University of Nottingham
UK | CHINA | MALAYSIA

MATH2019 Introduction to Scientific Computation

— Coursework 1 (10%) —

Submission deadline: ***3pm, Tuesday, 1 Nov 2022***

Note: This is currently **version 3-3** of the PDF document. (24th October, 2022)

No more new questions will be added anymore.

This coursework contributes **10%** towards the overall grade for the module.

Rules:

- Each student is to submit their own coursework.
- You are allowed to work together and discuss in small groups (2 to 3 people), but *you must write your own coursework and program all code by yourself*.
- Please be informed of the UoN Academic Misconduct Policy (incl. plagiarism, false authorship, and collusion).

Coursework Aim and Coding Environment:

- In this coursework you will develop Python code related to algorithms that solve nonlinear equations, and you will study some of the algorithms' behaviour.
- As discussed in the lecture, you should write (and submit) plain Python code (.py), and you are strongly encouraged to use the **Spyder IDE** (integrated development environment). Hence you should *not* write IPython Notebooks (.ipynb), and you should *not* use Jupyter.

How and Where to run Spyder:

- Spyder comes as part of the Anaconda package (recall that Jupyter is also part of Anaconda). Here are three options on how to run Spyder:

(1) You can choose to install Anaconda on you personal device (if not done already).

(2) You can open Anaconda on any University of Nottingham computer.

(3) You can open a UoN Virtual Desktop on your own personal device, which is virtually the same as logging onto a University of Nottingham computer, but through the virtual desktop. The simply open Anaconda. Here is further info on the UoN Virtual Desktop.

- The A18 Computer Room in the Mathematical Sciences building has a number of computers available, as well as desks with dual monitors that you can plug into your own laptop.

Time-tabled Support Sessions (Mondays 12-1pm, Fridays 11-12noon, ESLC C13 Computer Room):

- You can work on the coursework whenever you prefer.
- We especially encourage you to work on it during the (optional) time-tabled drop-in sessions.

Piazza (https://piazza.com/class/l7z5sbys7m74fq):

- You are allowed and certainly encouraged(!) to also ask questions using Piazza to obtain clarification of the coursework questions, as well as general Python queries.
- However, when using Piazza, please ensure that you are not revealing any answers to others. Also, please ensure that you are not directly revealing any code that you wrote. Doing so is considered Academic Misconduct.
- When in doubt, please simply attend a drop-in session to meet with a PGR Teaching Assistant or the Lecturer.

Helpful resources:

• Python 3 Online Documentation – Spyder IDE (integrated development editor) – NumPy User Guide – Matplotlib Quick Start Guide – Moodle: Core Programming 2021-2022 – Moodle: Core Programming 2020-2021

• You are expected to have basic familiarity with Python, in particular: logic and loops, functions, NumPy and matplotlib.pyplot. Note that it will always be assumed that the package numpy is imported as np, and matplotlib.pyplot as plt.

Some Further Advice:

• Write your code as neatly and read-able as possible so that it is easy to follow. Add some comments to your code that indicate what a piece of code does. Frequently run your code to check for (and immediately resolve) mistakes and bugs.

• Coursework Questions with a "⋆" are more tedious, but can be safely skipped, as they don't affect follow-up questions.

Submission Procedure:

• Submission will open after 21 October 2022.

• To submit, simply upload the requested .py-files on Moodle. (Your submission will be checked for plagiarism using *turnitin*.)

• Your work will be marked (mostly) automatically: This is done by running your functions and comparing their output against the true results.

Getting Started:

• Download the contents of the "Coursework 1 Pack" folder from Moodle into a single folder.
(More files may be added in later weeks.)

• Open Spyder (see instructions above), and if you wish you can watch a basic intro video from the Spyder website (click "Watch video").

• You can also watch the recording of Lecture 1, second hour, where a brief demonstration was given.

▶ Bisection method

Let $f(x) = x^3 + x^2 - 2x - 2$. Note that this is the same function as in Lecture 1. Consider the bisection method for finding the root of f in the interval $[1, 2]$.

1 (This is a warm up exercise. No marks will be awarded for it. Its solution will appear on Moodle. Further clarification of what you need to do to obtain marks will become clear once more questions are released.)

Open the py-file `just_trying_out.py` from the Coursework 1 Pack (from Moodle), and add your code to it:

• First simply plot the function to get an idea of what it looks like.

Hint: Use matplotlib.pyplot; see, e.g., the Matplotlib Quick Start Guide.

• Then try implementing the bisection algorithm as explained in Lecture 1.

Hint: The Lecture 1 SlidesAndNotes PDF contains a pseudo-code algorithm for bisection, as well as a simple Python code. This was also demonstrated in Lecture 1.

• Verify that the first few approximations are correct (e.g., compare with hand

calculations). What value do your approximations seem to converge to? Does that seem correct to you?

2 (This is a warm up exercise. No marks will be awarded for it. Its solution will appear on Moodle. Further clarification of what you need to do to obtain marks will become clear once more questions are released.)

Next, you will repeat what you have done above, but now by using two files: `rootfinders.py` and `main.py`. The file `rootfinders.py` will be a *module* (recall that a module can be imported and contains Python definitions and statements). The idea is that `rootfinders.py` contains your function definitions (algorithms for finding roots), while `main.py` is a testing *script* that is used to test your module by importing it and calling its functions.

To get started, download both files into the same folder, and open the file `rootfinders.py`. Note that this file contains already an unfinished function with the following signature:

```
def bisection(f,a,b,Nmax)
```

This function is to return a `numpy.ndarray` called `p_array`, shape `(Nmax, )`, which is a 1-D array of the approximations p_n ($n = 1, 2, \ldots$) computed by the bisection method. The input `f` can be a lambda function or function defined using `def`, `a` and `b` define the initial interval $[a, b]$ of interest, and `Nmax` is the maximum number of iterations.

- Complete this function so that it implements the bisection algorithm, and provides the output as required above.
- Test your function by running the `main.py` file, which contains the following:

```
#%% Question 2

import numpy as np
import matplotlib.pyplot as plt
import rootfinders as rf

# Initialise
f = lambda x: x**3 + x**2 - 2*x - 2
a = 1
b = 2
Nmax = 5

# Run bisection
p_array = rf.bisection(f,a,b,Nmax)

# Print output
print(p_array)
```

(turn page)

(The below has been added on 14 Oct 2022.)

▶ **Fixed-point iteration I**

We now consider the fixed-point problem $g(p) = p$. To numerically find a fixed point, we consider the fixed-point iteration method (p_0 given):

$$p_n = g(p_{n-1}), \qquad n = 1, 2, \ldots.$$

3 • Add in your file `rootfinders.py` the following function definition (to be completed):

← Correction: Added a colon ":" (**16 Oct, 5pm**)

```
def fixedpoint_iteration(g,p0,Nmax):
    """
    DESCRIPTION

    Parameters
    ----------
    g : TYPE
        DESCRIPTION.
    p0 : TYPE
        DESCRIPTION.
    Nmax : TYPE
        DESCRIPTION.

    Returns
    -------
    p_array : TYPE
        DESCRIPTION.

    """



    return p_array
```

This function should return a `numpy.ndarray` called `p_array`, with shape (`Nmax`,), which is a 1-D array of the approximations p_n ($n = 1, 2, \ldots$) computed by the fixed-point iteration method. The input `g` can be a lambda function or function defined using `def`, `p0` is a real number representing the initial approximation to start the fixed-point iteration, and `Nmax` is the maximum number of iterations.

• Complete this function so that it implements the fixed-point iteration algorithm, and provides the output as required above.

• Also complete the function documentation at the top of your function definition, the so-called `doc_string`. Simply replace `DESCRIPTION` and `TYPE` by appropriate ones. This documentation should become visible, whenever one types: `help(rf.fixedpoint_iteration)` or `rf.fixedpoint_iteration?`.

Hint: See the docstring of the bisection function, which was already given. See also the useful online pandas guide to writing a docstring.

• Test your function by running the `main.py` file, which has been updated (see Moodle) to additionally contain the below. Recall that this is a testing script, hence you're welcome to modify `main.py` in any way you like.

```
#%% Question 3

import numpy as np
import matplotlib.pyplot as plt
import rootfinders as rf

# Print help description
help(rf.fixedpoint_iteration)

# Initialise
g = lambda x: 1 - 1/2 * x**2
p0 = 1
Nmax = 5

# Run method
p_array = rf.fixedpoint_iteration(g,p0,Nmax)

# Print output
print(p_array)
```

For the above input, the output of `print(p_array)` should be:

```
[0.5         0.875       0.6171875   0.80953979  0.67232266]
```

► **Assessment**

As you may now realise, when submitting your coursework, you will only be asked to upload your `rootfinders.py` file.

Marks can be obtained for your fixedpoint_iteration definition for generating the required output, for certain set(s) of inputs for $\{g, p0, Nmax\}$. The correctness of the following will be checked: **[10 / 40]**

- The `type` of output `p_array`
- The `np.shape` (number of columns and rows) of output `p_array`
- The values of output `p_array`
- As well as for the output of "help(fixedpoint_iteration)".

Note that in marking your work, different sets of input(s) may be used.

(turn page)

► **Fixed-point iteration II**

We now consider the fixed-point iteration method with a stopping criterion. In particular, the iterations should prematurely stop when

$$\left|p_k - g(p_k)\right| \leq \texttt{TOL}\,.$$

4 • Add in your file `rootfinders.py` the following function definition (to be completed):

← Correction: Added a colon ":" (**16 Oct, 5pm**)

```
def fixedpoint_iteration_stop(g,p0,Nmax,TOL):
    """
    DESCRIPTION

    Parameters
    ----------
    g : TYPE
        DESCRIPTION.
    p0 : TYPE
        DESCRIPTION.
    Nmax : TYPE
        DESCRIPTION.
    TOL : TYPE
        DESCRIPTION.

    Returns
    -------
    p_array : TYPE
        DESCRIPTION.

    """



    return p_array
```

This function should return a `numpy.ndarray` called `p_array`, with shape $(k,\ )$, which is a 1-D array of the approximations p_n $(n = 1, 2, \ldots, k)$ computed by the fixed-point iteration method with the above stopping criterion. The value of k is the smallest integer such that the stopping criterion holds, unless `Nmax` iteration have been performed, in which case $k =$ `Nmax`.

• Complete this function so that it implements the output as required above.

Hint: To calculate the absolute value, one can use the command `abs`, `np.abs` or `np.absolute`; see numpy.org/doc.

Hint: One can use the command `return` at any time in your code to exit the function prematurely.

• Also complete the function documentation at the top of your function definition, the so-called `doc_string`. Simply replace `DESCRIPTION` and `TYPE` by appropriate ones. This documentation should become

visible, whenever one types: `help(rf.fixedpoint_iteration_stop)` or `rf.fixedpoint_iteration_stop?`.
Hint: See the docstring of the bisection function, which was already given. See also the useful online pandas guide to writing a docstring.

• Test your function by adding the below to your `main.py` file. Recall that this is a testing script, hence you're welcome to modify `main.py` in any way you like.

```
#%% Question 4

import numpy as np
import matplotlib.pyplot as plt
import rootfinders as rf

# Print help description
help(rf.fixedpoint_iteration_stop)

# Initialise
g = lambda x: 1 - 1/2 * x**2
p0 = 1
Nmax = 15
TOL = 10**(-2)

# Run method
p_array = rf.fixedpoint_iteration_stop(g,p0,Nmax,TOL)

# Print output
print(p_array)
```

← Replacement:
1e-2 by 10**(-2)
21 Oct, 11am

For the above input, the iterations should stop prematurely, and the shape (`np.shape`) of `p_array` ~~should be (12,),~~ should be (13,), instead of (15,).
However, when `Nmax = 5` and `TOL = 10**(-2)`, or when `Nmax = 15` and `TOL = 10**(-3)`, the iterations should not stop prematurely (but simply stop when `Nmax` iterations have been performed).

Sentence corrected on **16 Oct, 5pm** ←

← Replacement:
1e-2 by 10**(-2)
1e-3 by 10**(-3)
21 Oct, 11am

▶ Assessment

When submitting your coursework, you will only be asked to upload your `rootfinders.py` file.

Marks can be obtained for your fixedpoint_iteration_stop definition for generating the required output, for certain set(s) of inputs for {`g,p0,Nmax,TOL`}. The correctness of the following will be checked: **[10 / 40]**

- The `type` of output `p_array`
- The `np.shape` (number of columns and rows) of output `p_array`
- The values of output `p_array`
- As well as for the output of "help(fixedpoint_iteration_stop)".

Note that in marking your work, different sets of input(s) may be used.

(turn page)

(The below has been added on 21 Oct 2022.)

▶ Newton's method

We now consider the rootfinding problem $f(p) = 0$. To numerically find a root, we consider Newton's method (p_0 given):

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \qquad n = 1, 2, \ldots.$$

To stop iterations prematurely, we consider the stopping criterion:

$$\left|p_k - p_{k-1}\right| \leq \texttt{TOL}\,.$$

5 • Add in your file `rootfinders.py` the following function definition (to be completed):

← Correction: Added a colon ":" (**21 Oct, 5pm**)

```
def newton_stop(f,dfdx,p0,Nmax,TOL):
    """
    DESCRIPTION

    Parameters
    ----------
    f : TYPE
        DESCRIPTION.
    dfdx : TYPE
        DESCRIPTION.
    p0 : TYPE
        DESCRIPTION.
    Nmax : TYPE
        DESCRIPTION.
    TOL : TYPE
        DESCRIPTION.

    Returns
    -------
    p_array : TYPE
        DESCRIPTION.

    """



    return p_array
```

This function should return a `numpy.ndarray` called `p_array`, with shape $(k,)$, which is a 1-D array of the approximations p_n $(n = 1, 2, \ldots, k)$ computed by Newton's method with the above stopping criterion. The value of k is the smallest integer such that the stopping criterion holds, unless `Nmax` iterations have been performed, in which case $k =$ `Nmax`. The inputs `f` and `dfdx` can be lambda functions or functions defined using `def`, and correspond to the function f and its

derivative f', respectively. The input `p0` is a real number representing the initial approximation and `Nmax` is the maximum number of iterations.

• Complete this function so that it provides the output as required above.

Hint: To calculate the absolute value, one can use the command `abs`, `np.abs` or `np.absolute`; see numpy.org/doc.

Hint: One can use the command `return` at any time in your code to exit the function prematurely.

• Also complete the function documentation at the top of your function definition, the so-called `doc_string`. Simply replace `DESCRIPTION` and `TYPE` by appropriate ones. This documentation should become visible, whenever one types: `help(rf.newton_stop)` or `rf.newton_stop?`.

Hint: See the docstring of the bisection function, which was already given. See also the useful online pandas guide to writing a docstring.

• Test your function by adding the below to your `main.py` file. Recall that this is a testing script, hence you're welcome to modify `main.py` in any way you like.

```
#%% Question 5

import numpy as np
import matplotlib.pyplot as plt
import rootfinders as rf

# Print help description
help(rf.newton_stop)

# Initialise
f    = lambda x:  np.cos(x) - x
dfdx = lambda x: -np.sin(x) - 1
p0 = 1
Nmax = 5
TOL = 10**(-3)

# Run method
p_array = rf.newton_stop(f,dfdx,p0,Nmax,TOL)

# Print output
print(p_array)
```

For the above input, the iterations should stop prematurely, and the shape (`np.shape`) of `p_array` should be (3,), instead of (5,). The corresponding output of `print(p_array)` should be:

```
[0.75036387 0.73911289 0.73908513]
```

However, when `Nmax = 5` and `TOL = 10**(-16)`, the iterations should not stop prematurely (but simply stop when `Nmax` iterations have been performed).

► **Assessment**

When submitting your coursework, you will only be asked to upload your `rootfinders.py` file.

Marks can be obtained for your newton_stop definition for generating the required output, for certain set(s) of inputs for {`f,dfdx,p0,Nmax,TOL`}. The correctness of the following will be checked: **[10 / 40]**

- The `type` of output `p_array`
- The `np.shape` (number of columns and rows) of output `p_array`
- The values of output `p_array`
- As well as for the output of "help(newton_stop)".

Note that in marking your work, different sets of input(s) may be used.

(turn page)

(The below has been added on 23 Oct 2022.)

► **Convergence behaviour: Plotting the errors**

Assuming one knows the exact solution p, one can plot the (absolute value of the) error, $e_n = |p - p_n|$, at each iteration $n = 1, 2, 3, \ldots$.

6 • Add in your file `rootfinders.py` the following import statement and function definition for `plot_convergence` (to be completed):

```
import matplotlib.pyplot as plt

def plot_convergence(p,f,dfdx,g,p0,Nmax):

    # Fixed-point iteration
    p_array = fixedpoint_iteration(g,p0,Nmax)
    e_array = np.abs(p - p_array)
    n_array = 1 + np.arange(np.shape(p_array)[0])

    # Newton method


    # Preparing figure, using Object-Oriented (OO) style; see:
    # https://matplotlib.org/stable/tutorials/introductory/quick_start.html
    fig, ax = plt.subplots()
    ax.set_yscale("log")
    ax.set_xlabel("n")
    ax.set_ylabel("|p-p_n|")
    ax.set_title("Convergence behaviour")
    ax.grid(True)

    # Plot
    ax.plot(n_array , e_array , "o", label="FP iteration",linestyle="--")


    # Add legend
    ax.legend();

    return fig, ax
```

← Replacement: '...' by "..." 24 Oct, 5:15pm

As you can see, this function currently creates a figure with a plot of the errors for the fixed-point iteration method, using the function `fixedpoint_iteration` that you completed in Question 3.

• Complete this function so that it also plots (in the same figure using the same axes) the errors for *Newton's method*. Compute the Newton-method approximations using your `newton_stop`, based on the input for `(f,dfdx,p0,Nmax)` and setting `TOL` to a very small value, e.g., `TOL = 10**(-16)`.

– Ensure the two graphs can be distinguished from one another (i.e., have different markers, linestyle, etc); see https://matplotlib.org/stable/tutorials/introductory/quick_start.html

• Test your function by adding the below to your `main.py` file. Recall that this is a testing script, hence you're welcome to modify `main.py` in any way you like:

```python
#%% Question 6

import numpy as np
import matplotlib.pyplot as plt
import rootfinders as rf

# Initialise
f    = lambda x: x - np.cos(x)
dfdx = lambda x: 1 + np.sin(x)
g    = lambda x: np.cos(x)
p0 = 1
Nmax = 20
p = np.float64(0.739085133215160641655312070 47)

# Plot convergence
fig, ax = rf.plot_convergence(p,f,dfdx,g,p0,Nmax)
```

For the above input, the figure should look like the one on Slide 11 from the Lecture 3 Slides (but without the results for the Secant method).

► **Assessment**

When submitting your coursework, you will only be asked to upload your `rootfinders.py` file.

Marks can be obtained for your plot_convergence definition for generating the required output, for certain set(s) of inputs for {`p,f,dfdx,g,p0,Nmax`}. The correctness of the following will be checked: **[4 / 40]**

- The correctness of the graph plotted for each method
- The ability to identify which graph corresponds to which method

(turn page)

► **Optimizing the fixed-point iteration method** (this is a challenging question!)
In this last part, we return to the fixed-point iteration method with stopping criterion (Question 4), and aim to optimise the parameter c within the following function $g(\cdot)$:

$$g(x) = x - c\, f(x)\,.$$

(Recall from Lecture 2 that this $g(\cdot)$ is useful when trying to use fixed-point iteration to solve the rootfinding problem $f(p) = 0$.)

7⋆ • Add in your file `rootfinders.py` the following function definition for `optimize_FPmethod` (to be completed):

```
def optimize_FPmethod(f,c_array,p0,TOL):

    return c_opt, n_opt
```

This function should return the real number `c_opt` and corresponding integer `n_opt`, which correspond to the optimal value of c in the following sense: The optimal $c =$ `c_opt` is an element within the given array `c_array`, for which the fixed-point iteration method is the *fastest* to converge to a given tolerance value `TOL`, using starting value `p0`. In other words, ~~the absolute value of the error~~ $|p_k - g(p_k)|$ reaches `TOL` in the *least* amount of iterations `n_opt`.

← Sentence corrected on **24 Oct, 5:15pm**

Hint: For simplicity, you may assume that the optimal number of iterations `n_opt` is (much) smaller than 100. Hence, call your `fixedpoint_iteration_stop` with `Nmax` set to 100.

Hint 2: You may find useful the following `numpy` functions: `numpy.argmin`, `numpy.amin`, and `min`.

• Test your function by adding the below to your `main.py` file. Recall that this is a testing script, hence you're welcome to modify `main.py` in any way you like:

```
#%% Question 7

import numpy as np
import matplotlib.pyplot as plt
import rootfinders as rf


# Initialise
f = lambda x: x**3 + x**2 - 2*x - 2
p0 = 1
TOL = 10**(-6)
c_array = np.linspace(0.01,0.4,40)

# Find optimal c
c_opt, n_opt = rf.optimize_FPmethod(f,c_array,p0,TOL)

# Print output
print(n_opt)
print(c_opt)
```

Note that the above f has root $p = \sqrt{2}$.

► **Assessment**

When submitting your coursework, you will only be asked to upload your `rootfinders.py` file.

Marks can be obtained for your optimize_FPmethod definition for generating the required output, for certain set(s) of inputs for {`f`,`c_array`,`p0`,`TOL`}. The correctness of the following will be checked: **[6 / 40]**

- The correctness of `n_opt` and `c_opt`.
- Also, it is required that your function completes its run within a few minutes.

► Finished!