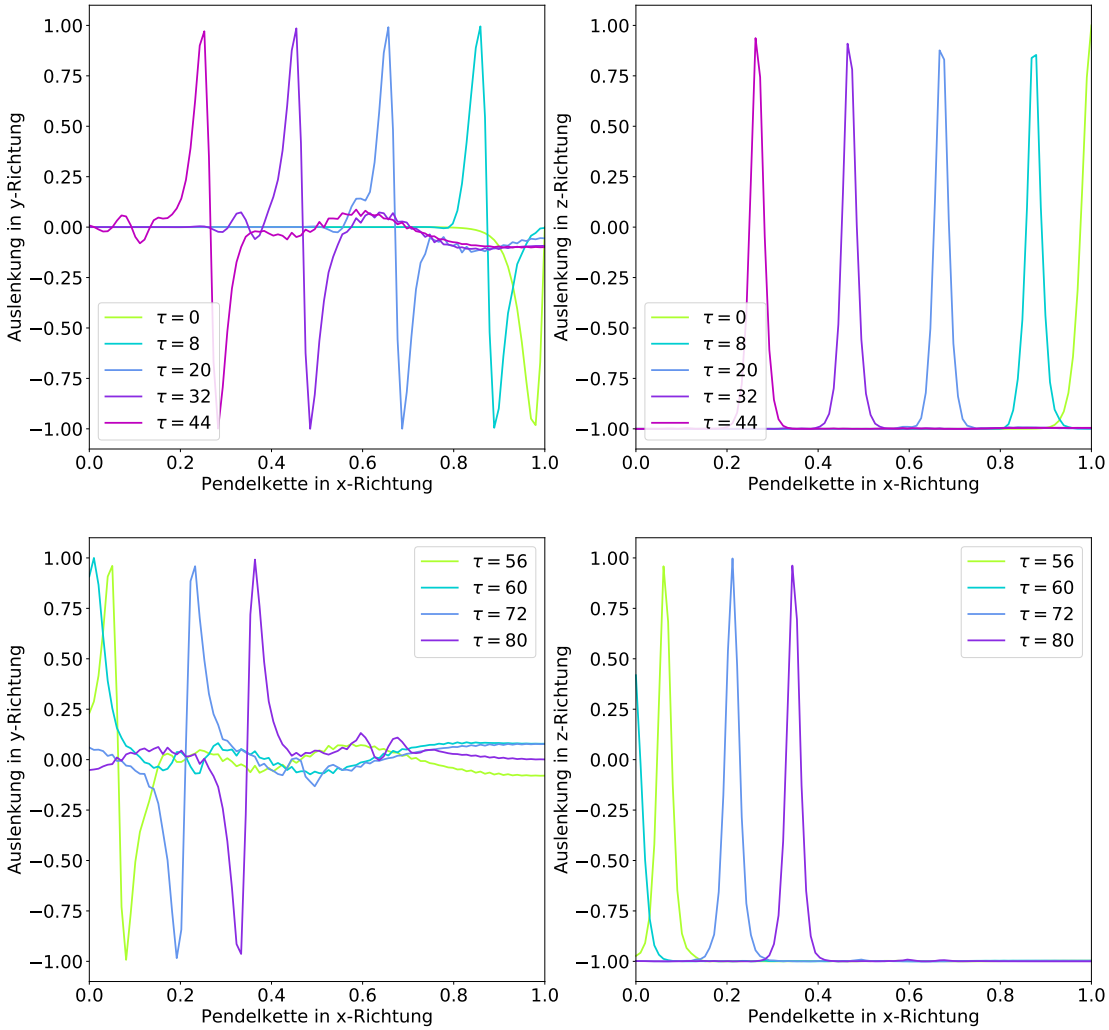


Abbildung 5.4: Zeitentwicklung der Pendelauslenkung in y- und z-Richtung mit einem Antisoliton $\varphi_A(x,t) = 4\arctan(\exp(-(50(x-1)-ct)/\sqrt{\nu^2-c^2}))$ als Anfangsbedingung ohne Reibung mit $\nu^2 \simeq 6.371$, $c = 0.85\nu$ und $\Delta\tau = 0.001$.

Für ein Antisoliton als Anfangsbedingung ergeben sich in Abbildung 5.4 analog zum Soliton zwei Peaks, der eine in und der andere entgegen der y-Richtung. Da sich Soliton und Antisoliton lediglich um ein Vorzeichen unterscheiden, wird entsprechend der Parametrisierung aus Gleichung (2.1) ein Vorzeichenwechsel in den y-Koordinaten und unveränderte z-Koordinaten erwartet. Ein Vergleich von Abbildung 5.3 und 5.4 zeigt, dass sich das Antisoliton im Gegensatz zum Soliton mit dem positiven Peak fortbewegt. Anschaulich bedeutet dies, dass sich die Drehrichtung der Pendel umgedreht hat. Zur Zeit $\tau \approx 58$ wird das Antisoliton wieder am offenen Ende reflektiert. In z-Richtung sind erneut einfache Peaks Richtung positiver z-Achse zu erkennen, welche nun von $x = 1$ nach $x = 0$ und wieder zurück wandern.

Eine weitere Lösung der Sine-Gordon-Gleichung ist die Soliton-Soliton-Kollision, bei der zwei Solitonen aufeinander treffen. Als Funktion wird

$$\varphi_{SK}(x,t) = 4\arctan\left(\frac{c\sinh(\frac{50(x-0.5)}{\sqrt{\nu^2-c^2}})}{\cosh(\frac{ct}{\sqrt{\nu^2-c^2}})}\right),$$

mit $c = 0.85\nu$ gewählt.

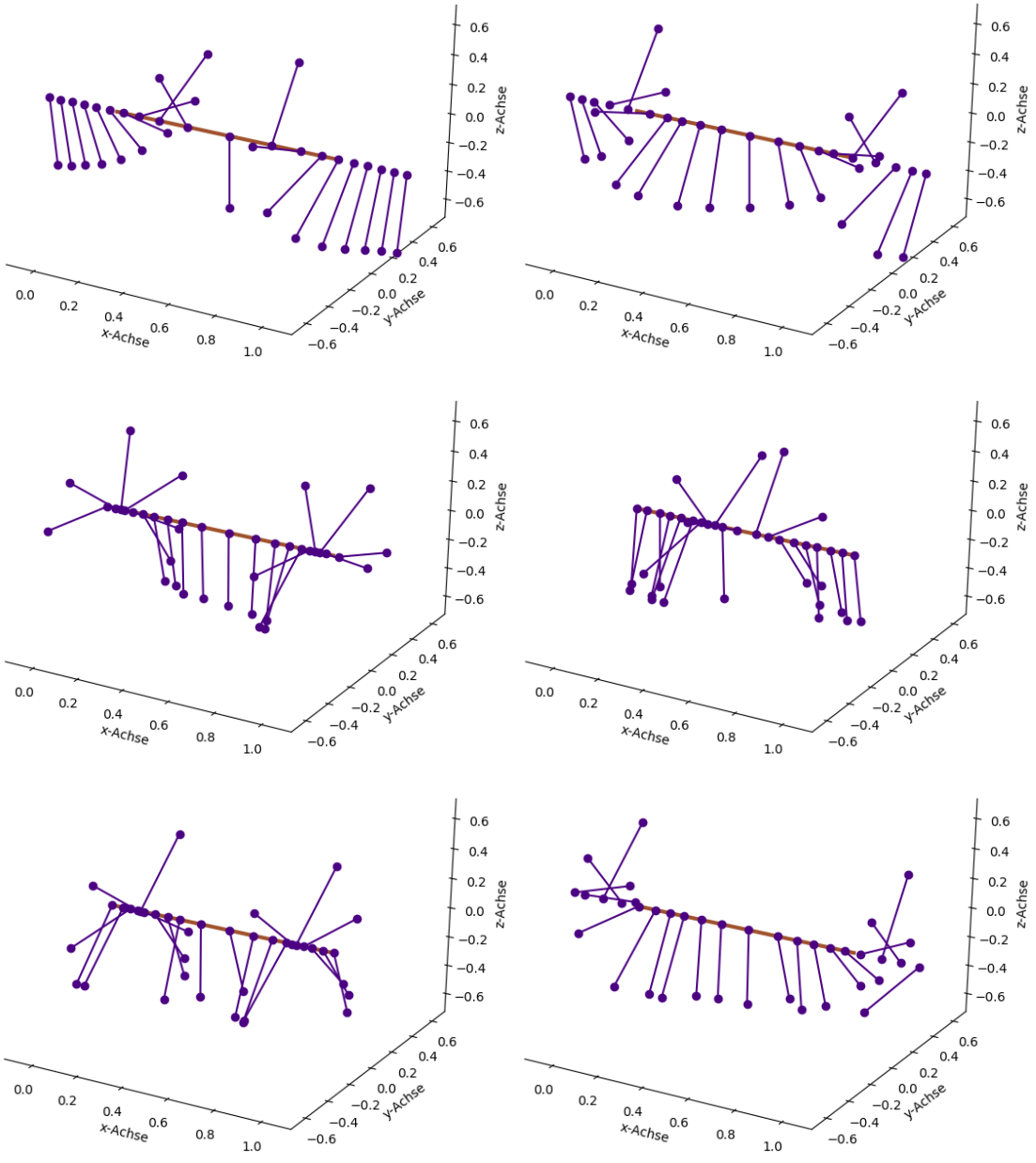


Abbildung 5.5: Ausschnitt aus der Animation einer Soliton-Soliton-Kollision $\varphi_{SK}(x,t) = 4\arctan(c\sinh(50(x-0.5)/\sqrt{\nu^2-c^2})/\cosh(ct/\sqrt{\nu^2-c^2}))$ auf der linearen Pendelkette von 20 Pendeln mit $\nu^2 \simeq 6.371$, $c = 0.85\nu$ und $\Delta\tau = 0.001$.

In Abbildung 5.5 sind einige Ausschnitte aus der Animation der Soliton-Soliton-Kollision zu sehen. Die Auslenkung in y- oder z-Richtung ist in Abbildung 5.6 dargestellt. In y-Richtung ist zu Beginn $\tau = 0$ eine Überlagerung beider Solitonen zu erkennen, die danach in entgegengesetzte Richtung bis an die Enden der Pendelkette weiterlaufen. Dort angekommen ($\tau \simeq 25$) werden sie am offenen Ende reflektiert und bewegen sich wieder aufeinander zu, bis sie sich für $\tau = 50$ überlagern und wieder umkehren. In z-Richtung können zwei Peaks symmetrisch zur Mitte der Pendelkette beobachtet werden, die sich zunächst voneinander weg bewegen, an den Wenden reflektiert werden und wieder aufeinander zubewegen. Bis auf kleinere Schwingungen, die der y-Auslenkung überlagert sind, entspricht dies der erwarteten Soliton-Soliton-Kollision.

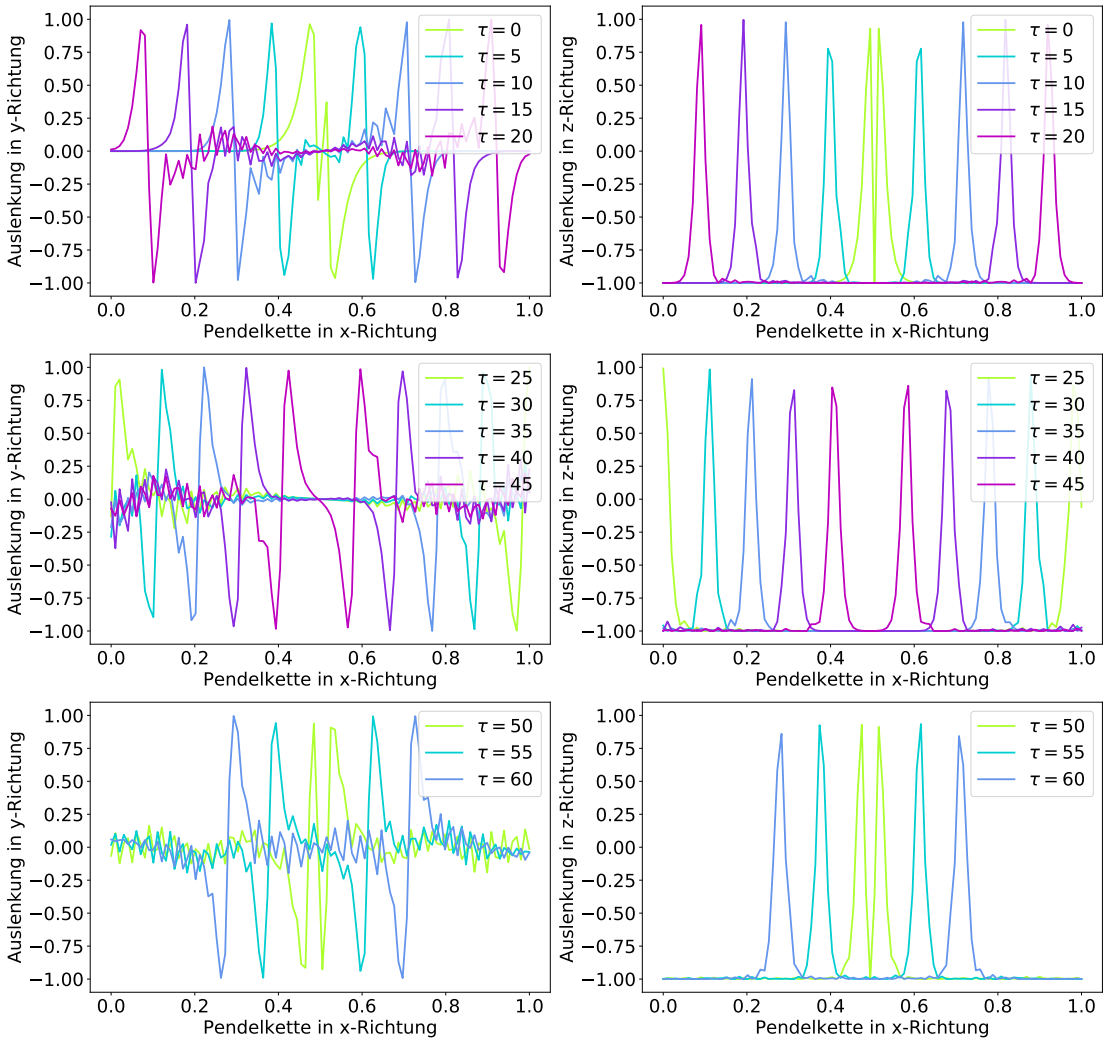


Abbildung 5.6: Zeitentwicklung der Pendelauslenkung in y- und z-Richtung mit einer Soliton-Soliton-Kollision $\varphi_{SK}(x,t) = 4\arctan(c\sinh(50(x - 0.5)/\sqrt{\nu^2 - c^2})/\cosh(ct/\sqrt{\nu^2 - c^2}))$ als Anfangsbedingung ohne Reibung mit $\nu^2 \simeq 6.371$, $c = 0.85\nu$ und $\Delta\tau = 0.001$.

Zuletzt wird eine Breather-Lösung als Anfangsbedingung gewählt und die Pendel damit simuliert. Dazu wird die Funktion in Gleichung (4.7) entsprechend

$$\varphi_B(x,t) = 4\arctan\left(\frac{\sqrt{1-\omega^2}\sin(\omega t)}{\omega\cosh(\sqrt{1-\omega^2}50(x-0.5))}\right),$$

mit $\omega = 0.95$ modifiziert.

Die Breather-Lösung, auch atmende Lösung genannt, beschreibt im Wesentlichen eine oszillierende stehende Welle mit einem Wellenbauch. Bei der Pendelkette liegen die Knoten nahe der Ränder der Pendelkette. Für alle Zeiten ist der Wellenbauch wie bei einer stehenden Welle deutlich zu erkennen, siehe dazu die Auslenkung in y-Richtung in Abbildung A.1 und in Abbildung A.2 in z-Richtung im Anhang für die Zeiten $0 \leq \tau \leq 15$. Ein Auschnitt der Animation für 50 Pendel kann in Abbildung 5.7 betrachtet werden.

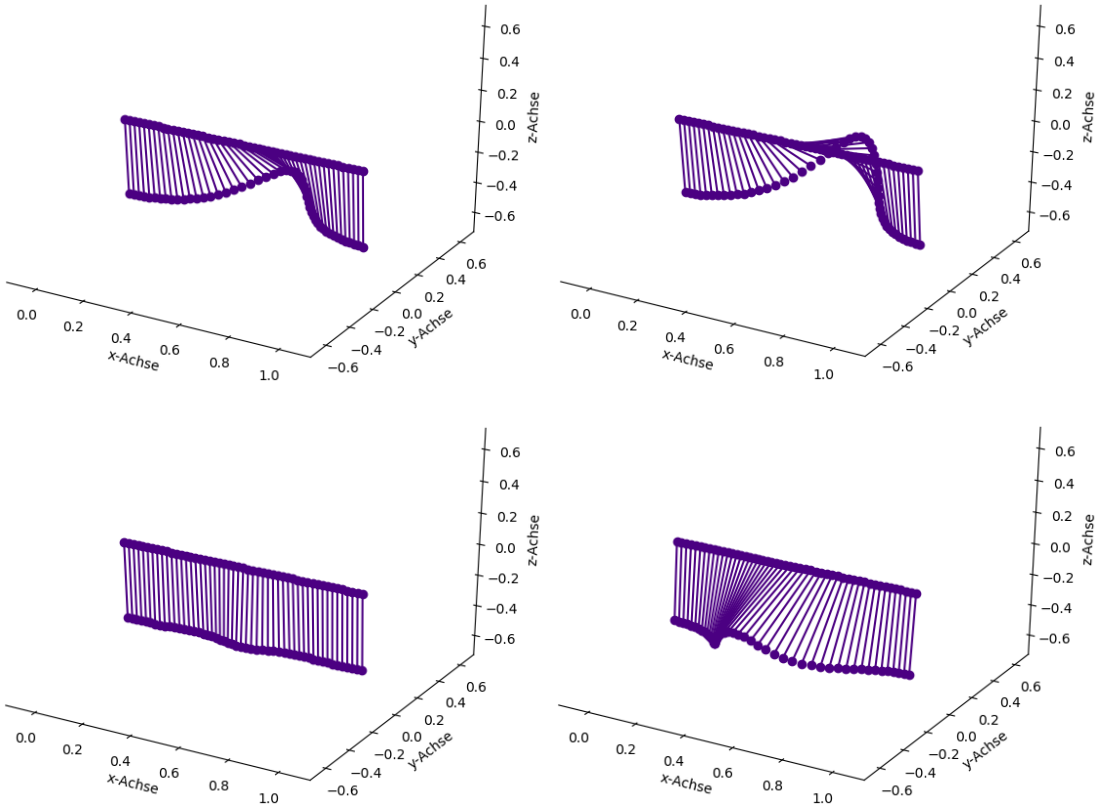


Abbildung 5.7: Ausschnitt aus der Animation einer Breather-Funktion $\varphi_B(x,t) = 4\arctan(\sqrt{1-0.95^2}\sin(0.95t)/0.95\cosh(\sqrt{1-0.95^2}50(x-0.5)))$ auf der linearen Pendelkette von 50 Pendeln mit $\nu^2 \simeq 6.371$ und $\Delta\tau = 0.001$.

Mit der Zeit verformt sich der Wellenbauch. Die mittleren Pendel hängen mit der Zeit zunehmend hinterher. Dennoch zeigt die Simulation, dass die Breatherlösung das System der gekoppelten Pendelkette sehr gut beschreibt.

Die Animation derselben Anfangsfunktion für eine Kreispendelkette mit einer Länge von $S = 5$ mit 50 Pendeln ist in Abbildung 5.8 in Ausschnitten dargestellt. Es ist deutlich zu erkennen, dass sich eine Welle auf der Kreispendelkette mit der Zeit ausbreitet.

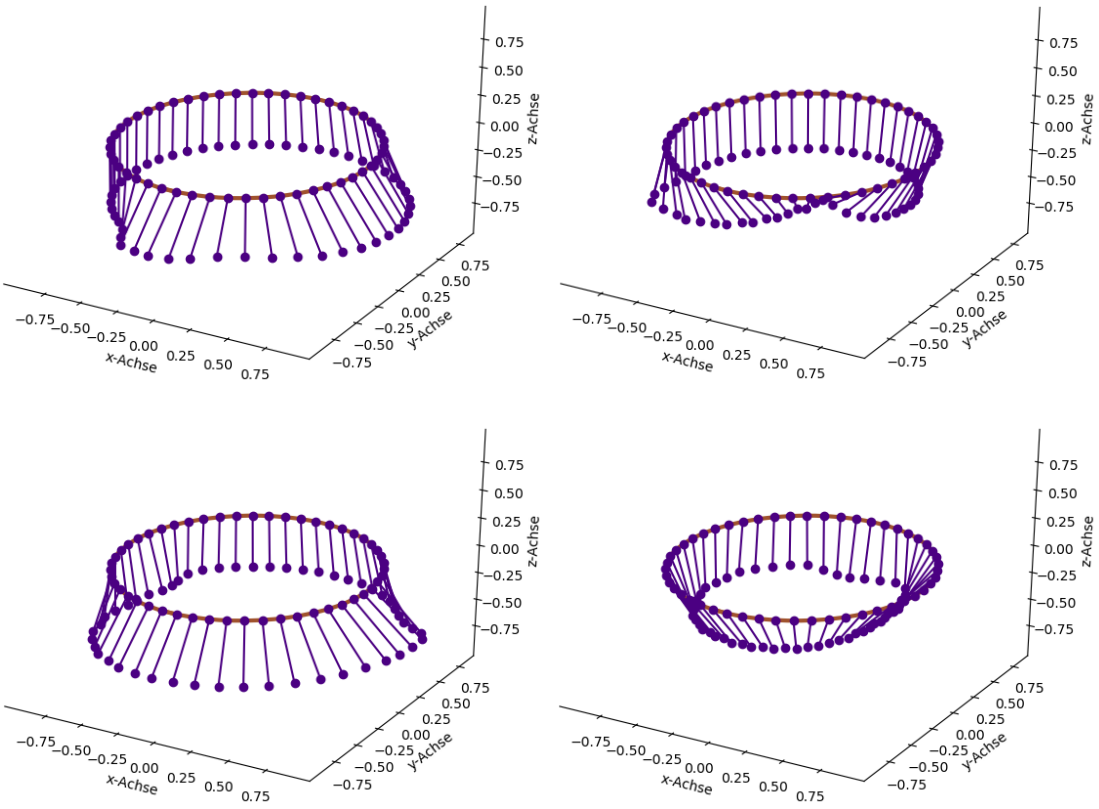


Abbildung 5.8: Ausschnitt aus der Animation einer Breather-Funktion $\varphi_B(x,t) = 4\arctan(\sqrt{1-0.95^2}\sin(0.95t)/0.95\cosh(\sqrt{1-0.95^2}50(x-0.5)))$ auf der Kreispendelkette von 50 Pendeln und $S = 5$ mit $\nu^2 \simeq 6.371$ und $\Delta\tau = 0.001$.

5.2. Beispiel: Stokesche Reibung

Der Fall Stokescher Reibung wird exemplarisch für vier verschiedene Werte des Reibungsfaktors Λ untersucht.

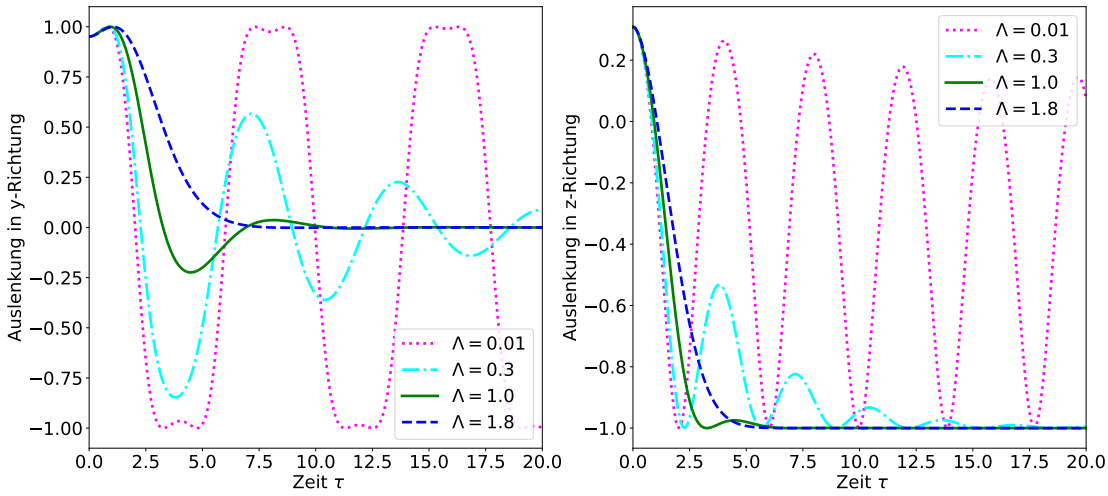


Abbildung 5.9: Geplottet sind ist die Auslenkung in y- und z-Richtung des ersten Pendels in Abhängigkeit der Zeit, wenn zu Beginn die Hälfte der Pendel um $3/5\pi$ ausgelenkt wird mit $n = 100$ und $\Delta\tau = 0.001$ für vier verschiedene Reibungsfaktoren Λ

Für kleine Werte von Λ ist eine Schwingung mit abnehmender Amplitude zu erkennen. Für $\Lambda = 0.3$ ist die charakteristische exponentielle Abnahme der Amplitude der Schwingung zu beobachten. Für $\Lambda = 1.0$ schwingt das System nur zweimal über die Ruhelage hinaus und nähert sich dann der Ruhelage. Für $\Lambda = 1.8$ ist der aperiodische Grenzfall zu erkennen. Für größere Reibungsfaktoren wäre der Kriechfall zu beobachten. Diese Beobachtungen können in allen Methoden gemacht werden und unterscheiden sich nicht.

5.3. Vergleich der numerischen Methoden

Um die zuvor vorgestellten Methoden zur numerischen Lösung der Bewegungsgleichung zu vergleichen, wird zum einen die Berechnungszeit für die jeweilige Zeit τ_e und zum anderen die Abweichung zur Python-Routine ϵ, die noch vorgestellt wird, untersucht.

5.3.1. Performance

In allen Verfahren zur Lösung der Bewegungsgleichung ist ein exponentieller Zusammenhang zwischen dem Rechenaufwand und der Gesamtlaufzeit τ_e festzustellen. Da die iterative Methode die Winkel φ_n für alle Zeiten in jedem Iterationsschritt berechnet, ist zu erwarten, dass sie die rechenaufwendigste Methode ist. Die Python-Routine greift auf Fortran Bibliotheken zurück, welche performant programmiert sind. In Abbildung 5.10 sind die Berechnungszeiten der Python-Routine und der expliziten Berechnung der Winkel aus der zeitdiskretisierten Bewegungsgleichung in Abhängigkeit der Gesamtlaufzeit τ_e für drei verschiedene Zeitschrittweiten $\Delta\tau$ aufgetragen, wobei zu Beginn ein Pendel um $\pi/4$ ausgelenkt wird.

Bei der Python-Routine liegen die Werte für die verschiedenen Schrittweiten sehr nahe beieinander und die benötigte Zeit nimmt nur leicht mit $\Delta\tau$ zu. Für die zeitdiskretisierte Bewegungsgleichung erhöht sich der Rechenaufwand um den Faktor, um den sich die Schrittweite verringert. Dadurch, dass die Berechnung mit einer Schleife über $1, ..., M$ läuft, erhöht sich die Anzahl der Berechnungen mit M. Für dieselbe Gesamtlaufzeit τ_e werden für kleinere $\Delta\tau$ Werte somit mehr Zeitschritte benötigt. Ein Vergleich beider Methoden in Abbildung 5.10 und A.3 im Anhang, bei der eine Solitonanfangsbedingung gewählt wird, zeigt, dass die Zeitdiskretisierung für $\Delta\tau = 0.1$ auch für sehr große Zeiten um etwa einen Faktor 10 schneller erfolgt, als die Python-Routine. Für $\Delta\tau = 0.01$ liegen beide Kurven sehr eng beieinander und für größere Zeitschrittweiten ist die Python-Routine schließlich schneller.

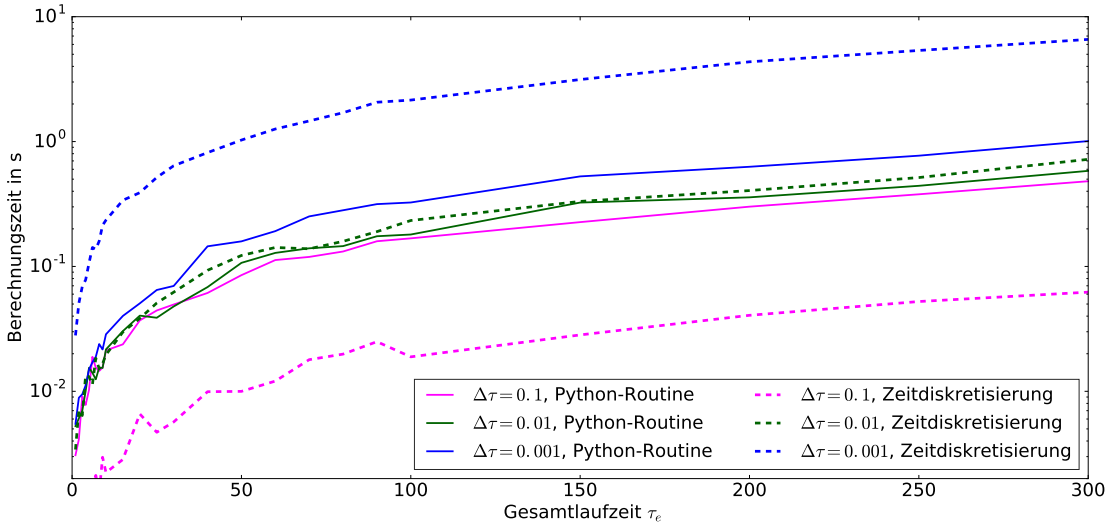


Abbildung 5.10: Aufgetragen sind die benötigten Berechnungszeiten für 50 Pendel in Abhängigkeit von der Gesamtzeit τ_e für drei verschiedene Schrittweiten $\Delta\tau$ für die Python-Routine und der Zeitdiskretisierung, wobei zu Beginn ein Pendel um $\pi/4$ ausgelenkt wird.

Das Iterationsverfahren mit dem Schwingungsansatz und der Variation der Konstanten dagegen benötigt, wie erwartet, eine längere Berechungszeit, wie in Abbildung 5.11 mit einem Soliton als Anfangsbedingung zu sehen ist. Auch mit nur einem Pendel um $\pi/4$ ausgelenkt, zeigt sich das gleiche Resultat wie im Anhang in Abbildung A.4 dargestellt. Auch die Methode mit Variation der Konstanten benötigt mit abnehmender Zeitschrittweite eine längere Berechnungszeit. Für jeden Zeitschritt werden die vorkommenden Integrale mit Hilfe von Summen diskretisiert. Da die Summe mit jedem weiteren Zeitschritt lediglich erweitert wird, kann hier zwar auf eine Rechenoperation verzichtet werden, allerdings ist die Schleife über die Zeitschritte nicht vermeidbar. Deswegen erhöht sich der Rechenaufwand um den Faktor, um den sich die Zeitschritte erhöhen.

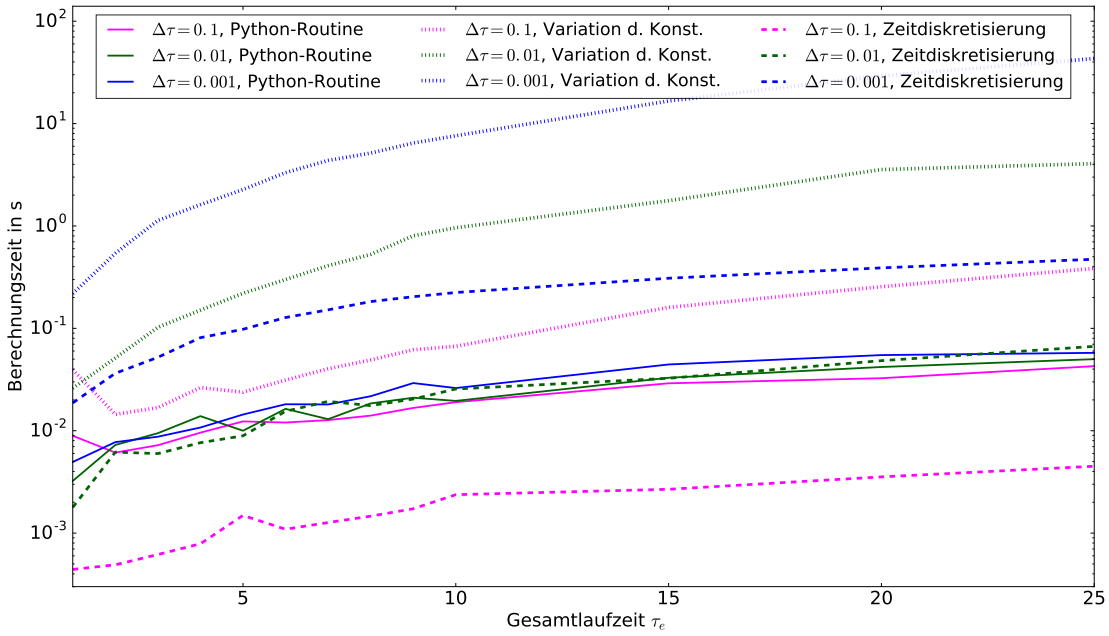


Abbildung 5.11: Aufgetragen sind die benötigten Berechnungszeiten für 50 Pendel in Abhängigkeit von der Gesamtzeit τ_e für drei verschiedene Schrittweiten $\Delta\tau$ für die Python-Routine, die Zeitdiskretisierung und Variation der Konstanten, mit einer Solitonenfunktion als Anfangsbedingung.

Das Iterationsverfahren für den Schwingungsansatz mit Variation der Konstanten konvergiert, unabhängig von der gewählten Zeitschrittweite, mit den gewählten Parametern ab einer Gesamtlaufzeit von $\tau_e = 30.0$ auch nach 1000 Iterationsschritten nicht mehr. Demnach können für diese Zeiten keine Untersuchungen angestellt werden.

5.3.2. Abweichung zur Python-Routine

Zur Untersuchung der Genauigkeit der Verfahren, werden die Werte für den Winkel φ zur Zeit τ_{end} für verschiedene $\Delta\tau$ verglichen. Dabei werden die Werte der Python-Routine als Referenzwerte für φ gewählt und mit $\hat{\varphi}$ bezeichnet und die Differenz aus diesen und den aus den Verfahren gewonnenen Werten gebildet, quadriert und über alle Pendel aufsummiert. Damit folgt dann der Fehler ϵ

$$\epsilon = \sqrt{\sum_{n=1}^{N} (\hat{\varphi}_n(\tau_{end}) - \varphi_n(\tau_{end}))^2}. \tag{5.1}$$

Der Fehler wird für vier verschiedene $\Delta\tau$, zwei verschiedene Anfansgbedingungen (Soliton und Auslenkung eines Pendels) und Anzahl an Pendeln (50 und 100) berechnet. Sowohl in der iterativen Methode, als auch bei der direkten expliziten Berechnung der diskretisierten Differentialgleichung zeigt sich ein exponentieller Zusammenhang zwischen $\epsilon/\sqrt{N}$ und der Zeit τ_{end} (siehe Abbildung 5.12 bis 5.14).

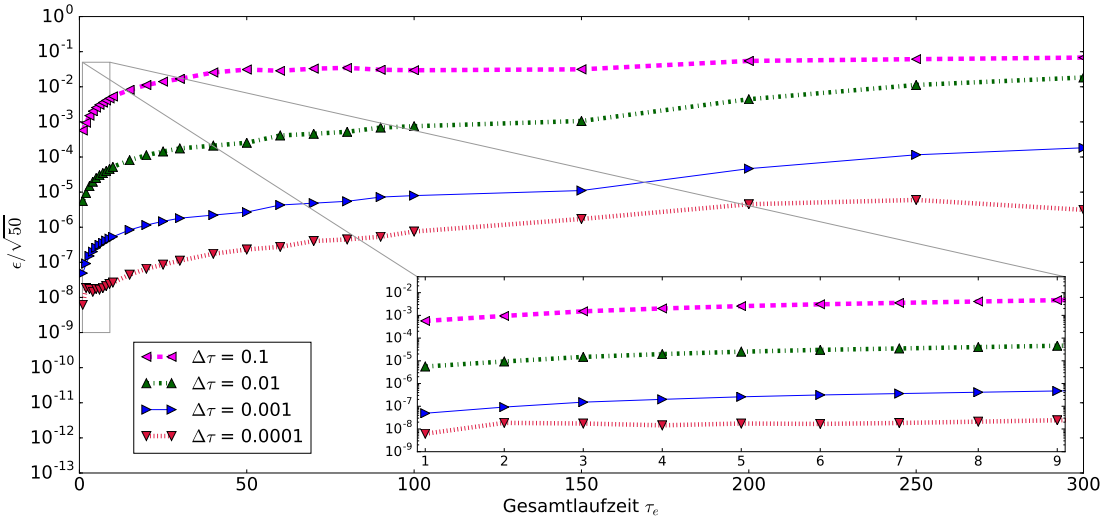


Abbildung 5.12: Aufgetragen sind die Abweichungen zur Python-Routine $\epsilon/\sqrt{50}$ der diskretisierten Differentialgleichung für 50 Pendel und einer Soliton-Anfangsbedingung für vier verschiedene $\Delta\tau$ zur jeweiligen Laufzeit τ_e. Der Zeitabschnitt 1 bis 9 wird vergrößert dargestellt.

Ein Vergleich zwischen den Anfangsbedingungen Soliton (Abbildung 5.12) und ein Pendel um $\pi/4$ ausgelenkt (Abbildung 5.13) zeigt, dass letztere geringere Fehler aufweist. Dadurch, dass die Solitonenanfangsbedingung alle Pendel stark auslenkt, ist dieses System fehleranfälliger. Der Vergleich zwischen 50 und 100 Pendeln mit einem Pendel zu Beginn ausgelenkt zeigt, dass der Fehler scheinbar unabhängig von der Anzahl der Pendel ist. Bis auf einen Faktor von $\sqrt{2}$ unterscheiden sie sich nicht.

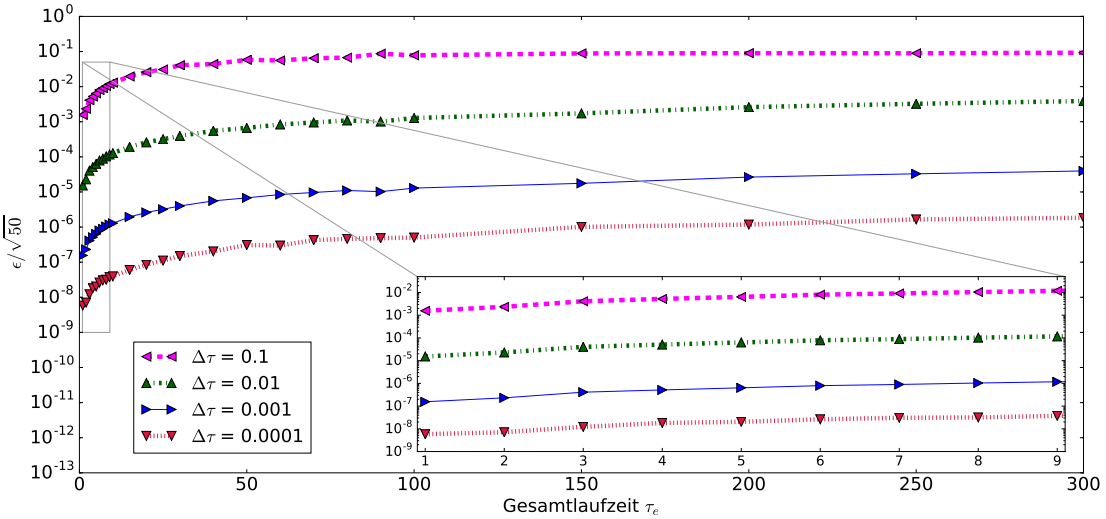
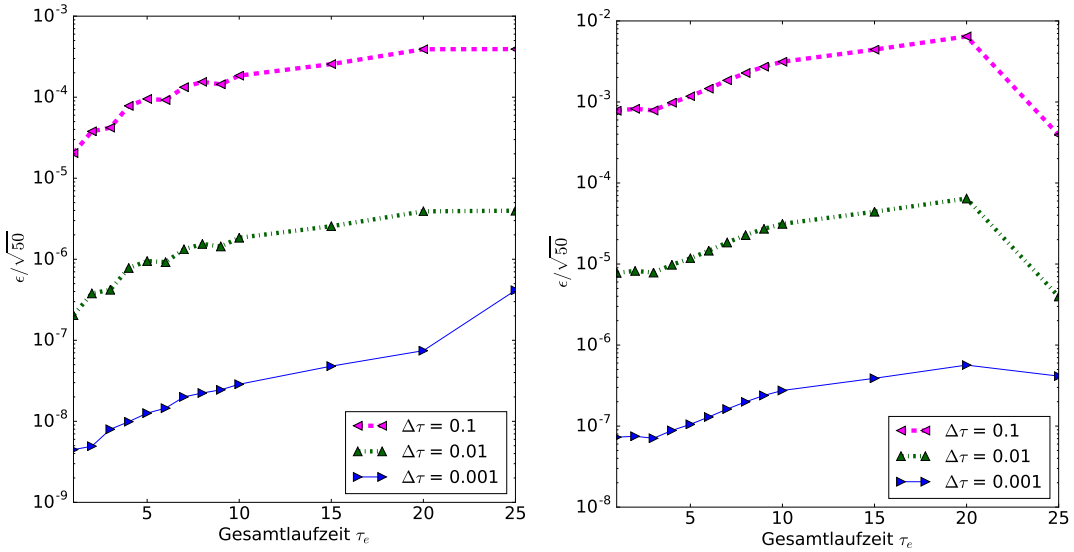


Abbildung 5.13: Aufgetragen sind die Abweichungen zur Python-Routine $\epsilon/\sqrt{50}$ der diskretisierten Differentialgleichung für 50 Pendel und Anfangsbedingung, bei der ein Pendel um $\pi/4$ ausgelenkt wird, für vier verschiedene $\Delta\tau$ zur jeweiligen Laufzeit τ_e. Der Zeitabschnitt 1 bis 9 wird vergrößert dargestellt.

Es wird deutlich, dass die Solitonenanfangsbedingung in Abbildung 5.12 größere Fehler aufweist als in Abbildung 5.13, bei der nur ein Pendel zu Beginn ausgelenkt wird. Außerdem nähert sich die Abweichung zur Python-Routine asymptotisch einem maximalen Fehler an. Im Gegensatz zur Zeitdiskretisierung, hängt die Abweichung des Iterationsverfahrens zur Python-Routine stark von der Anfangsbedingung ab. Ein Vergleich von Abbildung 5.14a und 5.14b zeigt, dass die Abweichungen sich um einen Faktor 100 unterscheiden. Die Solitonenanfangsbedingung weicht stärker ab, als die, bei der nur ein Pendel um $\pi/4$ ausgelenkt wird.



(a) Anfangsbedingung, bei der ein Pendel um $\pi/4$ ausgelenkt wird

(b) Soliton-Anfangsbedingung

Abbildung 5.14: Aufgetragen sind die Abweichungen zur Python-Routine $\epsilon/\sqrt{50}$ des Schwingungsansatzes mit Variation der Konstanten für 50 Pendel und zwei verschiedenen Anfangsbedingungen für drei verschiedene $\Delta\tau$ zur jeweiligen Laufzeit τ_e.

Zudem weicht das Iterationsverfahren um ein bis zwei Größenordnungen weniger von der Python-Routine ab, als die Zeitdiskretisierung. Allerdings konvergiert es nur bis $\tau_e = 25.0$.

6. Zusammenfassung und Ausblick

Das Ergebnis dieser Arbeit ist ein sehr umfrangreiches Python-Programm, welches eine entlang der Aufhängung gekoppelte, Pendelkette in verschiedenen Formen und mit variablen Anfangsbedingungen numerisch berechnet und live visualisieren und speichern kann.

Der mathematisch hergestellte Zusammenhang für unendlich viele Pendel mit der Sine-Gordon-Gleichung kann anhand der Simulation beobachtet und bestätigt werden. Solitonen breiten sich auf einer Pendelkette aus und überlagern sich mit anderen Solitonenlösungen störungsfrei. Auch Reibungseffekte können mit dem Programm untersucht werden.

Ein Vergleich der drei verschiedenen vorgestellten Methoden zeigt, dass das Iterationsverfahren sehr nahe an die Python- beziehungsweise Fortran-Routine herankommt und selbst bei großem $\Delta\tau = 0.1$ weniger als drei Größenordnungen abweicht. Allerdings konvergiert es nur für eine begrenzte Zeit. Grund hierfür sind in dem Paper [4] eingeführten Größen, deren Werte angeben, ob das System konvergieren wird. Diese könnten fortführend genauer untersucht werden. Mit der Zeitdiskretisierung weichen die Werte zwar um ein bis zwei Größenordnungen mehr von der Python-Routine ab, aber auch für sehr große Zeiten können noch Winkel berechnet werden. Für große Zeiten bleiben die Abweichungen zur Python-Routine für großes $\Delta\tau = 0.1$ bei weniger als 10^{-1}. Der große Vorteil der Zeitdiskretisierung gegenüber dem Iterationsverfahren ist die benötigte Berechnungszeit der Werte. Diese liegt für die Zeitdiskretisierung um ein bis zwei Größenordnungen unter der für das Iterationsverfahren. Für große $\Delta\tau$ ist die Zeitdiskretisierung sogar um den Faktor 10 schneller als die Python-Routine.

Gegenstand ergänzender Untersuchungen zu dieser Arbeit könnten beispielsweise verschiedene Medien oder Kopplungen sein. Außerdem könnten die Definitionsbereiche der Solitonenlösungen genauer untersucht und auf die Pendel, insbesondere für den Pendelkreis mit periodischen Randbedingungen, angewendet werden. Da alle Klassen des Programms einfach zu erweitern sind, können weitere Anfangsbedingungen und Verfahren untersucht werden.

Literatur

[1] https://www.youtube.com/watch?v=SAbQ4MvDqEE, letzter Zugriff 19. April 2018

[2] https://www.youtube.com/watch?v=uktMdcYI_vk, letzter Zugriff 19. April 2018

[3] Dr. S. Gurevich. Numerische Methoden für komplexe Systeme. Kapitel 5: Sine-Gordon-Gleichung. 2013/14.
https://www.uni-muenster.de/Physik.TP/archive/typo3/en/menu/teaching/archiv/numerische-methoden-fuer-komplexe-systeme-ws-1314.html, letzter Zugriff 19. April 2018

[4] H. Temimi und A. Ansari. A computational iterative method for solving nonlinear ordinary differential equations. *LMS J. Comput. Math.*, 18(1), 2015.

[5] A. C. Hindmarsh. Brief Description of ODEPACK.2001.
http://www.netlib.org/odepack/opkd-sum, letzter Zugriff 19. April 2018

[6] P. N. Brown, A. C. Hindmarsh.
http://www.netlib.org/ode/vode.f, letzter Zugriff 19. April 2018

A. Anhang

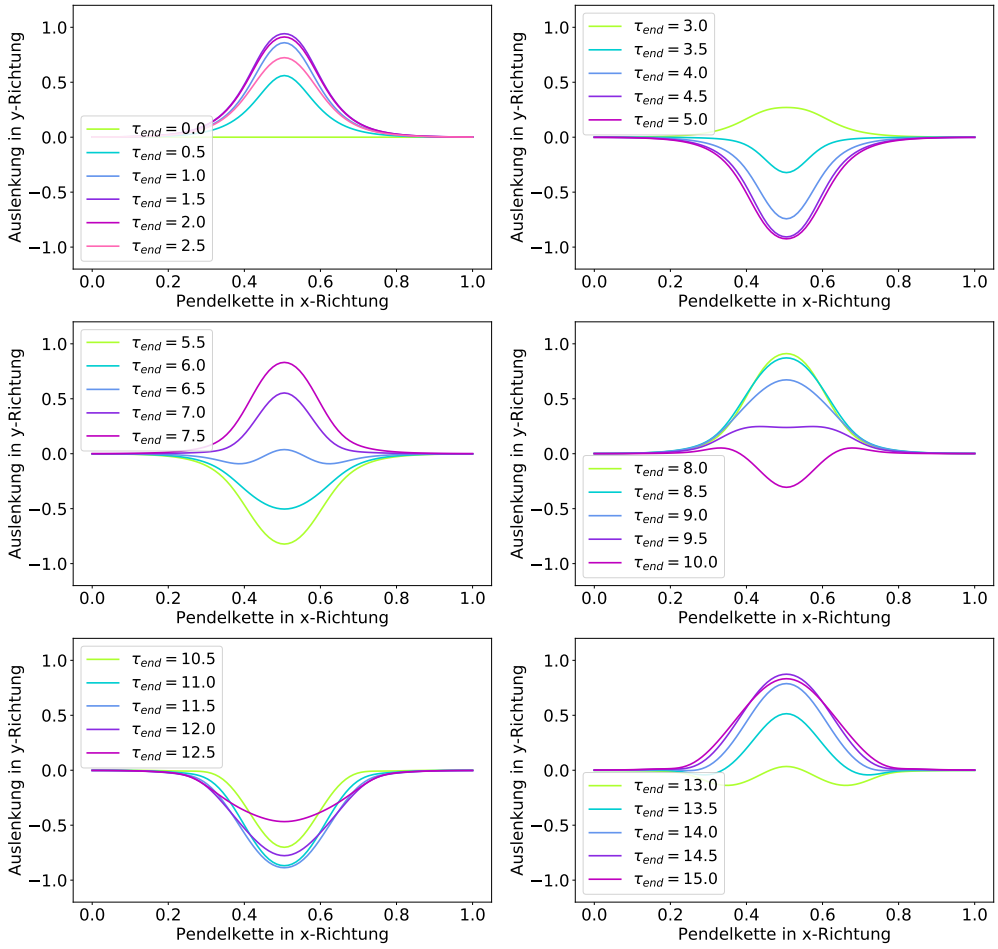


Abbildung A.1: Zeitentwicklung der Pendelauslenkung in y-Richtung mit einer Breather-Funktion $\varphi_B(x,t) = 4\arctan(\sqrt{1-0.95^2}\sin(0.95t)/0.95\cosh(\sqrt{1-0.95^2}50(x-0.5)))$ als Anfangsbedingung ohne Reibung mit $\nu^2 \simeq 6.371$ und $\Delta\tau = 0.0001$.

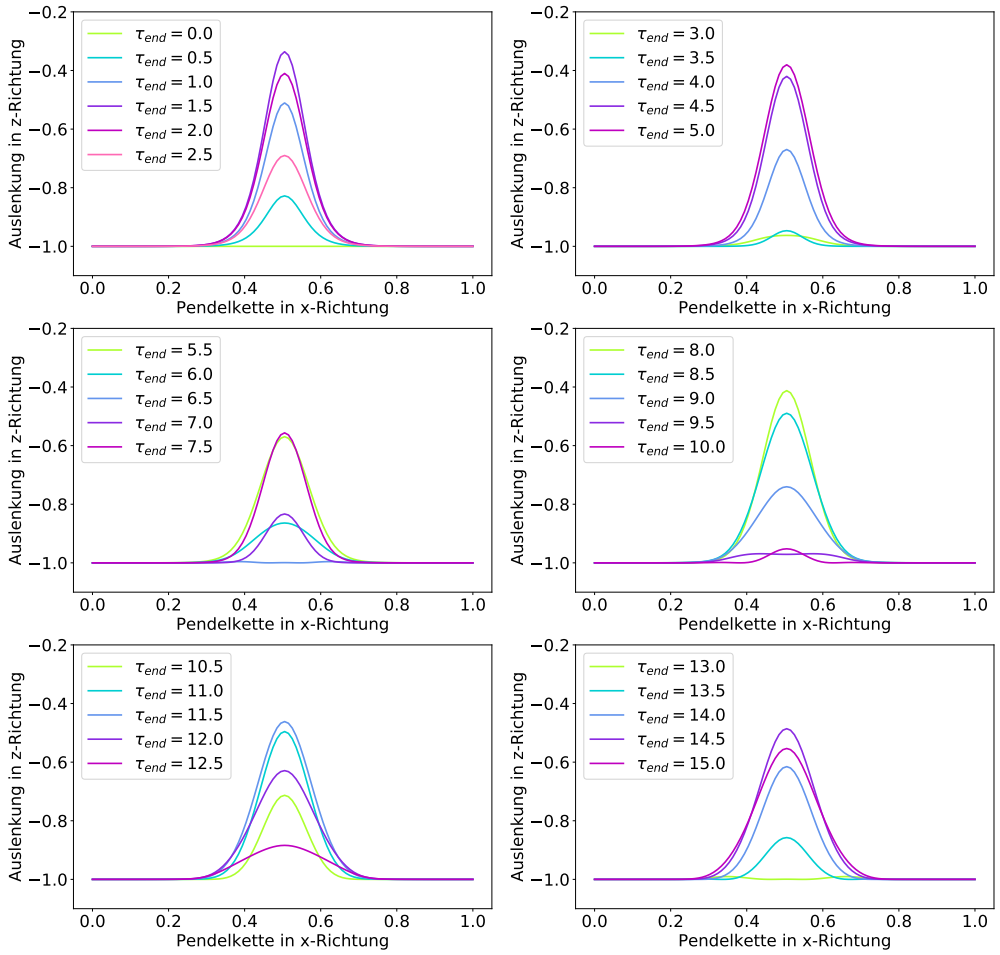


Abbildung A.2: Zeitentwicklung der Pendelauslenkung in z-Richtung mit einer Breather-Funktion $\varphi_B(x,t) = 4\arctan(\sqrt{1-0.95^2}\sin(0.95t)/0.95\cosh(\sqrt{1-0.95^2}50(x-0.5)))$ als Anfangsbedingung ohne Reibung mit $\nu^2 \simeq 6.371$ und $\Delta\tau = 0.0001$.

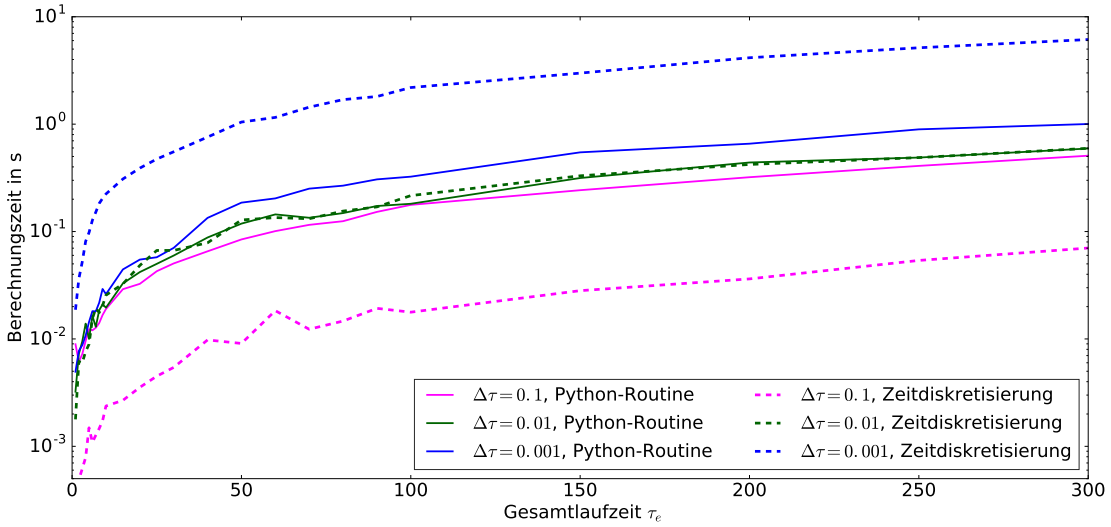


Abbildung A.3: Aufgetragen sind die benötigten Berechnungszeiten für 50 Pendel in Abhängigkeit von der Gesamtzeit τ_e für drei verschiedene Schrittweiten $\Delta\tau$ für die Python-Routine und der Zeitdiskretisierung, mit einer Solitonenfunktion als Anfangsbedingung.

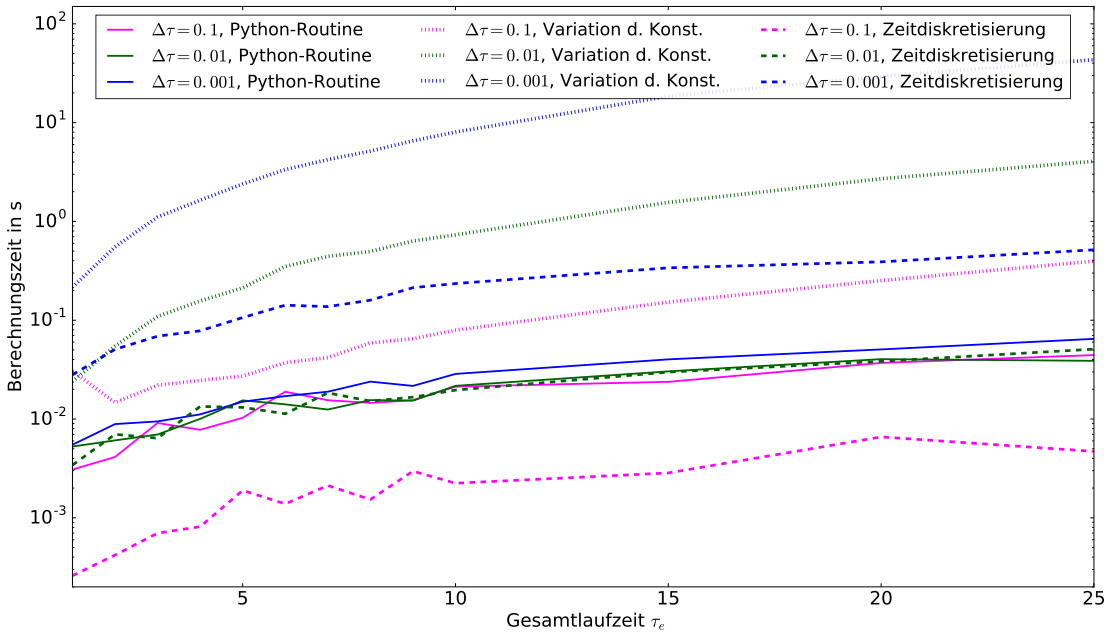


Abbildung A.4: Aufgetragen sind die benötigten Berechnungszeiten für 50 Pendel in Abhängigkeit von der Gesamtzeit τ_e für drei verschiedene Schrittweiten $\Delta\tau$ für die Python-Routine, die Zeitdiskretisierung und Variation der Konstanten, wobei zu Beginn ein Pendel um $\pi/4$ ausgelenkt wird.

Auflistung 1: Klasse zur Berechnung der Anfangsauslenkung und -geschwindigkeit

```
import numpy as np

class StartFunc(object):
    def __init__(self, Faktor, cond):
        """Initialisierung der Klasse zur Berechnung der
        Anfangsauslenkung und -geschwindigkeit von phi

        :param Faktor: Parameter aus der DGL
        :param cond: Anfangsbedingung
        """
        self.Condition = {
            'Soliton': cond,
            'NewKink-Kink': cond,
            'Antisoliton': cond,
            'Kink-Kink': cond,
            'Breather': cond,
            'EinPendel': cond,
            'HalfPendel': cond
        }
        self.Condition = self.Condition.get(cond, False)
        if(self.Condition==False):
            raise NameError(cond + ' ist keine gueltige Anfangsbedingung.')
        self.factor = Faktor
        self.c = 0.85*np.sqrt(self.factor)
        self.om = 0.95

    def phi_null(self, x):
        """Berechnet fuer die gegebenen x Werte die
        Anfangsauslenkung phi zum Zeitpunkt t = 0

        :param x: x Werte
        :return: array mit phi(0)
        """
        # Anfangsbed. ein Pendel ausgelenkt
        ret = np.zeros(len(x))
        ret[0] = np.pi / 4


        pen = np.zeros(len(x))
        for i in range(0, round(len(x)/2)):
            pen[i] = 3*np.pi/5


        result = {
            'Soliton': (4 * np.arctan(np.exp((x-0)*len(x)/2
                                             / (np.sqrt(self.factor - self.c**2))))),
            'Antisoliton': (- 4 * np.arctan(np.exp((x-x[len(x)-1])*len(x)/2
                                                   / (np.sqrt(self.factor - self.c**2))))),
            'Kink-Kink': 4*np.arctan(self.c*np.sinh((x-(2*x[-1]-x[-2])/2)*len(x)/2
                                                    /np.sqrt(self.factor-self.c**2))),
```

```
            'Breather': np.zeros(len(x)),
            'EinPendel': ret,
            'HalfPendel': pen
        }
        return result.get(self.Condition, 'Error')

    def w(self, x):
        """Berechnet fuer die gegebenen x Werte die Anfangsgeschwindigkeit
        fuer t = 0

        :param x: x Werte
        :return: array mit d/dt phi(0)
        """
        result = {
            'Soliton': (-2 * self.c / (np.sqrt(self.factor - self.c ** 2))
                        / (np.cosh((x-0)*len(x)/2 / (np.sqrt(self.factor - self.c ** 2))))),
            'Antisoliton': (+2 * self.c / (np.sqrt(self.factor - self.c ** 2))
                            / (np.cosh((x-x[len(x)-1])*len(x)/2
                                       / (np.sqrt(self.factor - self.c ** 2))))),
            'Kink-Kink': np.zeros(len(x)),
            'Breather': 4*np.sqrt(1-self.om**2)
                        / (np.cosh(np.sqrt(1-self.om**2)*(x-(2*x[-1]-x[-2])/2)*len(x)/2)),
            'EinPendel': np.zeros(len(x)),
            'HalfPendel': np.zeros(len(x))
        }
        return result.get(self.Condition, 'Error')
```

Auflistung 2: Klasse zur impliziten numerischen Berechnung der DGL für endlich viele gekoppelte Pendel

```
from scipy import integrate
import numpy as np


class PythPendelSim(object):

    def __init__(self, Anzahl, sFphi0, sFw0, matrix, time_number, time_steps, Reibung=0):
        """Initializer

        :param Anzahl: der Pendel
        :param sFphi0: liefert Anfangsauslenkung von phi
        :param sFw0: liefert Anfangsgeschwindigkeit von phi
        :param matrix: Matrix aus der DGL
        :param time_number: Anzahl der Zeitschritte
        :param time_steps: diskretes Zeitintervall
        :param Reibung: Reibungskoeffizient aus der DGL
        """
        self.n = Anzahl

        # Anfangsauslenkung
        self.phi0 = sFphi0
        # Anfangsgeschwindigkeit
        self.phi0 = np.append(self.phi0, sFw0)
```

```
24
25          self.etha = matrix
26          self.Lambda = Reibung
27
28
29          # Intervall fuer die numerische Berechnung
30          self.time = np.linspace(0, time_steps*time_number, time_number+1)
31
32
33      def dgl(self, phi_array, dphi_array):
34          """Berechnet die Differentialgleichung
35
36          :param phi_array: Winkel der n Pendel
37          :param dphi_array: Winkelgeschwindigkeit der n Pendel
38          :return: d^2/dt^2 phi_n fuer n = 1,...,N
39          """
40          res = - np.sin(phi_array) \
41              + np.dot(self.etha, phi_array)-self.Lambda*dphi_array
42          return res
43
44
45      def deriv(self, y, t):
46          """Differentialgleichungssystem 1. Ordnung
47          von d/dt phi_n fuer n = 1,..,2N
48
49          :param y: Variablen des Differentialgleichungssystems
50          :param t: Zeit, wird automatisch mit uebergeben
51          :return: erste Ableitung von y
52          """
53          end = int(self.n)
54          # d/dt phi_n fuer n = N+1,...,2N
55          phis = y[0:end]
56          # d/dt phi_n fuer n = 1,...,N
57          dphis = y[end:]
58          return np.append(dphis, self.dgl(phis, dphis))
59
60
61      def solver(self):
62          """ Loest das DGL System mit LSODA aus FORTRAN
63
64          :return: Matrix, Spalten entsprechen den Pendeln
65                  Reihen der Zeitentwicklung
66          """
67          sol = integrate.odeint(self.deriv, self.phi0, self.time)
68          return sol
```

Auflistung 3: Klasse, die wahlweise die Python-Routine, Zeitdiskretisierung oder das iterative Verfahren verwendet, um endlich viele Pendel zu simulieren.

```
1  import matplotlib
2  from mpl_toolkits.mplot3d import Axes3D
3  import mpl_toolkits.mplot3d.art3d as art3d
4  import matplotlib.pyplot as plt
```

```
5  from matplotlib import animation
6  import numpy as np
7  import threading
8
9  # eigene Klassen
10 from StartFunc import StartFunc
11 from PythPendelSim import PythPendelSim
12 from DiskPendelSim import DiskPendelSim
13 from IterativePendelSim import IterativePendelSim
14
15
16 class PendelNumerisch:
17
18     def __init__(self, Anzahl, dtau, Zeitschritte, Plotanteil=1,
19                  Startbedingung = 'Soliton', Methode = 'Python', Form='Linear',
20                  Kettenlaenge=1, Federkonst=5, Pendellaenge=0.5, Masse=0.0001,
21                  Ortsfaktor=9.81, Reibungsfaktor=0
22                  ):
23         """Initialisiert die Parameter fuer die Simulierung
24
25
26         :param Anzahl: der dargestellten Pendel
27         :param dtau: diskretisierte Zeit dt
28         :param Zeitschritte: Anzahl der Zeitschritte
29         :param Kettenlaenge: Laenge der Pendelkette
30         :param Federkonst: der Feder
31         :param Pendellange: Laenge des Pendels
32         :param Masse: der Pendel
33         :param Ortsfaktor:  Fallbeschleunigung g
34         :param Kopplung: der Feder mit den Pendeln
35         :param Startbedingung: Wahl der Anfangsauslenkung und -geschwindigkeit
36         :param Methode: zur Loesung der DGL
37         :param Form: Kette oder Kreis
38         :rtype: object
39         Werte die oben in __init__ schon zugewiesen
40         werden, sind Default Werte
41         """
42
43
44
45         # bisherige Grenze liegt bei knapp 150 Pendeln
46         # bei mehr Pendeln ist die Animation sehr langsam
47
48         self.dt = dtau
49         self.Zeit = Zeitschritte
50         self.Plotanteil = Plotanteil
51         self.d = Kettenlaenge
52
53         if(Anzahl>0):
54             self.n = Anzahl
55         else:
56             raise ValueError('Anzahl der Pendel muss groesser als 0 sein.')
57
```

```
58          self.g = Ortsfaktor
59          self.l = Pendellaenge
60          self.alpha = 0.05*self.l
61          self.sigma = self.alpha/self.l
62          self.k = Federkonst
63          self.m = Masse
64          self.Gamma = Reibungsfaktor
65
66          self.kappa = self.k * self.d
67          self.rho = self.m / self.d
68
69          self.Form = Form
70
71          '''neu eingefuehrte Parameter'''
72
73          # Werte fuer die Differentialgleichung
74          self.Omega_g_quad = self.g / self.l / (self.sigma**2 + 1)
75          self.Omega_k_quad = self.k / self.m * self.sigma**2 / (self.sigma**2 + 1)
76          self.dgl_param = self.Omega_k_quad/self.Omega_g_quad
77
78          # Matrix etha_nl definieren
79          self.etha = np.zeros((self.n, self.n))
80          if(self.n > 1):
81              self.etha[self.n - 1][self.n - 2] = 1
82              self.etha[0][1] = 1
83
84          for i in range(0, self.n):
85              if (not (i == 0 or i == self.n - 1)):
86                  self.etha[i][i] = -2
87                  self.etha[i][i - 1] = 1
88                  self.etha[i][i + 1] = 1
89              else:
90                  self.etha[i][i] = -1
91
92          if(self.Form=='Kreis'):
93              K = np.zeros((self.n, self.n))
94              K[0][0] = -1
95              if (self.n > 1):
96                  K[self.n-1][self.n-1] = -1
97                  K[0][self.n - 1] = 1
98                  K[self.n - 1][0] = 1
99              self.etha = self.etha + K
100         self.etha = self.etha * self.dgl_param
101
102         self.AUSWAHL = Startbedingung
103         self.meth = Methode
104
105
106         # Klasse die Anfangsfunktionen phi_null und w liefert
107         self.sF = StartFunc(self.dgl_param, self.AUSWAHL)
108
109         self.mylines = np.array([])
110         self.mypoints = np.array([])
```

```
111
112         self.pause = False
113         self.p = 0
114         self.ani = None
115
116
117
118         "Parametrisierung der Pendel zur Simulierung der DGL fuer " \
119         "endlich viele Pendel"
120
121         if (self.Form == 'Linear'):
122             self.Kette = np.array(range(0, self.n)) * self.d / self.n
123             self.phi0_array = self.sF.phi_null(self.Kette)
124             self.w0_array = self.sF.w(self.Kette)
125             self.XS = self.Kette + self.alpha*self.phi0_array
126             self.ZS = np.array(-self.l * np.cos(self.phi0_array))
127             self.YS = np.array(self.l * np.sin(self.phi0_array))
128             self.X = np.copy(self.XS)
129             self.Y = np.zeros(self.n)
130             self.Z = np.zeros(self.n)
131
132         elif(self.Form=='Kreis'):
133             self.beta_null = self.beta_init(self.n)
134             self.Kette = self.beta_null/(self.d/2*np.pi)
135             self.phi0_array = self.sF.phi_null(self.Kette)
136             self.w0_array = self.sF.w(self.Kette)
137             self.XS = (self.d/(2*np.pi) + self.l*np.sin(self.phi0_array)) \
138                       * np.sin(self.beta_null)
139             self.YS = -(self.d/(2*np.pi) + self.l*np.sin(self.phi0_array)) \
140                       * np.cos(self.beta_null)
141             self.ZS = - self.l * np.cos(self.phi0_array)
142             self.X = self.d/(2*np.pi) * np.sin(self.beta_null)
143             self.Y = -self.d/(2*np.pi) * np.cos(self.beta_null)
144             self.Z = np.zeros(self.n)
145
146         else:
147             raise NameError(self.Form + ' ist keine gueltige Form der Pendelkette')
148
149         if (self.meth == 'Python'):
150             # loest die DGL fuer endlich viele Pendel
151             self.solution = PythPendelSim(
152                 self.n, self.phi0_array, self.w0_array, self.etha, self.Zeit,
153                 self.dt, self.Gamma
154             ).solver()
155             self.T = len(self.solution)
156
157         elif (self.meth == 'Diskret'):
158             # loest die DGL iterativ fuer endlich viele Pendel
159             self.solution = np.transpose(DiskPendelSim(
160                 self.n, self.phi0_array, self.w0_array, self.etha, self.Zeit,
161                 self.dt, self.Gamma
162             ).startComp())
163             print("computing was finished")
```

```
164             self.T = len(self.solution)
165
166         elif (self.meth == 'Iterative'):
167             self.solution = np.transpose(IterativePendelSim(
168                 self.n, self.phi0_array, self.w0_array, self.etha, self.Zeit,
169                 self.dt, self.Gamma
170             ).startIter())
171             self.T = len(self.solution)
172
173         else:
174             raise NameError(self.meth + ' ist keine gueltige Methode.')
175
176
177
178         ''' Einstellungen des Plots, Achsen usw '''
179
180         self.fig = plt.figure()
181         self.ax = Axes3D(self.fig)
182         #self.ax.axes.set_pane_color(255,255,255,0)
183         self.ax.w_xaxis.set_pane_color((1, 1, 1, 0))
184         self.ax.w_yaxis.set_pane_color((1, 1, 1, 0))
185         self.ax.w_zaxis.set_pane_color((1, 1, 1, 0))
186
187         self.ax.grid(False)
188
189         self.ax.set_xlabel('x-Achse')
190         self.ax.set_ylabel('y-Achse')
191         self.ax.set_zlabel('z-Achse')
192
193         #self.ax.set_axis_off()
194
195         if(self.Form=='Linear'):
196             self.ax.set_xlim(-0.1, self.d+0.1)
197             self.ax.set_ylim(-self.l - 0.2, self.l + 0.2)
198             self.ax.set_zlim(-self.l - 0.2, self.l + 0.2)
199             self.ax.plot(
200                 [0, self.Kette[-1]], [0, 0], [0, 0], c='sienna', ls='-', lw=3
201             )
202
203         elif(self.Form=='Kreis'):
204             r = self.d/(2*np.pi)
205             self.ax.set_xlim(-r - 0.2, r + 0.2)
206             self.ax.set_ylim(-r - 0.2, r + 0.1)
207             self.ax.set_zlim(-r - 0.2, r + 0.2)
208             p = matplotlib.patches.Circle(
209                 (0, 0), r, fill=False, color='sienna', lw=3
210             )
211             self.ax.add_patch(p)
212             art3d.pathpatch_2d_to_3d(p, z=0, zdir="z")
213             self.ax.plot(
214                 [self.X[-1], self.X[0]], [self.Y[-1], self.Y[0]], [0, 0],
215                 c='sienna', ls='-', lw=3
216             )
```

```
217
218
219     def beta_init(self, number):
220         """Berrechnet die Anfangswinkel beta
221         der Kreispendelkette
222
223         :param number: Anzahl der Pendel
224         :return: beta
225         """
226         vec_beta_null = np.array([])
227         for i in range(1, number + 1):
228             com_beta_null = (number - i) * 2 * np.pi / number
229             vec_beta_null = np.append(vec_beta_null, [com_beta_null])
230
231         return vec_beta_null
232
233     def initPlot(self):
234         """Plottet die Pendel fuer t = 0, initialisiert
235         mylines und mypoints: LineObjects deren Werte einzelnd
236         in loop() durch die Animation animate() neu gesetzt werden
237
238         """
239
240         if(len(self.mylines)==0):
241             for i in range(0, self.n):
242                 myline = self.ax.plot(
243                     [self.X[i], self.XS[i]], [self.Y[i], self.YS[i]],
244                     [self.Z[i], self.ZS[i]], c='indigo', marker='o'
245                 )
246                 self.mylines = np.append(self.mylines, [myline])
247         else:
248             self.loop(self.XS, self.YS, self.ZS)
249
250
251
252     def onClick(self, event):
253         """Funktion die es erlaubt, die Animation
254         per Klick zu pausieren
255
256         :param event: hier: Klicken
257         """
258         if self.pause:
259             self.ani.event_source.stop()
260             self.pause = False
261         else:
262             self.ani.event_source.start()
263             self.pause = True
264
265     def loop(self, newxs, newys, newzs, newx=None, newy=None):
266         """Ueberschreibt die Werte von mylines und mypoints
267         mit den neuen phi-Werten
268
269         :param newx: neue X Werte
```

```
270         :param newy: neue Y Werte
271         :param newz: neue Z Werte
272         """
273         if newy is None:
274             Y = self.Y
275             X = newxs
276         else:
277             X = newx
278             Y = newy
279 
280         for k in range(0, self.n):
281             self.mylines[k].set_xdata([X[k], newxs[k]])
282             self.mylines[k].set_ydata([Y[k], newys[k]])
283             self.mylines[k].set_3d_properties([self.Z[k], newzs[k]])
284 
285 
286     def animate(self, phi):
287         """aktualisiert die Winkel und plottet diese
288 
289         :param phi: Winkel zur jeweiligen zu plottenden Zeit
290         """
291 
292         vec_phi_tpluseins = phi[:self.n]
293 
294         if(self.Form=='Linear'):
295             newx = self.X + self.alpha * vec_phi_tpluseins
296             newys = np.array(self.l * np.sin(vec_phi_tpluseins))
297             newzs = np.array(-self.l * np.cos(vec_phi_tpluseins))
298             threading.Thread(
299                 target=self.loop, args=(newx, newys, newzs)
300             ).start()
301 
302         elif(self.Form=='Kreis'):
303             betanew = self.beta_null \
304                       + self.alpha/(self.d/(2*np.pi)) * vec_phi_tpluseins
305             newxs = +(self.d / (2 * np.pi) + self.l * np.sin(vec_phi_tpluseins))\
306                     * np.sin(betanew)
307             newys = -(self.d / (2 * np.pi) + self.l * np.sin(vec_phi_tpluseins))\
308                     * np.cos(betanew)
309             newzs = - self.l * np.cos(vec_phi_tpluseins)
310             newx = self.d / (2 * np.pi) * np.sin(betanew)
311             newy = -self.d / (2 * np.pi) * np.cos(betanew)
312             threading.Thread(
313                 target=self.loop, args=(newxs, newys, newzs, newx, newy)
314             ).start()
315 
316     def show(self):
317         """startet die Animation
318 
319         """
320 
321         self.fig.canvas.mpl_connect('button_press_event', self.onClick)
322 
```

```
323         try:
324             self.ani = animation.FuncAnimation(
325                 self.fig, self.animate, init_func=self.initPlot,
326                 frames=self.solution[::self.Plotanteil], interval=1, repeat=False,
327                 save_count=len(self.solution[::self.Plotanteil]), blit=False
328             )
329             plt.show()
330 
331         except AttributeError:
332             print('Simulation wurde beendet')
333 
334 
335 
336     def save(self, name='pendelMovie', dtype='ffmpeg'):
337         """Ermoeglicht das Speichern der zuvor gezeigten
338         Simulation
339 
340         :param name: Name der Datei
341         :param dtype: Dateityp zum Speichern
342         """
343         Writer = animation.writers['ffmpeg']
344         writer = Writer(metadata=dict(artist='Me'), bitrate=1800, fps=30)
345         self.ani.save(name + '.mp4', writer=writer)
346 
347         if(dtype=='html'):
348             file = open(name + '.html', "w")
349             file.write(
350                 '<!DOCTYPE html> <html> <head> <meta charset="utf-8"/>'
351                 '<title>Bachelor-Thesis Lea Otterbeck</title>'
352                 '</head> <body> <div align="center"> <h1>Simulation Pendelkette</h1>'
353                 '<video width="50%" controls> <source src="' + name + '.mp4' +
354                 '" type="video/mp4"/> Your browser does not support the video tag.'
355                 '</video> </div> </body> </html> '
356             )
357             file.close()
358 
359 
360 if "__main__" == __name__:
361 
362     pendel = PendelNumerisch(
363         Anzahl=50, dtau=0.01, Zeitschritte=10000, Plotanteil=1,
364         Startbedingung='Breather', Methode='Python', Form='Kreis',
365         Kettenlaenge=5, Federkonst=5, Pendellaenge=0.5, Masse=0.0001,
366         Ortsfaktor=9.81, Reibungsfaktor=0
367     )
368 
369     pendel.show()
370     #pendel.save(name='Soliton', dtype='mp4')
```

Auflistung 4: Klasse, welche das nichtlineare Differentialsystem für endlich viele Pendel durch Zeitdiskretisierung der Bewegungsgleichung numerisch löst.

```
import numpy as np


class DiskPendelSim(object):
    def __init__(self, Anzahl, phi_null, w_null, matrix, time_number, time_steps, Reibung=0):
        """Initializer

        :param Anzahl: der Pendel
        :param phi_null: liefert Anfangsauslenkung von phi
        :param w_null: liefert Anfangsgeschwindigkeit von phi
        :param matrix: aus der DGL
        :param time_number: Anzahl der Zeitschritte
        :param time_steps: diskretes Zeitintervall
        :param Reibung: Reibungskoeffizient aus der DGL
        """

        self.n = Anzahl
        self.phi0 = phi_null
        self.w0 = w_null

        self.etha = matrix
        self.R = Reibung
        self.Lambda = Reibung
        self.dtau = time_steps
        self.factor = self.dtau*self.Lambda/2

        self.NT = time_number


    def expDiskSim(self):
        """Explizite Berechnung von phi mit diskretisierter DGL
        nur fuer Stokesche Reibung

        :return: phi
        """

        phi = np.zeros((self.n, self.NT+1))
        phi[:, 0] = self.phi0
        phi[:, 1] = self.phi0 + self.dtau * self.w0 + self.dtau ** 2 / 2 \
                    * ( - np.sin(self.phi0) + np.dot(self.etha, self.phi0)
                        - self.Lambda * self.w0**self.R)
        for i in range(1, self.NT):
            phi[:, i + 1] = 1/(1 + self.factor) * \
                            (2 * phi[:, i] - (1-self.factor) * phi[:, i - 1]
                             + self.dtau ** 2 * (- np.sin(phi[:, i])
                                                 + np.dot(self.etha, phi[:, i])))
        return phi


```

```
50      def startComp(self):
51          """Startet die Berechnung von phi
52
53          :return: phi, wobei in den Reihen die Pendel und in den Spalten die Zeiten stehen
54          """
55          solphi = self.expDiskSim()
56
57          return solphi
```

Auflistung 5: Klasse, welche das nichtlineare Differentialsystem für endlich viele Pendel durch eine iterative Berechnung mit Variation der Konstanten numerisch löst.

```
1   import numpy as np
2
3
4   class IterativePendelSim(object):
5       def __init__(self, Anzahl, phi_null, w_null, matrix, time_number, time_steps, Reibung=0):
6           """Initializer
7
8           :param Anzahl: der Pendel
9           :param phi_null: liefert Anfangsauslenkung von phi
10          :param w_null: liefert Anfangsgeschwindigkeit von phi
11          :param matrix: aus der DGL
12          :param time_number: Anzahl der Zeitschritte
13          :param time_steps: diskretes Zeitintervall
14          :param Reibung: Reibungskoeffizient aus der DGL
15          """
16
17          self.n = Anzahl
18          self.phi0 = phi_null
19          self.w0 = w_null
20          self.etha = - matrix
21          self.R = Reibung
22          self.Lambda = Reibung
23          self.dtau = time_steps
24
25          self.NT = time_number
26          self.time = np.linspace(0, time_steps * time_number, time_number + 1)
27
28
29          self.EW, self.AT = np.linalg.eig(self.etha)
30          self.A = np.transpose(self.AT)
31          # Da ein Eigenwert 0 ist, liefert die Numerik
32          # teilweise sehr kleine aber negative Werte,
33          # daher wird der Betrag genommen
34          self.W = np.sqrt(np.abs(self.EW))
35          self.WTau = self.W[:, np.newaxis] * self.time[np.newaxis, :]
36          self.S = np.sin(self.WTau)
37          self.C = np.cos(self.WTau)
38
39          self.alpha_init = np.dot(self.A, self.phi0)
40          self.beta_init = np.dot(self.A, self.w0)
```

```
41
42
43      def nonLin(self, phi, dphi):
44          """Berrechnet den nichtlinearen Anteil
45
46          :param phi: Werte phi
47          :param dphi: Ableitung von phi
48          :return: nichtlinearer Anteil der DGL
49          """
50
51          return np.dot(self.A, (np.sin(phi) + self.Lambda*dphi))
52
53      def initGuess(self):
54          """Berrechnet die Startewerte der Iteration
55
56          :return: Startwerte fuer die Iteration
57          """
58          Phi_init = self.alpha_init[:, np.newaxis]*self.C \
59                     + self.beta_init[:, np.newaxis]*self.S/self.W[:, np.newaxis]
60          dphi_init = - self.phi0[:, np.newaxis]*self.S*self.W[:, np.newaxis] \
61                      + self.w0[:, np.newaxis]*self.C
62          phi_init = np.dot(self.AT, Phi_init)
63          return phi_init, dphi_init
64
65      def diskInt(self, f):
66          """Berrechnet numerisch die diskreten Integrale
67
68          :param f: Teil der zu iterierenden Funktion
69          :return: alpha und beta
70          """
71
72          int1 = self.S * f
73          int2 = -self.C * f
74          alpha = np.zeros((self.n, self.NT + 1))
75          beta = np.zeros((self.n, self.NT + 1))
76
77          for zeit in range(1, self.NT + 1):
78              alpha[:, zeit] = alpha[:, zeit - 1] + int1[:, zeit] + int1[:, zeit-1]
79              beta[:, zeit] = beta[:, zeit - 1] + int2[:, zeit] + int2[:, zeit-1]
80
81          alpha = alpha*self.dtau/2.0
82          beta = beta * self.dtau / 2.0 + self.beta_init[:, np.newaxis]
83
84          alpha = alpha / self.W[:, np.newaxis]
85          beta = beta / self.W[:, np.newaxis]
86          alpha = alpha + self.alpha_init[:, np.newaxis]
87
88          return alpha, beta
89
90
91      def mpeinsIteration(self, phim, dphim):
92          """Fuehrt den naechsten Iterationsschritt aus
93
```

```
94          :param phim: phi-Werte aus der vorherigen Iteration
95          :param dphim: Ableitung der phi-Werte
96          :return: phi und dphi aus dem aktuellen Iterationsschritt
97          """
98
99          AN = self.nonLin(phim, dphim)
100         alpha, beta = self.diskInt(AN)
101         Phimpluseins = self.C * alpha + self.S*beta
102         dPhimpluseins = self.W[:, np.newaxis]*(self.C*beta - self.S*alpha)
103         phimpluseins = np.dot(self.AT, Phimpluseins)
104         dphimpluseins = np.dot(self.AT, dPhimpluseins)
105
106         return phimpluseins, dphimpluseins
107
108     def startIter(self, itersteps=300):
109         """Startet die Iteration
110
111         :param itersteps: Anzahl der maximalen Iterationsschritte
112         :return: phi-Werte aus dem letzten Iterationsschritt
113         """
114
115         print('Initialisierung...')
116         phim, dphim = self.initGuess()
117
118         for iter in range(0, itersteps):
119             print('Starte Iteration...')
120             oldphim = np.copy(phim)
121             phim, dphim = self.mpeinsIteration(phim, dphim)
122
123             abs = np.max(np.absolute(oldphim - phim))
124             if(np.max(abs) < 10**(-5)):
125                 print('Iteration konvergierte nach ', iter, ' Schritten')
126                 self.iternumber = iter
127
128                 return phim
129
130             if(iter == itersteps-1):
131                 print('Iteration ist nicht konvergiert.')
132                 print('Zuletzt berechnete Werte werden zurueckgegeben '
133                       'und sind mit Vorsicht zu geniessen!')
134
135                 return phim
```

Name, Vorname: Otterbeck, Lea

Erklärung

Hiermit erkläre ich, dass ich die von mir eingereichte Abschlussarbeit (Bachelor-Thesis) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Stellen der Abschlussarbeit, die anderen Werken dem Wortlaut oder Sinn nach entnommen wurden, in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

....
Datum Unterschrift

Erklärung

Hiermit erkläre ich mich damit einverstanden, dass meine Abschlussarbeit (Bachelor-Thesis) wissenschaftlich interessierten Personen oder Institutionen und im Rahmen von externen Qualitätssicherungsmaßnahmen des Studiengangs zur Einsichtnahme zur Verfügung gestellt werden kann. Korrektur- oder Bewertungshinweise in meiner Arbeit dürfen nicht zitiert werden.

....
Datum Unterschrift