APPM 4600 — HOMEWORK # 5 Solution

For all homeworks, you can use any approved coding language such as Matlab, C, C++, Python, etc. **Do not use** symbolic software such as Maple or Mathematica.

1. (30 points) Suppose we want to find a solution located near $(x, y) = (1, 1)$ to the nonlinear set of equations

$$\begin{aligned} f(x,y) &= 3x^2 - y^2 = 0, \\ g(x,y) &= 3xy^2 - x^3 - 1 = 0 \end{aligned} \tag{1}$$

   (a) Iterate on this system numerically (for example with Matlab), using the iteration scheme

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - \begin{bmatrix} 1/6 & 1/18 \\ 0 & 1/6 \end{bmatrix} \begin{bmatrix} f(x_n, y_n) \\ g(x_n, y_n) \end{bmatrix}, \quad n = 0, 1, 2, \ldots$$

   starting with $x_0 = y_0 = 1$, and check how well it converges.

   (b) Provide some motivation for the particular choice of the numerical 2×2 matrix in the equation above.

   (c) Iterate on (1) using Newton's method, using the same starting approximation $x_0 = y_0 = 1$, and check how well this converges.

   (d) Spot from your numerical result what the exact solution is, and then verify that analytically.

Soln: The code for all parts of this problem are below.

   (a) The iteration converges with an absolute error in 30 iterations.

   (b) The matrix $\begin{bmatrix} 1/6 & 1/18 \\ 0 & 1/6 \end{bmatrix}$ is the inverse of the Jacobian evaluated at $\boldsymbol{x}_0$. Thus this iteration scheme is Lazy Newton.

   (c) Newtons method converges in just 9 iterations.

   (d) The exact solution is $x = 1/2$ and $y = \sqrt{3}/2$. $f(1/2, \sqrt{3}/2) = 0$ and $g(1/2, \sqrt{3}/2) = 0$

2. Consider the nonlinear system of equations

$$\begin{cases} x = \dfrac{1}{\sqrt{2}}\sqrt{1+(x+y)^2} - \dfrac{2}{3} \\ y = \dfrac{1}{\sqrt{2}}\sqrt{1+(x-y)^2} - \dfrac{2}{3} \end{cases}$$

Theorem 10.6 in the textbook (on page 633 in the 9^{th} edition reads as follows:

Let $D = \{(x_1, x_2, \ldots, x_n)^t : a_i \leq x_i \leq b_i,\}$ some collection of constants $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$. Supposed that G is a continuous function from $D \subset \mathbb{R}^n$ into $\mathbb{R}^n$ with the property that $\mathsf{G}(\boldsymbol{x}) \in D$ whenever $\boldsymbol{x} \in D$. Then G has a fixed point in D

Moreover, supposed that all the component functions of G have continuous partial derivative and a constant $K \leq 1$ exists with

$$\left|\frac{\partial g_i(\boldsymbol{x})}{\partial x_j}\right| \leq \frac{K}{n}$$

whenever $\boldsymbol{x} \in D$, for each $j = 1, \ldots, n$ and each component function g_i. Then the sequence $\{\boldsymbol{x}^{(k)}\}_{k=0}^{\infty}$ defined by an arbitrary selected $\boldsymbol{x}^{(0)}$) in D and generated by

$$\boldsymbol{x}^{(k)} = \mathsf{G}(\boldsymbol{x}^{(k-1)}), \quad \text{for each } k \geq 1$$

converges to the unique fixed point $\boldsymbol{p} \in D$ and

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{p}\|_\infty \leq \frac{K^k}{1-K}\|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}\|_\infty.$$

Based on this theorem, find a region D in the $x, y-$plane for which the fixed point iteration

$$\begin{cases} x_{n+1} = \dfrac{1}{\sqrt{2}}\sqrt{1+(x_n+y_n)^2} - \dfrac{2}{3} \\ y_{n+1} = \dfrac{1}{\sqrt{2}}\sqrt{1+(x_n-y_n)^2} - \dfrac{2}{3} \end{cases}$$

is guaranteed to converge to a unique solution for any starting point $(x_0, y_0) \in D$.

Soln:

For this problem $n = 2$.

We want $\left|\frac{\partial g_i(\boldsymbol{x})}{\partial x_j}\right| \leq \frac{1}{2}$ for all $\boldsymbol{x} \in D$.

$$\left|\frac{\partial g_1(\boldsymbol{x})}{\partial x}\right| = \left|\frac{x+y}{\sqrt{2}\sqrt{1+(x+y)^2}}\right|$$

$$\left|\frac{\partial g_1(\boldsymbol{x})}{\partial y}\right| = \left|\frac{x+y}{\sqrt{2}\sqrt{1+(x+y)^2}}\right|$$

$$\left|\frac{\partial g_2(\boldsymbol{x})}{\partial x}\right| = \left|\frac{x-y}{\sqrt{2}\sqrt{1+(x-y)^2}}\right|$$

$$\left|\frac{\partial g_2(\boldsymbol{x})}{\partial x}\right| = \left|\frac{x-y}{\sqrt{2}\sqrt{1+(x-y)^2}}\right|$$

The only way that all these partial derivatives can be less than $\frac{1}{2}$ is if $|x|+|y| \leq 1$. This means that $D = \{(x,y) \in \mathbb{R}^2 : |x|+|y| \leq 1\}$.

We must show that G applied to $\boldsymbol{x} \in D$ is in D.

Let $\boldsymbol{x} \in D$. Then $|x|+|y| \leq 1$.

Now

$$|g_1(\boldsymbol{x})| = |\frac{1}{\sqrt{2}}\sqrt{1+(x+y)^2} - \frac{2}{3}| \leq |1-\frac{2}{3}| = 1/3$$

and

$$|g_2(\boldsymbol{x})| = |\frac{1}{\sqrt{2}}\sqrt{1+(x-y)^2} - \frac{2}{3}| \leq |1-\frac{2}{3}| = 1/3$$

The point $(1/3, 1/3)$ is in D thus the mapping G is onto. The fact that is continuous follows from the fact that g_1 and g_2 are discontinuo.

3. (30 points) Let $f(x,y)$ be a smooth function such that $f(x,y)=0$ defines a smooth curve in the $x,y-$plane. We want to find some point on this curve that lies in the neighborhood of a start guess (x_0,y_0) that is off the curve. i.e. your goal is to move from an initial guess to the curve $f(x,y)=0$.

   (a) Derive the iteration scheme

$$\begin{cases} x_{n+1} = x_n - d f_x \\ y_{n+1} = y_n - d f_y \end{cases}$$

   for solving the task outlined above. Here $d = f/(f_x^2 + f_y^2)$, and it is understood that f, f_x, f_y are all evaluated at the location (x_n, y_n).

   Hint: One way to proceed is to look for a new iterate (x_{n+1}, y_{n+1}) that (i) lies on the gradient line through (x_n, y_n) and (ii) also obeys $f(x,y) = 0$. Apply Newton to this 2×2 system.

   (b) The iteration scheme above generalizes in an obvious way to moving from a start location (x_0, y_0, z_0) onto a surface $f(x,y,z) = 0$. With this iteration, find a point the ellipsoid $x^2 + 4y^2 + 4z^2 = 16$ when starting from $x_0 = y_0 = z_0 = 1$. Give numerical evidence showing that the iteration indeed is quadratically convergent.

Soln:

(a) The equation of the normal line at (x_n, y_n) is

$$l_n(x,y) = \frac{x - x_n}{f_x} - \frac{y - y_n}{f_y} = 0.$$

Note that at (x_n, y_n), $l(x_n, y_n) = 0$.
We are going to apply one step of Newton's method to the following system of equations

$$l_n(x,y) = 0$$

and

$$f(x,y) = 0.$$

The Jacobian of this system is

$$\mathsf{G} = \begin{bmatrix} \frac{1}{f_x} & -\frac{1}{f_y} \\ f_x & f_y \end{bmatrix}$$

and it's inverse is

$$\mathsf{G}^{-1} = \frac{f_x f_y}{f_x^2 + f_y^2} \begin{bmatrix} f_y & \frac{1}{f_y} \\ -f_x & \frac{1}{f_x} \end{bmatrix}$$

Thus Newtons applied to this system is

$$
\begin{aligned}
\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} &= \begin{bmatrix} x_n \\ y_n \end{bmatrix} - \mathsf{G}^{-1}|_{(x_n,y_n)} \begin{bmatrix} l_n(x_n,y_n) \\ f(x_n,y_n) \end{bmatrix} \\
&= \begin{bmatrix} x_n \\ y_n \end{bmatrix} - \frac{f_x f_y}{f_x^2 + f_y^2} \begin{bmatrix} f_y & \frac{1}{f_y} \\ -f_x & \frac{1}{f_x} \end{bmatrix} \begin{bmatrix} 0 \\ f(x_n,y_n) \end{bmatrix} \\
&= \begin{bmatrix} x_n - df_x \\ y_n - df_y \end{bmatrix}
\end{aligned}
$$

(b) The scheme converges in 6 iterations to with an absolute convergence error of $1e-14$. Table 1 reports the value of the ratio

$$
r_k = \frac{\|\boldsymbol{\alpha} - \boldsymbol{x}_k\|}{\|\boldsymbol{\alpha} - \boldsymbol{x}_{k-1}\|^2}
$$

for each iteration. Note that it is converging to a constant which means that the method is second order. Below is the code.

k	r_k
1	0.339876735691327
2	0.242518401255263
3	0.253243077063564
4	0.253393911370371

Table 1: Ratio r_k of the errors at each iteration

```
import numpy as np
import math
import time
from numpy.linalg import norm

def driver():

    tol = 1e-14

    Nmax = 100

    x0 = np.array([1,1,1])

    [xstar,ier,it] = test_iteration(x0,tol,Nmax)
    print(xstar)
    print('the error message reads:', '%d' % ier)
    print('number of iterations:', '%d' % it)


def test_iteration(x0,tol,Nmax):
```

```
    for it in range(Nmax):

      dtmp = evald(x0[0],x0[1],x0[2])
      fx = evalfx(x0[0],x0[1],x0[2])
      fy =evalfy(x0[0],x0[1],x0[2])
      fz =  evalfz(x0[0],x0[1],x0[2])
      tmp = np.array([fx,fy,fz])
      xk = x0- dtmp*tmp
      if (norm(x0-xk)<tol):
         xstar = xk
         ier = 0
         return[xstar,ier,it]
      x0 = xk

    xstar = xk
    ier = 1
    return[xstar,ier,it]


def evalf(x,y,z):
      f = x**2+4*y**2+4*z**2-16
      return f

def evalfx(x,y,z):
      fx = 2*x
      return fx

def evalfy(x,y,z):
      fy = 8*y
      return fy

def evalfz(x,y,z):
    fz = 8*z
    return fz

def evald(x,y,z):
    d = evalf(x,y,z)/(evalfx(x,y,z)**2+evalfy(x,y,z)**2+evalfz(x,y,z)**2)
    return d

if __name__ == '__main__':
  # run the drivers only if this is called from the command line
  driver()
```