# homework4

December 1, 2025

## 1 Homework 4

study number: s234857

[17]:
```
import numpy as np
```

### 1.1 Multiple choice:

A: 3

B: 1

C: 2

D: 3

### 1.2 2) Newton's method

#### 1.2.1 1.

The given system and Jacomian matrix is written as a function taking X and returning the system evaluated in X (FX) and the Jacobian matrix (dFX), both as NumPy arrays,

[19]:
```
def F(X):
    FX = np.array([
        -X[0]**2 - X[0] + 2*X[1] - 18,
        (X[0] - 1)**2 + (X[1] - 6)**2 - 25
    ])

    dFX = np.array([
        [-2*X[0] - 1, 2],
        [2*X[0]- 2, 2*X[1] - 12]
    ])

    return FX, dFX
```

[20]:
```
F(np.array([1.5,11]))
```

[20]:
```
(array([0.25, 0.25]),
 array([[-4.,  2.],
        [ 1., 10.]]))
```

### 1.2.2 2.

The gradients of $F$ doesn't seem to blow up anywhere. Suiting starting points are therefore simply chosen close to the roots.

The graph suggests that the roots of the system (i.e the intersection between the individual expressions) are around $X = [-2, 10]$ and $[1.5, 11]$. These points are therefore chosen.

### 1.2.3 3.

The Newtonsys function is from the exercises is imported and used for the exercise:

[21]:
```
def Newtonsys(FdF, X0, kmax):
    """Solve system f(x) = 0 by Newton's method.

F(x) and F'(x) are computed by the user-supplied function FdF with the
 call [F,dF] = FdF(x), where F is a np.array holding the computed vector F(x)␣
 ↪and
 dF is a np.array holding the computed matrix F'(x).

The starting point is x0.

The function carries out kmax iterations, and stores all kmax iterations
as columns on the output matrix Xiterations.

Per Christian Hansen and Hans Bruun Nielsen, DTU Compute, April 11, 2016.

Translated to Python spring 2023.

    Args:
        FdF (function): Function that computes F(x) and F'(x). Must return two␣
 ↪np.arrays.
        X0 (np.array): The starting point
        kmax (int): Maximal number of iterations

    Returns:
        np.array: Iterations, stored columnwise
    """

    # Initialization
    X = X0
    Fx, dFx = FdF(X)
    H = np.linalg.solve(dFx,Fx) # This is the first step.

    # Create the array to store the iterations.

    Xiterations = []
    # Now iterate.
    for k in range(1, kmax+1):
```

```
        X = X - H
        Xiterations.append(X)
        Fx, dFx = FdF(X)
        H = np.linalg.solve(dFx,Fx)


    return np.array(Xiterations)
```

[22]:
```
X0s = [np.array([1.5,11]), np.array([-2,10])]
kmax = 100
for X0 in X0s:
    Xit = Newtonsys(F, X0, kmax)
    print((Xit[-1]))
```

```
[ 1.54694647 10.96999493]
[-2. 10.]
```

With 4 significant digits the found solution becomes

$$r_1 = [1.547, 10.97]$$
$$r_2 = [-2, 10]$$

### 1.2.4 4.

To examine convergence, we look at the series

$$\left\{ \frac{e_{k+1}}{e_k^2} \right\}_{k=0}^{6}.$$

If there is quadratic convergence, the series is expected to converge to some constant.

[23]:
```
X0 = np.array([0, 10])
max_iter = 8

X = Newtonsys(F, X0, max_iter)
Xstar = X[-1]
e = np.array([
    np.linalg.norm(X[k] - Xstar) for k in range(7)
])
print(([e[k+1] / e[k]**2 for k in range(6)]))
```

```
[np.float64(0.057679595628006036), np.float64(0.11122976486773244),
np.float64(0.1910709454299745), np.float64(0.255900032388262),
np.float64(0.27143660144657744), np.float64(0.2720111842618869)]
```

The series seems to approximately converge towards 0.27, i.e. indicating quadratic convergence.

## 1.3 3) Sensitivity analysis

### 1.3.1 1.

Using that $\delta|b_i| \leq 0.005, \forall i \in \{1, ..., 100\}$ and the triangle inequality,

$$\begin{aligned}\|\delta b\|_2 &= \left(\sum_{k=1}^{100} |b_1|^2\right)^{\frac{1}{2}} \\ &\leq \left(\sum_{k=1}^{100} 0.005^2\right)^{\frac{1}{2}} \\ &= 0.05.\end{aligned}$$

### 1.3.2 2.

Using the inequality for relative error from week 11 - slide 22„ we have

$$\begin{aligned}\frac{\|\delta x\|_2}{\|x\|_2} &\leq \kappa(A)\frac{\|\delta b\|_2}{\|b\|_2} \\ &\leq 2.2\frac{0.05}{1.4} \\ &= 0.07857\end{aligned}$$

giving the upper bound for $\|\delta x\|_2$