# Homework6

Sunday, October 8, 2023 11:04 PM

Tony Samour
APPM 4600

1.) 1. The nonlinear system

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

has two real solutions. In the last homework you used Newton's method to solve this problem with following initial guesses:

(i) $x = 1$, $y = 1$
(ii) $x = 1$, $y = -1$
(iii) $x = 0$, $y = 0$

In this assignment, use the two quasi-Newton methods with the different initial guesses. Is the performance better or worse that of Newton's methods?

Goal: Compare the 3 methods using the given initial guesses.

For use in the Jacobian:

$$\left.\begin{array}{ll} f_x(x,y) = 2x & g_x(x,y) = e^x \\ f_y(x,y) = 2y & g_y(x,y) = 1 \end{array}\right\} \Rightarrow J = \begin{bmatrix} 2x & 2y \\ e^x & 1 \end{bmatrix}$$

Output:

```
$ python3 Problem1.py
C:\Users\tonys\Documents\APPM4600\Samour_APPM4600\Homework\Homework6\Problem1.py
:109: RuntimeWarning: overflow encountered in exp
  F[1] = np.exp(x[0]) + x[1] - 1
Initial Guess: (1,1)
[-1.81626407  0.8373678 ]
Newton: the error message reads: 0
Newton: took this many seconds: 0.0006199216842651367
Netwon: number of iterations is: 7
[nan nan]
Lazy Newton: the error message reads: 1
Lazy Newton: took this many seconds: 0.0
Lazy Newton: number of iterations is: 99
[-1.81626407  0.8373678 ]
Broyden: the error message reads: 0
Broyden: took this many seconds: 0.0
Broyden: number of iterations is: 12

Initial Guess: (1,-1)
[ 1.00416874 -1.72963729]
Newton: the error message reads: 0
Newton: took this many seconds: 0.00031257152557373046
Netwon: number of iterations is: 5
[ 1.00416874 -1.72963729]
Lazy Newton: the error message reads: 0
Lazy Newton: took this many seconds: 0.0
Lazy Newton: number of iterations is: 36
[ 1.00416874 -1.72963729]
Broyden: the error message reads: 0
Broyden: took this many seconds: 0.0
Broyden: number of iterations is: 6
Traceback (most recent call last):
  File "C:\Users\tonys\Documents\APPM4600\Samour_APPM4600\Homework\Homework6\Pro
blem1.py", line 225, in <module>
    driver()
    ^^^^^^^^
  File "C:\Users\tonys\Documents\APPM4600\Samour_APPM4600\Homework\Homework6\Pro
blem1.py", line 77, in driver
    [xstar,ier,its] =  Newton(x0_2,tol,Nmax)
                       ^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\tonys\Documents\APPM4600\Samour_APPM4600\Homework\Homework6\Pro
blem1.py", line 128, in Newton
    Jinv = inv(J)
           ^^^^^^
  File "C:\Users\tonys\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\numpy\linalg\
linalg.py", line 561, in inv
    ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\tonys\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\numpy\linalg\
linalg.py", line 112, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix
```

For the initial guess $(1,1)$:
Newton converged in the fewest iterations but took longest. Lazy Newton did not converge

For the initial guess $(1,-1)$:
All three methods converged, and Lazy Newton took many more iterations than the other two. The Broyden method only required one more iteration than Newton's method.

For the initial guess $(0,0)$:
An error message appeared saying singular matrix.

Let's check: $(x_0, y_0) = (0,0) \Rightarrow J_0 = J(0,0) = \begin{bmatrix} 2(0) & 2(0) \\ e^{(0)} & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$

$\det(J_0) = (0)(1) - (0)(1) = 0 \Rightarrow$ Singular matrix $\Rightarrow$ Matrix is not invertible.

Since the initial Jacobian cannot be inverted, none of the methods can be used.

2.) 2. Consider the nonlinear system

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

Using your own codes test the following three techniques for approximating the solution to the nonlinear system to within $10^{-6}$:

- Newton's method
- Steepest descent method
- First Steepest descent method with a stopping tolerance of $5 \times 10^{-2}$. Use the result of this as the initial guess for Newton's method.

Using the same initial guess, which technique converges the fastest? Try to explain the performance.



$F(x,y,z) = x + \cos(xyz) - 1$ $\quad g(x,y,z) = (1-x)^{1/4} + y + 0.05z^2 - 0.15z - 1$ $\quad h(x,y,z) = -x^2 - 0.1y^2 + 0.01y + z - 1$

$\frac{\partial F}{\partial x} = 1 - yz \cdot \sin(xyz)$ $\quad \frac{\partial g}{\partial x} = -\frac{1}{4}(1-x)^{-3/4}$ $\quad \frac{\partial h}{\partial x} = -2x$

$\frac{\partial F}{\partial y} = -xz \cdot \sin(xyz)$ $\quad \frac{\partial g}{\partial y} = 1$ $\quad \frac{\partial h}{\partial y} = -0.2y + 0.01$

$\frac{\partial F}{\partial z} = -xy \cdot \sin(xyz)$ $\quad \frac{\partial g}{\partial z} = 0.1z - 0.15$ $\quad \frac{\partial h}{\partial z} = 1$

We can use the partial derivatives above to create the Jacobian, J:

$$J = \begin{bmatrix} F_x & F_y & F_y \\ g_x & g_y & g_z \\ h_x & h_y & h_z \end{bmatrix}$$

Output:

```
tonys@TonyStudio MINGW64 ~/Documents/APPM4600/Samour_APPM4600/Homework/Homework6
 (main)
$ python3 Problem2Newton.py
[-0.98606765 -0.08586088  1.97392522]
Newton: the error message reads: 0
Newton: took this many seconds: 0.0003202342987060547
Netwon: number of iterations is: 5

tonys@TonyStudio MINGW64 ~/Documents/APPM4600/Samour_APPM4600/Homework/Homework6
 (main)
$ python3 Problem2Steepest.py
[-0.98450405 -0.08600293  1.96944596]
Steep: the error message reads: 0
Steep: took this many seconds: 0.0047102212905883786
g evaluated at this point is  4.949975984623329e-06

tonys@TonyStudio MINGW64 ~/Documents/APPM4600/Samour_APPM4600/Homework/Homework6
 (main)
$ python3 Problem2Hybrid.py
[-0.98606765 -0.08586088  1.97392522]
Hybrid: the error message reads: 0
Hybrid: took this many seconds: 0.0006297063827514648
Netwon: number of iterations is: 3

tonys@TonyStudio MINGW64 ~/Documents/APPM4600/Samour_APPM4600/Homework/Homework6
 (main)
$ 
```

All codes were run from the same initial point: (0, 0, 0).

Based on the output above, it appears that Newton's Method appears fastest. However, since it requires taking the inverse of the Jacobian at every step, it is computationally "expensive". So, using steepest descent to converge more quickly and then Newton's Method to "fine-tune" the results is an effective method.