Palmer Edholm
Dr. Joe Koebbe
MATH 4610
October 8, 2021

Tasksheet 4

Task 1 The following is my code for my abs_error() function which finds the absolute error in approximating one value with another.

```
1  def abs_error(x, y):
2      # Find absolute error in approximating x with y
3      output = abs(x - y)
4      return output
```

The following is my code for my rel_error() function which finds the relative error in approximating one value with another.

```
1  def rel_error(x, y):
2      # Find relative error in approximating x with y
3      output = (abs(y - x) / abs(x))
4      return output
```

All my code will be archived in a Python package at the end since I am not aware of a way to add files to an already built Python package.

Task 2 The following is my code to graph functions.

```
1  from matplotlib import pyplot as plt
2  import numpy as np
3
4
5  def graphics(expression, xlow=-10.0, xhigh=10.0, ylow=-10.0,
6               yhigh=10.0):
7      # Set size of graph
8      plt.xlim([xlow, xhigh])
9      plt.ylim([ylow, yhigh])
10     # Create values for x-axis
11     xvals = np.linspace(xlow, xhigh, 10000)
12     # Hard code labels for axes
13     plt.xlabel('x')
14     plt.ylabel('y')
15     # Loop through all strings in array
16     for i in range(len(expression)):
17         # Create anonymous function for ith function in expression
18         func = lambda x: eval(expression[i])
19         # Compute range values of function
20         yvals = [func(y) for y in xvals]
21         # Plot the function
22         plt.plot(xvals, yvals, label=f'y={expression[i]}')
```

```
23        plt.legend(loc='best')
24        plt.show()
```

Note that functions are entered as strings in an array when calling the function and they must be passed in as python language (e.g., Numpy methods like np.log() must be used to graph more complex functions). The default grid size is $[-10, 10] \times [-10, 10]$ but can be changed when calling graphics. This code will be archived at the end.

Task 3 The following is my base code for the fixed point iteration method of root-finding.

```
1  import numpy as np
2
3
4  def fxd_pt_iter(x0, f, tol, max_iter):
5      def g(x):
6          return x-f(x)
7      iters = 0
8      error = 10 * max_iter
9      while error > tol and iters < max_iter:
10         x1 = g(x0)
11         error = abs(x1-x0)
12         x0 = x1
13         iters += 1
14     return x1, iters
```

As a note, the objective function has to be passed in as an anonymous function via the use of the lambda function. To make this function work for specific function, g will have to be modified. This code will be archived at the end.

Task 4 Looking at the graph of $f(x) = xe^{3x^2} - 7x$, we can see that there is a root near $x = 1$. If we call fxd_pt_iter, and pass in $x0 = 1$, the routine ends early because the routine diverges and does not find a root. However, if we modify our function g as suggested (i.e., $g(x) = x - \epsilon f(x)$) and set ϵ equal to some small number, say 10^{-2}, the routine converges on the value 0.8055789837178884 with a tolerance of 0.0001.

Task 5 The following is my code for the bisection method.

```
1  import numpy as np
2
3
4  def bisection(a, b, f, tol):
5      fa = f(a)
6      fb = f(b)
7      if fa*fb > 0:
8          raise Exception('Root not in specified interval')
9      error = abs(b-a)
10     k = int((np.log(error / tol) / np.log(2))+1)
11     for i in range(k):
12         c = 0.5 * (a+b)
13         fc = f(c)
```

```
14          if fa * fc < 0:
15              b = c
16          else:
17              a = c
18      return c
```
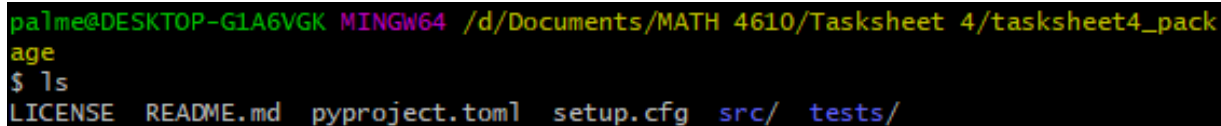
In order for this method to work, a closed interval $[a, b]$ that contains exactly one root has to be passed in. If there is no root in the interval passed in, an exception will be raised. Given a specified level of tolerance, we can compute the exact number of iterations needed to obtain that level of accuracy:

$$\text{iterations} = \frac{\log\left(\frac{\text{error}}{\text{tolerance}}\right)}{\log(2)}.$$

Passing in the closed and bounded interval $[0.5, 1.5]$, the function defined in task 4, and the tolerance level 0.0001, we get the value 0.80535888671875 which is very close to what we got in task 4.

Now that I have all the code for this tasksheet, I will go ahead and create my python package. First, I'll create the necessary file structure as shown in figure 1.

Figure 1: File structure

My source code is all in the src directory. Once I have this all set up, I can run the command

```
py -m build
```

and my python package is built.

Task 6 Root finding methods are used in economics occasionally. In [1], the author talked about how when deriving a Marshallian demand function, it's common practice to invert the demand function to find the market clearing price for some good. However, depending on the market and the good in question, it is often impossible to analytically derive the market clearing price so some root finding method must be employed. The author states how Newton's method is one oft-used by economists. However, the author notes that there are drawbacks to Newton's method if the first derivative is not easily calculated.

References

[1] http://fmwww.bc.edu/ec-c/F2003/316/EC316a.F2003.sect1.pdf