Bsp 2

a) Für eine SPD (Symmetrisch, Pos. Def.) Matrix $A \in \mathbb{R}^{n\times n}$ ist $\|x\|_A = (x^T A x)^{1/2}$ eine Vektornorm auf $\mathbb{R}^n$.

(1) zz: $\|x\|_A \geq 0$, $\|x\|_A = (\underbrace{x^T A x}_{>0 \text{ für } x\neq 0 \text{ und } =0 \text{ für } x=0})^{1/2} \geq 0$ ✓

(2) zz: $\|x\|_A = 0 \Leftrightarrow x = 0$ (denn A pos. definit)

"$\Rightarrow$" $\|x\|_A = 0 = (x^T A x)^{1/2} \Rightarrow x = 0$

"$\Leftarrow$" $x = 0 \Rightarrow (x^T A x)^{1/2} = 0$ ✓

(3) zz: $\|\alpha x\|_A = |\alpha| \cdot \|x\|_A$

$\|\alpha x\|_A = (\alpha x^T A \alpha x)^{1/2} = (\alpha^2 x^T A x)^{1/2} = |\alpha| \cdot (x^T A x)^{1/2} = |\alpha| \cdot \|x\|_A$

(4) z.z.: $\|x+y\|_A \leq \|x\|_A + \|y\|_A$

$\|x+y\|_A^2 = (x+y)^T A (x+y) \overset{A \text{ symm.}}{=} x^T A x + 2x^T A y + y^T A y$

$\leq \|x\|_A^2 + 2|x^T A y| + \|y\|_A^2 = \|x\|_A^2 + 2\cdot\langle x, Ay\rangle + \|y\|_A^2$

$\overset{\text{C.S.}}{\leq} \|x\|_A^2 + 2\cdot\underbrace{(\langle x, Ax\rangle)^{1/2}\cdot(\langle y, Ay\rangle)^{1/2}}_{=(x^T A x)^{1/2}\cdot(y^T A y)^{1/2}} + \|y\|_A^2 \leq (\|x\|_A + \|y\|_A)^2$ ✓

=||A^(1/2)x|| *||A^(1/2)y||

wobei mit A=VLV^T, A^(1/2)= VL^(1/2)V^T

$\Rightarrow$ a ist richtig ✓

b) Für eine Matrix $S = \{s_{i,j}\}_{i,j=1}^n$ mit positiven Einträgen $s_{i,j} > 0$ ist $\|x\|_S = \|Sx\|_2$ eine Vektornorm auf $\mathbb{R}^n$.

Da für S nicht vorausgesetzt ist, dass S regulär ist kann $Sx = 0$ sein obwohl $x \neq 0$

$\Rightarrow$ b ist falsch. ✗

c) Mit der konvexen Glättung der Betragsfunktion

$|x|_\varepsilon = \sqrt{x^2+\varepsilon^2} \approx |x|$, $\varepsilon > 0$

ist eine Vektornorm auf $\mathbb{R}^n$ gegeben durch

$\|x\|_{1,\varepsilon} = \sum_{i=1}^n |x_i|_\varepsilon$, $x = \{x_i\}_{i=1}^n$

wenn $x = 0$: $\|0\|_{1,\varepsilon} = \sum_{i=1}^n |0|_\varepsilon = \sum_{i=1}^n \sqrt{0^2+\varepsilon^2} = n\cdot\varepsilon > 0$ $\Rightarrow$ keine Vektornorm

$\Rightarrow$ c ist falsch ✗

d) Für jedes $n \in \mathbb{N}$ $n \geq 2$ und für jedes $\delta > 0$ gibt es eine Vektornorm $\|\cdot\|_\delta$ auf $\mathbb{R}^n$, die erfüllt

$|\|x\|_\delta - |x_1|| \leq \delta \|x\|_1$, $\forall x = \{x_i\}_{i=1}^n \in \mathbb{R}^n$

Sei $\|x\|_\delta = \sum_{i=2}^n |x_i|\cdot\delta + |x_1|$

$\rightsquigarrow \big|\|x\|_\delta - |x_1|\big| = \sum_{i=2}^n |x_i|\cdot\delta$

$\rightarrow \sum_{i=2}^n |x_i|\cdot\delta \leq \delta \sum_{i=1}^n |x_i| = \delta|x_1| + \delta\sum_{i=2}^n |x_i|$

$0 \leq \underbrace{\delta}_{>0}\underbrace{|x_1|}_{\geq 0}$ ✓

noch zz $\|x\|_\delta$ ist eine Vektornorm

(1) $\sum_{i=2}^{n} \underbrace{|x_i|}_{\geq 0} \cdot \underbrace{\delta}_{>0} + \underbrace{|x_1|}_{\geq 0} \geq 0$ ✓

(2) $\|x\|_\delta = 0 = \sum_{i=2}^{n} |x_i| \cdot \underbrace{\delta}_{>0} + |x_1| \Rightarrow x = 0$, $x = 0 \Rightarrow \sum_{i=2}^{n} 0 \cdot \delta + 0 = 0$ ✓

(3) $\|\alpha \cdot x\|_\delta = |\alpha x_1| + \sum_{i=2}^{n} |\alpha x_i| \delta = |\alpha| \left(|x_1| + \sum_{i=2}^{n} |x_i| \delta \right) = |\alpha| \|x\|_\delta$ ✓

(4) $\|x + y\|_\delta = |x_1 + y_1| + \sum_{i=2}^{n} |x_i + y_i| \delta \leq |x_1| + |y_1| + \sum_{i=2}^{n} |x_i| \delta + \sum_{i=2}^{n} |y_i| \delta = \|x\|_\delta + \|y\|_\delta$

$\Rightarrow$ Vektornorm ✓

$\Rightarrow$ d) ist richtig ✓

Übungszettel 4 Beispiel 4

Wir werden dieses Beispiel zuerst theoretisch lösen und anschließend durch Rechnung mit MatLab überprüfen:

Es lässt sich leicht überprüfen, dass 1 ein Eigenwert von B ist. Laut der Vorlesung ist jede Matrixnorm größer oder gleich dem betragsmäßig größten Eigenwert der Matrix. Es existiert also eine reguläre Matrix A, so dass $B = A^{-1}*D*A$ mit D Diagonalmatrix oder einer Jordanmatrix. Damit gilt $B^n=A^{-1}*D^n*A$. Also ist B nur konvergent, wenn D^n konvergent ist. Da aber auf der Hauptdiagonale von D ein Eintrag den Wert 1 hat, kann D nicht konvergent sein.

Dasselbe Argument kann man auch für Antwort c verwenden. Die Antworten a und c sollten somit falsch sein.

Bei den Antworten b und d wird beide Male die Einheitsmatrix von B bzw C abgezogen, womit die 1 als Eigenwert nicht mehr vorkommt, sondern die 0 d.h. diese Matrizen sollten konvergent sein. Damit sollten die Antworten b und d stimmen.

<u>Überprüfung mittels Matlab:</u>

```
kappa=0.1;
tau=0.1;
v=0.1;
T=6;
time=[];
N=7;
h=1/N;
Q=spdiags(ones(N-1,2),[0,1],N-1,N);
b=-2*speye(N-1);
c=zeros(N-1,1);
b=[b,c];
Q=full(Q+b);
D=Q'*Q;

B=speye(N)-tau*kappa/h^2*D;
C=speye(2*N-1)+tau*v/h*[diag(0,N-2),Q;-Q',diag(0,N-1)];

%a) Setze tau*kappa/h^2=1/2
Ba=speye(N)-0.5*D;
Ba=Ba^200;
maximuma=max(max(Ba))

%b) Setze tau*kappa/h^2=1/4
Bb=0.25*D;
Bb=Bb^500;
maximumb=max(max(Bb))

%c) Setze tau*v/h=1/2
Cc=speye(2*N-1)+0.5*[diag(0,N-2),Q;-Q',diag(0,N-1)];
```

Die Matrizen B und C werden konstruiert

Zuerst wird der jeweilige Faktor bestimmt, mit dem die Matrix multipliziert wird.

Anschließend multiplizieren wir die Matrix sehr oft miteinander und geben und zum Schluss den maximalen Eintrag des Ergebnisses aus (bei allen Antworten analog)

```
Cc=Cc^200;
maximumc=max(max(Cc))

%d) Setze tau*v/h=h/4
Cd=-h/4*[diag(0,N-2),Q;-Q',diag(0,N-1)];
Cd=Cd^300;
maximumd=max(max(Cd))
```

Die Ergebnisse, die uns Matlab liefert sind:

```
maximuma =  0.14286
maximumb =    2.6819e-12
maximumc =    2.3116e+28
maximumd = 0
```

Also konvergieren die Matrizen der Antworten b und d wirklich gegen 0, wohingegen die Matrizen der Antworten a und c nicht konvergieren.

UE Numerik

Aufgabe 6:

a) SGS: $x^{k+\frac{1}{2}} = (D+L)^{-1}[b - U x^k]$

$x^{k+1} = (D+U)^{-1}[b - L x^{k+\frac{1}{2}}]$

$= (D+U)^{-1}[b - L(D+L)^{-1}[b - U x^k]]$

$= (D+U)^{-1}[I - L(D+L)^{-1}]b + (D+U)^{-1}L(D+L)^{-1}U x^k$

$T = (D+U)^{-1}L(D+L)^{-1}U$

a wahr

c) $M^{-1} = (D+U)^{-1}[I - L(D+L)^{-1}]$

NR: $I - L(D+L)^{-1} = I - (L+D)(D+L)^{-1} + D(D+L)^{-1}$

$L = I - I + D(D+L)^{-1} = D(D+L)^{-1}$

$= (D+U)^{-1}D(D+L)^{-1}$

$\Rightarrow M = (D+L)D^{-1}(D+U)$

c wahr

b) SSOR: $x^{k+\frac{1}{2}} = (D+\omega L)^{-1}[(1-\omega)D - \omega U]x^k + \omega(D+\omega L)^{-1}b$

$x^{k+1} = (D+\omega U)^{-1}[(1-\omega)D - \omega L]x^{k+\frac{1}{2}} + \omega(D+\omega U)^{-1}b$

$= (D+\omega U)^{-1}[(1-\omega)D - \omega L](D+\omega L)^{-1}[(1-\omega)D - \omega U]x^k$

$+ [(D+\omega U)^{-1}[(1-\omega)D - \omega L](D+\omega L)^{-1}\omega(D+\omega L)^{-1} + \omega(D+\omega U)^{-1}]b$

$T = (D+\omega U)^{-1}[(1-\omega)D - \omega L](D+\omega L)^{-1}[(1-\omega)D - \omega U]$

b falsch, weil

- $(1-\omega)D - \omega L \neq L$, analog für $(1-\omega D) - \omega U \neq U$
- Konkretes Gegenbeispiel mit

$A = \begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix}$, $w = 1/2$

d) $M^{-1} = (D+\omega U)^{-1}[(1-\omega)D - \omega L]\omega(D+\omega L)^{-1} + \omega(D+\omega U)^{-1}$

$= (D+\omega U)^{-1}[\omega[(1-\omega)D - \omega L](D+\omega L)^{-1} + \omega I]$

$= \omega(D+\omega U)^{-1}[[2D - \omega D - (D+\omega L)](D+\omega L)^{-1} + I]$

$= \omega(D+\omega U)^{-1}[(2-\omega)D(D+\omega L)^{-1} - I + I]$

$= \omega(2-\omega)(D+\omega U)^{-1}D(D+\omega L)^{-1}$

5.

Damit M gleich wie in Angabe, muss

$\omega(2-\omega) = 1$ sein $\quad (\Leftrightarrow -\omega^2 + 2\omega - 1 = 0 \quad \Leftrightarrow \quad \omega = 1)$

also d ~~im Allgemeinen~~ falsch

Übersicht über die benötigten Funktionen:

Biexponentielle Funktion:

num1_blatt4_bsp8.m | jacobi_method.m | biexp.m | gauss_seidel_meth

```
1    %Biexponentielle Funktion mit Parametern nach Blatt
2    function [x] = biexp( k, c1, c2, lamb1, lamb2 )
3    x = c1 * exp(lamb1 *  k) + c2 * exp(lamb2 * k);
4    end
5
```

Jacobi-Algorithmus:

num1_blatt4_bsp8.m | jacobi_method.m | biexp.m | gauss_seidel_method.m

```
1   function [x, F] = jacobi_method( x_0, A, b, N, u )
2   %Inputs: A, b das zu loesende GLS, x_0 Start-Iteration,
3   %N Anzahl Iterationen, u "richtige" Loesung
4   %Outputs: letzte Iterierte (nicht unbedingt notwendig fuer Beispiel)
5   % und Array
6   d = diag(A);
7   n = length(b); %Laenge von b berechnen
8   x = x_0;
9   for it = 1:N %analog zu Skriptum, Folie 80
10      for i = 1:n
11          r(i) = b(i);
12          for j = 1:n
13              r(i) = r(i) - A(i,j) * x(j); %Residuum wird berechnet
14          end
15          dx(i) = r(i)/d(i); %entspricht Mult. des Residuums mit D^(-1)
16      end
17      for i = 1:n
18          x(i) = x(i) + dx(i); %neue Iterierte berechnen
19      end
20      x = reshape(x, [N,1]); %transponieren
21      norm(u-x); %2-Norm berechnen
22      F(it) = norm(u-x); %erstellt Array mit N Eintraegen:
23                         %jeweils Abstand der aktuell Iterierten von u
24  end
25  F = reshape(F, [N,1]); %transponieren
26  end
```

Gauss-Seidel-Algorithmus:

num1_blatt4_bsp8.m | jacobi_method.m | biexp.m | gauss_seidel_method.m*

```
function [x, F] = gauss_seidel_method(x_0, A, b, N, u)
%Inputs: A, b das zu loesende GLS, x_0 Start-Iteration,
%N Anzahl Iterationen, u "richtige" Loesung
%Outputs: letzte Iterierte (nicht unbedingt notwendig fuer Bsp)und Array
d = diag(A);
n = size(A,1);
x = x_0;
for it = 1:N %Algorithmus wie im Skriptum, Folie 81
%dies ist genau genommen nicht der eigentliche Gauss-Seidel-Alg., wie er
%auf Folie 82 ganz unten steht, sondern hier verwende ich die Rechnung,
%die auf dieser Folie ganz oben steht: x_k+1 = D^(-1) [b - L*x_k+1 - U*x_k]
%Mein r ist also nicht das Residuum im klassischen Sinn
    for i = 1:n
        r(i) = b(i);
        for j = 1:n
            if j ~=i
              r(i) = r(i) - A(i,j) * x(j);
            end %fuer j < i wird mit neuer Iterierter gerechnet,
                %fuer j > i mit alter Iterierter
        end
        x(i) = r(i)/d(i);
    end
    x = reshape(x, [N,1]); %transponieren
    F(it) = norm(u-x); %erstellt Array mit N Eintraegen:
                       %jeweils Abstand der aktuell Iterierten von u
end
F = reshape(F, [N,1]); %transponieren
end
```

Genereller Code

num1_blatt4_bsp8.m | jacobi_method.m | biexp.m | gauss_seidel_method.m

```
clc;
clear;

%Angabe vom Blatt
N = 100;
nu = 0.05;
mu = 0.005;
h = 1/N;
us = [zeros(round(N/2),1);ones(N-round(N/2),1)];
v = us + nu * randn(N,1);
I = speye(N);
A = spdiags([-ones(N-1,1),ones(N-1,1)], [0,1],N-1,N);
A = A'*A;
A = (I + mu*A/h^2);
u = A\v;

%wende die Algorithmen an, v jeweils als Start-Iterierte
[x_j,F_j] = jacobi_method(v, A, v, N, u);
[x_g,F_g] = gauss_seidel_method(v, A, v, N, u);

%berechne Arrays mit den verschieden parametrisierten Biexp.-funktionen
for i = 1:N
    B1(i) = biexp(i,0.5,0.7,-0.09,-0.02);
    B2(i) = biexp(i,0.7,0.5,-0.09,-0.02);
    B3(i) = biexp(i,0.6,0.6,-0.1,-0.03);
    B4(i) = biexp(i,0.4,0.4,-0.1,-0.03);
end
```

```
28
29      %transponieren
30 -    B1 = reshape(B1, [N,1]);
31 -    B2 = reshape(B2, [N,1]);
32 -    B3 = reshape(B3, [N,1]);
33 -    B4 = reshape(B4, [N,1]);
34
35      %berechne jeweils die Abstaende
36 -    v1 = norm(B1-F_j)
37 -    v2 = norm(B2-F_j)
38 -    v3 = norm(B3-F_g)
39 -    v4 = norm(B4-F_g)
40
41      %Bedingungen fuer Beispiel
42 -    if v1 < v2
43 -        disp('a richtig, b falsch')
44 -    else
45 -        disp('b richtig, a falsch')
46 -    end
47 -    if v3 < v4
48 -        disp('c richtig, d falsch')
49 -    else
50 -        disp('d richtig, c falsch')
51 -    end
```

Ausgabe/Ergebnis:

```
v1 =

    0.4782


v2 =

    1.0761


v3 =

    0.3840


v4 =

    0.8762

a richtig, b falsch
c richtig, d falsch
>>
```

```
% Zeige, dass der Code vom Herrn Kapp mit Gauss-Seidel uebereinstimmt:
x = b; X = b;
for it=1:itmax
    y = x;
    for i=1:n
        r(i) = b(i);
        for j=1:n
            if (j ~= i)
                r(i) = r(i) - A(i,j)*x(j);
            end
        end
        x(i) = r(i)/A(i,i);
    end
   R = b - A*X;
   X = X + (D+L)\R;
   disp(sprintf('||X-x|| = %0.2e',norm(X-x)))
   if (norm(x-y) < tol*norm(x))
      break;
   end
end
norm(x-(A\b))
```