Prof. Dr. Karsten Urban
M. Sc. Lewin Ernst

Winter term 2024/2025
Sheet 01, 30 Points

Numerical Optimization
Solution to exercise sheet

review on 23.10.2024 during the exercise class

Modus Operandi of the exercise class:

- You choose a group (four members at max) for the exercises on Moodle. There will be a poll, where you can choose your group. If you have no group by 18th of October you will get automatically assign to one.
- There will be an exercise sheet, uploaded every Wednesday before the exercise class starts. It usually consists of three up to four tasks, covering theoretical aspects or numerical experiments (with matlab).
- You have one week to work on the sheet with your group.
- Before the subsequent exercise class you and your group can pick on moodle the tasks for which you have a solution. You have to upload the solutions on moodle. Your group will already get 50 percent of the points for those tasks. For the submitted tasks, you must pick also at least one expert of your group, who is willing to present the solution. The experts will get 100 percent of the points for those tasks.
- During the exercise class I will pick for each task an expert, who presents then his/her solution to the class.
- In the end you should have at least 70 percent of all points to pass the preliminaries (a.k.a. Vorleistung).

1. *(Derivation of Gauß-Newton Method)*
Derive the Gauß-Newton method by using Taylor's expansion of the residual function $F : \mathbb{R}^n \to \mathbb{R}^m$ with a second order remainder term, i.e. expand $F(x)$ at the current iteration $x^{(k)} \in \mathbb{R}^n$ to approximate $F(x^{(k)} + s^{(k)})$ with an affine linear function.

(10 Points)

Solution: We expand the residual function $F(x)$ at $x^{(k)}$ with Taylor's expansion to get

$$F(x) = F(x^{(k)}) + J_F(x^{(k)})(x - x^{(k)}) + o(\|x - x^{(k)}\|_2), \quad \text{for } x \to x^{(k)},$$

where $J_F(x^{(k)}) \in \mathbb{R}^{m \times n}$ is the Jacobian of F at $x^{(k)}$. We define the step $s^{(k)} := x - x^{(k)}$ to derive

$$\begin{aligned} g(x^{(k)} + s^{(k)}) &\approx \frac{1}{2}\|F(x^{(k)}) + J_F(x^{(k)})s^{(k)}\|_2^2 =: \tilde{g}(s^{(k)}) \\ &= \frac{1}{2}F(x^{(k)})^T F(x^{(k)}) + F(x^{(k)})J_f(x^{(k)})s^{(k)} + \frac{1}{2}(s^{(k)})^T \left(J_F(x^{(k)})\right)^T J_f(x^{(k)})s^{(k)}. \end{aligned}$$

The first term is constant with respect to $s^{(k)}$ and the minimization of $\tilde{g}$ instead of g becomes

$$\min_{s^{(k)} \in \mathbb{R}^n} \tilde{g}(s^{(k)}) \quad \Leftrightarrow \quad \min_{s^{(k)} \in \mathbb{R}^n} F(x^{(k)})J_f(x^{(k)})s^{(k)} + \frac{1}{2}(s^{(k)})^T \left(J_F(x^{(k)})\right)^T J_f(x^{(k)})s^{(k)}. \tag{1}$$

This is a quadratic function, which has a unique minimum s^* with $\nabla\tilde{g}(s^*) = 0$ if $J_F(s^*)$ has full rank. We shall assume this in the following. So we deduce for the minimum

$$\nabla\tilde{g}(s^{(k)}) = 0 \Leftrightarrow \left(J_F(x^{(k)})\right)^T J_F(x^{(k)})s^{(k)} + \left(J_F(x^{(k)})\right)^T F(x^{(k)}) = 0$$

and eventually

$$s^{(k)} = -\left((J_F(x^{(k)}))^T J_F(x^{(k)})\right)^{-1} (J_F(x^{(k)}))^T F(x^{(k)}). \tag{2}$$

The next iteration is then defined by

$$x^{(k+1)} = x^{(k)} + s^{(k)} = x^{(k)} - \left((J_F(x^{(k)}))^T J_F(x^{(k)})\right)^{-1} (J_F(x^{(k)}))^T F(x^{(k)}),$$

which is exactly line 3 and 4 in the Algorithm 1.2.1.
Some remarks on the Gauß-Newton Method:

- The method can be seen as a quasi Newton scheme, because we apply the Newton method to the function ∇g and approximate the Jacobian $J_{\nabla g} = H_g$.
- We transfered the idea of Newton's method for nonlinear root-finding to nonlinear least-squares problems (LSQP). Namely, we have reduced the problem of solving one nonlinear LSQP to solving a sequence of linear LSQP.
- In this exercise, the approximation started earlier than it does in the lecture notes, namely at the very beginning by linearizing the nonlinear residual function F. In the lecture notes we approximate the Hessian of g at the end of the derivation.
- We could apply a pcg-method to solve (1) or we could solve (2) with a QR-decomposition. So we see, that there are many ways to solve the arising subproblems with methods from previous numerics courses.

As we have seen in the exercise class, we can derive the Gauß-Newton method in another way. Namely, by applying Newton's method to the residual function F. In order to do so, lets recall the Newton's method: We face the following problem

$$\text{Find } x \in \mathbb{R}^n : \quad F(x) = 0, \quad F : \mathbb{R}^n \to \mathbb{R}^m, m \geq n.$$

Note that this is different to the problem in Numerical Analysis, where we had $m = n$. We shall see what the influence of this is. We derive Newtons method for this problem: Let

$$J_F(x) = \begin{pmatrix} \nabla F_1(x) \\ \vdots \\ \nabla F_m(x) \end{pmatrix}$$

be the Jacobian of F, than we have by Taylor around $x^{(k)}$

$$F(x) = F(x^{(k)}) + J_F(x^{(k)})(x - x^{(k)}) + o(\|x - x^{(k)}\|) \quad x \to x^{(k)}.$$

We use the root of the linear polynomial of this as the next iterate $x^{(k+1)}$

$$x^{(k+1)} = x^{(k)} - J_F(x^{(k)})^{-1} F(x^{(k)}).$$

The Newton update is then

$$J_F(x^{(k)}) s^{(k)} = -F(x^{(k)}) \tag{3}$$

$$x^{(k+1)} = x^{(k)} + s^{(k)}. \tag{4}$$

How is the first equation (3) to be understood? On the left side we have the matrix $J_F(x^{(k)}) \in \mathbb{R}^{m \times n}$ and we notice now where $m \geq n$ comes into play. The equation (3) is to be understood as a linear least squares problem. By using `s_k = -JF_k \ F_k` in MATLAB to solve (3), a linear least squares solver is picked automatically and therefore MATLAB solves the normal equation

$$\left((J_F(x^{(k)}))^T J_F(x^{(k)}) \right) s^{(k)} = -(J_F(x^{(k)}))^T F(x^{(k)})$$

internally so that the algorithm is equivalent to Gauß-Newton.
The correct way to apply Newton's method to solve a minimization problem like

$$\text{Find } x \in \mathbb{R}^n : \quad \min_{x \in \mathbb{R}^n} g(x), \quad g : \mathbb{R}^n \to \mathbb{R}.$$

is to apply it to the problem

$$\text{Find } x \in \mathbb{R}^n : \quad \nabla g(x) = 0.$$

In this way we use the correct Hessian of g and we see the difference to the Gauß-Newton algorithm where we approximate the Hessian. By the way, in chapter 1.2.1 in the lecture notes we apply Newton's method in the first few lines (eq. (1.2.5)).

2. *(Convergence of Gauß–Newton)*
Consider for a parameter $\lambda \in \mathbb{R}$ the parametrized function

$$F_\lambda : \mathbb{R} \to \mathbb{R}^2, \qquad F_\lambda(x) = \begin{pmatrix} x + 1 \\ \lambda x^2 + x - 1 \end{pmatrix}$$

and the nonlinear least squares problem

$$g(x) := \frac{1}{2} \|F_\lambda(x)\|_2^2 \to \min. \tag{5}$$

a) Prove that the function g has a local minimum at $x^* = 0$, if $\lambda < 1$ and it's the only local minimum, if $\lambda < 7/16$.
b) Prove that $x^* = 0$ is a repulsive fixpoint[1] of the Gauß-Newton-Method if $\lambda < -1$, i.e. there exists a $\delta > 0$, such that

$$|x_{k+1} - 0| > |x_k - 0| \qquad \text{for all } x_k \text{ with } 0 < |x_k - 0| < \delta.$$

(4 + 4 = 8 Points)

Solution: Let

$$F_\lambda : \mathbb{R} \to \mathbb{R}^2, \qquad F_\lambda(x) = \begin{pmatrix} x+1 \\ \lambda x^2 + x - 1 \end{pmatrix}. \tag{6}$$

We consider the parameterized function $g_\lambda : \mathbb{R} \to \mathbb{R}$ with

$$g_\lambda(x) := \frac{1}{2}\|F_\lambda(x)\|_2^2 = \frac{1}{2}F_\lambda(x)^T F_\lambda(x) = \frac{1}{2}\left((x+1)^2 + (\lambda x^2 + x - 1)^2\right)$$

and the corresponding derivatives

$$\begin{aligned} g_\lambda'(x) &= (x+1) + (\lambda x^2 + x - 1)(2\lambda x + 1) = x\left[2\lambda^2 x^2 + 3\lambda x - 2(\lambda - 1)\right], \\ g_\lambda''(x) &= 6\lambda^2 x^2 + 6\lambda x - 2(\lambda - 1). \end{aligned}$$

a) The necessary condition for the existence of local minima gets us

$$g_\lambda'(x) = x\left[2\lambda^2 x^2 + 3\lambda x - 2(\lambda - 1)\right] \stackrel{!}{=} 0 \qquad \Leftrightarrow \qquad x^* = 0 \quad \text{oder} \quad 2\lambda^2 x^2 + 3\lambda x - 2(\lambda - 1) \stackrel{!}{=} 0.$$

In addition we have: $g_\lambda''(0) = -2(\lambda - 1) > 0 \quad \Leftrightarrow \quad \lambda < 1$. Therefore, at $x^* = 0$ is a local minima (i.e. x^* is a local minimizer), if $\lambda < 1$ holds. Other critical points are

$$2\lambda^2 x^2 + 3\lambda x - 2(\lambda - 1) \stackrel{!}{=} 0 \qquad \Leftrightarrow \qquad x_{1,2} = \frac{-3 \pm \sqrt{16\lambda - 7}}{4\lambda}.$$

If $\lambda < 7/16$, the radicant is negative. Therefore, the necessary condition for other local minimas can not be fulfilled and hence $x^* = 0$ is the only local minima, if $\lambda < 7/16$ holds.

b) The fixpoint function of the Gauß-Newton method is given by

$$\Phi^{GN}(x) = x - \left[F'(x)^T F'(x)\right]^{-1} F'(x)^T F(x) = x - \left[F'(x)^T F'(x)\right]^{-1} \nabla g(x). \tag{7}$$

We plug (6) into (7) and we get:

$$\Phi_\lambda^{GN}(x) = x - \frac{x[2\lambda^2 x^2 + 3\lambda x - 2(\lambda - 1)]}{1 + (2\lambda x + 1)^2} = \frac{4\lambda^2 x + 1}{4\lambda(2\lambda^2 x^2 + 2\lambda x + 1)} + \frac{x}{2} - \frac{1}{4\lambda}.$$

Let now $\lambda < -1$. We notice that Φ_λ^{GN} is at least continuously differentiable in a neighbourhood of $x^* = 0$ with

$$\frac{d}{dx}\Phi_\lambda^{GN}(x) = \frac{1}{2} - \frac{4\lambda^3 x^2 + 2\lambda x - 2\lambda + 1}{2(2\lambda^2 x^2 + 2\lambda x + 1)^2}$$

With $\frac{d}{dx}\Phi_\lambda^{GN}(0) = \lambda$ it also holds $\left|\frac{d}{dx}\Phi_\lambda^{GN}(0)\right| > 1$, if $\lambda < -1$. Due to the continuity of $\frac{d}{dx}\Phi_\lambda^{GN}$, there is a neighbourhood of $x^* = 0$ with $\left|\frac{d}{dx}\Phi_\lambda^{GN}(x)\right| > 1$ for all $x \in B_\varepsilon(0)$ with $\varepsilon > 0$ small enough. With the mean value theorem it follows

$$|x_{k+1} - 0| = |\Phi_\lambda^{GN}(x_k) - \Phi_\lambda^{GN}(0)| = \left|\frac{d}{dx}\Phi_\lambda^{GN}(\xi)\right| |x_k - 0| > |x_k - 0|,$$

if x_k is close enough at $x^* = 0$. Therefore $x^* = 0$ is a repulsive fixpoint of the Gauß-Newton method, if $\lambda < -1$.

[1]You can ask ChatGPT what a repulsive fixpoint is.

3. *(Gauß-Newton method,* MATLAB*)*

a) Write a MATLAB function

`x = GaussNewton(F, dF, x0, maxIt, tol),`

which computes the solution $\mathtt{x} \in \mathbb{R}^n$ of the nonlinear least squares problem

$$\mathtt{x} = \arg\min \|F(x)\|_2^2,$$

by using the Gauß-Newton-Method. You should be able to explain what happens in the code!
Hint: You are allowed to use ChatGPT (and you actually should do this).

b) Write a MATLAB function

`x = Newton(F, J, x0, maxIter, tol),`

which computes the solution $\mathtt{x} \in \mathbb{R}^n$ of the nonlinear root finding problem

$$\text{find } \mathtt{x} \in \mathbb{R}^n : F(x) = 0,$$

by using the Newton-Method, where `J` is the Jacobian of `F`.
Hint: You are allowed to use ChatGPT (and you actually should do this).

c) Extend the functions `GaussNewton.m` and `Newton.m`, such that all iterates `x` are returned. Further, adjust `GaussNewton.m` in a way, that for all iterations the relative error of the Hessian approximation of g is calculated and returned. This means calculate for all iterations $k = 1, 2, ...$

$$\frac{\|(\nabla^2 g)(x^{(k)}) - \left(F'(x^{(k)})\right)^T F'(x^{(k)})\|_2}{\|(\nabla^2 g)(x^{(k)})\|_2}$$

and return it.

d) Write a script in which you apply your MATLAB-functions `GaussNewton.m` and `Newton.m` to the nonlinear least squares-problem (5) using the initial value $x_0 = 10$ and various values for λ with $\lambda < 7/16$. Plot the error $|0 - x^{(k)}|$ over the iterations k in a loglog-plot and also plot the relative error of the Hessian over the iterations. What do you observe?

(2 + 2 + 3 + 5 = 12 Points)

Solution:

a) The function could look like:

```
function [x, res, iter, error_hess] = gaussNewton_ChatGPT(func, ...
                                                          jacobian, hess, ...
                                                          x0, maxIter, tol)
    % GAUSSNEWTON Solve nonlinear least squares using the Gauss-Newton method.
    %
    % Inputs:
    %   func       - Handle to the function f(x) = residuals (nx1 vector)
    %   jacobian   - Handle to the Jacobian of the function J(x) (nxm matrix)
    %   hess       - Handle to the hessian of the function g
    %   x0         - Initial guess for the parameters (mx1 vector)
    %   maxIter    - Maximum number of iterations (default 100)
    %   tol        - Tolerance for convergence (default 1e-6)
    %
```

```
    % Outputs:
    %   x          - Optimized parameter vector (mx1) for all iterations
    %   res        - Final residual norm ||f(x)||
    %   norm_JJ    - Norm of (J' * J)

    % Set defaults if not provided
    if nargin < 4, maxIter = 100; end
    if nargin < 5, tol = 1e-6; end

    % Initialize variables
    x = x0;
    for iter = 1:maxIter
        % Evaluate the function and its Jacobian at the current x
        r = func(x(iter));                % Residual vector (n x 1)
        J = jacobian(x(iter));            % Jacobian matrix (n x m)

        % Norm of (J' * J)
        error_hess(iter) = norm((J' * J)-hess(x(iter)),2)/norm(hess(x(iter)),2);

        % Solve the linear system: J'J dx = -J'r
        dx = -(J' * J) \ (J' * r);  % Parameter update

        % Update the parameter vector
        x(iter+1) = x(iter) + dx;

        % Check for convergence
        if norm(dx) < tol
            fprintf('Converged in %d iterations.\n', iter);
            break;
        end
    end

    % Compute final residual norm
    res = norm(func(x(iter)));

    % Display a message if the method did not converge
    if iter == maxIter
        warning(['Gauss-Newton did not converge within the maximum number ' ...
            'of iterations.']);
    end
end
```

b) The function could look like:

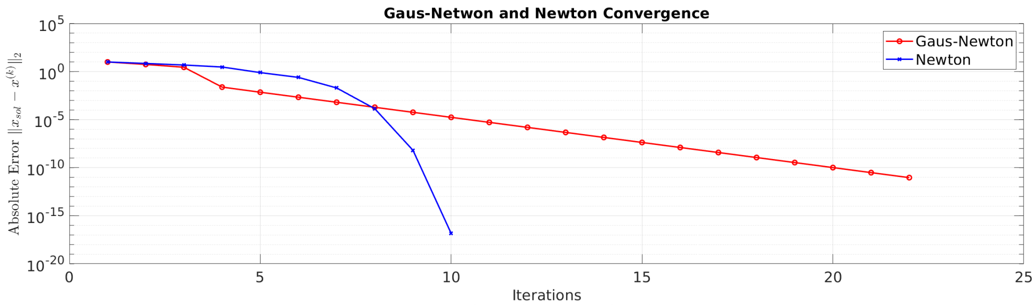
```
function [x, iter] = newtonsMethod(F, J, x0, maxIter, tol)
    % NEWTONSMETHOD Solve a system of nonlinear equations using Newton's method.
    %
    % Inputs:
    %   F          - Handle to the function F(x) = 0 (nx1 vector of equations)
    %   J          - Handle to the Jacobian matrix J(x) (nxn matrix)
    %   x0         - Initial guess for the solution (nx1 vector)
    %   maxIter    - Maximum number of iterations (default 100)
    %   tol        - Tolerance for convergence (default 1e-6)
    %
    % Outputs:
    %   x          - Solution vector (nx1) for all iterations

    % Set defaults if not provided
    if nargin < 4, maxIter = 100; end
    if nargin < 5, tol = 1e-6; end
```

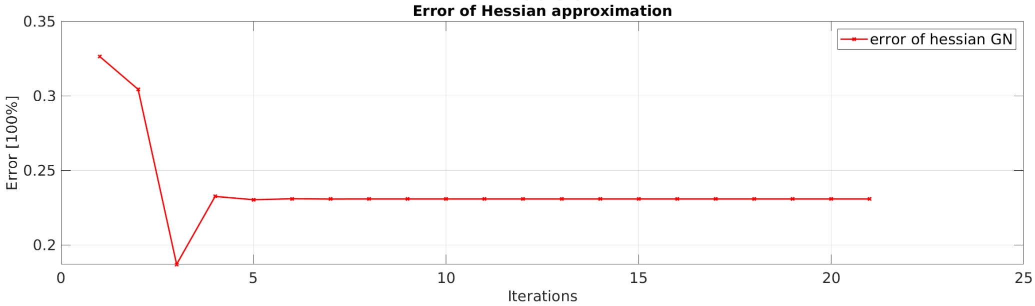
```
    % Initialize variables
    x = x0;

    for iter = 1:maxIter
        % Evaluate function and Jacobian at the current x
        Fx = F(x(iter));    % Function value (n x 1)
        Jx = J(x(iter));    % Jacobian matrix (n x n)

        % Solve for the Newton step: J(x) * dx = -F(x)
        dx = -Jx \ Fx;

        % Update the solution
        x(iter+1) = x(iter) + dx;

        % Check for convergence
        if norm(Fx,2) < tol
            fprintf('Newton''s method converged in %d iterations.\n', iter);
            break;
        end
    end

    % Display warning if max iterations are reached
    if iter == maxIter
        warning(['Newton''s method did not converge within the maximum' ...
            ' number of iterations.']);
    end
end
```

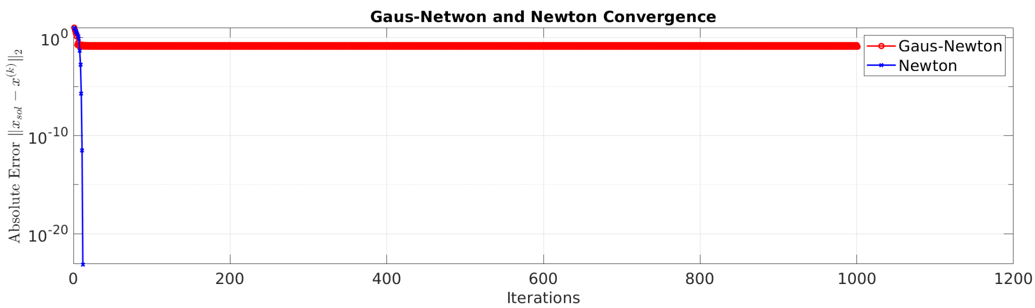
c) The script could look like:

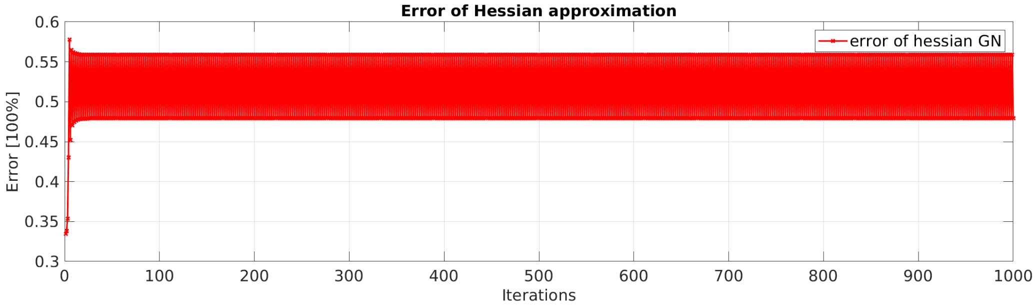
```
% Gauss-Newton Method on a Nonlinear Oscillatory Problem
clc; clear; close all;

% Define the parameter
lambda = -0.01;

% Define the residual function
f = @(x) [ x + 1; lambda*x.^2 + x - 1 ];

% Define the Jacobian of the residual function
J = @(x) [ 1; 2*lambda*x + 1 ];

% Function for newtons method
g       = @(x) 0.5*((x+1).^2+(lambda*x.^2+x-1).^2);
nabla_g = @(x) x*(2*lambda.^2*x.^2+3*lambda*x-2*(lambda-1));
Hess_g  = @(x) 6*lambda.^2*x.^2+6*lambda*x-2*(lambda-1);

% Define the initial value and termination parameters
x0    = 10;
maxIt = 1e3;
tol   = 1e-10;

% Apply the Gauss-Newton method
[x_GN, res, iter_GN, error_hess] = gaussNewton_ChatGPT(f, J, Hess_g, x0, maxIt, tol);
% Apply the Newton method
[x_N, iter_N] = newtonsMethod(nabla_g, Hess_g, x0, maxIt, tol);

% Plot the path taken by the Gauss-Newton method
figure(1);
subplot(2,1,1);
semilogy(1:iter_GN+1, vecnorm(x_GN,2,1), 'r-o', 'MarkerSize', 5, ...
```

```
      'LineWidth', 1.5);
hold on
semilogy(1:iter_N+1, vecnorm(x_N,2,1), 'b-x', 'MarkerSize', 5, ...
      'LineWidth', 1.5);

title("Gaus-Netwon and Newton Convergence");
xlabel('Iterations');
ylabel('Absolute Error $\Vert x_{sol}-x^{(k)} \Vert_2$',Interpreter='latex');
legend('Gaus-Newton','Newton');
grid on;
hold off
fontsize(16,"points")

subplot(2,1,2);
plot(1:iter_GN, error_hess, 'r-x', 'MarkerSize', 5, ...
      'LineWidth', 1.5);
hold on
title("Error of Hessian approximation");
xlabel('Iterations');
ylabel('Error [100%]');
legend('error of hessian GN');
grid on;
hold off;
fontsize(16,"points")
```








Error of Hessian approximation
error of hessian GN
Error [100%]
Iterations
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0
100
200
300
400
500
600
700
800
900
1000