Homework 4

Panupong (Peach) Chaphuphuang

March 9, 2024

Github URL for homework 4:
`https://github.com/pach2648/APPM4600/tree/main/Homework/HW4`

Problem 1

Solution

a) For the code, click **Github Link for 1a**

```
Converge to solution: x =  0.500000000040521  y =  0.8660254038535676
Iterations:  33
0.029541730880737305
```

Figure 1: The result of 1a (The last line is the time that this algorithm runs)

b) The matrix can be any 2×2 matrix that is invertible. It can be the partial derivative of $f(x, y)$ & $g(x, y)$ like the example below.

$$\begin{bmatrix} \frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial y} \\ \frac{\partial g(x,y)}{\partial x} & \frac{\partial g(x,y)}{\partial y} \end{bmatrix}$$

c) For the code, click **Github Link for 1c**

$$\mathrm{J} = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial y} \\ \frac{\partial g(x,y)}{\partial x} & \frac{\partial g(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 6x & -2y \\ 3y^2 - 3x^2 & 6xy \end{bmatrix}$$

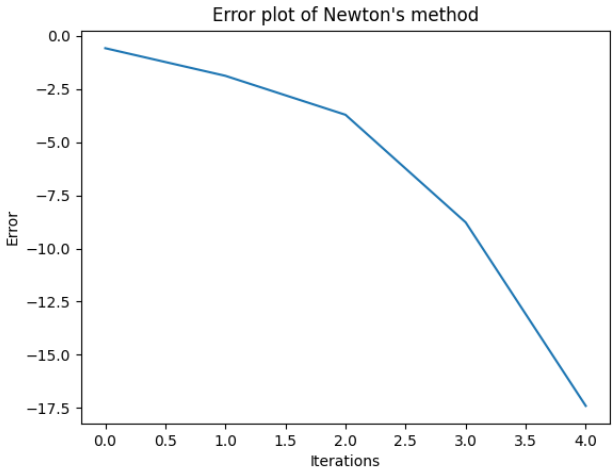

Figure 2: The result of 1c

```
[0.5       0.8660254]
0.003015518188476562 5
```

Figure 3: The result of 1c (The last line is the time that this algorithm runs)

d) The exact solution is $(\frac{1}{2}, \frac{\sqrt{3}}{2})$. We can prove this by plugging in the solution in both equations, and they should be true (have to be zero).

Another way to find the exact solution is to plot both graphs and find the intersection between those two equations. The figure is shown at figure 4

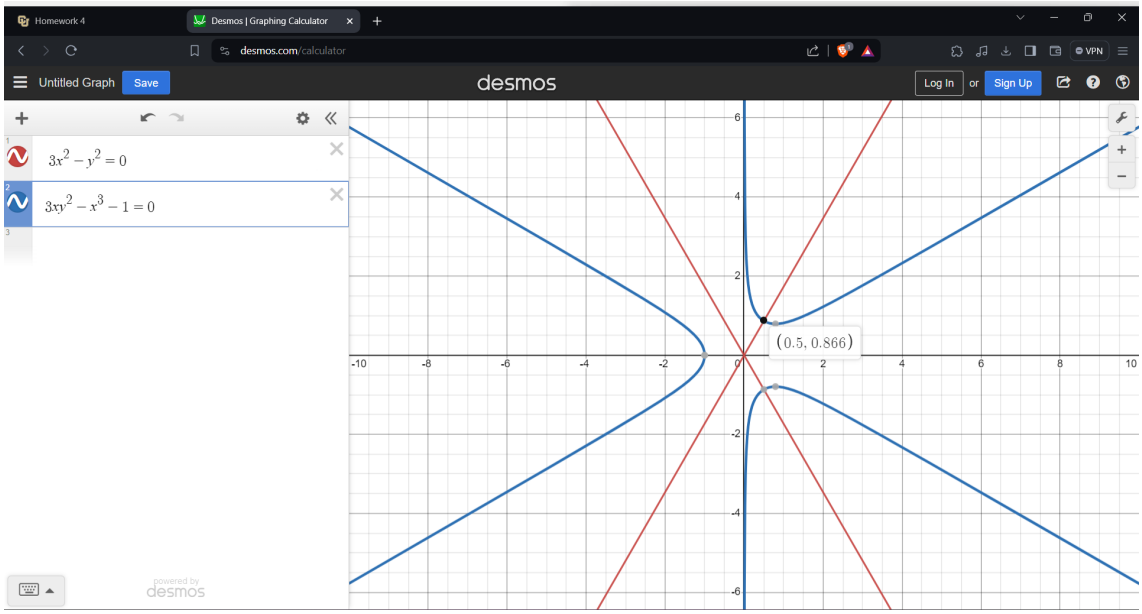

Figure 4: The exact solution from graph

By checking the result from part a and part c, they both converge to the exact solution.

Problem 2

Solution

For the code, click **Github Link for 2** For each subpart, you have to change the initial guesses to get the result from each part (Line 22).

$$\mathrm{J} = \begin{bmatrix} 2x & 2y \\ e^x & 1 \end{bmatrix}$$

a) $x = 1, y = 1$

From the code, the approximate root is $x = -1.81626407, y = 0.8373678$. The Newton's method and Broyden converge, but the Lazy Newton diverges.

```
|--n--|----xn----|---|f(xn)|---|
|--0--|1.4142136|3.3747676|
|--1--|4.3235744|15.0989046|
|--2--|4.3575159|15.0002843|
|--3--|2.6377492|2.9579991|
|--4--|2.0806894|0.3296943|
|--5--|2.0015721|0.0063052|
|--6--|2.0000006|0.0000025|
|--7--|2.0000000|0.0000000|
Newton method converged, n=8, |F(xn)|=0.0e+00
```

Figure 5: The result of Newton's

```
[-1.81626407  0.8373678 ]
|--n--|----xn----|---|f(xn)|---|
|--0--|1.4142136|3.3747676|
|--1--|4.3235744|15.0989046|
|--2--|1.4303279|2.1708520|
|--3--|1.9799298|0.9766453|
|--4--|2.6079734|2.9005702|
|--5--|2.2259748|1.0338327|
|--6--|2.1301901|0.5460298|
|--7--|1.9814143|0.0880741|
|--8--|1.9920107|0.0327319|
|--9--|2.0015004|0.0062216|
|--10--|2.0000190|0.0000813|
|--11--|2.0000009|0.0000038|
|--12--|2.0000000|0.0000001|
|--13--|2.0000000|0.0000000|
Broyden method converged, n=14, |F(xn)|=1.5e-14
```

Figure 6: The result of Broyden

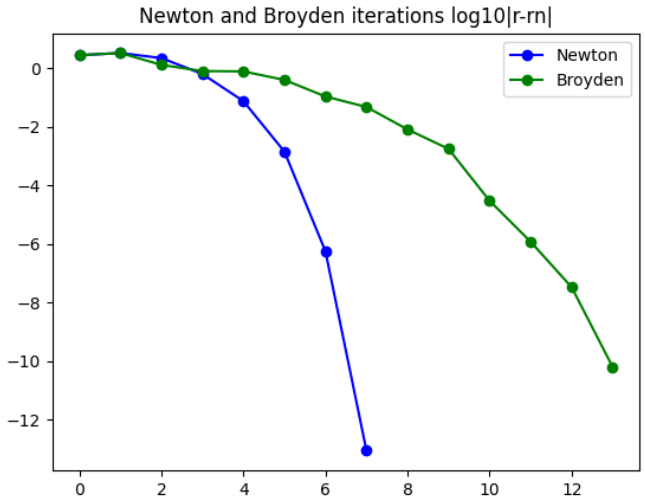

Figure 7: The Error Comparison of each method for 2a

b) $x = 1, y = -1$

From the code, the approximate root is $x = 1.00416874, y = -1.72963729$. The Newton's method and Broyden converge, but the Lazy Newton diverges.

```
|--n--|----xn----|---|f(xn)|---|
|--0--|1.4142136|2.1250715|
|--1--|2.2045293|0.8599867|
|--2--|2.0097602|0.0396404|
|--3--|2.0000238|0.0001024|
|--4--|2.0000000|0.0000000|
|--5--|2.0000000|0.0000000|
Newton method converged, n=6, |F(xn)|=0.0e+00
```

Figure 8: The result of Newton's

```
|--n--|----xn----|---|f(xn)|---|
|--0--|1.4142136|2.1250715|
|--1--|2.2045293|0.8599867|
|--2--|1.9597072|0.1596518|
|--3--|1.9979029|0.0084978|
|--4--|1.9999963|0.0000265|
|--5--|2.0000005|0.0000022|
|--6--|2.0000000|0.0000000|
|--7--|2.0000000|0.0000000|
Broyden method converged, n=8, |F(xn)|=9.2e-16
```

Figure 9: The result of Broyden

```
|--34--|2.0000000|0.0000000|
|--35--|2.0000000|0.0000000|
|--36--|2.0000000|0.0000000|
Lazy Newton method failed to converge, n=1000, |F(xn)|=1.3e-10
```

Figure 10: The result of Lazy Newton

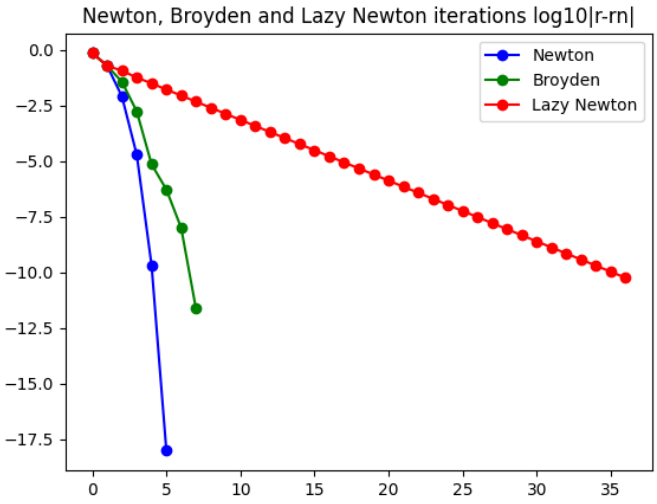

Figure 11: The Error Comparison of each method for 2b

c) $x = 0, y = 0$

From the code, all methods diverge since the initial guesses are on the line that makes the Jacobian matrix a singular matrix (No inverse of J)

$$\mathrm{J} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

which is non-invertible.

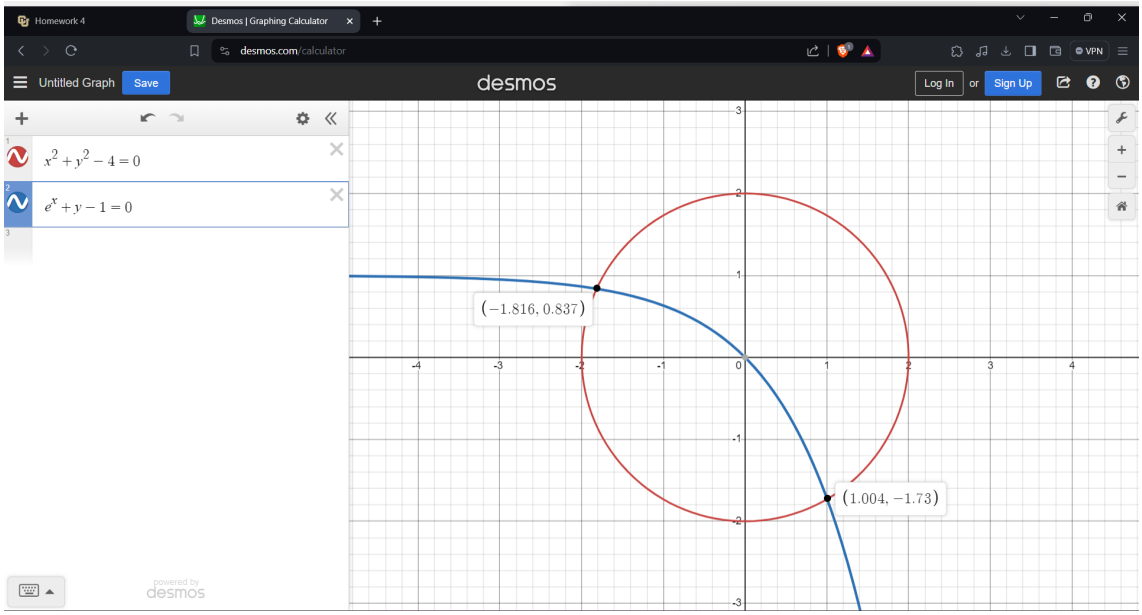

Figure 12: The exact solutions from graph

From figure 7 and figure 11, we can see that the Newton's method perform the best compared to the other two quasi-Newton methods. However, to get this performance, we need to find a good initial guess. If the initial

guesses are not near the solution, Newton's can also be slow or be divergent. Moreover, the reason we get the different results from different initial guesses is that the solution will depend on the near initial guesses. As we can see from figure 12, the initial guesses from part a are near the solution of $(-1.816, 0.837)$ **while the guesses from part b are near the solution of** $(1.004, -1.73)$.

Problem 3

Solution

a) Newton's method

For the code, click **Github Link for 3a**

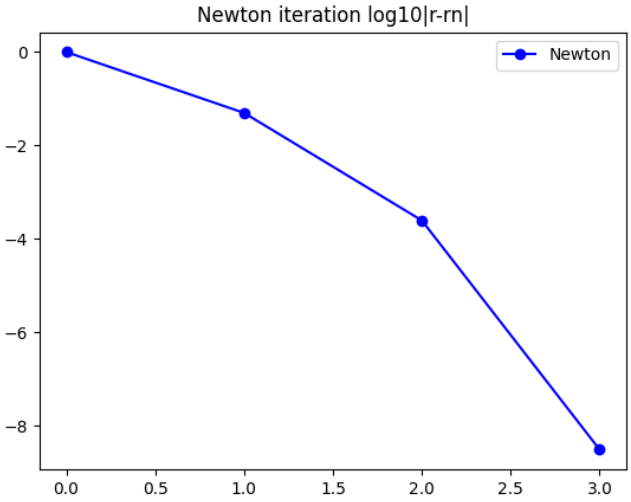

Figure 13: Error plot of Newton's method Vs Iterations for 3a

```
|--n--|----xn----|---|f(xn)|---|
|--0--|0.0000000|1.0000000|
|--1--|1.0096729|0.0499008|
|--2--|1.0047396|0.0002479|
|--3--|1.0049876|0.0000000|
Newton method converged, n=4, |F(xn)|=2.2e-16

Roots:  [0.  0.1 1. ]
Time:  0.07946062088012695
```

Figure 14: Results for 3a

b) Steepest descent method

For the code, click **Github Link for 3b**

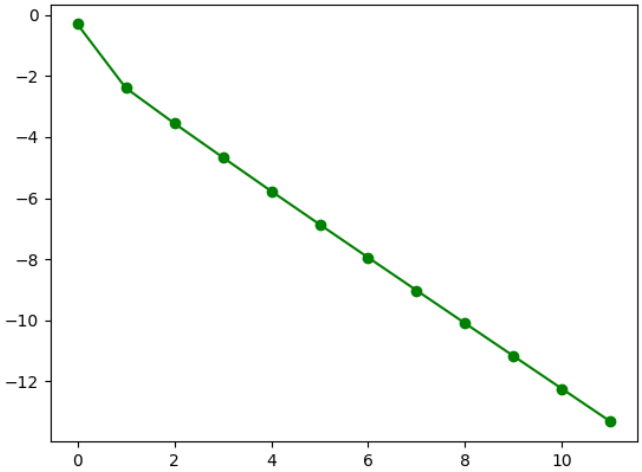

Figure 15: Error plot of Steepest descent method Vs Iterations for 3b

```
|--n--|-alpha-|----|xn|----|---|f(xn)|---|---|Gf(xn)|---|
|--0--|1.00000|0.0000000|0.5000000|1.0000500|
|--1--|1.00000|1.0000500|0.0040500|0.0928827|
|--2--|1.00000|1.0006734|0.0002830|0.0254425|
|--3--|1.00000|1.0052184|0.0000217|0.0072029|
|--4--|1.00000|1.0046692|0.0000017|0.0020458|
|--5--|1.00000|1.0049970|0.0000001|0.0005884|
|--6--|1.00000|1.0049657|0.0000000|0.0001700|
|--7--|1.00000|1.0049895|0.0000000|0.0000493|
|--8--|1.00000|1.0049860|0.0000000|0.0000143|
|--9--|1.00000|1.0049878|0.0000000|0.0000042|
|--10--|1.00000|1.0049874|0.0000000|0.0000012|
Roots:  [1.85971612e-07 9.99998023e-02 1.00000004e+00]
Time:  0.1214444637298584
```

Figure 16: Results for 3b

c) Stopping tolerance of 5×10^{-2}

For the code, click **Github Link for 3c.** The same codes as part a and part b but change the variable 'tol' for each method to get the results.

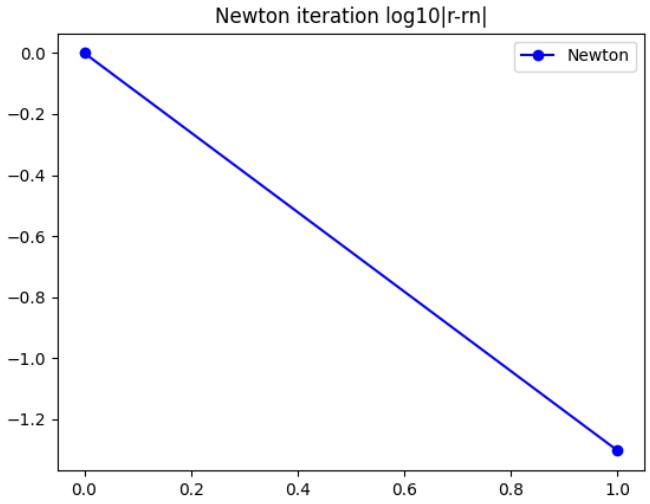

Figure 17: Error plot of Newton's method Vs Iterations for 3c

```
|--n--|----xn----|---|f(xn)|---|
|--0--|0.0000000|1.0000000|
|--1--|1.0096729|0.0499008|
Newton method converged, n=2, |F(xn)|=2.5e-04

Roots:  [0.         0.09998767 0.999752  ]
Time:  0.06823205947875977
```

Figure 18: Results for 3c Newton's Method

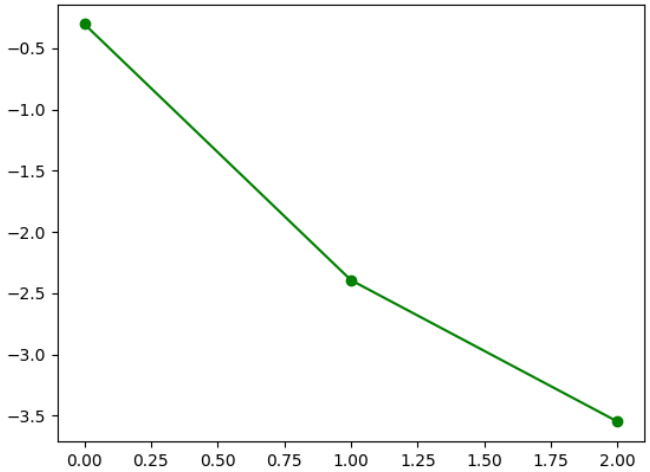

Figure 19: Error plot of Steepest descent method Vs Iterations for 3c

```
|--n--|-alpha-|----|xn|----|---|f(xn)|---|---|Gf(xn)|---|
|--0--|1.00000|0.0000000|0.5000000|1.0000500|
|--1--|1.00000|1.0000500|0.0040500|0.0928827|
Roots:  [-0.0225      0.09999928  0.99541   ]
Time:  0.03811764717102051
```

Figure 20: Results for 3c Steepest descent method

From all the results, we can see that with the first condition (tolerance within 10^{-6}), Newton's method can give us accurate results with faster running time while the steepest descent method can give us the approximation numbers and run more slowly than Newton's. Even if we change the tolerance to be 5×10^{-2}, the steepest runs a little bit faster but uses more iteration than Newton's. Generally, Newton's method can perform better in terms of results and running time if we have good initial guesses.

Problem 4

Solution

a)

$$\begin{aligned} P_n(x) &= y_0L_0(x) + y_1L_1(x) + \cdots + y_nL_n(x) \\ &= \sum_{j=0}^{n} y_iL_i(x) \end{aligned}$$

$$\begin{aligned} L_i(x) &= \frac{\prod_{j\neq i}(x - x_j)}{\prod_{j\neq i}(x_i - x_j)} \\ &= \prod_{j\neq i} \frac{(x - x_j)}{x_i - x_j} \end{aligned}$$

Calculate each $L_i(x)$

$$\begin{aligned} L_0(x) &= \frac{(x-2)(x-3)(x-5)(x-8)}{(0-2)(0-3)(0-5)(0-8)} = \frac{x^4 - 18x^3 + 111x^2 - 278x + 240}{240} \\ L_1(x) &= \frac{(x-0)(x-3)(x-5)(x-8)}{(2-0)(2-3)(2-5)(2-8)} = \frac{x^4 - 16x^3 + 79x^2 - 120x}{-36} \\ L_2(x) &= \frac{(x-0)(x-2)(x-5)(x-8)}{(3-0)(3-2)(3-5)(3-8)} = \frac{x^4 - 15x^3 + 66x^2 - 80x}{30} \\ L_3(x) &= \frac{(x-0)(x-2)(x-3)(x-8)}{(5-0)(5-2)(5-3)(5-8)} = \frac{x^4 - 13x^3 + 46x^2 - 48x}{-90} \\ L_4(x) &= \frac{(x-0)(x-2)(x-3)(x-5)}{(8-0)(8-2)(8-3)(8-5)} = \frac{x^4 - 10x^3 + 31x^2 - 30x}{720} \end{aligned}$$

Therefore,

$$\begin{aligned} P_4(x) &= -125L_0(x) - 27L_1(x) - 8L_2(x) + 27L_4(x) \\ &= x^3 - 15x^2 + 75x - 125 \end{aligned}$$

b) Use the table below to make it easier to calculate the coefficient of Newton interpolation

x	**0**	**2**	**3**	**5**	**8**
y	**-125**	**-27**	**-8**	**0**	**27**
1st order	$\frac{-27-(-125)}{2-0} = 49$	$\frac{-8-(-27)}{3-2} = 19$	$\frac{0-(-8)}{5-3} = 4$	$\frac{27-0}{8-5} = 9$	
2nd order	$\frac{19-49}{3-0} = -10$	$\frac{4-19}{5-2} = -5$	$\frac{9-4}{8-3} = 1$		
3rd order	$\frac{-5-(-10)}{5-0} = 1$	$\frac{1-(-5)}{8-2} = 1$			
4th order	$\frac{1-1}{8-0} = 0$				

Table 1: Newton Interpolation Table

Therefore,

$$\begin{aligned} P_4(x) &= -125 + 49(x-0) - 10(x-0)(x-2) + 1(x-0)(x-2)(x-3) + 0 \\ &= x^3 - 15x^2 + 75x - 125 \end{aligned}$$

c) **The order higher than 3 should vanish. We know that we get the data from $(x-5)^3$, so the higher order differences in Newton interpolation would vanish (ideally, the fourth order tend to be zero which is what we see in both part a and b)**