# Assignment 2: Optimization Methods

Steinnes L.

*Institute of Physics, University of Oslo*

*lasse.steinnes@fys.uio.no*

October 28, 2020

**Abstract**

The following sections give answers to Assignment 2 in the UiO course MAT4110: Introduction to Numerical Analysis. The codes producing the results discussed in Problem 2 and Problem 3, are available at https://github.com/lasse-steinnes/MAT4110/tree/master/Assignment2.

Consider the ordinary differential equation (ODE)

$$y''(x) = f(y(x)) \quad \text{for x} \in (0,1), \tag{1}$$

with the following boundary conditions (BCs)

$$y(0) = \alpha, \quad y(1) = \beta. \tag{2}$$

$f \in C(\mathbb{R})$ is a continuous function, and $\alpha, \beta \in \mathbb{R}$. $y : [0,1] \to \mathbb{R}$ is the unknown function, which is a solution for eq. 1 with BCs 2, if these are satisfied at all points $x \in (0,1)$ and at the boundaries.

## Problem 1

### a)

If one partitions the domain $x \in [0,1]$ into N intervals, with

$$x_i = ih, \qquad \text{where} \quad h = \frac{1}{N}, \quad i = (0,1,...,N). \tag{3}$$

Then $y(x_i) \approx z(x_i)$. An approximation to eq.1 is

$$\frac{d^2y}{dx^2} \approx \frac{\frac{y(x+h)-y(x)}{h} - \frac{y(x)-y(x-h)}{h}}{h} = \frac{y(x+h)-2y(x)+y(x-h)}{h^2}, \tag{4}$$

where h is small, but non-zero. Now using eq. 1, then eq. 4 becomes

$$z_{i-1} + 2z_i + z_{i+1} = h^2 f(z_i). \tag{5}$$

Thus $z_i$ can be found for inner points $i = (1,2,...N)$, where $z_0 = \alpha$ and $z_N = \beta$.

**b)**

The problem can be written as

$$Az = b(z), \tag{6}$$

where $A \in \mathbb{R}^{N+1} \times \mathbb{R}^{N+1}$ is a tridiagonal matrix, and $z, b(z) \in \mathbb{R}^{N+1}$. $z = [z_0, z_1, ..., z_N]^T$, and $b(z) = [\alpha, h^2(f(z_1), h^2 f(z_2))..., h^2 f(z_{N-1}), \beta]^T$. Then,

$$A = \begin{bmatrix} 1 & 0 & 0 & . & . & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & . \\ . & 0 & . & . & . & 0 \\ 0 & . & 0 & 1 & -2 & 1 \\ 0 & 0 & . & . & 0 & 1 \end{bmatrix}. \tag{7}$$

**c)**

Want to show that A is invertible from the assumption that A is diagonally dominant, i.e. there is at least one index $i$ such that $A_{i,i} > \sum_{j \neq i} |A_{i,j}|$. This can be proven by contradiction. If $Az = 0$ for $z \neq 0$, then A is singular (not invertible). Denote the elements of A by $a_{i,j}$. A row $r$ of the new vector $Az$ can be written

$$\sum_{j=0}^{N} a_{r,j} z_j = 0. \tag{8}$$

Now, move $a_{r,r}z_r$ over to the right hand side, where $z_r = \sup|z|$, and division by $z_r$ yields

$$a_{r,r} = -\sum_{j \neq r}^{N} a_{r,j} \frac{z_j}{z_r} \leq \sum_{j \neq r}^{N} \left| a_{r,j} \frac{z_j}{z_r} \right| \leq \sum_{j \neq r}^{N} |a_{r,j}|, \tag{9}$$

where the last inequality is a result of $|z_j z_r| \leq 1$. Eq. 9, resulting from a singular system, is in contradiction with the assumption that A is diagonally dominant. Thus A must be invertible.

**d)**

For a general matrix $B \in \mathbb{R}^{n\times n}$, to solve the system $Bu = c$ with $u, c \in \mathbb{R}^n$ takes O($n^3$) floating point operations (FLOPs). This can be explained by what operations would be needed for applying Gaussian elimination on the system. First, consider row-reduction. For the extended matrix $B|c$ one would first perform $n+1$ multiplications per row and row reduce $n-1$ rows. This is the first reduction. To row reduce $B$ to echeleon form require

$$\sum_{j=0}^{n-1}(n+(1-j))(n-(1+j)) = \sum_{j=0}^{n-1} n^2 - 2jn + (1-j)(1+j) \tag{10}$$

FLOPs. Or by using the sum of the $n-1$ first integer numbers,then

$$\text{eq. } 10 = (n-1)n^2 - 2n\frac{(n-1)n}{2} + n - \frac{1}{6}(n-2)(2n-1)n = O(n^3). \tag{11}$$

Likewise, backward-substitution to solve the equation would require $n^2 + O(n)$, because assume we first divide by $a_{ii}$ and operate on n-1 rows. Then one only needs two multiplications per row. Thus

$$\sum_{j=0}^{n-1} 2(n-j) = 2n(n-1) - \frac{2}{2}n(n-1) = 1n^2 + O(n). \tag{12}$$

Thus Gaussian elimination is at highest $O(n^3)$ operations.

However, in the case of a tridiagonal and diagonally dominant matrix, one would only need maximum 3 multiplication per row for the extended matrix $B|c$ to reduce it to echeleon form. Here, because of the zeroes, only one operation is need for the first and last iteration, whereas the other reduction steps only operates on two rows. Thus forward substitution requires

$$3(1+1) + 3 \cdot 2n = 6n + 6 \tag{13}$$

FLOPs. Likewise backward substitution would require $O(n)$ flops. Thus, for a tridiagonal, diagonally dominant matrix, one only needs $O(n)$ FLOPs.

## Problem 2

### a)

A fixed point iteration can be found from eq. 6, which approximates $z$. If a fixed point iteration $g$ is a contraction, then Banach's fixed point theorem promise convergence towards a fixed point $z^*$. Now let

$$z^{k+1} = g(z^k) = A^{-1}b(z^k). \tag{14}$$

Since $b$ depends on $z$, the ODE (*eq*.3) can be a non-linear system with no analytical solution. Therefore, one can make an initial guess for $z_0 \in \mathbb{R}^{n+1}$, and use a fixed point method to solve for the unknown $z$.

### b)

A function `fixedpointODE(f, alpha, beta, N, tolerance)` is written in python and takes input described in Problem 1 (sec. ). It runs the fixed point iteration from problem **a)** until $||z^{k+1} - z^k|| <$ tolerance, and returns the final iteration $z^{k+1}$. The L2-norm is used to compute the difference between iterations. For further details, see the Optimize class at [1].

### c)

The code was tested for two cases

$$\alpha = 0, \;\; \beta = 1, \;\; f(z) = -1 \tag{15}$$

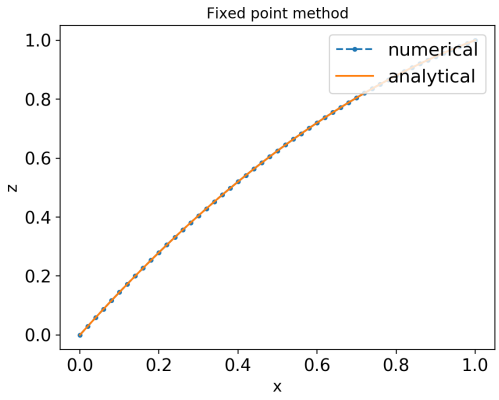
and

$$\alpha = 1, \;\; \beta = 1, \;\; f(y) = y^2. \tag{16}$$

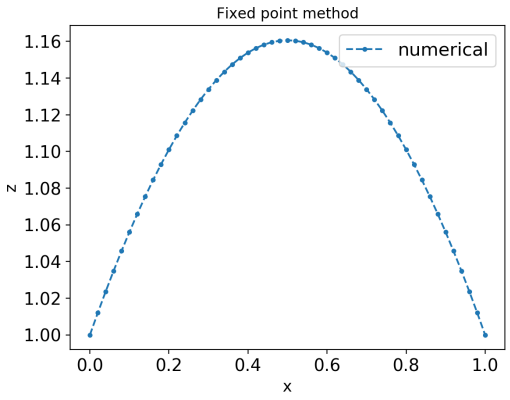
In both cases $N = 50$, $z^0 = 0$ and tolerance $= 10^{-12}$ were used as parameters. Eq. 15 has analytical solution

$$y = \frac{1}{2}x(3-x), \tag{17}$$

which can easily be shown by simple integration and use of the BCs. The results are presented in fig. 1




(a) Case 1 (eq. 15)




(b) Case 2 (eq. 16)

Figure 1: **Numerical fixed point ODE solver**: **a)** Case 1 is in agreement with the analytical solution **b)** Case 2 fulfill the boundary conditions. In both cases $N = 50$, $z^0 = 0$ and tolerance $= 10^{-12}$ were used as parameters.

### d)

Number of floating point operations per iteration are of the order $O(n^3)$ when using `numpy.linalg.inv`. The programme works by taking the inverse of A one time only, with $O(n^3)$ FLOPs. Then to solve for $z^{n+1}$, 3 multiplications for $n+1$ rows yields $O(n^3) + O(n)$ operations.. However, as seen in Problem 1 **d**, theoretically on could have only O($n$) FLOPs with `numpy.linalg.solve` if the tridiagonality of A is taken into account in the solver. With the mu

## Problem 3

### a)

Now let's use Newton's method to solve the contraction

$$z^* = g(z^*) = Az^* - b(z^*) = 0. \tag{18}$$

Newton's method for finding zeroes, can be applied on g(z), so that

$$z^{n+1} = z_n - \left[\nabla g(z^n)^T\right]^{-1} g(z^n), \tag{19}$$

which can be rewritten to

$$\nabla g(z^n)z^{n+1} = \nabla g(z^n)z^n - g(z^n). \tag{20}$$

Hence, it is this equation one must solve for $z^{n+1}$ in each iteration. In this case

$$\nabla g(z) = A - \nabla b(z), \tag{21}$$

where

$$\nabla b(z) = h^2 \begin{bmatrix} 0 & 0 & . & . & . & 0 \\ 0 & \left[\frac{\partial f}{\partial z}\right]_{z=z_1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \left[\frac{\partial f}{\partial z}\right]_{z=z_2} & 0 & . & . \\ . & 0 & 0 & . & 0 & . \\ 0 & . & . & 0 & \left[\frac{\partial f}{\partial z}\right]_{z=z_{n-1}} & 0 \\ 0 & 0 & . & . & 0 & 0 \end{bmatrix}. \tag{22}$$

**b)**

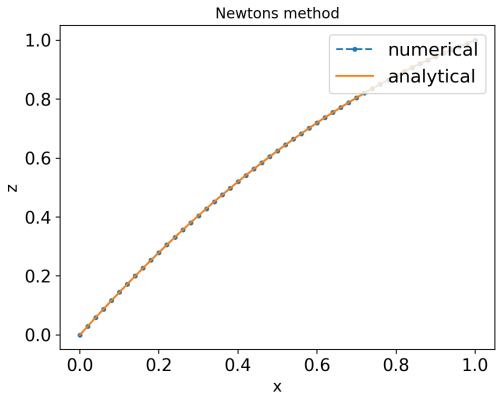
A programme applying Newton's method to find zeroes, is implemented in python which can be called by `newtonODE(f, df, alpha, beta, N, tolerance)`, where $df$ is the derivative of $f$ with respect to $z$. The L2-norm is again used to check the difference between $z^{n+1}$ and $z^n$ in each iteration. The programme can be found in the Optimize class at [1].

**c)**

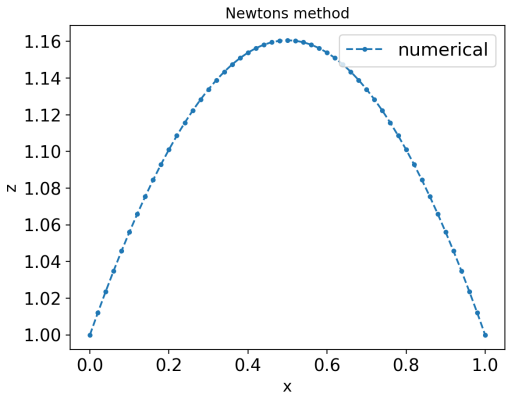
The `newtonODE` function is run for the two cases eq. 15 and eq. 16 with same parameters as in problem 2 **c)**. The results (fig. 2) are the same as with `fixedpointODE`.

**d)**

Number of floating point operations per iteration are probably of order $O(n^3)$ when using `numpy.linalg.solve` to solve eq. 20, if the tridiagonality of A is not taken into account in the solver. Calculating the right hand side of eq.20 is only a vector transformation with approximately $O(n)$ operations. If `numpy.linalg.solve` takes into account the tridiagonlity of A, then each iteration has $O(n)$ operations.




(a) Case 1 (eq. 15)




(b) Case 2 (eq. 16)

Figure 2: **Numerical Newton method ODE solver**: **a)** Case 1 is in agreement with the analytical solution **b)** Case 2 fulfill the boundary conditions. In both cases $N = 50$, $z^0 = 0$ and tolerance $= 10^{-12}$ were used as parameters.

**e)**

For the fixed point iteration, it is expected that `numpy.linalg.solve` would take longer time, than just inverting the matrix once. However, this is not the case, suggesting the module option `numpy.linalg.solve` provides some matrix specific operations.

Results are provided for time usage and number of iterations until convergence (tab. 1), for fixed point ODE with A inverted only once, and Newton's method using the option `numpy.linalg.solve`. It becomes evident that Newton's method converges a lot faster (for less iterations) for the case 2. Since the run-times have some stochasticity, the best approach would be to calculate CPU-times from averages of several runs, however, this is not done here. As expected the two methods are close in runtime, in accordance with the disussion in Problem 2**d)** and 3**d)**.

Table 1: **Benchmarks for ODE optimization methods**: CPU times and number of iterations until convergence is given for fixed point iteration and Newton's method for two cases (eqs.15-16). First guess for $z$ was $z^0 = 0$. Parameters: $N = 50$, tolerance $= 10^{-}12$.

| | | Fixed point iteration | Newton's method |
|---|---|---|---|
| **Case 1** | | | |
| | Number of iterations | 2 | 2 |
| | CPU-time (ms) | 602.684 | 481.707 |
| **Case 2** | | | |
| | Number of iterations | 18 | 5 |
| | CPU-time (ms) | 1185.87 | 994.246 |

## References

[1] Lasse Steinnes. Assignment2-GIT, 2020.