Numerische Mathematik 1
Wintersemester 2020, Übungsblatt 2

Ausarbeitung über Moodle bis 16. Oktober 2020. Nach diesem Datum erscheinen die nachträglichen Kommentare und die Lösungen der Teilnehmer.

1. Die Lösung $y(t)$ der gewöhnlichen Differentialgleichung $y'(t) = -y(t)$ wird durch die Differenzengleichung approximiert,

$$\frac{y_{k+1} - y_k}{\tau} = -2y_k + y_{k-1}, \quad k \in \mathbb{N}$$

wobei $y_k \approx y(t_k)$, $t_k = k\tau$, $k \in \mathbb{N}_0$, $\tau > 0$. Seien $r_{1,2}$ die Nullstellen der charakteristischen Gleichung $(r^2 - r)/\tau = -2r + 1$. (Sehen Sie Seiten 34–35 im Lehrbuch.) Kreuzen Sie bei den wahren Behauptungen an.

   (a) Die exakte Lösung der Differenzengleichung mit gegebenen Anfangswerten y_0 und y_1 ist $y_k = c_1 r_1^k + c_2 r_2^k$ wobei $c_1 + c_2 = y_0$ und $c_1 r_1 + c_2 r_2 = y_1$.

   (b) Mit fehlerhaften Anfangswerten $\tilde{y}_0 \approx y_0$ und $\tilde{y}_1 \approx y_1$ sind die entstehenden Fehler $E_k = y_k - \tilde{y}_k$ gegeben durch $E_k = \tilde{c}_1 r_1^k + \tilde{c}_2 r_2^k$, wobei $\tilde{c}_1 + \tilde{c}_2 = \tilde{y}_0$ und $\tilde{c}_1 r_1 + \tilde{c}_2 r_2 = \tilde{y}_1$.

   (c) Für alle Anfangswerte $\tilde{y}_0 \neq y_0$ und $\tilde{y}_1 \neq y_1$ wachsen die entstehenden Fehler $E_k = y_k - \tilde{y}_k$ exponentiell, wenn τ ausreichend groß ist.

   (d) Die Iteration ist stabil für $\tau \in (0, 2/3)$.

   Kommentare: Sehen Sie diesen Code. Der Betrag einer einzigen Nullstelle der charakteristischen Gleichung bleibt in $(0, 1)$. Sei r_1 diese Nullstelle. Es gelten dann $|r_1| < 1$, $\forall \tau > 0$ und $|r_2| < 1$, $\forall \tau \in (0, 2/3)$. Wenn $(y_0, y_1) \neq (\tilde{y}_0, \tilde{y}_1)$ so ausgewählt werden, dass $c_1 \neq \tilde{c}_2$ und $c_2 = \tilde{c}_2$ gelten, dann folgen $E_k = y_k - \tilde{y}_k = (c_1 - \tilde{c}_1) r_1^k$ und $|E_k| \leq |c_1 - \tilde{c}_1|$, $\forall k \in \mathbb{N}_0$. Für allgemeine Störungen gilt $|E_k|/r_2^k \to |c_2 - \tilde{c}_2|$, und daher wächste der Fehler exponentiell.

2. Die Nullstelle $\hat{x} = 1 + 2^{24}$ der Funktion $f(x) = \arctan[(x - \hat{x})/\sqrt{3}]$ soll mit einem iterativen Verfahren annäherungsweise bestimmt werden. Die Funktion wird mit doppelter Genauigkeit ausgewertet, aber die Iterierten x_k werden in der Menge `S` der Zahlen gespeichert, die mit einfacher Genauigkeit dargestellt werden können. Es gilt $\lim_{k\to\infty} |x_k - \hat{x}| = \min_{x \in \mathtt{S}} |x - \hat{x}|$, aber $x_k = \hat{x}$ kann für kein k gelten. Weiters sind zwei aufeinanderfolgende Iterierte nie gleich. Kreuzen Sie bei den wahren Behauptungen an.

   (a) Das kleinste Intervall $[x_1, x_2]$ mit $x_1, x_2 \in \mathtt{S}$ und $\hat{x} \in [x_1, x_2]$ hat die Endpunkte $x_1 = \frac{1}{2} + 2^{24}$ und $x_2 = \frac{3}{2} + 2^{24}$.

   (b) Der kleinste Wert der Toleranz ε, mit dem die Iteration durch das Abbruchskriterium $|x_k - x_{k-1}| \leq \varepsilon |x_k|$ beendet werden kann, ist $\varepsilon = 2^{-23}$.

   (c) Der kleinste Wert der Tolderanz δ, mit dem die Iteration durch das Abbruchskriterium $|x_k - x_{k-1}| \leq \delta$ beendet werden kann, ist $\delta = 2$.

   (d) Der kleinste Wert der Toleranz η, mit dem die Iteration durch das Abbruchskriterium $|f(x_k)| \leq \eta$ beendet werden kann, ist $\eta = \pi/6$.

   Kommentare: Die relative Fehlertoleranz ϵ verlangt kein Vorwissen über die Funktion. Wie auf Seite 32 im Skriptum hingeweisen wird, sind ca `1.0e-6` für einfache Genauigkeit und ca `1.0e-15` für doppelte Genauigkeit geeignet für eine relative Fehlertoleranz.

3. Das Polynom $(1-x)^3$ wird durch die folgenden Methoden mit 2-dezimalziffriger Genauigkeit und Aufrundung an der Stelle $x = 0.75$ ausgewertet:

   Summe: $p \approx 1 - 3x + 3x^2 - x^3$
   $p \leftarrow 1, t \leftarrow 3 \cdot x, p \leftarrow p - t, u \leftarrow x \cdot x, v \leftarrow 3 \cdot u, p \leftarrow p + v, w \leftarrow x \cdot u, p \leftarrow p - w$

   Horner: $q \approx 1 + x(-3 + x(3 - x))$
   $q \leftarrow 3 - x, q \leftarrow x \cdot q, q \leftarrow q - 3, q \leftarrow x \cdot q, q \leftarrow q + 1$

   Faktorisierung: $r \approx (1-x)^3$
   $t \leftarrow 1 - x, r \leftarrow t \cdot t, r \leftarrow t \cdot r$

   Kreuzen Sie bei den wahren Behauptungen an.

   (a) Der relative Fehler in p ist 28%.

   (b) Der relative Fehler in q ist 28%.

   (c) Der relative Fehler in r ist 2.4%.

   (d) Es gelten $p, q, r > 0$.

   Kommentare: Sehen Sie diesen Code. Sehen Sie Beispiel 5 und die Matlab Funktion `round`, um diese Rechnungen mit einem Matlab Code für ein beliebiges Polynom durchzuführen.

4. Ein beliebiges Polynom $p(x) = a_0 + a_1 x + \cdots a_n x^n$, $n \in \mathbb{N}$, wird durch die folgenden Methoden an einer gegebenen Stelle `x` ausgewertet. Sei `a = (a_0,a_1,...a_n)` ein Feld mit `a_k` $= a_k$, $k = 0, \ldots, n$.

```
p = a_0
for k from 1 to n
    p = p + a_k * x^k
end
```

```
q = a_n
for i from n to 1
    q = a_(i-1) + q * x
end
```

   Kreuzen Sie bei den wahren Behauptungen an.

   (a) Die Anzahl der `flops` für `p` ist $\mathcal{O}(n)$.

   (b) Die Anzahl der `flops` für `p` ist $\mathcal{O}(n^2)$.

   (c) Die Anzahl der `flops` für `q` ist $\mathcal{O}(n)$.

   (d) Die Anzahl der `flops` für `q` ist $\mathcal{O}(n^2)$.

   Kommentare: Sie können die Anzahl `f` der `flops` automatisch folgendermaßen ausrechnen:

```
f = 0
p = a_0
for k from 1 to n
    p = p + a_k * x^k
    f = f + k+1
end
print f
```

```
f = 0
q = a_n
for i from n to 1
    q = a_(i-1) + q * x
    f = f + 2
end
print f
```

5. Wie auf Seite 34 im Skriptum hingewiesen, wird $x \in \mathbb{R}$ mit d-dezimalziffriger Genauigkeit und Aufrundung mit der Matlab Funktion `round(x,d,'s')` berechnet. Hier soll eine eigene Funktion `myround(x,d)` programmiert werden, die diese Rechnung durchführt. Weiters sollen für gewisse Übungsbeispiele sollen alle Operationen `+`, `-`, `*` bzw. `/` durch `myround(a+b,d)`, `myround(a-b,d)`, `myround(a*b,d)` bzw. `myround(a/b,d)` implementiert werden. Die benötigte Funktion ist gegeben durch

```
function c = myround(x,d)
    if ((x == 0) || isnan(x))
        return;
    end
    s = floor(log10(abs(x)));
    s = 10^(s-d+1);
    c = __________;
end
```

wobei die leere Stelle mit einer unten stehenden Funktion eingefüllt werden soll. Kreuzen Sie bei den wahren Behauptungen an.

(a) `round(x/s)*s;`

(b) `sign(x)*floor(abs(x)/s+1/2)*s;`

(c) `sign(x)*ceil(abs(x)/s-1/2)*s;`

(d) `fix(x/s+1/2)*s;`

Kommentare: `round(x,d,'s')` oder `myround(x,d)` soll für *alle* `flops` in anderen Beispielen verwendet, in denen die dezimale Genauigkeit vorgegeben ist. Wie können diese Funktionen für vorgegebene *binäre* Genauigkeit angepasst werden und das Ergebnis in binärer Form ausgegeben?

6. Das folgende Gleichungssystem soll mit Gaußscher Elimination und Rückwärts Substitution gelöst werden:

$$\begin{array}{rcrcrcr} 4x & - & y & + & z & = & 8 \\ 2x & + & 5y & + & 2z & = & 3 \\ x & + & 2y & + & 4z & = & 11 \end{array}$$

Kreuzen Sie bei den wahren Behauptungen an.

(a) Mit 1-dezimalziffriger Genauigkeit und Aufrundung ist das Ergebnis
$$x = 1,\ y = -1,\ z = 1.$$

(b) Mit 2-dezimalziffriger Genauigkeit und Aufrundung ist das Ergebnis
$$x = 1.0,\ y = -0.98,\ z = 2.9.$$

(c) Mit 3-dezimalziffriger Genauigkeit und Aufrundung ist das Ergebnis
$$x = 1.00,\ y = -1.00,\ z = 3.00.$$

(d) Die exakte Lösung des Gleichungssystems wird nicht mit endlich-dezimalziffriger Genauigkeit berechnet.

Kommentare: Sehen Sie diesen Code. Genau der Algorithmus auf Seite 42 im Skriptum soll implementiert werden, wobei *jede* Operation $(+, -, \times, \div)$ mittels einer Funktion aus Beispiel 5 durchgeführt wird, um die genannte Genauigkeit einzuhalten.

7. Seien $p, q, n \in \mathbb{N}$. Eine Bandmatrix $A \in \mathbb{R}^{n\times n}$ sei mit der Bandbreite $p+q+1$ gegeben, wobei p die linke und q die rechte Halbbandbreiten sind, d.h. die Anzahl der streng unteren Diagonalen ist p, und die Anzahl der streng oberen Diagonalen ist q. Der Pseudo-Code für Gaußsche Elimination auf Seite 42 im Skriptum lässt sich folgendermaßen anpassen,

```
for k=1,...,n-1
  ...
  for i=k+1,...,min(n,k+p)
    ...
    for j=k+1,...,min(n,k+q)
```

um die Anzahl $\alpha(n,p,q)$ der `flops` für diesen Algorithmus zu minimieren. Kreuzen Sie bei den wahren Behauptungen an.

(a) Für beliebige $1 \le p, q \le n$ gilt $\alpha(n,p,q) \le (n-1)p(1+2q)$.

(b) Für beliebige $1 \le p, q \le n$ gilt $\alpha(n,p,q) \le (1+2p)(1+2q)$.

(c) Für $p = q = N$ und $n = N^2$ gilt $\alpha(N^2,N,N) = \mathcal{O}(N^3)$.

(d) Für $p = q = N$ und $n = N^2$ gilt $\alpha(N^2,N,N) = \mathcal{O}(N^4)$.

Kommentare: Sehen Sie diesen Code. Um die Berechnung einer oberen Schranke zu erleichtern, können die Schleifengrenzen `min(n,k+p)` und `min(n,k+q)` mit `k+p` ($\ge$ `min(n,k+p)`) bzw. `k+q` ($\ge$ `min(n,k+q)`) ersetzt werden. Um die Berechnung einer unteren Schranke zu erleichtern, kann die Schleifengrenze `n-1` für das Pivot-Index mit `min(n-p,n-q) = n - max(p,q)` ersetzen werden, und dann gelten `min(n,k+p) = k+p` und `min(n,k+q) = k+q`.

8. Basierend auf Seiten 38-39 im Skriptum sei der folgende Matlab Code gegeben,

```
tic;
N1    = N-1;
N4    = round(N/4);
z     = zeros(N4,1);
o     = ones(N-2*N4,1);
ustar = [z;o;z];
ustar = kron(ustar,ustar');
nu    = 0.1;  mu   = 0.1;
v     = ustar + nu*randn(N,N);
h     = 1/N;
d   = spdiags([-ones(N1,1), ...
          ones(N1,1)],[0,1],N1,N);
i   = speye(N);
Dx  = kron(i,d);
Dy  = kron(d,i);
D   = Dx'*Dx + Dy'*Dy;
I   = speye(N*N);
u   = (I + mu*D/h^2) \ v(:);
u   = reshape(u,N,N);
tau = toc;
```

wobei `N` $= N \in \mathbb{N}$ und `mu` $= \mu > 0$. Das rauschfreie Bild ist `ustar`. Die Rauschstufe im Bild `v` ist `nu` $= \nu > 0$. Das entrauschte Bild ist `u`. Diese Bilder lassen sich mit dem Matlab Befehl `imagesc` folgendermaßen darstellen:

```
subplot(1,3,1); imagesc(ustar); colormap('gray'); axis image; axis off;
subplot(1,3,2); imagesc(v);     colormap('gray'); axis image; axis off;
subplot(1,3,3); imagesc(u);     colormap('gray'); axis image; axis off;
```

Für ein gegebenes $N \gg 1$ sei $\tau_N =$ `tau` die Laufzeit, die vom Code berechnet wird. Laut Rechnungen gilt

(a) $\tau_N = \mathcal{O}(N)$, wenn `u` durch `u = (I + mu*D/h^2) \ v(:)` berechnet wird.

(b) $\tau_N = \mathcal{O}(N^2)$, wenn `u` durch `u = (I + mu*D/h^2) \ v(:)` berechnet wird.

(c) $\tau_N = \mathcal{O}(N^4)$, wenn `u` durch `u = full(I + mu*D/h^2) \ v(:)` berechnet wird.

(d) $\tau_N = \mathcal{O}(N^6)$, wenn `u` durch `u = full(I + mu*D/h^2) \ v(:)` berechnet wird.

Kommentare: Sehen Sie diesen Code. Da Gaußsche Elimination für eine $n \times n$ Matrix $\mathcal{O}(n^3)$ `flops` kostet, kostet Gaußsche Elimination für die Matrix `(I + mu*D/h^2)` $\mathcal{O}((N^2)^3)$ `flops`. Obwohl die Beziehung zwischen Laufzeit und `flops` nicht einfach ist, ist $\tau_N = \mathcal{O}(N^6)$ zu erwarten, wenn alle Elemente von `full(I + mu*D/h^2)` bearbeitet werden müssen. Die Anzahl der `flops` und die Laufzeit sind aber signifikant weniger, wenn `sparse(I + mu*D/h^2)` verwendet wird.