

ETHZ, D-MATH
Probeprüfung
Numerische Methoden D-PHYS, FS 2019
Dr. V. Gradinaru
29.03.2019

Auf der Vorlesungshompagie stehen Templates für die Probeprüfung bereit.
Prüfungsdauer : 90 Minuten.

1. (8 Punkte) Lobatto-Quadratur

Lobatto-Quadraturformeln besitzen die maximale Ordnung unter jenen Quadraturformeln, die die Intervallendpunkte als Knoten haben:

$(b_i, c_i)_{i=1}^s$ mit $c_1 = 0$, $c_s = 1$ der Ordnung $2s - 2$.

- a) Geben Sie die Lobatto-Quadraturformeln der Ordnungen 2 und 4 an ($s = 2$ und $s = 3$).

Solution:

Klar: $p = 2$ ist die Trapezregel, $p = 4$ ist Simpson-Regel.

- b) Berechnen Sie die Knoten und Gewichte der Lobatto-Quadraturformel der Ordnung 6 ($s = 4$).

Hint: : Die Berechnung vereinfacht sich, wenn Sie verwenden, dass die gesuchte Quadraturformel symmetrisch ist.

Solution:

Lobatto Quadrature der Ordnung $p = 2s - 2 = 6$ (d.h. $s = 4$ Knoten)

QF Lobatto symmetrisch: $c_1 = 0$, $c_4 = 1$, $c_3 = 1 - c_2$, $b_1 = b_4$, $b_2 = b_3$ also nur 3 Unbekannte b_1, b_2, c_2

Exakte Quadratur für Polynome vom Grad:

$q = 1$ gibt $b_1 + b_2 + b_3 + b_4 = 1$

$q = 3$ gibt $b_2 \cdot c_2^2 + b_3 \cdot c_3^2 + b_4 \cdot 1^2 = 1/3$

$q = 5$ gibt $b_2 \cdot c_2^4 + b_3 \cdot c_3^4 + b_4 \cdot 1^4 = 1/5$

```
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
```

```
def f(v):
    x = v[0]; y = v[1]; z = v[2]
    r1 = x+y-0.5
```

Bitte wenden!

```

r2 = x+ y*(z**2 + (1-z)**2) - 1./3.
r3 = x + y*(z**4 +(1-z)**4) - 1./5.
res = np.array([r1,r2,r3])
return res
b1,b2, c2 = optimize.fsolve(f,[0.24, 0.25, 0.3])
print(b1,b2,c2)
# 0.08333333333326 0.416666666667 0.276393202253
b1+b2
# 0.5
0.5 - np.sqrt(1./20)
# 0.27639320225002106

```

- c) Sei die symmetrische Quadraturformel auf $[-1,1]$ gegeben durch die Knoten $0, \pm\sqrt{\frac{5}{11} - \frac{2}{11}\sqrt{5/3}}, \pm\sqrt{\frac{5}{11} + \frac{2}{11}\sqrt{5/3}}, \pm 1$ und entsprechenden Gewichte $\frac{256}{525}, \frac{124+7\sqrt{15}}{350}, \frac{124-7\sqrt{15}}{350}, \frac{1}{21}$.

Finden Sie die Ordnung dieser Quadraturformel. Ist das auch eine Lobatto-Quadraturformel?

Solution:

Nach Definition, eine Quadraturformel Q mit $n+1$ Knoten besitzt Ordnung $p+1$, wenn sie Polynome von Grad maximal p exakt integriert. Da Q symmetrisch ist, man erwartet, dass sie gerade Ordnung hat. Außerdem, da die Anzahl der Knoten 7 ist, hat Q die Ordnung $p \leq 14$. Exakte Quadratur für polynome liefert

```

import numpy as np
from scipy import optimize

def Q(f):
    x = [0., -np.sqrt(5./11. - 2./11.*np.sqrt(5./3.)), np.sqrt
        ↪ (5./11. - 2./11.*np.sqrt(5./3.)), -np.sqrt(5./11. +
        ↪ 2./11.*np.sqrt(5./3.)), np.sqrt(5./11. + 2./11.*np.
        ↪ sqrt(5./3.)), 1., -1.]
    w = [256./525., (124.+7.*np.sqrt(15.))/350., (124.+7.*np.sqrt
        ↪ (15.))/350., (124.-7.*np.sqrt(15.))/350., (124.-7.*np.
        ↪ sqrt(15.))/350., 1./21., 1./21.]

    q = 0.;
    for k in range (0,len(x)):
        q = q + w[k]*f(x[k])

    return q

for k in range (0,15):
    f = lambda x: x**k
    I = (1.-(-1.)**(k+1))/(k+1)
    q = Q(f)

```

Siehe nächstes Blatt!

```

if (abs(q-I) <= 1e-15) :
    print("Die Quadraturformel ist exakt fuer polynome
          ↪ von Grad " + str(k))
else :
    print("Das erste falsche Ergebnis ist " + str(k))
    break

# Die Quadraturformel ist exakt fuer polynome von Grad 0
# Die Quadraturformel ist exakt fuer polynome von Grad 1
# Die Quadraturformel ist exakt fuer polynome von Grad 2
# Die Quadraturformel ist exakt fuer polynome von Grad 3
# Die Quadraturformel ist exakt fuer polynome von Grad 4
# Die Quadraturformel ist exakt fuer polynome von Grad 5
# Die Quadraturformel ist exakt fuer polynome von Grad 6
# Die Quadraturformel ist exakt fuer polynome von Grad 7
# Die Quadraturformel ist exakt fuer polynome von Grad 8
# Die Quadraturformel ist exakt fuer polynome von Grad 9
# Die Quadraturformel ist exakt fuer polynome von Grad 10
# Die Quadraturformel ist exakt fuer polynome von Grad 11
# Das erste falsche Ergebnis ist 12

```

Dann Q besitzt Ordnung 12. Da die Endpunkte -1 und 1 Knoten der Quadraturformel Q sind, ist Q eine Lobatto-Quadraturformel.

2. (8 Punkte) Splitting

Sei $\epsilon = 0.001$, $\omega = 0.01$ und die Differentialgleichung

$$\begin{cases} \ddot{u} = -u + \epsilon \cos(\omega t) \\ u(0) = 1, \dot{u}(0) = 0 \end{cases} \quad (1)$$

a) **Schreiben** Sie die Differentialgleichung (1) in einem System Differentialgleichungen erster Ordnung um:

$$\begin{cases} \dot{y}_1 = f_1(t, y_0, y_1) \\ \dot{y}_2 = f_2(t, y_0, y_1) \\ y_1(0) = 1 \\ y_2(0) = 0 \end{cases} \quad (2)$$

Solution:

Wir betrachten folgende Definitionen

$$y_1(t) := u(t), \quad y_2(t) := \dot{u}(t) \quad (3)$$

Dann gilt

$$\begin{cases} \dot{\mathbf{y}} = \mathbf{F}(t, \mathbf{y}) \\ \mathbf{y} = \mathbf{y}_0 \end{cases}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad \mathbf{F}(t, \mathbf{y}) = \begin{bmatrix} y_2 \\ -y_1 + \epsilon \cos(\omega t) \end{bmatrix}, \quad \mathbf{y}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (4)$$

b) Implementieren Sie folgende Verfahren:

- Explizites Eulerverfahren
- Implizites Eulerverfahren
- Implizite Mittelpunktsregel

Alle Funktionen sollen die rechte Seite der ODE `rhs`, den Startwert `y0`, die Endzeit `T` und die Anzahl Schritte `N` als Parameter akzeptieren, z.B:

```
1 def explicit_euler(rhs, y0, T, N):  
2     # Implementieren Sie hier das explizite Eulerverfahren.
```

Hinweis 1. Lösen Sie die auftretenden nicht-linearen Gleichungssysteme $F(y) = 0$ mit `scipy.optimize.fsolve`.

Hinweis 2. Für diese Aufgabe können Sie `ode_solvers.py` verwenden.

Solution:

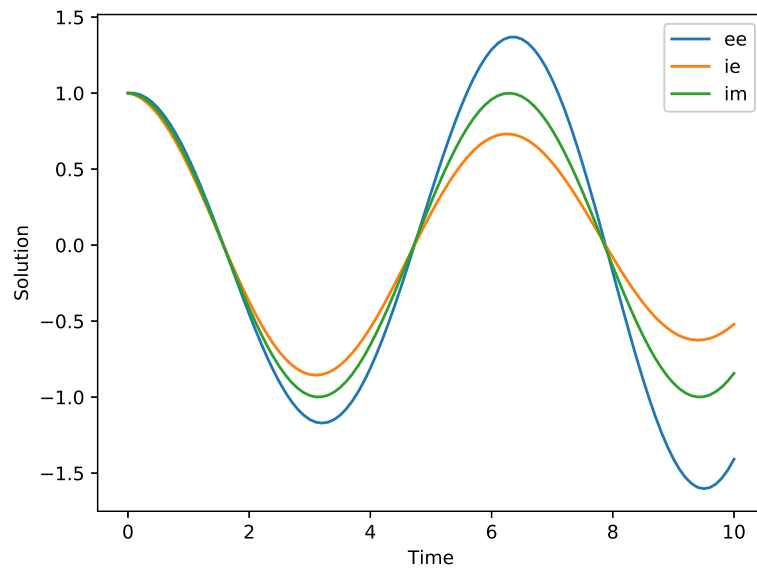
Den Mustercode finden Sie auf der Homepage.

c) Testen Sie Ihren Code mit Endzeit $T = 10$ und $N = 100$ und ploten Sie die entsprechenden Lösungen u in einem Bild.

Hinweis 2. Für diese Aufgabe können Sie `Differential.py` verwenden.

Solution:

Siehe nächstes Blatt!



d) Bestimmen Sie die Konvergenzordnung von

$$|y_N - y(T)| \quad (5)$$

mit $T = 1$. Betrachten Sie dazu die Folge der Schrittenanzahl $N_k = 2^k$, $k \in \{4, \dots, 11\}$ und für jede k , berechnen Sie $|u_{N_k} - y(T)|$. Erstellen Sie ein doppelt logarithmisches Plot des Fehlers. Vergleichen Sie diese mit der theoretischen Konvergenzordnung.

Hinweis. Benutzen Sie `ode45.py` um eine hervorragende Approximation von $y(T)$ zu berechnen. Sie finden `ode45` im Template `ode45.py`.

Solution:

```
<function explicit_euler at 0x117e31488>
rate: 1.0163626039019065
```

```
<function implicit_euler at 0x117e31598>
rate: 0.9825374689736645
```

```
<function implicit_mid_point at 0x117e316a8>
rate: 2.009554451220402
```

