D-ITET, D-MATL

# Musterlösung Prüfung Numerische Methoden, Sommer 2012

Dr. Lars Kielhorn

**1. a)**

$$z = \exp(\mathrm{i}\varphi) \quad \Longrightarrow \quad \mathrm{d}z = \mathrm{i}\exp(\mathrm{i}\varphi)\,\mathrm{d}\varphi$$

$$\Longrightarrow \qquad c_n[f] = \frac{1}{2\pi}\int_0^{2\pi} f(\exp(\mathrm{i}\varphi))\exp(-\mathrm{i}\varphi n)\,\mathrm{d}\varphi$$

**b)**

- Allgemeine zusammengesetzte Trapezregel

$$Q[f] = \frac{1}{2}(x_1 - a)f(a) + \frac{1}{2}\sum_{k=1}^{m-1}(x_{k+1} - x_{k-1})f(x_k) + \frac{1}{2}(b - x_{m-1})f(b)$$

- Wir integrieren auf dem Einheitskreis, d.h., $a = b = \varphi_0$ und $f(a) = f(b) = f(\varphi_0)$. Außerdem gilt aufgrund der Wahl äquidistanter Stützstellen $x_1 - x_{m-1} = x_{k+1} - x_{k-1} = 2\Delta\varphi$. Damit vereinfacht sich die Trapezregel zu

$$Q[f] = \Delta\varphi\sum_{k=0}^{m-1} f(x_k)$$

Einsetzen liefert:

$$c_n[f] \approx Q_n[f] = \frac{1}{N}\sum_{k=0}^{N-1} f(\exp(2\pi\mathrm{i}k/N))\,\exp(-2\pi\mathrm{i}kn/N)$$

**c)** Die Komplexität beträgt $\mathcal{O}(N\log N)$, da die Quadraturen schnell durch die Fouriertransformation durchgeführt werden können. Mit den Einheitswurzeln $\omega_N^{nk} := \exp(-2\pi\mathrm{i}kn/N)$ ist die Quadratur

$$Q_n[f] = \frac{1}{N}\sum_{k=0}^{N-1}\omega_N^{nk} f_k \qquad f_k := f(\omega_N^{-k}) \quad \Longleftrightarrow \quad \mathbf{q} = \frac{1}{N}\mathbf{F}\mathbf{f}$$

**d)** Mit $f(z) = z^p$ gilt $f_k = \left(\omega_N^{-k}\right)^p = \omega_N^{-pk}$. Damit folgt für die Quadraturformel

$$Q_n[f] = \frac{1}{N}\sum_{k=0}^{N-1}\omega_N^{nk}\omega_N^{-pk} = \frac{1}{N}\sum_{k=0}^{N-1}\omega_N^{(n-p)k} = \begin{cases} 1 & n = p \\ 0 & \text{sonst} \end{cases}$$

**Bitte wenden**

**2.** **a)** Man kann die Gleichung beispielsweise zu

$$x = \frac{c}{x}$$

umformen. Die zugehörige Fixpunktiteration mit Startwert $x_0$ ist dann:

$$x^{(k+1)} = \frac{c}{x^{(k)}}, \quad x^{(0)} = x_0.$$

**b)** Für eine Gleichung der Form $f(x) = 0$ lautet das Newtonverfahren allgemein:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}, \quad x^{(0)} = x_0.$$

Hier gilt

$$f(x) = x^2 - c$$

und weiter

$$f'(x) = 2x.$$

Damit ergibt sich das Newtonverfahren zu

$$x^{(k+1)} = x^{(k)} - \frac{(x^{(k)})^2 - c}{2x^{(k)}}, \quad x^{(0)} = x_0,$$

was sich zu

$$x^{(k+1)} = \frac{x^{(k)}}{2} + \frac{c}{2x^{(k)}}, \quad x^{(0)} = x_0.$$

vereinfachen lässt.

**c)**
```
function x = NewtonSqrt( x0, c)
%gibt eine der Naeherung x der Wurzel von c zurueck
maxit = 10; % Maximale Anzahl Iterationen
tol = 1e-6; % Toleranz fuer das Abbruchkriterum
x = x0;     % Startwert
for i = 1:maxit
  x = x/2 + c/(2*x);
  if abs(x*x-c) < tol
    return
  end
end
error('Nicht konvergiert')
```

**d)** Die Iterationsvorschrift des Newtonverfahrens lässt sich schreiben als

$$x^{(k+1)} = \Phi(x^{(k)}), \quad \Phi(x) = \frac{x}{2} + \frac{c}{2x}$$

**Siehe nächstes Blatt!**

Das führt zu

$$\Phi'(x) = \frac{1}{2} - \frac{c}{2x^2}$$

und

$$\Phi'(\sqrt{c}) = \frac{1}{2} - \frac{c}{2\sqrt{c}^2} = \frac{1}{2} - \frac{1}{2} = 0.$$

Mit Lemma 1.2.7 folgt damit, dass das Newtonverfahren lokal mit mindestens zweiter Ordnung konvergiert.

Alternativ kann sich auf die Aussage auf Seite 59 stützen, die sagt, dass das Newtonverfahren lokal quadratisch konvergiert, falls die erste Ableitung am Fixpunkt nicht verschwindet: Zu überprüfen bleibt also

$$f'(\sqrt{x}) = 2\sqrt{c} \neq 0,$$

was gilt, da wir $c > 0$ angenommen haben.

**e)** Wir schätzen die Konvergenzordnung mit

$$\frac{\log e^{(k+1)} - \log e^{(k)}}{\log e^{(k)} - \log e^{(k-1)}}$$

Wähle $k$ dabei so, dass wir in der asymptotischen Phase sind, aber vor Erreichen der Maschinengenauigkeit.

V1 ($k = 3$):

$$\frac{-22 - (-10)}{-10 - (-4.1)} = \frac{-12}{-5.9} \approx 2$$

Wir beobachten also quadratische Konvergenz.

V2 ($k = 9$):

$$\frac{-12 - (-11)}{-11 - (-9.6)} = \frac{1}{1.4} \approx 0.7$$

Alternativ sieht man aus dem halblogarithmischen Diagramm, dass das Verfahren linear konvergiert, da die Fehlerkurve eine Gerade darstellt.

V3 ($k = 5$):

$$\frac{-29 - (-17)}{-17 - (-9.8)} = \frac{12}{7.2} \approx 1.7$$

**f)** Das Verfahren V1 ist zu bevorzugen, da es die höchste Konvergenzordnung hat und der Fehler somit am schnellsten abnimmt.

**3.** **a)** **i)** Wir haben 2. Ableitungen, daher ist das System von zweiter Ordnung.

**ii)** Vektorschreibweise: $\mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix}$

$$\mathbf{y}''(t) = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \mathbf{y}'(t) + \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix} \mathbf{y}(t) + \begin{bmatrix} t^2 \\ t \end{bmatrix}$$

Mit $\mathbf{z}(t) := \mathbf{y}'(t)$ folgt

$$\mathbf{z}'(t) = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \mathbf{z}(t) + \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix} \mathbf{y}(t) + \begin{bmatrix} t^2 \\ t \end{bmatrix} .$$

Die Definition von $\mathbf{w}(t) := \begin{bmatrix} \mathbf{y}(t) \\ \mathbf{z}(t) \end{bmatrix}$ liefert schließlich

$$\mathbf{w}'(t) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 0 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix} \mathbf{w}(t) + \begin{bmatrix} 0 \\ 0 \\ t^2 \\ t \end{bmatrix} .$$

Die Anfangsbedingungen sind

$$\mathbf{w}_0 = \mathbf{w}(0) = \begin{bmatrix} y_1(0) \\ y_2(0) \\ z_1(0) = y_1'(0) \\ z_2(0) = y_2'(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix}$$

**iii)** Nein, das Gleichungssystem ist von der Form $\mathbf{w}'(t) = \mathbf{f}(t, \mathbf{w}(t))$. Damit ist es nicht autonom.

**b)** **i)** Wegen der rechten Seite $f(t_{n+1}, y_{n+1})$ ($y_{n+1}$ kommt als Argument vor) wird das Verfahren implizit sein.

**ii)** Es wird ein 2-stufiges Mehrschrittverfahren sein, da $\sum_{i=0}^{s}(\cdots)$ mit $s = 2$ gilt.

**iii)** Das Interpolationspolynom ist

$$\begin{aligned}(\Pi y)(t) = {} & y_{n-1} \frac{t - t_n}{t_{n-1} - t_n} \frac{t - t_{n+1}}{t_{n-1} - t_{n+1}} \\ & + y_n \frac{t - t_{n-1}}{t_n - t_{n-1}} \frac{t - t_{n+1}}{t_n - t_{n+1}} \\ & + y_{n+1} \frac{t - t_{n-1}}{t_{n+1} - t_{n-1}} \frac{t - t_n}{t_{n+1} - t_n} .\end{aligned}$$

Unter Beachtung von $t_{n+1-i} - t_{n-i} = h$ erhält man

$$\begin{aligned}(\Pi y)(t) = {} & \frac{y_{n-1}}{2h^2}(t - t_n)(t - t_{n+1}) \\ & - \frac{y_n}{h^2}(t - t_{n-1})(t - t_{n+1}) \\ & + \frac{y_{n+1}}{2h^2}(t - t_{n-1})(t - t_n)\end{aligned}$$

**Siehe nächstes Blatt!**

**iv)** Die Ableitung $(\Pi y)'$ an der Stelle $t_{n+1}$ ist dann

$$\begin{aligned}(\Pi y)'(t_{n+1}) &= \frac{1}{2h^2} y_{n-1}(t_{n+1} - t_n) \\ &\quad - \frac{1}{h^2} y_n (t_{n+1} - t_{n-1}) \\ &\quad + \frac{1}{2h^2} y_{n+1}(t_{n+1} - t_{n-1}) + \frac{1}{2h^2} y_{n+1}(t_{n+1} - t_n) \\ &= \frac{1}{2h} y_{n-1} - \frac{2}{h} y_n + \frac{3}{2h} y_{n+1} \approx y'(t_{n+1})\end{aligned}$$

Für die Koeffizienten $\alpha_i$ gilt damit

$$\boldsymbol{\alpha} = \frac{1}{2h} \begin{bmatrix} 1 \\ -4 \\ 3 \end{bmatrix} .$$

**v)** Adams-Verfahren beruhen auf einer Interpolation der rechten Seite. Dieses Verfahren hingegen approximiert den Differentialquotienten durch eine Rückwärtsdifferenz. Es gehört damit zu den BDF-Verfahren, genauer ist es ein BDF2.

**vi)** *Vorteil(e)*: Das Verfahren ist A-stabil. Explizite Runge-Kutta-Verfahren können nicht A-stabil sein. Eignet sich daher für steife DGLs.
*Nachteil(e)*: Runge-Kutta-Verfahren ermöglichen eine einfache Schrittweiten-Steuerung. Das ist hier nur schwer möglich. Mehrschrittverfahren benötigen anfangs Einschrittverfahren, um die benötigten Wertepaare zur Verfügung zu stellen. Runge-Kutta-Verfahren als Einschrittverfahren sind daher einfacher zu handhaben.

**4.** **a)** Es muss gelten: $\chi(a) = -1$ und $\chi(b) = 1$. Daraus folgt

$$\xi = \chi(t) = \frac{2}{b-a} t - \frac{b+a}{b-a}$$

**b)** **i)** Es gilt: $\cos(\varphi_k) = 0$ für $\varphi_k = (2k+1)\frac{\pi}{2}$, $k = 0, 1, \ldots$
Mit $\varphi_k = N \arccos \xi_k$ folgt damit $\xi_k = \cos\left(\frac{\pi}{N}(k + \frac{1}{2})\right)$.

**ii)** Auswerten der Interpolationsgleichung an den Stellen $\xi_k$, Multiplikation der Interpolationsgleichung mit $T_j(\xi_k)$ und Summation über die Nullstellen liefert

$$\sum_{k=0}^{N-1} f(t_k) T_j(\xi_k) = \sum_{k=0}^{N-1} \sum_{\ell=0}^{N-1} \alpha_\ell T_\ell(\xi_k) T_j(\xi_k) \qquad t_k = \chi^{-1}(\xi_k) .$$

Das inverse Mapping kann mit

$$t = \chi^{-1}(\xi) = \frac{b-a}{2} \xi + \frac{b+a}{2}$$

angegeben werden. Muss es aber nicht.
Umsortierung der Summation auf der rechten Seite

$$\sum_{k=0}^{N-1} f(t_k) T_j(\xi_k) = \sum_{\ell=0}^{N-1} \alpha_\ell \sum_{k=0}^{N-1} T_\ell(\xi_k) T_j(\xi_k) .$$

**Bitte wenden**

Berücksichtigung der Orthogonalitätseigenschaft liefert dann

$$\alpha_0 = \frac{1}{N}\sum_{k=0}^{N-1} f(\chi^{-1}(\xi_k))\, T_0(\xi_k) = \frac{1}{N}\sum_{k=0}^{N-1} f(\chi^{-1}(\xi_k))$$

$$\alpha_j = \frac{2}{N}\sum_{k=0}^{N-1} f(\chi^{-1}(\xi_k))\, T_j(\xi_k)\,, \quad j = 1, \ldots, N-1\,.$$

**iii)** Die Ausdrücke $T_j(\xi_k)$ bilden die Tschebyschow-Matrix $C \in \mathbb{R}^{N\times N}$. Ihre Einträge sind

$$C_{j,k} = T_j(\xi_k) = \cos\left(\tfrac{\pi}{N} j (k + \tfrac{1}{2})\right)\,.$$

```
function C = chebmat( N )
% ============================================================
%
% Berechnung der Tschebyschow-Matrix
%
% INPUT
%   -- N: Interpolationsordnung
%
% OUTPUT
%   -- Tm: [N x N]-Tschebyschow-Matrix
%
% ============================================================
C = zeros( N, N );    % initialisiere Matrix mit Nullen
for j = 0:N-1
  for k = 0:N-1
    C( j+1, k+1 ) = cos( pi/N * j * (k + 0.5) );
  end
end
```

**iv)**

```
function alpha = cheb_interpol( func, N, a, b )
% ============================================================
%
% Berechnung der Koeffizienten der Tschebyschow-Interpolation
%
% INPUT
%   -- func: Funktionen handle
%   -- N   : Interpolationsordnung
%   -- a   : Untere Intervallgrenze
%   -- b   : Obere Intervallgrenze
%
% OUTPUT
%   -- alpha: [Nx1]-array der Interpolationskoeffizienten
%
% ============================================================
% Berechnung der Tschebyschow-Matrix
```

**Siehe nächstes Blatt!**

```
C       = chebmat( N );
% Berechnung der Tschebyschow-Knoten
xsi_k    = cheb_points( N );
% Transformation lokal -> global
t_k      = loc_to_glob( xsi_k, a, b );
% Funktion func an den Stuetzstellen auswerten
ft_k     = func( t_k );
% Berechnung der Koeffizienten
alpha    = 2.0 / N * C * ft_k;
% Anpassen des 1. Koeffizienten
alpha(1) = alpha(1) / 2.0;
```

**c)** Es sind $N$ Koeffizienten zu berechnen. Pro Koeffizient ist eine Summation der Länge $N$ durchzuführen. Die Komplexität beträgt daher $\mathcal{O}(N^2)$.

Bei der Tschebyschow-Interpolation liegt eine trigonometrische Basis vor. Die Berechnung der Koeffizienten kann daher effizient, d.h., mit einer Komplexität von $\mathcal{O}(N \log N)$, mittels der Kosinus- oder der Fourier-Transformation durchgeführt werden. Diese realisieren eine schnelle Anwendung der Tschebyschow-Matrix.

**d)** Die Tschebyshow-Interpolation ist immer noch eine Polynominterpolation. Die Wahl von $N = m + 1$ führt daher zu einem exakten Ergebnis, d.h. der Fehler ist Null. Kleinere Fehler gibt es nicht.