EngSci331 Lab4 - NLE - Report

343733502 - Daniel Clark - dcla189

PART 1:

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).

```
Method       Estimate     Value       Iterations  Func Evals  Deriv Evals  Exit Flag
-------------------------------------------------------------------------------------------
Bisection    0.58578491   8.6296e-06          16          19            0  ExitFlag.Converged
Secant       0.58578856  -1.2015e-05           6           8            0  ExitFlag.Converged
RegulaFalsi  0.58579882  -7.0026e-05           7           9            0  ExitFlag.Converged
Newton       0.58578431   1.2015e-05           3           4            3  ExitFlag.Converged
Combined     0.58578431   1.2015e-05           3           6            3  ExitFlag.Converged


Process finished with exit code 0
```

- The bisection method is the least efficient root-finding method of the 5 shown above. It takes 16 iterations and executes a whopping 19 function evaluations for it to converge to within the set tolerance. It does however provide a certainty of convergence that some of the more efficient methods do not provide.
- Comparatively, the secant method converges much faster to within the set tolerance. However, while it does converge successfully for the given function $f(x)=2x^2-8x+4$, it is not guaranteed to converge for all other functions.
- The Regula Falsi method is similar to the secant search method, because it is a very similar algorithm. It only requires one more iteration than the secant method, but is guaranteed to find a root if there is one, while the secant method is not.
- Newton's method is the most efficient of all of the techniques, features only 7 total function/derivative evaluations. This is as expected
- The combined newton's and bisection method is in reality taking the exact same steps as the Newton method, so is almost as efficient, however the combined method also has to allow for use of the bisection method, so requires more set-up at the start of the function where it evaluates the function at two additional points, making it slightly less efficient in this case. Overall though this is the best method as it is guaranteed to find a root if there is one, while Newton's method is not.

2. Explain what is being depicted in the various graphs, and comment on 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 f2(x).

(b) Comment on the methods' performance on function f3(x).

(c) Comment on the methods' performance on function f4(x).

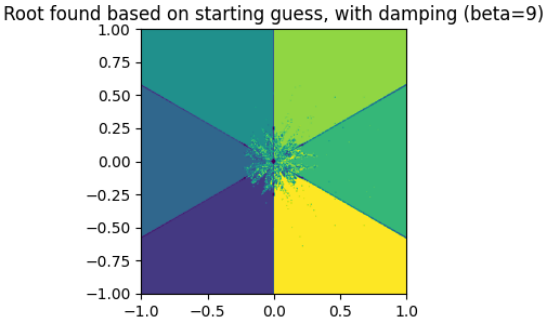
- On function $f_2(x)$, all methods except the secant method converge to the correct root at x=-1. The method that converges the fastest (in terms of iterations) is the combined method, followed by Newton's method. Both converge very quickly, within only 4-5 iterations, but the combined method is able to converge faster because it starts at -1.25, which is closer to the true root than where Newton starts at -3. The Regula Falsi and bisection search methods both converge much slower, as we would expect them to, however the secant method does not converge at all. This is because the secant method is not guaranteed to find a root on a given interval.
- On function $f_3(x)$, all methods except Newton's method successfully find the root at x=0. Combined is the fastest, followed by secant and Regula Falsi (who for this function have acted identically), and then bisection is the slowest. Newton's method meanwhile is not even close to finding the root, as it ends up predicting about -20000 after 6 iterations. This is because Newton's method is not guaranteed to find roots on a given interval, and can become caught in a situation where the x-value being predicted diverges rapidly from the true value.
- On function $f_4(x)$, it is only the three guaranteed methods that successfully converge to the true root on the given interval [-2, 1.5]. Both of the methods that can fail (Secant and Newton) diverge to other values which are correct roots of the equation (which has many valid roots), but are not on the given interval as required. The combined method quickly converges to the correct root at x=-1.605, and the Regula Falsi and bisection methods also reach this value, but significantly slower. Meanwhile, the secant method instead converges to x=1.605, and the Newtons method converges to a root at approximately x=-13.

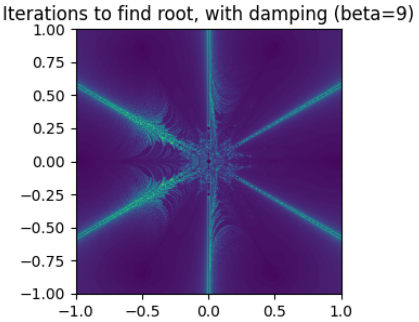
3. You have little control over which root your algorithms converge to, other than by modifying the initial interval/estimate. Briefly outline a strategy for systematically finding multiple roots of a general continuous nonlinear function f(x). State any assumptions that need to be made in order to find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

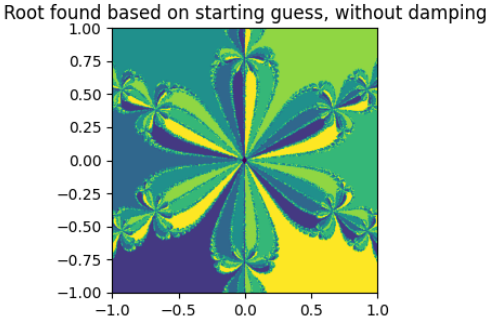
- We would have to run our chosen root finding algorithm (I would choose Combined) many times over multiple different intervals to ensure that we could find every root of our equation. Start by running it over a very wide domain e.g (-inf,inf). Once it has found one root (X) on this interval you would split the domain into two new regions, (-inf,X) and (X,inf) and run the root finding intervals on these new domains. This process is done recursively, and any domains without roots are discarded, until all intervals have been discarded, and we have a complete list of all roots for that particular nonlinear continuous function.

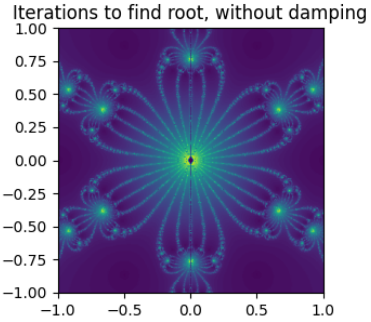
PART 2:

4. Create two pairs of plots using task3.py: one pair without damping, and one with damping, and describe what is shown in the plots and discuss the differences. (You can use the included equation, $x^6 = 1$, or change this to something else with at least two roots in the domain.)









5. Briefly outline two different reasons why newton multi could fail to converge to a solution; give an example of a starting point that fails to converge to a solution for f6.

- One way it can fail to converge is if it encounters a situation where the Jacobian is a singular matrix. When this is the case, we are unable to solve the equation Jx=f(x), and therefore will be unable to continue with this solution.
- The other way that Newton's method can fail is if Newton's method ever gets caught in a 'critical cycle', which is where the update for a certain x-value is 0, and therefore the x-value will never change, and will remain as that value for ever, never getting closer to finding a root.
- One situation that will encounter this problem is when we start with the initial point (0,0), as this will lead to a singular Jacobian matrix, and the root-finding method will fail.

6. Explain what the convergence plots are showing about the speed of convergence. You will need to change the code in task5.py (around lines 38-42) to show the log of the (absolute) function values.

- This plot shows that the convergence of this function takes around 5-6 iterations to fully convergence (for our chosen tolerance). For the first two starting points it takes 5 iterations, and takes 6 iterations from the third starting point. This shows that the root finding method is fairly consistent and that the starting point (as long it is in the

general area) is not too relevant on how quickly the method converges. It converges fast.

7. Now change the function to f7 (line 17) and comment on and explain the convergence for this in comparison to that of f6.

- F7 is very similar to f6, but does not include a -2 in the function, this has the effect of shifting the whole graph up. Because this has changed the way that our graph will converge, we end up converging slower for f7 than f6, as our roots are further from the initial starting positions. Instead of 5, 5, and 6 iterations it now takes 16, 16, and 15 iterations to correctly converge.

8. Provide the output of task7.py for the polynomial above.

- This is shown in the appendix, or can be found by running my task 7 code.

9. Comment on the number of iterations it takes to converge to the various roots of the polynomial, compared to the other methods you have implemented. (Of course this method is finding all the roots, not just one.)

- For finding the first four roots, this method takes 5, 3, 3, and 3 iterations respectively. This is fairly quick, but as each time we find a root we are deflating our equation, it continues to get faster over time (as the equation becomes less complex (lower order)). For the final two roots it takes only 1 iteration to converge to the correct root! This is extremely fast and efficient because by now the polynomial has been deflated to a second and then first order polynomial, so is easy to solve.

Appendix:

Finding root 1
The current polynomial is: (1.000e+00 + 0.000e+00i)x^6 + (2.000e+00 + 0.000e+00i)x^5 + (0.000e+00 + 0.000e+00i)x^4
- (4.000e+00 + 0.000e+00i)x^3 + (2.000e+00 + 0.000e+00i)x^2 + (0.000e+00 + 0.000e+00i)x^1
+ (4.000e+00 + 0.000e+00i)x^0

it	re(x)		im(x)	|p(x)|	|p'(x)|	|p''(x)|	|step|
0	0.4166	+	0.0102i	4.0883	0.046	2.212	1.486
1	1.9023	+	0.0267i	80.9417	244.707	626.747	1.591
2	0.5325	+	0.8357i	8.5546	20.360	75.694	0.503
3	0.9504	+	0.5560i	1.8669	20.636	88.653	0.088
4	1.0177	+	0.4997i	0.0075	22.111	95.488	0.000
5	1.0181	+	0.4996i	0.0000	22.136	95.575	

Root (1.0180771259727968+0.49964553247379473j) may be inaccurate

Finding root 2
The current polynomial is: (1.000e+00 + 0.000e+00i)x^5 + (3.018e+00 + 4.996e-01i)x^4 + (2.823e+00 + 2.017e+00i)x^3
- (2.134e+00 + 3.464e+00i)x^2 - (1.903e+00 + 2.460e+00i)x^1 - (3.166e+00 + 1.554e+00i)x^0

it	re(x)		im(x)	|p(x)|	|p'(x)|	|p''(x)|	|step|
0	0.8252	+	0.2986i	12.2927	25.376	63.364	0.750
1	1.0000	-	0.4304i	1.5077	20.191	72.654	0.072
2	1.0180	-	0.4997i	0.0008	22.149	77.644	0.000
3	1.0181	-	0.4996i	0.0000	22.151	77.649	

Root (1.0180771259479477-0.4996455325030335j) may be inaccurate

Finding root 3
The current polynomial is: (1.000e+00 + 0.000e+00i)x^4 + (4.036e+00 - 2.924e-11i)x^3 + (6.932e+00 - 1.354e-10i)x^2
+ (4.924e+00 - 2.704e-10i)x^1 + (3.110e+00 - 2.281e-10i)x^0

it	re(x)		im(x)	|p(x)|	|p'(x)|	|p''(x)|	|step|
0	0.3684	+	0.1937i	6.0655	12.197	24.805	0.849
1	-0.1648	+	0.8541i	0.8299	7.642	17.366	0.125
2	-0.2804	+	0.8080i	0.0024	5.794	14.130	0.000
3	-0.2800	+	0.8079i	0.0000	5.798	14.136	

Root (-0.2800245515347949+0.8078646877931953j) may be inaccurate

Finding root 4
The current polynomial is: (1.000e+00 + 0.000e+00i)x^3 + (3.756e+00 + 8.079e-01i)x^2 + (5.228e+00 + 2.808e+00i)x^1

+ (1.191e+00 + 3.437e+00i)x^0

it	re(x)		im(x)	|p(x)|	|p'(x)|	|p''(x)|	|step|
0	0.5660	+	0.1617i	8.4393	11.495	11.211	1.256
1	-0.2078	-	0.8280i	0.2882	4.104	7.106	0.075
2	-0.2801	-	0.8079i	0.0001	3.588	6.667	0.000
3	-0.2800	-	0.8079i	0.0000	3.588	6.668	

Finding root 5
The current polynomial is: (1.000e+00 + 0.000e+00i)x^2 + (3.476e+00 - 1.286e-11i)x^1 + (4.254e+00 - 1.928e-11i)x^0

it	re(x)		im(x)	|p(x)|	|p'(x)|	|p''(x)|	|step|
0	0.1243	+	0.4329i	4.7936	3.824	2.000	1.982
1	-1.7381	+	1.1106i	0.0000	2.221	2.000	

Finding root 6
The current polynomial is: (1.000e+00 + 0.000e+00i)x^1 + (1.738e+00 + 1.111e+00i)x^0

it	re(x)		im(x)	|p(x)|	|p'(x)|	|p''(x)|	|step|
0	0.5621	+	0.1743i	2.6347	1.000	0.000	2.635
1	-1.7381	-	1.1106i	0.0000	1.000	0.000	

The roots of the polynomial are: 1.018e+00 + 4.996e-01i
1.018e+00 - 4.996e-01i
-2.800e-01 + 8.079e-01i
-2.800e-01 - 8.079e-01i
-1.738e+00 + 1.111e+00i
-1.738e+00 - 1.111e+00i

Process finished with exit code 0