Out: March 2, 2020
Due: March 15, 2020

NTNU – Trondheim
Norwegian University of
Science and Technology

TTK4130 Modeling and Simulation

Assignment 8

Introduction

Newton's method is one of the most important and powerful root-finding methods. Under the correct circumstances (see "Fourier conditions"), the convergence of this method is quadratic.

In this course, we will use Newton's method to solve the implicit equations that arise from the implementation of implicit Runge-Kutta schemes (IRK), and from the simulation of implicit ODEs and DAEs.

Before we dwell into IRK schemes and implicit differential equations, we will implement and test our own Newton's method. Under the testing, we will be specially interested in the situations where Newton's method fails. Finally, we will use the developed Newton's method to simulate models using the implicit Euler scheme.

Newton's method and their most relevant aspects for this course are explained in the additional lecture notes, which can be found on Blackboard.

Problem 1 (Newton's method)

(a) Implement Newton's method as a Matlab function. This function should have as arguments the objective function f, whose root we want to find, its Jacobian J and an initial estimate x_0. You are free to add arguments for the tolerance and the maximum number of iterations, or to hardcode some values for these parameters. The Matlab function should return at least the final root estimate.

Hint: A template for such a Matlab function can be found on Blackboard.

Solution: An example implementation that returns all root estimates is

```
function X = NewtonsMethod(f,J,x0,tol,N)
    if nargin < 5
        N = 100;
    end
    if nargin < 4
        tol = 1e-6;
    end
    X = NaN(length(x0),N+1);
    X(:,1) = x0;
    xn = x0;
    n = 1;
    fn = f(xn);
    iterate = norm(fn,Inf) > tol;
    while iterate
        xn = xn - J(xn)\fn;
        n = n+1;
        X(:,n) = xn;
        fn = f(xn);
        iterate = norm(fn,Inf) > tol && n <= N;
    end
    X = X(:,1:n);
end
```

(b) Use your implemented Newton's method to solve the equation

$$xy = 2 \tag{1a}$$

$$\frac{x^4}{4} + \frac{y^3}{3} = 1. \tag{1b}$$

Use the initial estimate $x_0 = y_0 = -1$.
Add a semilogarithmic plot of the infinity norm of the residuals ($||f(x_k)||_\infty$) to your answer.
Display the iteration values, and comment on the results.

Solution: In this case,

$$f(x) = \begin{bmatrix} x_1 x_2 - 2 \\ \frac{x_1^4}{4} + \frac{x_2^3}{3} - 1 \end{bmatrix}$$

$$F = \begin{bmatrix} x_2 & x_1 \\ x_1^3 & x_2^2 \end{bmatrix}$$

$$x_0 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

Figure 1 shows the infinity norm of the residuals.

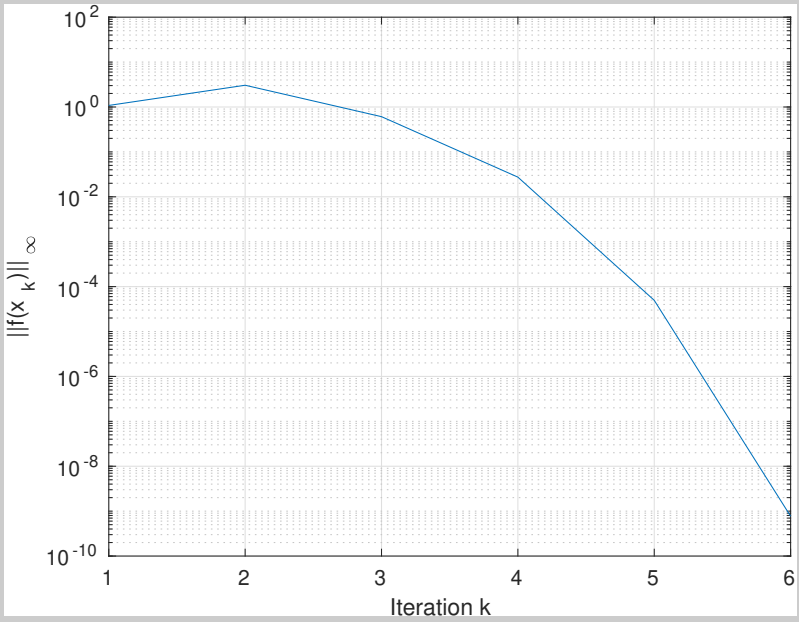

Figure 1: Infinity norm of the residuals.

The iterations are

k	$x_{k,1}$	$x_{k,2}$
0	-1.000000000000000	-1.000000000000000
1	-2.041666666666667	-0.958333333333333
2	-1.702689187972853	-1.138703714488933
3	-1.605757784225688	-1.239437100091089
4	-1.602360572320280	-1.248140068297339
5	-1.602375263119968	-1.248147076489921
6	-1.602375262956673	-1.248147076552865

Note that after 6 iterations we have achieved machine precision in the root estimate. Not bad!

(c) Test your implemented Newton's method on the function

$$f(x) = (x-1)(x-2)(x-3)+1 = x^3 - 6x^2 + 11x - 5. \tag{2}$$

Use the initial estimate $x_0 = 3$.
Add a plot of the obtained results to your answer.
Explain the reason behind these results.
How you would modify Newton's method to improve the root estimates?
Hint: What happens if one iteration is near a singular point?

Solution:
The results are shown in Figure 2.
We observe that the iterations go away from the root several times, i.e. the estimation error is increased after the iteration. This is because these iterations are very close to singular points, i.e. $\boldsymbol{J}(\boldsymbol{x}_k) \simeq \boldsymbol{0}$, which gives very large steps since

$$\Delta \boldsymbol{x}_k = -\left(\boldsymbol{J}(\boldsymbol{x}_k)\right)^{-1} \boldsymbol{f}(\boldsymbol{x}_k).$$

One way to improve the results is to reduce the iteration steps by an scaling factor, i.e.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \left(\boldsymbol{J}(\boldsymbol{x}_k)\right)^{-1} \boldsymbol{f}(\boldsymbol{x}_k), \quad \alpha_k \in (0,1].$$

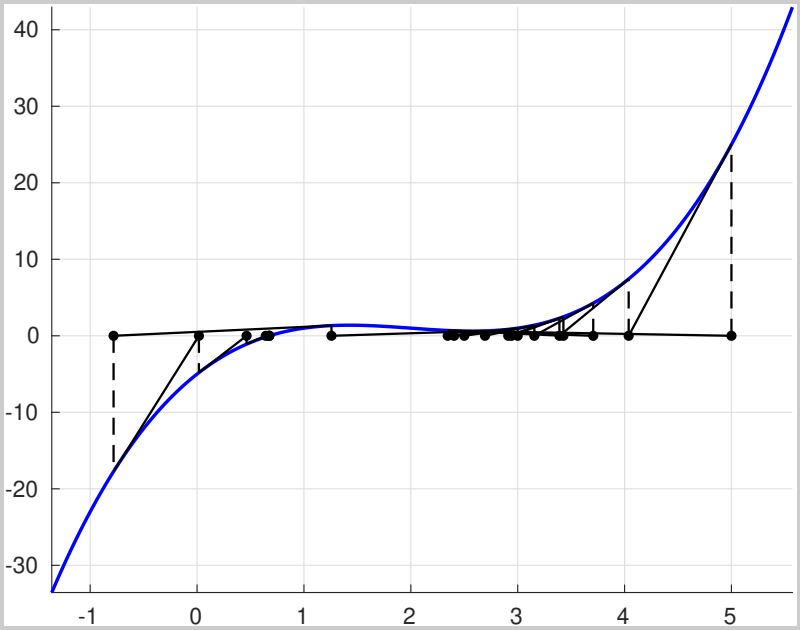

Figure 2: Objective function (blue) and iterations (black).

Note that initial estimates close to 3.0557 make the iterations diverge.

(d) Test your implemented Newton's method on the function

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} x_1 - 1 + (\cos(x_2)x_1 + 1)\cos(x_2) \\ -x_1 \sin(x_2)(\cos(x_2)x_1 + 1) \end{bmatrix}. \tag{3}$$

Use the initial estimate $\boldsymbol{x}_0 = [1, 3]^\top$.
Add a semilogarithmic plot of the infinity norm of the residuals ($||\boldsymbol{f}(\boldsymbol{x}_k)||_\infty$) to your answer.
Display the iteration values, and explain the reason behind the obtained results.
How you would modify Newton's method to improve the root estimates?
Hint: The closest root to $\boldsymbol{x}_0$ is $[1, \pi]^\top$. What is the Jacobian at this root?

Solution: Figure 3 shows the infinity norm of the residuals.

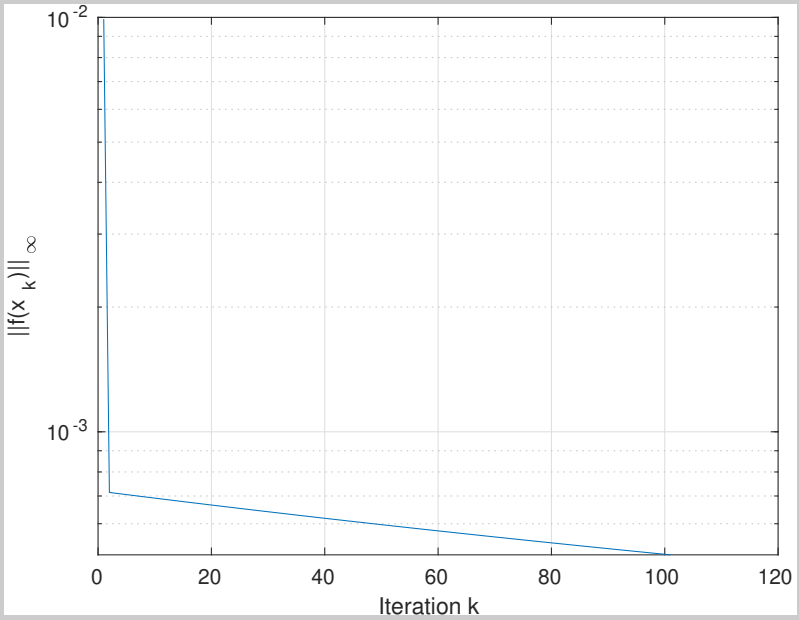

Figure 3: Infinity norm of the residuals.

The iterations get stuck and the convergence is very slow. The reason for this is that the Jacobian at the root is rank-deficient, as it is immediate to verify:

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{bmatrix} 1+\cos^2(x_2) & -\sin(x_2)(2x_1\cos(x_2)+1) \\ -\sin(x_2)(2x_1\cos(x_2)+1) & -x_1(\cos(x_2)-x_1+2x_1\cos^2(x_2)) \end{bmatrix}$$
$$\Rightarrow \boldsymbol{J}\left(\begin{bmatrix} 1 \\ \pi \end{bmatrix}\right) = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}.$$

The results can be improved by regularizing the Jacobian, i.e. by replacing $\boldsymbol{J}(\boldsymbol{x}_k)$ with $\boldsymbol{J}(\boldsymbol{x}_k) + \alpha_k \boldsymbol{I}$, where $\alpha_k > 0$ is a sufficiently small number.

Since a necessary condition for an extremum of a scalar function is that the gradient is zero at that point, Newton's method can also be used to find maxima and minima of a scalar function by finding the roots of the gradient.

(e) Use your implemented Newton's method to find the minimum of the function

$$f(\boldsymbol{x}) = 100(x_2 - x_1)^2 + (x_1 - 1)^4. \tag{4}$$

Use the initial estimate $\boldsymbol{x}_0 = [10, 10]^\top$.
What is the converge order of the iterations?
Explain the reason behind the obtained results.

Solution: We use Newton's method to find the root of the gradient of f, which we will denote by $\boldsymbol{g}$. Note that the Jacobian of the gradient is the Hessian. Hence,

$$\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x})^\top = \begin{bmatrix} 200(x_1 - x_2) + 4(x_1 - 1)^3 \\ 200(x_2 - x_1) \end{bmatrix}$$
$$\boldsymbol{J}(\boldsymbol{g})(\boldsymbol{x}) = \boldsymbol{H}(f)(\boldsymbol{x}) = \begin{bmatrix} 200 + 12(x_1 - 1)^2 & -200 \\ -200 & 200 \end{bmatrix}.$$

Since the actual root is $[1, 1]^\top$, the order of convergence can be easily calculated. The following table summarizes the iterations and their corresponding order:

k	$x_{k,1}$	$x_{k,2}$	order
0	10.000000000000000	10.000000000000000	-
1	9.666666666666666	9.666666666666666	1.039225551275191
2	9.333333333333332	9.333333333333332	1.040827441375583
3	8.999999999999998	8.999999999999998	1.042565776585920
4	8.666666666666664	8.666666666666664	1.044458771017486
5	8.333333333333330	8.333333333333330	1.046528032735783
6	7.999999999999997	7.999999999999997	1.048799393197894
7	7.666666666666664	7.666666666666664	1.051303992530660
8	7.333333333333331	7.333333333333331	1.054079717745691
9	6.999999999999998	6.999999999999998	1.057173135534751
10	6.666666666666665	6.666666666666665	1.060642130241614
11	6.333333333333332	6.333333333333332	1.064559566787685
12	5.999999999999999	5.999999999999999	1.069018475646280
13	5.666666666666666	5.666666666666666	1.074139553210767
14	5.333333333333333	5.333333333333333	1.080082281937281
15	5.000000000000000	5.000000000000000	1.087061888817103
16	4.666666666666667	4.666666666666667	1.095376065772228
17	4.333333333333334	4.333333333333334	1.105448713601580
18	4.000000000000001	4.000000000000001	1.117904889901083
19	3.666666666666667	3.666666666666667	1.133706495849563
20	3.333333333333334	3.333333333333334	1.154415278665706
21	3.000000000000001	3.000000000000001	1.182748963535320
22	2.666666666666667	2.666666666666667	1.223901085741545
23	2.333333333333334	2.333333333333334	1.289224226994100
24	2.000000000000001	2.000000000000001	1.409420839653211
25	1.666666666666667	1.666666666666667	1.709511291351430
26	1.333333333333337	1.333333333333337	43.972094003430804
27	1.000000000000019	1.000000000000019	-

We observe that the convergence is slightly superlinear with the exception of the last iteration. The convergence is not quadratic since the Hessian of f at the minimum is singular.

Problem 2 (Implicit Euler method)

(a) Based on the root-finding method developed in the previous problem, implement the implicit Euler method for a generic vector field $\boldsymbol{f}(t, \boldsymbol{x})$ (i.e. $\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x})$) as a Matlab function.

Explain what implicit equation has to be solved at each iteration of this IRK.

Add the implemented code to your answer.

Hint: A template for such a Matlab function can be found on Blackboard.

Solution: In order to implement the implicit Euler method, the equation

$$\boldsymbol{k}_1 = \boldsymbol{f}(t_{n+1}, \boldsymbol{x}_n + \Delta t_n \boldsymbol{k}_1)$$

has to be solved for $\boldsymbol{k}_1$ at each iteration.

Therefore, at each iteration, we define the function

$$g_n(\boldsymbol{k}) = \boldsymbol{k} - \boldsymbol{f}(t_{n+1}, \boldsymbol{x}_n + \Delta t_n \boldsymbol{k}).$$

The Jacobian of g_n is

$$\boldsymbol{J}(g_n)(\boldsymbol{k}) = \boldsymbol{I} - \Delta t_n \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(t_{n+1}, \boldsymbol{x}_n + \Delta t_n \boldsymbol{k}).$$

Hence, an example implementation of the implicit Euler method is

```
function x = ImplicitEuler(f,dfdx,T,x0)
    Nt = length(T);
    Nx = length(x0);
    dT = diff(T);
    x = zeros(Nx,Nt);
    x(:,1) = x0;
    xt = x0;
    for nt=2:Nt
        t = T(nt);
        dt = dT(nt-1);
        g = @(k) k-f(t,xt+dt*k);
        G = @(k) eye(Nx)-dt*dfdx(t,xt+dt*k);
        K = NewtonsMethod(g,G,xt,1e-6,1000);
        k = K(:,end);
        xt = xt + dt*k;
        x(:,nt) = xt;
    end
end
```

(b) Test the implemented integrator on the classic test system:

$$\dot{x} = \lambda x, \tag{5}$$

where $\lambda = -2$ and $x(0) = 1$.
Simulate until a final time $t_f = 2$ with a time-step $\Delta t = 0.2$.
Compare the results to the actual solution, and add a plot to your answer.

Solution: The results are shown in Figure 4.
Since the test system is relatively simple, an explicit expression for the iterations can be derived, which is

$$x_n = \frac{1}{(1 - \Delta t \lambda)^n} x_0.$$

We observe that the iterations obtained using Newton's method are very accurate approximations to the theoretical values.

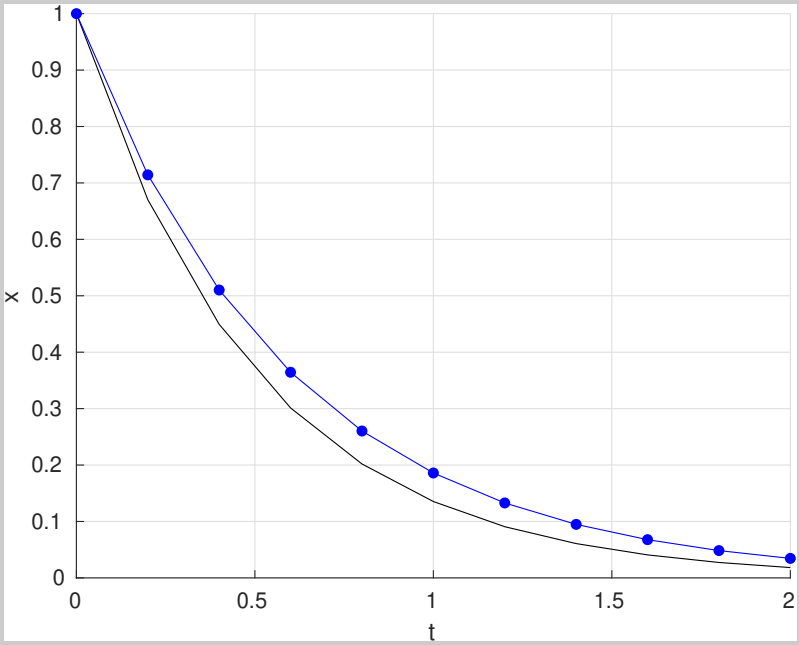

Figure 4: Actual solution (black) and approximation using the implicit Euler method (blue).