Norwegian University of Science
and Technology
Department of Mathematical
Sciences

TMA4215 Numerical mathematics Autumn 2017

Project 1

This project counts for 10% of the final grade in the course.
The deadline is **October 9, at 23:59**.

The report and the codes should be sent electronically to Abdullah Abdulhaque via email (abdullah.abdulhaque@ntnu.no). Collect all the material in a compressed zip-file and mark it with your candidate number(s), not your name(s).

Some hints:

- Before starting, make sure that you master PYTHON or MATLAB. You can also consult the *Getting started page* at the MATLAB documentation. For Python, we recommend the books posted at the homepage or the internet sites
  - https://www.python.org
  - https://docs.scipy.org/doc/
  - http://matplotlib.org
- Start as simple as possible. Do one thing at the time, and check that it is correct before proceeding. Compare with hand calculations on simple problems, if you find this easier.
- You should hand in a written answer to all the questions, in LaTeX. It is sufficient to just answer the questions. Do not write a traditional report.

1 **a)** Assume that the function f has a root ξ of multiplicity m. Prove by induction that all derivatives up to order m can be expressed as $f^{(n)}(x) = (x-\xi)^{m-n}h_n(x)$, where h_n is a function such that $h_n(\xi) \neq 0$.

b) Show that if f has a root ξ of multiplicity m, then g has a simple root ξ, where

$$g(x) = \frac{f(x)}{f'(x)}$$

Use this to modify Newton's method, given by

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

c) Show that if f has a root ξ of multiplicity m, then g has a simple root ξ, where

$$g(x) = \frac{f(x)}{f'(x)} + \frac{f''(x)f(x)^2}{2f'(x)^3}$$

Use this to modify Olver's method, given by

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} - \frac{f''(x_k)f(x_k)^2}{2f'(x_k)^3}$$

You can use MAPLE to calculate the derivatives since they are complicated.

For **d)**, follow the instructions at the end of the paper.

d) Create the program `Nonlinear_Solver` with the following submodules:

- `XML_Extraction`, for reading data from XML-files.
- `Newton`, for starting the naive algorithm in **b)**.
- `Improved_Newton`, for starting the improved algorithm in **b)**.
- `Olver`, for starting the naive algorithm in **c)**.
- `Improved_Olver`, for starting the improved algorithm in **c)**.
- `Plot_Function`, for plotting the function.

The error tolerance should be 10^{-14}. Check that the program is correct by testing both algorithms on the equations

$$x^2 - 4x + 3 = 0$$
$$e^{2x} - 6e^x + 8 = 0$$

e) Modify the previous algorithms such that they can also return an array with the errors $e_k = |x_k - x_{k-1}|$ and its length, in addition to the root. For both algorithms, plot e_{k+1}/e_k to see whether the error tends to zero. For Newton and Olver, plot respectively e_{k+1}/e_k^2 and e_{k+1}/e_k^3. This requires that you create a new submodule called `Plot_Convergence`, using `semilogy`.
As a test problem, use $x_0 = 0.4$ as initial value, and the function

$$f(x) = x^2 e^{-x^2}$$

Verify the convergence order of the four algorithms numerically. Describe your observations.

2 **a)** Find the Jacobian of the system

$$x^4 + x - y^3 - 1 = 0$$
$$x^2 + xy + y - 2 = 0$$

Show that Newton's method converges in $0 \leq x, y \leq 1$.

b) Find the Jacobian of the system

$$3x^2 + 7z - 2 = 0$$
$$5x^2 + x + y^4 - 8z + 2 = 0$$
$$-x^5 + y^3 + 4y + 5z^2 - 1 = 0$$

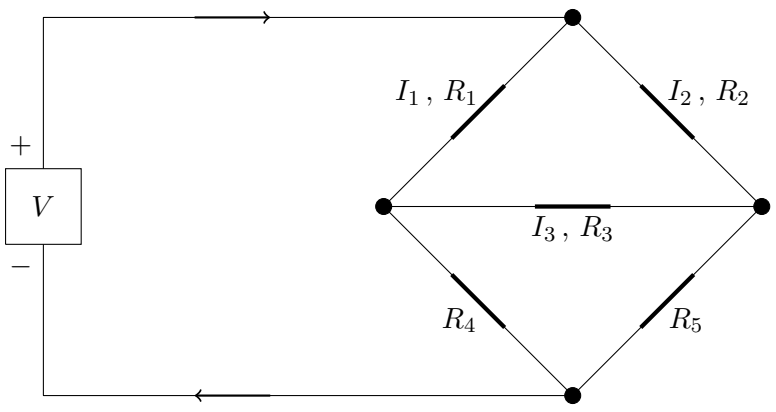
Show that Newton's method converges in $0 \leq x, y, z \leq 1$.

c) Create the program `Nonlinear_Solver_R2`. The input are the entries of the initial guess vector $\mathbf{x}_0$, and the error tolerance should be 10^{-14}. Define the nonlinear system and its Jacobi matrix as function handles before the iteration, and use this program to find the zeros of the previous nonlinear systems (for the second one, create `Nonlinear_Solver_R3`).

d) We will now examine an electric circuit. For such problems, it is necessary to apply Kirchhoff's rules:

$$\sum I = 0 \qquad \text{(junction)}$$
$$\sum V = 0 \qquad \text{(loop)}$$

The voltage in the battery is the sum of the other voltages in the corresponding loop. Use these given informations to formulate the linear system of equations describing the currents. Assuming that the resistances are positive, derive an algebraic criterion on the determinant for ensuring that the solution is unique.



e) Create the program `Electric_Current` for finding the currents $\{I_i\}_{i=1}^3$. Choose the following voltages and resistances:

Case	V	R_1	R_2	R_3	R_4	R_5
1	5.6	3.7	6.3	5.2	2.4	0.6
2	6.4	4.1	1.7	7.3	2.8	2.0
3	1.7	4.3	5.1	5.8	2.2	0.4
4	5.6	4.3	1.4	2.7	2.1	2.4
5	9.2	5.2	3.8	2.3	0.8	0.2
6	13.2	8.6	9.1	6.3	3.8	3.1

List up all the answers you get for the various combinations.

This time, the program must open an XML-file with the voltage V and the resistances $\{R_i\}_{i=1}^5$. They will be used for defining the matrix and vector of the linear system.
If you get negative currents, do not worry. It means that the current goes in the opposite direction of what you had assumed.

Tips on XML-parsing and general coding

The purpose of XML is storing data in separate files such that they can be reused in an efficient way without modifying the source code of the main program. In this way, you make the source code more general and generic. The underlying process is nearly identical for MATLAB and PYTHON, but there are some small differences.

Python

For PYTHON, the main ingredient is the library `xml.etree.ElementTree`, which has been included with some other ones in the template `Header.py`. The name of the XML-file for input can be anything, but the ending must always be `.xml`. If you use *XMLFILE* as a variable for the file name, you write the syntax

```
tree = et.parse(XMLFILE)
root = tree.getroot()
```

Thus, you have created a tree with a root, and elements are accessed like in a linked list. All the XML-variables are strings by default. For example, if element number 3, 6 and 8 in the tree are respectively string, integer or float, you write

```
var3 = str(root[3].text)
var6 = int(root[6].text)
var8 = float(root[8].text)
```

If a variable is a function expression depending on x, say XML-variable 0, you create a function handle by writing

```
func = lambda x : eval(root[0].text)
```

For multivariate functions, the procedure is slightly different. Let us consider the function $F(x, y) = [x^2 + y^2 - 8, x^2 - y^4]^T$. We create the function handle by writing

```
F = lambda x,y : np.array([eval('x**2+y**2-8'),eval('x**2-y**4')])
```

The code for finding the ∞-norm of a vector $\mathbf{x}$ is

```
np.linalg.norm(x,ord=inf)
```

In order to solve the linear system $\mathbf{Ax} = \mathbf{b}$, the code is

```
x = np.linalg.solve(A,b)
```

Matlab

For MATLAB, you need the syntax

```
tree = xmlread(XMLFILE);
theStruct = parseChildNodes(tree);
```

The first one, `xmlread`, is a default MATLAB-function, and the other one is included with the other XML facilities at the home page.

```
var1 = theStruct.Children(2).Children.Data;
var2 = str2num(theStruct.Children(4).Children.Data);
var3 = str2double(theStruct.Children(6).Children.Data);
f = str2func(strcat('@(x)',theStruct.Children(8).Children.Data));
```

Creating multivariate functions is more straightforward than PYTHON:

```
F = @(x,y) [x^2+y^2-8; x^2-y^4];
```

The code for finding the ∞-norm of a vector $\mathbf{x}$ is

```
norm(x,Inf)
```