Dr. R. Käppeli
M. Sprecher

# Musterlösung 5

**1.** **a)**
```
function [m1,m2,r]=circle_linear_fit(x,y)

n=100;
A=[2*x 2*y ones(n,1)];
b=x.^2+y.^2;
l=A\b;
m1=l(1);
m2=l(2);
r=sqrt(l(3)+m1^2+m2^2);

end
```

Verwendet man die Daten auf der Vorlesungshomepage so erhält man $m1 = 0.0500$, $m2 = 0.0518$ und $r = 1.0002$.

**b)**
```
function [m1,m2,r] = circle_gaussnewton(x,y)

tol=10^(-14);
maxstep=100;

%initial guess
m1=0;
m2=0;
r=1;

for i=1:maxstep
    i
[m1new,m2new,rnew]=circle_gaussnewton_step(x,y,m1,m2,r);
nchange=norm([m1new m2new rnew]-[m1,m2,r])
if nchange<tol
    break
end
m1=m1new;
m2=m2new;
r=rnew;
```

```
end


function [m1,m2,r] = circle_gaussnewton_step(x,y,m1,m2,r)

%Distanz zum Kreismittelpunkt
l=sqrt((x-m1).^2+(y-m2).^2);

%Residuum
res=l-r;

A=[(m1-x)./l (m2-y)./l -ones(size(x))];
b=-res;

x=A\b;
m1=m1+x(1);
m2=m2+x(2);
r=r+x(3);


end
```

Verwendet man die Daten auf der Vorlesungshomepage so erhält man $m1 = 0.0499$, $m2 = 0.0516$ und $r = 0.9998$. Es sind 7 Iterationsschritte nötig.

**2.** Allgemein kann der Winkel von $(u, v)$ zu $(x, y)$ berechnet werden mit

$$\arctan\left(\frac{y-v}{x-u}\right).$$

Da

$$\arctan'(x) = \frac{1}{1+x^2}$$

bekommt man für die lineare Approximation bei $(x_0, y_0)$

$$\begin{aligned}
\arctan\left(\frac{y-v}{x-u}\right) \approx\ & \arctan\left(\frac{y_0-v}{x_0-u}\right) && (1)\\
& +\frac{(v-y_0)}{(x_0-u)^2+(y_0-v)^2}(x-x_0) && (2)\\
& +\frac{-(u-x_0)}{(x_0-u)^2+(y_0-v)^2}(y-y_0). && (3)
\end{aligned}$$

**Siehe nächstes Blatt!**

```
x=[0;1;3;0];
y=[0;0;3;1];
angles=[44;63;225-180;26]/360*2*pi;


%Initial guess
m1=2;
m2=2;

%Parameter for Iteration
tol=10^(-14);
maxstep=100;

for i=1:maxstep
    i
[m1new,m2new]=pirate_gaussnewton_step(x,y,angles,m1,m2);
nchange=norm([m1new,m2new]-[m1,m2])
if nchange<tol
    break
end
m1=m1new;
m2=m2new;

end



function [m1,m2] = pirate_gaussnewton_step(x,y,angles,m1,m2)

%Denominator
den=(x-m1).^2+(y-m2).^2;

%Residuum
b=angles-atan((y-m2)./(x-m1));

%Matrix
A=[(y-m2)./den -(x-m1)./den];

x=A\b;
m1=m1+x(1);
m2=m2+x(2);

end
```

**Bitte wenden!**

Die Lösung ist $(2.0023, 1.9827)$

**3.** **a)** Wir erhalten

$$J = \begin{pmatrix} 1 & e^{-x_3 t_1} & -t_1 x_2 e^{-x_3 t} \\ \vdots & \vdots & \vdots \\ 1 & e^{-x_3 t_6} & -t_6 x_2 e^{-x_3 t_6} \end{pmatrix}$$

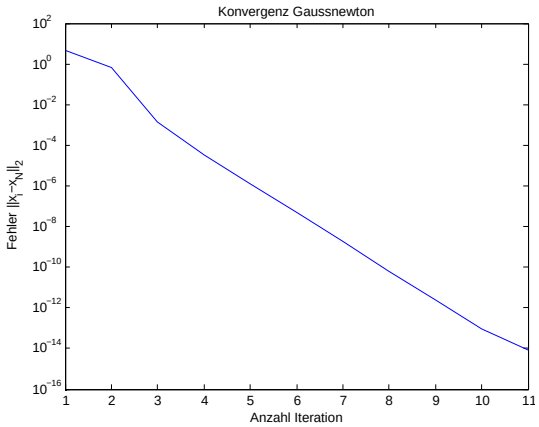
```
function [x] = gaussnewtonstep(x,t,T)
%berechnet einen Newtonschritt aus der Approximation x wird eine bes
%Approximation x berechnet
%hierbei sind x=(x_1,x_2,x_3) die Approximation der Koeffizienten
%t die Zeitpunkte an denen Messungen vorliegen
%T die Messdaten zu den Zeitpunkten t

%Funktion f
f=@(x,t,T) x(1)+x(2)*exp(-x(3)*t)-T;

%Gradient von f
Df=@(x,t) [1;exp(-x(3)*t);-x(2)*t*exp(-x(3)*t)];

%Matix und rechte Seite des linearen Problems
l=length(t);
A=zeros(l,3);
b=zeros(l,1);
for i=1:l
    A(i,:)=Df(x,t(i));
    b(i)=f(x,t(i),T(i));
end

%Korrektionsschritt
x=x-A\b;

end
```

Nach 11 Schritten ist die Norm der Korrektur kleiner als $10^{-13}$.

**b)**

```
%Startvektor
x=[25;45;0.3];

%Daten
t=2:2:12;
y=[47 36 31 27 24 23];
```

**Siehe nächstes Blatt!**




```
%Iteration
for l=1:1000
    x(:,end+1)=gaussnewtonstep(x(:,end),t,y);
    norm(x(:,end)-x(:,end-1))
    if norm(x(:,end)-x(:,end-1))<10^(-13)
        break;
    end
end
l

%Fehlerplot
err=zeros(1,l);
sol=x(:,end)
for i=1:l
    z=x(:,i);
    err(i)=norm(z-sol);
end

semilogy(1:l,err);
title('Konvergenz Gaussnewton');
xlabel('Anzahl Iteration');
ylabel('Fehler ||x_i-x_N||_2');
```

Die Konvergenz der Iteration scheint linear zu sein.

**c)** Für die Temperatur am Anfang erhalten wir $x_1 + x_2 \approx 63,5°C$. Herr Müller hat also keinen Grund sich zu beklagen, seine Suppe war wahrscheinlich heiss genug.

**4.** **a)**

```
function [x] = newtonstep(x,t,y)
```

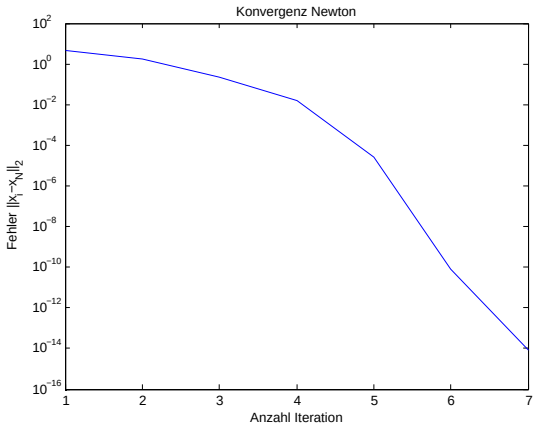
**Bitte wenden!**

```
%berechnet einen Newtonschritt aus der approximation x wird eine bes
%Approximation x berechnet
%hierbei sind x=(x_1,x_2,x_3) die Approximation der Koeffizienten
%t die Zeitpunkte an denen Messungen vorliegen
%y die Messdaten zu den Zeitpunkten t

%Funktion f
f=@(x,t,T) x(1)+x(2)*exp(-x(3)*t)-T;

%Gradient von f
Df=@(x,t) [1;exp(-x(3)*t);-x(2)*t*exp(-x(3)*t)];

%Hessian von f
D2f=@(x,t) exp(-x(3)*t)*[0 0 0;0 0 -t;0 -t x(2)*t^2];

%Gradient und Hessian vom Funktional
val=0;
grad=zeros(3,1);
hess=zeros(3);
for i=1:length(t)
    ti=t(i);
    Fi=f(x,ti,y(i));
    Ji=Df(x,ti);
    Hi=D2f(x,ti);
    val=val+.5*Fi^2;
    grad=grad+Ji*Fi;
    hess=hess+Ji*Ji'+Fi*Hi;
end
x=x-hess\grad;

end
```

Hier werden nur gerade 7 Iterationen benötigt damit der Fehler kleiner als $10^{-13}$ ist.

**b)**
```
x=[25;45;0.3];
t=2:2:12;
y=[47 36 31 27 24 23];

for l=1:1000
    x(:,end+1)=newtonstep(x(:,end),t,y);
    norm(x(:,end)-x(:,end-1))
    if norm(x(:,end)-x(:,end-1))<10^(-13)
        break;
```

**Siehe nächstes Blatt!**




```
    end
end
l

err=zeros(1,l);
sol=x(:,end)
for i=1:l
    z=x(:,i);
    err(i)=norm(sol-z);
end

semilogy(1:l,err);
title('Konvergenz Newton');
xlabel('Anzahl Iteration');
ylabel('Fehler ||x_i-x_N||_2');
```

Es scheint schneller als linear zu konvergieren (Das Newtonverfahren konvergiert quadratisch).

**c)** Man kommt auf das gleiche Resultat wie bei Aufgabe 3. Beide Verfahren approximieren dieselbe Lösung $x^*$.

**5.** Der Rang ist 3. Das Ausgleichsproblem besitzt daher keine eindeutige Lösung.

```
%tol fuer SVD Aschnitt
%tol = eps;      % --> grosser Fehler
tol = 10.*eps; % --> kleiner Fehler (WAHL von eps Problemabhaengig!)


A = [ 22, 10,  2,  3,  7; ...
```

**Bitte wenden!**

```
      14,  7, 10,  0,  8; ...
      -1, 13, -1,-11,  3; ...
      -3, -2, 13, -2,  4; ...
       9,  8,  1, -2,  4; ...
       9,  1, -7,  5, -1; ...
       2, -6,  6,  5,  1; ...
       4,  5,  0, -2,  2;];

b = [  0; ...
       1; ...
      11; ...
       4; ...
      -6; ...
       3; ...
      12; ...
      -5;];

% Groesse des Problems
[m,n]=size(A);

% SVD
[U,S,V] = svd(A);

% transformiere b --> b Hut
bh = U'*b;

% loese S xh = bh
xh = zeros(n,1);
for i=1:n
  if ( S(i,i) > tol )
    xh(i) = bh(i)/S(i,i);
  else
    xh(i) = 0;
  end
end

% transformiere xh zurueck --> x
x = V*xh;

disp('With SVD')
res = A*x - b;
res_norm = norm(res)
```

**Siehe nächstes Blatt!**

```
twonorm=norm(x)


disp('With Backslash')
xback=A\b
res=A*xback-b;
res_norm = norm(res)
twonorm=norm(xback)
```

Da beide Lösungen jeweils das Residuum minimieren sind die Residuen identisch. Die 2-norm ist bei der Lösung mit dem Backslashoperator grösser. Dies liegt daran, dass wir bei der Lösung mit der Singulärwertzerlegung die 2-norm minimiert haben. Der Backslashoperator versucht hingegen eine Lösung zu erhalten, die möglichst viele Nullen enthält.