PVK Probeprüfung FS 2017

Lucas Böttcher
Numerische Methoden
ETH Zürich

June 23, 2017

1. Radioaktiver Zerfall

Gegeben sind zwei radioaktive Substanzen, welche mit den Raten $\lambda_1 = 0.5$ und $\lambda_2 = 0.1$ zerfallen:

$$A \xrightarrow{\lambda_1} B \xrightarrow{\lambda_2} C.$$

Die Ratengleichungen für die Mengen Φ_A, Φ_B an A und B sind die Folgenden:

$$\begin{pmatrix} \dot{\Phi}_A(t) \\ \dot{\Phi}_B(t) \end{pmatrix} = \begin{pmatrix} -\lambda_1 \Phi_A(t) \\ \lambda_1 \Phi_A(t) - \lambda_2 \Phi_B(t) \end{pmatrix}. \tag{1}$$

(a) Löse das Ratengleichungssystem (1) mit der Variation der Konstanten. Die Anfangswerte seien $\Phi_A(0)$ und $\Phi_B(0)$.

Wir stellen fest, dass $\Phi_A(t) = \Phi_A(0)e^{-\lambda_1 t}$ und wählen den Ansatz $\Phi_B(t) = c(t)e^{-\lambda_2 t}$. Dieser Ansatz gibt $c(t) = \frac{\lambda_1}{\lambda_2 - \lambda_1}\Phi_A(0)e^{-(\lambda_1-\lambda_2)t}$. Wir addieren dann die homogene Lösung und die spezielle Lösung. Für die gegebene Anfangsbedingung erhält man somit $\Phi_B(t) = \left(\Phi_B(0) - \frac{\lambda_1}{\lambda_2-\lambda_1}\Phi_A(0)\right)e^{-\lambda_2 t} + \frac{\lambda_1}{\lambda_2-\lambda_1}\Phi_A(0)e^{-\lambda_1 t}$.

(b) Implementiere eine Pythonfunktion $g(\mathbf{z})$, welche die rechte Seite von Gleichung (1) berechnet. Dabei bezeichnet $\mathbf{z}$ den Vektor der aktuellen Φ-Werte.

siehe Code **radioaktiver_zerfall.py**.

(c) Nutze die **ode45** Funktion, um die Ratengleichungen (1) für $t \in [0, 15]$, $\Phi_A(0) = 1$ und $\Phi_B(0) = 0.5$ zu lösen. Plotte die berechneten Lösungen.

siehe Code **radioaktiver_zerfall.py**.

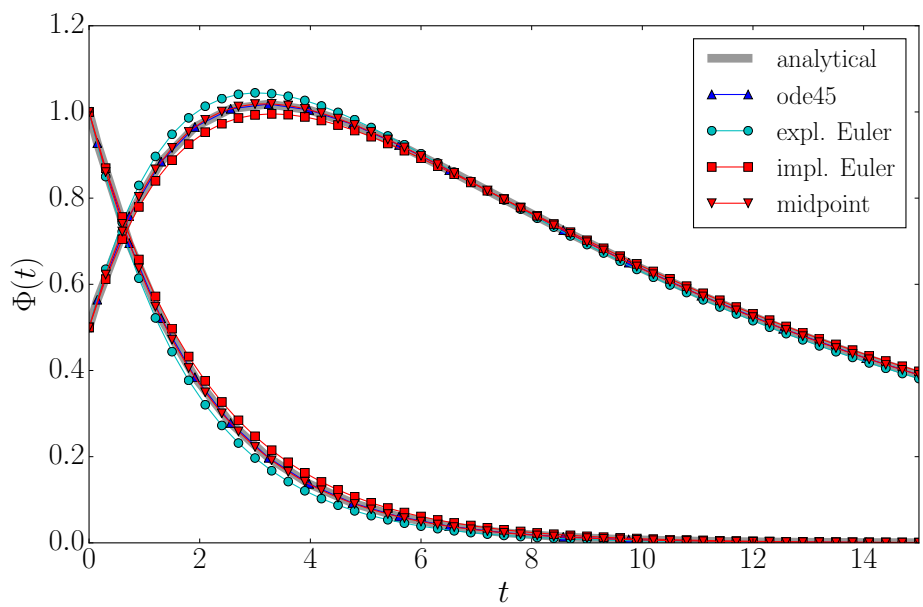
(d) Nun berechnen wir die Lösungen numerisch mit der expliziten und impliziten Eulermethode sowie der impliziten Mittelpunktsregel. Die Funktionen geben den Zeitvektor $\mathbf{t}$

und die numerisch berechneten Funktionswerte **y** in einem Tupel (**t**, **y**) zurück. Die Funktionsargumente sind: die rechte Seite **g**, Anfangs- und Endzeiten **t0**, **tend**, der Anfangswert **y0** und die Anzahl der Intervalle **N**.

Name	Funktion
explizites Eulerverfahren	**expleuler(g,t0,tend,y0,N)**
implizites Eulerverfahren	**impleuler(g,t0,tend,y0,N)**
Mittelpunktsregel	**mittelpkt(g,t0,tend,y0,N)**

siehe Code **radioaktiver_zerfall.py**.

(e) Plotte die Lösungen für die verschiedenen numerischen Verfahren mit $t \in [0, 15]$, $N = 50$, $\Phi_A(0) = 1$ und $\Phi_B(0) = 0.5$ und vergleiche diese mit der **ode45** und der analytischen Lösung.



Die **ode45** und die Mittelpunktsmethode liegen sehr nahe an der analytischen Lösung. Hingegen sind das explizite und implizite Eulerverfahren nicht so genau.

2. Approximation einer Ellipse

Gegeben seien $N = 6$ Punkte, die näherungsweise auf einer Ellipse liegen:

x_i	-0.1	6.7	6.7	-0.2	-7.3	-6.5
y_i	3.1	1.4	-1.0	-2.6	-1.1	1.3

.

Die Ellipsengleichung mit den Parametern a und b ist:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1. \tag{2}$$

Als Abweichungsmass d_i der Punkte (x_i, y_i) von der Ellipsengleichung (2) nutzen wir die Abweichung des Wertes $\frac{x_i^2}{a^2} + \frac{y_i^2}{b^2}$ von 1.

Wir wollen die Parameter a und b finden, welche die Summe der quadratischen Abweichungen $\sum_i d_i^2$ minimiert.

(a) Gebe die Gleichung für die Abweichung d_i sowie die der Funktion $\mathbf{F}\colon \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$ für das nichtlineare Ausgleichsproblem an. Was sind die Werte von n_1 und n_2?

Die Abweichung ist $d_i = \left|\frac{x_i^2}{a^2} + \frac{y_i^2}{b^2} - 1\right|$ und wir wollen die Summe $\sum_i d_i^2$ minimieren. Der zweidimensionale Parametervektor ist $\mathbf{z} = (z_0, z_1)^T = (a, b)^T$ und dementsprechend ist $\mathbf{F} : \mathbb{R}^2 \to \mathbb{R}^N$. In Komponenten ausgeschrieben findet man $\mathbf{F}_i(\mathbf{z}) = \frac{x_i^2}{z_0^2} + \frac{y_i^2}{z_1^2} - 1$.

(b) Definiere das Funktional

$$\Phi(\mathbf{z}) := \frac{1}{2}\|\mathbf{F}(\mathbf{z})\|^2.$$

Berechne die Jacobimatrix $\mathbf{DF}$ von $\mathbf{F}$ sowie den Gradient und die Hessematrix von Φ.

Das Funktional ist $\Phi(\mathbf{z}) = \frac{1}{2}\sum_{i=1}^N \left(\frac{x_i^2}{z_0^2} + \frac{y_i^2}{z_1^2} - 1\right)^2$. Für das Gauss-Newton-Verfahren brauchen wir die Jacobimatrix. Diese ist gerade gegeben durch

$$\mathbf{DF}(\mathbf{z}) = \begin{pmatrix} \nabla F_1(\mathbf{z})^T \\ \vdots \\ \nabla F_N(\mathbf{z})^T \end{pmatrix} \in \mathbb{R}^{N,2},$$

wobei $\nabla F_i(\mathbf{z}) = -2\begin{pmatrix} \frac{x_i^2}{z_0^2} & \frac{y_i^2}{z_1^3} \end{pmatrix}^T$. Nun berechnen wir den Gradienten und die Hessematrix von Φ, welche wir für das Newton-Verfahren brauchen. Der Gradient ist

$$\nabla\Phi(\mathbf{z}) = \begin{pmatrix} \frac{\partial\Phi}{\partial z_0} \\ \frac{\partial\Phi}{\partial z_1} \end{pmatrix}$$

und kann verschieden in Python implementiert werden, cf. Code **ellipse.py**. Und die Hessematrix ist dann

$$\mathbf{H}\Phi(\mathbf{z}) = \begin{pmatrix} \frac{\partial^2\Phi}{\partial z_0^2} & \frac{\partial^2\Phi}{\partial z_1 \partial z_0} \\ \frac{\partial^2\Phi}{\partial z_0 \partial z_1} & \frac{\partial^2\Phi}{\partial z_1^2} \end{pmatrix}$$

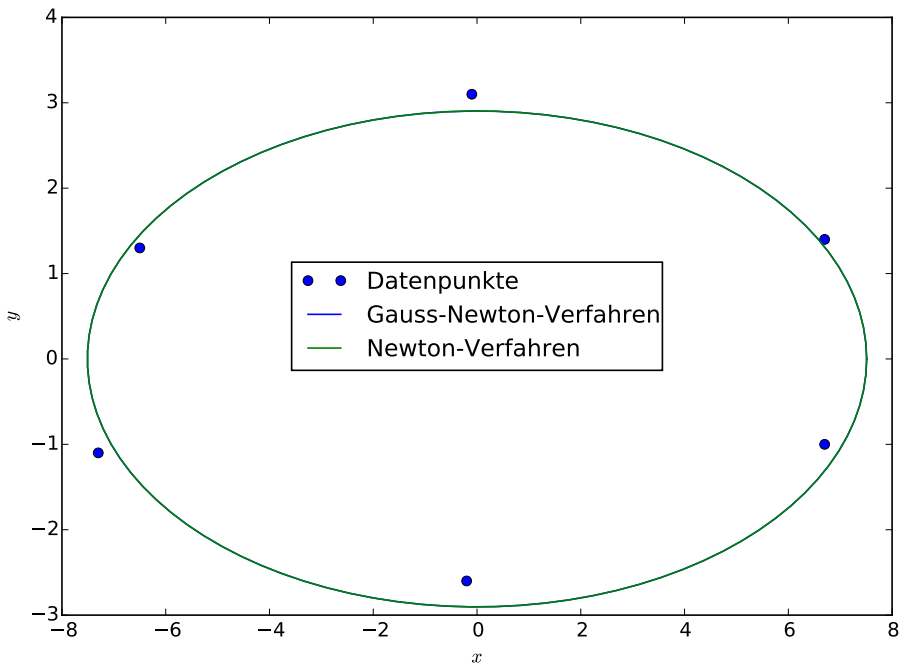
und kann auch wie der Gradient entweder noch analytisch vereinfacht werden oder direkt mit Summen in Python implementiert werden, cf. Code **ellipse.py**.

(c) Implementiere ein Gauss-Newton Verfahren in Python, welches das Funktional Φ minimiert.

siehe Code **ellipse.py**.

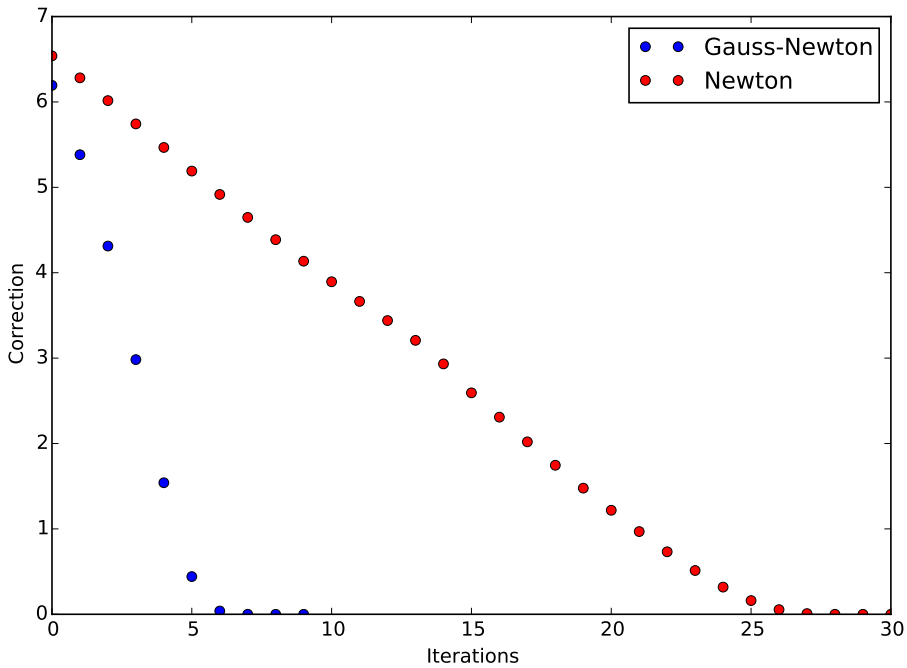
(d) Löse dasselbe Problem mit einem Newtonverfahren.

siehe Code **ellipse.py**. Die Lösungen für beide Verfahren sind in dem untenstehenden Plot gezeigt. Wir sehen, dass beide Verfahren die Datenpunkte gut mit einer Ellipse approximieren.



(e) Vergleiche die Konvergenz der Gauss-Newton und Newtonmethoden.

Der untenstehende Plot zeigt, dass die Gauss-Newton-Methode schneller konvergiert als die Newton-Methode. Man braucht bei der Gauss-Newton-Methode nur 5 Iterationen bis der Fehler zur finalen Lösung nahezu verschwindet. Jedoch sind 25 Iterationen für das Newton-Verfahren nötig.



3. Monte Carlo Integration

Wir möchten die Rungefunktion

$$f(t) = \frac{1}{1+25t^2}, \ t \in [-1, 1], \tag{3}$$

mit einem Monte Carlo Verfahren integrieren.

(a) Gebe die analytische Lösung der Stammfunktion von Gleichung (3) an. Man kann die Integrationskonstante vernachlässigen.

Die Stammfunktion ist $F(t) = 5^{-1} \arctan(5t)$.

(b) Implementiere eine Funktion **int_uniform(F, N, a, b)** in Python, welche das Integral **I** mit **N** zufälligen Stützstellen $t_i \sim U(a, b)$ approximiert und die Varianz **var** berechnet. Die Funktion soll das Tupel (**I**, **var**) zurückgeben.

siehe Code **monte_carlo.py**.

(c) Approximiere das Integral für $N = 10^k, \ k = 2, \ldots, 5$ und berechne jeweils den Fehler. Wie verhält sich die Varianz.

siehe Code **monte_carlo.py**.

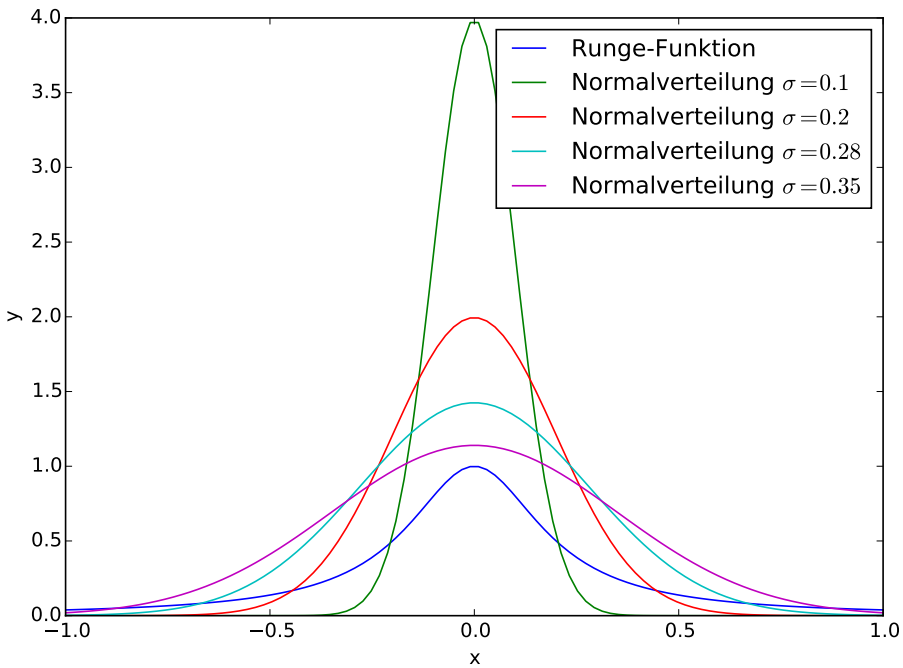
(d) Wir möchten die Varianz reduzieren und bemerken, dass die Rungefunktion ein ähnliches Verhalten zeigt wie eine Normalverteilung. Wir verwenden also ein *Importance*

Sampling mit $x_i \sim g(x)$, wobei

$$g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right). \tag{4}$$

Bevor wir das Importance Sampling implementieren, wählen wir angemessene Parameter μ und σ. Die Rungefunktion ist symmetrisch um den Ursprung. Wir wählen also $\mu = 0$. Variiere nun σ und plotte verschiedene Normalverteilungen zusammen mit $f(t)$ und finde ein geeignetes σ für welches die Normalverteilung ähnlich aussieht wie $f(t)$.

Wir wählen also $\mu = 0$ und lesen ein geeignetes σ aus dem Plot ab. Wir verwenden $\sigma = 0.28$.



(e) Implementiere eine Funktion **int_normal(F, N, a, b)**, welche das Integral mittels Importance Sampling an **N** zufälligen, normalverteilten Punkten approximiert. Verwende $\mu = 0$ und das σ aus Aufgabe **(d)**.

siehe Code **monte_carlo.py**.

(f) Führe das Importance Sampling aus für $N = 10^k,\ k = 2, \ldots, 5$ und berechne den Fehler von **I** sowie die Varianz der N Messwerte. Vergleiche die Varianzen von **int_uniform(F, N, a, b)** und **int_normal(F, N, a, b)**.

Wir sehen im **monte_carlo.py** Code, dass die Varianz und der Fehler kleiner werden. Wir konnten also mit dem Importance-Sampling eine Varianzreduktion erreichen.

4. Runge Kutta

Das Butcher Tableau der *England Formel* ist gegeben durch:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & & \\ 1 & 0 & -1 & 2 & \\ \hline & \frac{1}{6} & 0 & \frac{2}{3} & \frac{1}{6} \end{array}$$

.

(a) Implementiere ein Runge Kutta Verfahren mit dem Butcher Tableau der *England Formel*.

siehe Code **runge_kutta.py**.

(b) Finde die Konvergenzordnung des Verfahrens empirisch durch die Anwendung der Methode auf die Differentialgleichung $\ddot{y}(t) + y(t) = 0$ für $t \in [0, 10]$.

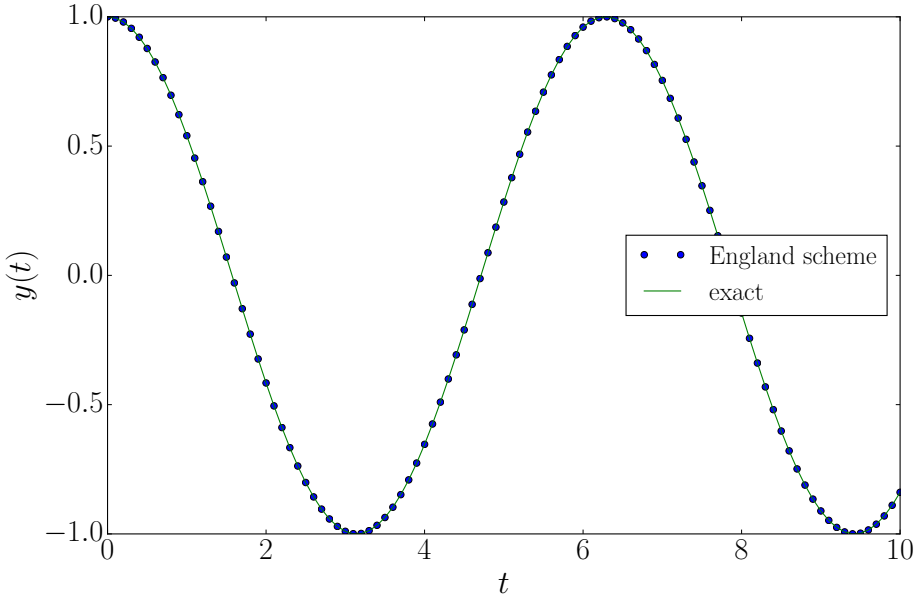
siehe Code **runge_kutta.py**.

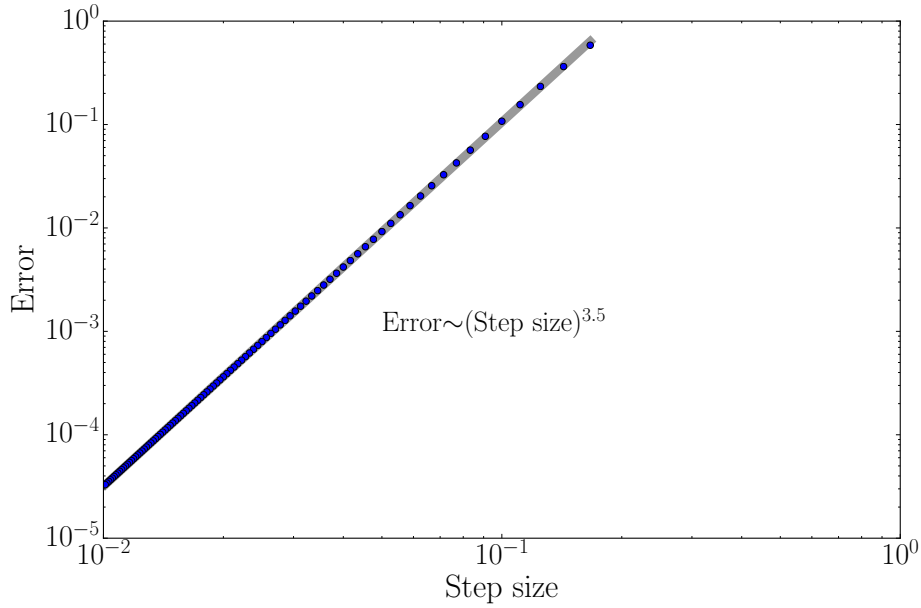
Wir nehmen an, dass $y(0) = 1$ und $\dot{y}(0) = 0$. Damit ist die Lösung zu obiger Differentialgleichung einfach $y(t) = \cos(t)$. Zuerst plotten wir zum Testen die numerische und die analytische Lösung. Im zweiten Schritt untersuchen wir die Konvergenzeigenschaften. Dazu berechnen wir den Fehler als Norm der Abstände von der numerischen zur analytischen Lösung. Wir finden, dass die Konvergenzordnung ungefähr einen Wert 3.5 annimmt und damit zwischen 3 und 4 liegt.

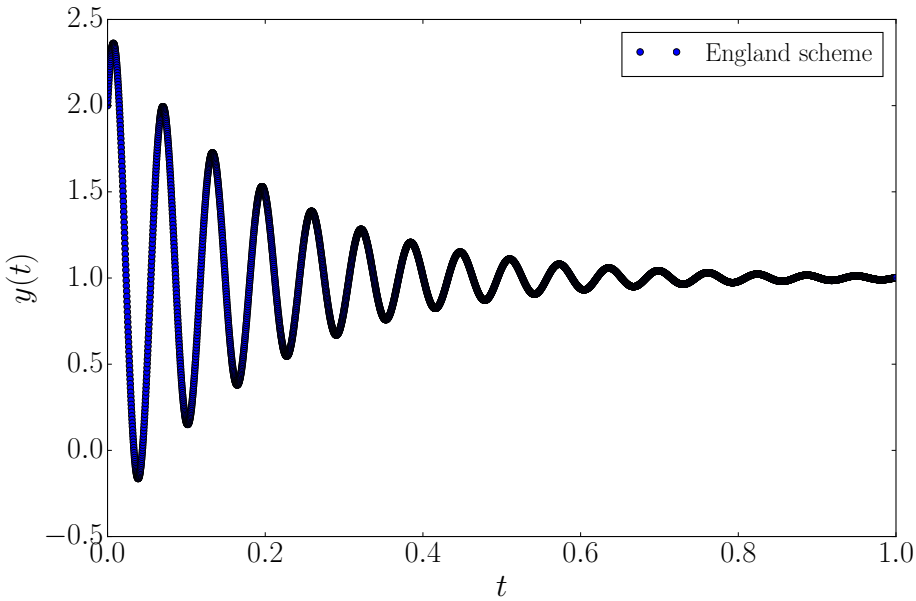
(c) Verwende diese Methode, um die Differentialgleichung $\ddot{y}(t) + 10\dot{y}(t) + 10025y(t) - 10025 = 0$ mit $y(0) = 2$, $\dot{y}(0) = 95$, $t \in [0, 1]$ und 2^{12} Zeitschritten zu lösen. Stelle die Lösung graphisch dar.

siehe Code **runge_kutta.py**.

Die Lösung der Funktion führt rasche, abfallende Oszillationen aus.


1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
$y(t)$
t
England scheme
exact


10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-2}
10^{-1}
10^0
Error
Step size
Error$\sim$(Step size)$^{3.5}$



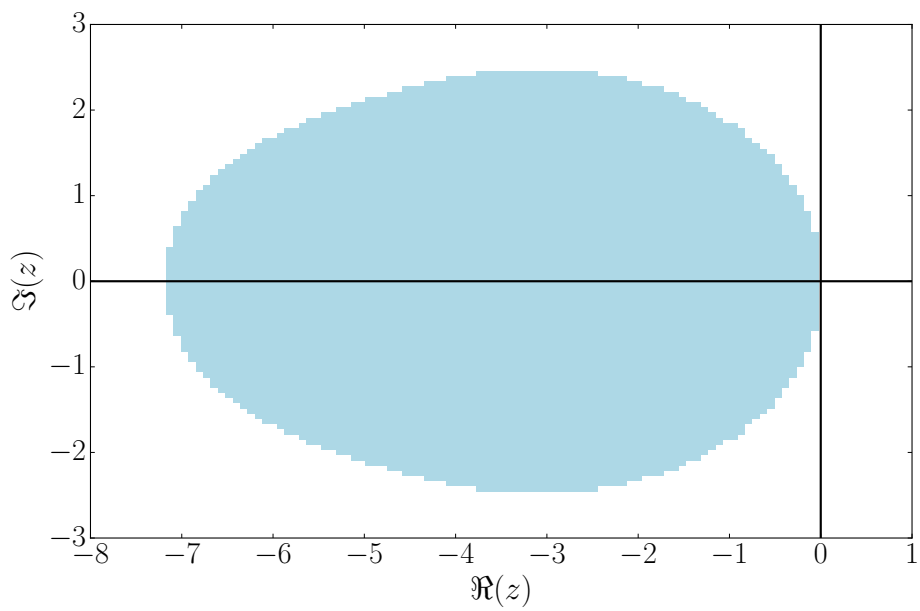
(d) Kann man mit dieser Methode in 5 Zeitschritten eine sinnvolle Annäherung der Lösung der letzten Gleichung zur Zeit $T = 1$ erhalten? Warum?

Man erhält keine sinnvolle Annäherung, da die Schrittweite zu gross ist und zu Instabilitäten bei der numerischen Lösung führt.

(e) Implementiere die Stabilitätsfunktion $S\colon \mathbb{C} \to \mathbb{C}$. Plotte diese Funktion und bestimme durch Ablesen aus diesem Bild eine maximale Zeitschrittweite h, die eine stabile Lösung für die letztgennante Differentialgleichung garantiert. Notiere die Herleitung und den Wert.

siehe Code **runge_kutta.py**.

Die Stabilitätsfunktion ist $S(z) = \frac{1}{54}((3+z)(6+z)(3+2z))$ und kann mit der Determinantenformel aus dem Skript berechnet werden. Aus dem untenstehenden Bild liest man ab, dass $h|\lambda| < 7$, cf. p. 281 im Skript wo $h < \sup\{t > 0; t\lambda \in S\}$.


3
2
1
0
−1
−2
−3
−8
−7
−6
−5
−4
−3
−2
−1
0
1
$\Im(z)$
$\Re(z)$