Dr. S. May D-ITET, D-MATL Sommer 2015

Prüfung Numerische Methoden

Wichtige Hinweise

- Die Prüfung dauert 90 Minuten.
- Erlaubte Hilfsmittel: 5 A4-Blätter doppelseitig (=10 Seiten) eigenhändig und handschriftlich verfasste Zusammenfassung, nicht ausgedruckt, nicht kopiert. Sonst keine Hilfsmittel zugelassen.
- Begründen Sie jeweils Ihre Aussagen. Unbegründete Lösungen (außer bei der Multiple-Choice-Aufgabe falls nicht explizit gefordert) werden nicht akzeptiert!

Name		Note
Vorname		
Studiengang		
Leginummer		
Prüfung	Numerische Methoden	
Datum	19.08.2015	

1	2	3	4	5	6	Bonuspunkte	Punkte
8	7	10	8	11	12		

- Legen Sie Ihre Legi auf den Tisch. Schalten Sie Ihr Handy aus.
- Beginnen Sie jede Aufgabe auf einer neuen Seite, und schreiben Sie Ihren Namen und Ihre Leginummer auf alle Blätter.
- Schreiben Sie nicht mit Bleistift. Verwenden Sie einen Stift mit blauer oder schwarzer Farbe (keinesfalls rot oder grün).
- Versuchen Sie Ihren Lösungsweg möglichst klar darzustellen und arbeiten Sie sorgfältig!
- **Schauen Sie das Prüfungsblatt erst an, wenn der Assistent das Signal dazu gibt!**

Viel Erfolg!

1. Kurze Fragen (8 P)

(a) Was können Sie über die lokale eindeutige Lösbarkeit des Anfangswertproblems

$$\begin{cases} \dot{y}(t) = & (y(t))^{1/3}, \quad t \in (0,T) \\ y(0) = & 1 \end{cases}$$

aussagen? Begründen Sie Ihre Antwort.

(b) Gegeben sei eine unendlich oft stetig differenzierbare Funktion $v : \mathbb{R} \to \mathbb{R}$ und der finite Differenzenquotient

$$\frac{v(x-2h) - 2v(x) + v(x+2h)}{4h^2}.$$

Welche Ableitung von v in x kann man mit diesem Differenzenquotienten approximieren?

- ☐ $v'(x)$
- ☐ $v''(x)$
- ☐ $v^{(3)}(x)$
- ☐ $v^{(4)}(x)$

Mit welcher Ordnung wird diese Ableitung approximiert?

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4

(c) Wir wollen $\int_0^1 f(x)dx$ mit numerischer Quadratur approximieren. Dazu verwenden wir die *zusammengesetzte* Quadraturformel $Q_N^{10}(f;0,1)$

- mit N Intervallen der Länge $h = \frac{1}{N}$,
- die auf jedem Intervall eine Newton-Cotes Formel der Ordnung 10 verwendet.

Wir betrachten

$$E(f;N) = \left| \int_0^1 f(x)dx - Q_N^{10}(f;0,1) \right|.$$

Welches Verhalten erwarten wir für $E(f;N)$ für $N \to \infty$ für die folgenden Funktionen:

- $f(x) = \sin(x)\cos(x)$: ...

- $f(x) = (x^3 - 1)(x^3 + 1)$: ...

- $f(x) = \sqrt{x}$: ...

Bitte wenden!

2. Numerische Quadratur (7 P)

(a) Wir wollen mithilfe von numerischer Quadratur ein Integral approximieren:

$$Q_n(f; a, b) = \sum_{i=1}^{n} \omega_i f(c_i) \approx \int_a^b f(x)dx, \quad a < b.$$

Die Gewichte ω_i und Knoten c_i sollen basierend auf *Gauss-Quadraturformeln* gewählt werden.

(i) Die Gauss-Quadraturformel mit n Knoten sei gegeben durch

$$\widehat{Q}_n(\widehat{f}; -1, 1) := \sum_{i=1}^{n} \widehat{\omega}_i \widehat{f}(\widehat{c}_i) \approx \int_{-1}^{1} \widehat{f}(x)dx.$$

Geben Sie eine Koordinatentransformation, um ω_i und c_i aus $\widehat{\omega}_i$ und $\widehat{c}_i$ herzuleiten.

(ii) Berechnen Sie $Q_2(f; 1, 2)$ mit $f(x) = x^2$.

Hinweis: $\widehat{Q}_2(\widehat{f}; -1, 1)$ verwendet die Knoten $\widehat{c}_i = \pm\sqrt{\frac{1}{3}}$ und die Gewichte $\widehat{\omega}_i = 1, i = 1, 2$.

(b) Es sei

$$h(x) = \begin{cases} x^3 & 0 \leq x \leq 2, \\ \exp(x) & 2 < x \leq 4. \end{cases}$$

Geben Sie einen möglichst effizienten Weg an (ohne adaptive Quadratur zu verwenden), um $\int_0^4 h(x)dx$ mithilfe von numerischer Quadratur *sehr genau* zu berechnen.

3. Runge-Kutta-Einschrittverfahren (10 P)

Für $\gamma \in [0, 1]$, sei das Runge-Kutta-Einschrittverfahren mit dem folgenden Butcher-Schema gegeben:

$$\begin{array}{c|c} \gamma & \gamma \\ \hline & 1 \end{array} \tag{1}$$

(a) Schreiben Sie für das Anfangswertproblem

$$\dot{y}(t) = f(t, y(t)), \quad y(0) = y_0,$$

das in (1) gegebene Verfahren in die Stufenform eines Runge-Kutta-Verfahrens um. Verwenden Sie hierbei den Schritt von y_n zu y_{n+1}.

(b) Für welche Werte von γ ist das Verfahren explizit/implizit?

- explizit: ...
- implizit: ...

(c) Welche Konsistenzordnung besitzt das Verfahren für $\gamma = 0$? Führen Sie eine Analyse mittels Taylorentwicklung durch!

Hinweis: Sie können annehmen, dass das Verfahren autonomisierungsinvariant ist.

(d) Bestimmen Sie die Stabilitätsfunktion dieses Verfahrens in Abhängigkeit von γ.

Siehe nächstes Blatt!

4. Steife Probleme (8 P)

Wir betrachten das lineare Anfangswertproblem

$$\begin{cases} \dot{\mathbf{y}}(t) = & \mathbf{A}\mathbf{y}(t), \\ \mathbf{y}(0) = & \mathbf{y}_0, \end{cases} \tag{2}$$

mit $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^{m\times m}$.

(a) Geben Sie die mathematische Definition an, die in der Vorlesung verwendet wurde, um Probleme der Form (2) als "steif" zu charakterisieren.

(b) Beschreiben Sie in Worten, wodurch ein steifes Problem für ein lineares System charakterisiert ist. Der Einfachheit halber konzentrieren Sie sich auf den Fall $\mathbf{A} \in \mathbb{R}^{2\times 2}$.

(c) Ist das folgende Problem (der mathematischen Definition aus der Vorlesung zufolge) steif?

$$\begin{cases} \dot{\mathbf{y}}(t) = & \begin{pmatrix} -2 & 0 & 0 \\ 0 & 500 & 0 \\ 0 & 0 & -1 \end{pmatrix} \mathbf{y}(t), \\ \mathbf{y}(0) = & \mathbf{y}_0. \end{cases}$$

(d) Wir wollen das Problem (2) für

$$\mathbf{A} = \begin{pmatrix} -2 & 0 \\ 0 & -10000 \end{pmatrix}$$

und den Endzeitpunkt $T = 1$ lösen.

(i) Nennen Sie zwei *verschiedene* Methoden 2. Ordnung, die dafür geeignet sind. (Man beachte, dass das Problem linear ist!)

- ...
- ...

(ii) Nennen Sie eine Methode 5. Ordnung, die dafür geeignet ist.

- ...

(iii) Welchen eingebauten MATLAB-Löser würden Sie verwenden, um das Problem mit hoher Genauigkeit *effektiv* zu lösen?

- ...

Hinweis: Es ist keine Begründung nötig für diese Teilaufgabe (**4**(d)).

5. Das Newton Verfahren (7 P + 4 P)

Wir betrachten das nichtlineare Anfangswertproblem

$$\begin{cases} \dot{\mathbf{y}}(t) = & \mathbf{f}(\mathbf{y}(t)), \quad t \in (0,T), \\ \mathbf{y}(0) = & \mathbf{y}_0, \end{cases} \tag{3}$$

mit $\mathbf{y}(t) \in \mathbb{R}^m$ und $\mathbf{f} : \mathbb{R}^m \to \mathbb{R}^m$, $m > 1$, und wollen es mit dem impliziten Euler Verfahren lösen. Die resultierenden nichtlinearen Gleichungen lösen wir mit dem Newton Verfahren. Vervollständigen Sie die folgenden MATLAB Funktionen, die dies tun.

Die MATLAB-Funktion `y = implicitEuler(f,Df,T,y0,N)` berechnet eine Approximation der Lösung $\mathbf{y}(t)$ von (3) basierend auf dem impliziten Euler Verfahren mit N äquidistanten Zeitschritten. Das nichtlineare Gleichungssystem wird mithilfe des Newton-Verfahrens gelöst.

```
function y = implicitEuler(f,Df,T,y0,N)
    %Schrittweite
    h = ...
    m = length(y0);
    %Alloziere Speicherplatz
    y = ...
    y(:,1) = y0;
    %Berechne die approximative Loesung
    for ii = 1:N
        %Funktion fuer Nullstellesuche (und Ableitung)
        F = ...
        DF = ...
        y(:,ii+1) = newton(...
    end
end
```

Siehe nächstes Blatt!

Die MATLAB-Funktion `x = newton(F,DF,x0)` berechnet eine Approximation der Nullstelle `x`* von `F` mithilfe des Newton-Verfahrens zu gegebenem Startwert `x0`, wobei `DF` die Ableitung (Jacobi-Matrix) von `F` ist.

```matlab
function x = newton(F,DF,x0)
    %Initialisierung
    x = ...
    %Toleranz
    tol = 1e-12;
    Nmax = 100;
    for i = ...
        x_old = x;
        % Berechne Newton-Korrektur
        dx = ...
        %Update
        x = ...
        %Abbruchkriterium
        if norm(DF(x_old)\F(x)) < tol;
            break
        end
    end
end
```

Bitte wenden!

6. Schrittweitensteuerung (12 P)

Wir betrachten das *skalare* Anfangswertproblem

$$\dot{y}(t) = f(y(t)), \quad y(t_0) = y_0,$$

mit $y : \mathbb{R} \to \mathbb{R}$ und $f : \mathbb{R} \to \mathbb{R}$. Wir wollen die Lösung am Endzeitpunkt T mit einem adaptiven Einschrittverfahren berechnen. Die 2 essentiellen Komponenten einer Schrittweitensteuerung (für den Zeitschritt $t_k \to t_{k+1}$) sind:

1. die Festlegung einer Toleranz tol_{k+1} 2. die Schätzung des Fehlers EST_{k+1}.

Für die Toleranz wollen wir erreichen, dass für alle k

$$tol_{k+1} \leq \text{absTol}$$

wobei absTol eine gegebene absolute Toleranz ist.
Für die Schätzung des Fehlers haben wir in der Vorlesung eingebettete Runge-Kutta Verfahren verwendet. Hier wollen wir den Fehler folgendermaßen schätzen: für einen gegebenen ODE-Löser, machen wir einen Schritt mit einer gegebenen Schrittweite H, ausgehend von y_k, und 2 Schritte mit Schrittweite $H/2$, ausgehend von y_k:

$$\begin{array}{ccccc} t_k & & \xrightarrow{\qquad\qquad H \qquad\qquad} & & t_{k+1} \\ y_k & & \xrightarrow{\qquad\qquad H \qquad\qquad} & & y_{k+1}^{H} \\ y_k & \xrightarrow{H/2} & y_{1/2} & \xrightarrow{H/2} & y_{k+1}^{H/2}. \end{array}$$

Daraus kann man den folgenden Fehlerschätzer herleiten (wird hier nicht gemacht):

$$EST_{k+1} = \frac{|y_{k+1}^{H} - y_{k+1}^{H/2}|}{1 - 2^{-p}}.$$

Hierbei bezeichnet p die Ordnung (des globalen Diskretisierungsfehlers) des ODE-Lösers, der in der Schrittweitensteuerung verwendet wird. Als Vorschlag für die neue Schrittweite im Falle eines gerade akzeptierten Schritts der Länge H verwenden wir wie in der Vorlesung

$$h = H \left(\frac{tol_{k+1}}{EST_{k+1}} \right)^{\frac{1}{p+1}}.$$

Ergänzen Sie die MATLAB-Funktion `adaptive.m`, die ein adaptives Einschrittverfahren für die *klassische RK 4 Methode* mit obigen Bausteinen implementiert. Der Rückgabewert der Funktion ist eine Approximation von $y(T)$. Um einen Schritt der klassischen Runge-Kutta 4 Methode mit Länge h ausgehend von (t_k, y_k) auszuführen, verwenden Sie den Funktionsaufruf

y = RK4_1Schritt(tk,h,yk,f);

Siehe nächstes Blatt!

```matlab
function y = adaptive(f,y0,t0,T,h0,absTol)
    % Initialisierung
    y = y0;  h = h0;  t = t0;
    hmin = 1e-8;
    mu = 2;
    rho = 0.8;
    % Ordnung des ODE-Losers
    p = ...
    while t < ...
        % 1 Schritt mit Schrittweite H
        y_H = ...
        % 2 Schritte mit Schrittweite H/2
        ...
        y_H_over_2 = ...
        % Schaetze den Fehler
        EST = ...

        % wird Schritt akzeptiert?
        if ...
            % Update y -- nimm die genauere Loesung
            y = ...
            t = t + h;
            % Update h
            % Vorschlag fuer neues h basierend auf Fehlerschaetzung
            vorschlag = ...
            h = max(hmin, min(mu*h,rho*vorschlag));
            % stelle sicher, dass Endzeit T nicht ueberschritten wird
            h = ...
        else
            h = h/2;
            if h < hmin
                fprintf('h zu klein\n')
                return
            end
        end
    end
end
```