ENGSCI 331 - Computational Techniques 2

Lab 1: NLE

Department of Engineering Science & Biomedical Engineering

Introduction

The lab consists of two parts.

Task 1 - Iterative Algorithms

Background and Aim: Implement the algorithms discussed in lectures to iteratively find one root of the nonlinear function $f_1(x) = 2x^2 - 8x + 4$, starting from either an initial interval $\left(x^{(0)}, x^{(1)}\right)$ or an initial guess $x^{(0)}$.

Methodology: In the extracted zip archive you find a file called `algorithms.py` containing the incomplete function:

– `def bisection(f, xl, xr, max_iter, tol)`

and headers for functions:

– `def secant(f, x0, x1, max_iter, tol)`

– `def regula_falsi(f, xl, xr, max_iter, tol)`

– `def newton(f, g, x0, max_iter, tol)`

– `def combined(f, g, xl, xr, max_iter, tol)`

The initial comments for each function have already been written, specifying the function input/output required for tasks 1 and 2, with details provided about each variable.

Pseudo-code is not provided, so it may be useful to write this yourself, based on the course notes.

Complete each function, such that it:

– produces a sequence of root estimates using the appropriate method and stores them in an array alongside the input initial root estimates;

- applies the following simple root finding test each iteration: $|f(x^{(k)})| < \Delta$ where Δ is some numerical tolerance value (`tol`);
- outputs the total number of iterations undertaken;
- resets the total number of function evaluations to 0; for methods using the gradient, gradient evaluations should be also reset to 0; and
- returns a termination flag, using the `ExitFlag enum`.

Note: each function you write should be ***well commented***, and you should seek to ***minimise the number of function evaluations***.

In `Newton` and `Combined`, you should use the algebraic derivative $g(x) = f'(x)$.

For `Combined`, you should combine the bisection method with Newton's method:

- Use the midpoint of the provided bracket, $\left(x^{(0)}, x^{(1)}\right)$ as the starting estimate for Newton's method i.e. $x^{(2)} = x^{(0)} + \dfrac{x^{(1)} - x^{(0)}}{2}$
- Attempt to use Newton's method to find the next root estimate, $x^{(k+1)}$.
- If $x^{(k+1)}$ from Newton's method lies outside of the current bracket, $\left(x^{(L)}, x^{(R)}\right)$, use the bisection method for this iteration to get a better estimate for $x^{(k+1)}$.
- Each iteration, use the latest root estimate $x^{(k+1)}$ to update the bracket, $\left(x^{(L)}, x^{(R)}\right)$.
- This algorithm should have both *guaranteed* and *fast* convergence.

Verification: The script `task1.py` calls `bisection`, `secant`, `regula_falsi`, `newton` and `combined` one-by-one to find a single root of $f(x)$ and constructs a table outlining the root estimates, and performance of the methods. You should not need to modify this script, except perhaps to comment out calls to incomplete functions.

1. Submit the output from `task1.py`, and comment on the performance of each of the various methods when applied to this function (both in terms of iterations and function / derivative evaluations).

Task 2 - Algorithm Comparison

Background and Aim: In Task 1, your algorithms were tested on a simple polynomial, $f(x) = 2x^2 - 8x + 4$, which has two real roots at $x = 2 \pm \sqrt{2}$. The aim of this task is to compare the behaviour of your algorithms for the following three nonlinear functions:

$$\begin{aligned} f_2(x) &= x^2 - 1, \\ f_3(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}}, \\ f_4(x) &= \cos(x) + \sin(x^2) - \frac{1}{2}. \end{aligned}$$

Methodology: Your Task 1 functions should each output a sequence of estimates for the root in a one-dimensional array:

$$\vec{x} = \left\{x^{(0)}, x^{(1)}, \ldots, x^{(k)}\right\}$$

Note that the script `task2.py` includes starting estimates of the roots, which you should not modify, as they have been chosen so as to exhibit specific behaviour in each method. It also produces a plot of $f(x)$ vs. x for your reference – you can set variable `disp_func = false` when you no longer wish to produce/view this plot.

Once you have run `task2.py`, answer the following questions:

2. Explain what is being depicted in the various graphs, and comment of the performance of the algorithms on the different functions. For example, ensure that you discuss the behaviour of Newton's method for each of the three functions. If Newton's method fails to find a root for any of the three functions, explain why this is the case and how this issue is fixed in your implementation of `Combined`.

   To help you structure your answer, discuss each function in a separately:

   (a) Comment on the methods' performance on function $f_2(x)$.
   (b) Comment on the methods' performance on function $f_3(x)$.
   (c) Comment on the methods' performance on function $f_4(x)$.

3. You have little control over which root your algorithms converge to, other than by modifying the initial interval/estimate. Briefly outline a strategy for systematically finding multiple roots of a general continuous nonlinear function $f(x)$. State any assumptions that need to be made in order to find all such roots.

Separate instructions will be made available on how to submit your completed code.