HOMEWORK # 06

02/28/2025

---

1 Question 1

For this question, we are given the following nonlinear system in 1 below which was solved in the previous assignment using *Newton's Method.*

$$\begin{cases} f(x,y) = x^2 + y^2 - 4 = 0 \\ g(x,y) = e^x + y - 1 = 0 \end{cases} \tag{1}$$

From last week's assignment, we know that this nonlinear system has two solutions which we were able to find using the following initial guesses for $\tilde{\mathbf{x}}_0$ where $\tilde{\mathbf{x}}_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$.

1. $\tilde{\mathbf{x}}_0 : x_0 = 1,\ y_0 = 1$
2. $\tilde{\mathbf{x}}_0 : x_0 = 1,\ y_0 = -1$
3. $\tilde{\mathbf{x}}_0 : x_0 = 0,\ y_0 = 0$

Additionally, the Jacobian, $\mathbf{J}$, was determined to be the following as shown below.

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

For this problem, we have been asked to use the two *Quasi-Newton* methods using different initial guesses to again solve this non-linear system. First, we will use the *Lazy-Newton Method* before using the *Broyden Method.* To summarize the output and convergence, consider the table below.

Method:	Initial Guess:	Convergence:	# Iterations:
Newton	(1,1)	Converged	8
	(1,-1)	Converged	5
	(0,0)	Did Not Converge	N/A
Lazy	(1,1)	Did Not Converge	N/A
	(1,-1)	Converged	26
	(0,0)	Did Not Converge	Did Not Converge
Broyden	(1,1)	Converged	13
	(1,-1)	Converged	7
	(0,0)	Did Not Converge	N/A

Table 1: Table of convergence for each method and given initial value.

1.1 Newton Method

Despite the homework stating that this problem was solved in bfHomework 05, I cannot seem to find that or remember doing so. A similar problem was proposed but the values and equations $f(x,y)$ and $g(x,y)$ were different as was the initial guess used. That being said, I also wrote code for a Newton's Method approach and found that both the first and second initial guesses converged. one converged to the point $(-1.81626,\ 0.83737)$ and $(1.00417,\ -1.72964)$ respectfully. The third initial value did not converge as the inverse of the jacobian, $\mathbf{J}^{-1}$, could not be found using *numpy.linalg.solve()* as $\mathbf{J}^{-1}$ is singular when the initial guess $(0,0)$ is used. That being said, the two guesses that did converge did so in 8 and 5 iterations respectfully.

1.2 Lazy-Newton Method

The *Lazy-Newton Method* had a difficult time converging to either of the two roots as the method only calculates the Jacobian, $\mathbf{J}$, once. Unlike the *Newton Method*, this caused large issues in this problem as one of the elements in the Jacobian was e^x. Since this value was never updated through the iterations, as the value of x grew larger, the value of e^x grew exponentially which blew up the value of x causing divergence rather than convergence to the root. Eventually, the value of x grew large enough that there was overflow and the method was forced to exit. This outcome occurred for both the first and third initial guesses listed above. As shown in the table, the second initial value converged to the root at $(1.00417,\ -1.72964)$ in 26 iterations; considerably slower than *Newton's Method*. This is understandable, however, as this method should only converge linearly.

1.3 Broyden Method

Like the *Newton Method*, the *Broyden Method* converged for the first and second initial guesses. These converged to the roots at $(-1.81626,\ 0.83737)$ and $(1.00417,\ -1.72964)$ respectively. As shown in the table above, this method converged to these points in 13 and 7 iterations respectively. In the case of the third initial guess, the method failed to converge as the Jacobian was singular as it was in the case for the *Newton Method*.

2 Question 2

Like Question 1, this question also revolves around solving a nonlinear system. For this problem, we are asked to consider the following nonlinear system found below in 2.

$$\begin{cases} x + cos(xyz) - 1 = 0 \\ (1-x)^{1/4} + y + 0.05z^2 - 0.15z - 1 = 0 \\ -x^2 - 0.1y^2 + 0.01y + z - 1 = 0 \end{cases} \tag{2}$$

For this question, we are tasked with finding approximate solution to the nonlinear system to within 10^{-6} using the following methods.

1. Newton's Method
2. Steepest Descent Method
3. First using the Steepest Descent method with a stopping tolerance of $5x10^{-2}$. Using the result as the initial guess, $\tilde{\mathbf{x}}_0$, for Newton's Method.

For each of the three methods, we are to use the same initial guess, $\tilde{\mathbf{x}}_0$, which was chosen to be $(0,1,1)$. Additionally, each of the above methods requires the use of the Jacobian of the system, $\mathbf{J}$ was calculated to be the following.

$$\mathbf{J} = \begin{bmatrix} -yz\sin(xyz) + 1 & -xz\sin(xyz) & -xy\sin(xyz) \\ -\frac{1}{4(1-x)^{(3/4)}} & 1 & 0.1z - 0.15 \\ -2x & 0.01 - 0.2y & 1 \end{bmatrix}$$

It is important to note that the value used for the initial guess, $\mathbf{x}_0$, was chosen very specifically. When using *Newton's Method* in solving nonlinear systems of equations, the Jacobian must be non-singular. This is required as the inverse of the Jacobian is used in the Newton step. Just by glancing at the system, the point $(x, y, z) = (0, 1, 1)$ appears as though it has a high likelihood of producing a non-singular Jacobian. While this can be determined analytically, I utilized the guess and check method. That is to say that this value for $\mathbf{x}_0$ was not determined on the first attempt. The results of the different methods where $\tilde{\mathbf{x}}_0 = (0, 1, 1)$ and the stopping tolerance is $tol = 10^-6$ can be found in the table below.

Method:	Convergence:	Converged To:	# Iterations:
Newton's	Converged	$(0, 0.1, 1)$	4
Steepest Descent	Converged	$(4.04030820 * 10^{-7}, 9.99997971 * 10^{-2}, 1.00000009)$	11
Mixed Method	Converged	$(0, 0.1, 1)$	4 (3,1)

Table 2: Table of results for each of the above methods using $\mathbf{x}_0 = (0, 1, 1)$.

2.1 Newton's Method

Using the same code from Question (1), the method converged to the point $(0, 0.1, 1)$ in 4 iterations. As seen in the table, this was the fastest single method when compared to the *Steepest Descent Method* alone. This makes sense as this method converges quadratically if it converges at all. All in all, we know that *Newton's Method* will converge the fastest and in the fewest number of iterations, but that requires a very good first/initial guess. As alluded to earlier, it took multiple attempts to find a $\mathbf{x}_0$ that would allow for this method to converge.

2.2 Steepest Descent

As shown in the table, the *Steepest Descent* method took many more iterations to converge. Similar to *Newton's Method*, this method also converged to the point $(x, y, z) = (4.04030820*10^{-7}, 9.99997971* 10^{-2}, 1.00000009)$. While this method did require more iterations to converge to the local minimum of the quadratic, it did so using fewer resources and a lower cost per iteration when compared to *Newton's Method*.

2.3 Steepest Descent and Newton's Method

The mixed method which used *Steepest Descent* to first refine the initial guess before continuing on with *Newton's Method* to fully converge within the desired tolerance was extremely efficient. As shown in the table, the mixed method took only four iterations in total to converge, matching *Newton's Method*. The mixed method took three iterations using *Steepest Descent* and only one using *Newton's Method*. While *Newton's Method* achieved this efficiency on its own, the beauty of the mixed method is the reduced average cost per iteration. Performing the *Steepest Descent* method first allows for a *"cheap"* refining of the initial guess $\mathbf{x}_0$. This can be seen as the mixed method only required a single iteration using Newton. When looking at performance, the mixed method balances efficiency and convergence with low cost per iteration. Thus making it the highest performer of the three.

3 Important Note

It is important to note that the code produced for this lab was built heavily on the code provided from class. Edits and changes in implementation were made to fit the needs of this assignment, but came directly from what was provided on the Canvas page. That being said, the code for this homework can be found in the GitHub repository under **Homework_06**.