University of Colorado
Boulder

HW5

Jesse Hettleman
APPM 4600

College of Engineering and Applied Science

Department of Applied Math

October 4, 2024

1 Problem 1

In this problem we are tasked with finding a solution near the point $(x,y) = (1,1)$ to the non-linear set of equations:

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

1.a

The first way we solve this system is by iterating the following equation:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - \begin{bmatrix} (1/6) & (1/18) \\ 0 & (1/6) \end{bmatrix} \begin{bmatrix} f(x_n,y_n) \\ g(x_n,y_n) \end{bmatrix} \tag{3}$$

When we iterate this sequence in python with an initial guess $x_0 = y_0 = 1$, we find that the sequence converges to $\begin{bmatrix} x_n = 0.5 \\ y_n = 0.8660254 \end{bmatrix}$ in seven iterations.

1.b

The reason we use the matrix $\begin{bmatrix} (1/6) & (1/18) \\ 0 & (1/6) \end{bmatrix}$ within this iteration can be investigated by looking at the Jacobian matrix for $\begin{bmatrix} f(x_n,y_n) \\ g(x_n,y_n) \end{bmatrix}$.

The Jacobian matrix for these functions is:

$$J_F(x,y) = \begin{bmatrix} f_x(x,y) & f_y(x,y) \\ g_x(x,y) & g_y(x,y) \end{bmatrix} = \begin{bmatrix} 6x & -2y \\ 3y^2 - 3x^2 & 6xy \end{bmatrix} \tag{4}$$

Therefore:

$$J_F(1,1) = \begin{bmatrix} 6 & -2 \\ 0 & 6 \end{bmatrix} \tag{5}$$

When we calculate the inverse of this matrix in python, we find it to be:

$$J_F^{-1}(1,1) = \begin{bmatrix} 0.16666667 & 0.05555556 \\ 0 & 0.16666667 \end{bmatrix} = \begin{bmatrix} (1/6) & (1/18) \\ 0 & (1/6) \end{bmatrix} \tag{6}$$

Therefore, the significance of the matrix used within this iteration is that it is the inverse of the Jacobian of our initial guess $(1,1)$. This is the equivalent of using the Lazy Newton (Chord) iteration in multiple dimensions.

1.c

When iterating on this problem using Newton's method and an error tolerance of 10^{-14}, we find that the sequence converges to $\begin{bmatrix} x_n = 0.5 \\ y_n = 0.8660254 \end{bmatrix}$ in five iterations. This is definitely improved from the last method! See 4 for code.

1.d The exact solution to this system of equations is $(\frac{1}{2}, \frac{\sqrt{3}}{2})$. This can be verified analytically by plugging these results into our equations:

$$\begin{bmatrix} f(\frac{1}{2}, \frac{\sqrt{3}}{2}) \\ g(\frac{1}{2}, \frac{\sqrt{3}}{2}) \end{bmatrix} = \begin{bmatrix} 3(\frac{1}{2})^2 - (\frac{\sqrt{3}}{2})^2 = \frac{3}{4} - \frac{3}{4} = 0 \\ 3(\frac{1}{2})(\frac{\sqrt{3}}{2})^2 - (\frac{1}{2})^3 - 1 = \frac{9}{8} - \frac{9}{8} = 0 \end{bmatrix} \tag{7}$$

2 Problem 2

In this problem we consider the nonlinear system of equations:

$$x = \frac{1}{\sqrt{2}}\sqrt{1+(x+y)^2} - \frac{2}{3} \tag{8}$$
$$y = \frac{1}{\sqrt{2}}\sqrt{1+(x-y)^2} - \frac{2}{3} \tag{9}$$

Using the given theorem, we will find a region $D \in \mathbb{R}^2$ for which the following fixed point iteration will converge:

$$G(x,y) = [G_1(x,y), G_2(x,y)] \text{ where} \tag{10}$$
$$G_1(x_n,y_n) = x_{n+1} = \frac{1}{\sqrt{2}}\sqrt{1+(x_n+y_n)^2} - \frac{2}{3} \tag{11}$$
$$G_2(x_n,y_n) = y_{n+1} = \frac{1}{\sqrt{2}}\sqrt{1+(x_n-y_n)^2} - \frac{2}{3} \tag{12}$$

According to the theorem, we want to find a region D such that $x_{n+1} = G_1(x_n, y+n)$ and $y_{n+1} = G_2(x_n, y+n)$. This also requires $G(x,y) \in D$ when $(x,y) \in D$, meaning $G(x,y)$ is a function mapping from $D \to D$.

To find a region satisfying these criteria, we will take the partial derivatives of $G(x,y)$ with respect to both variables:

$$G_{1_x}(x_n,y_n) = \frac{(x_n+y_n)}{\sqrt{2}}(1+(x_n+y_n)^2)^{-\frac{1}{2}} \tag{13}$$

$$G_{1_y}(x_n,y_n) = \frac{(x_n+y_n)}{\sqrt{2}}(1+(x_n+y_n)^2)^{-\frac{1}{2}} \tag{14}$$

$$G_{2_x}(x_n,y_n) = \frac{(x_n-y_n)}{\sqrt{2}}(1+(x_n-y_n)^2)^{-\frac{1}{2}} \tag{15}$$

$$G_{2_y}(x_n,y_n) = \frac{(x_n-y_n)}{\sqrt{2}}(1+(x_n-y_n)^2)^{-\frac{1}{2}} \tag{16}$$

Thus, we want to find D where:

$$max[|\tfrac{(x_n+y_n)}{\sqrt{2}}(1+(x_n+y_n)^2)^{-\frac{1}{2}}|,|\tfrac{(x_n-y_n)}{\sqrt{2}}(1+(x_n-y_n)^2)^{-\frac{1}{2}}|] \leq \tfrac{K\leq 1}{n=2} \leq \tfrac{1}{2}$$

First let's bound $|x_n+y_n| \leq |x_n|+|y_n|$ and $|x_n-y_n| \leq |x_n|+|y_n|$ by the triangle inequality. Then, bound $|x_n| \leq \frac{1}{2}$ and $|y_n| \leq \frac{1}{2}$ so that:

$$|\tfrac{(x_n+-y_n)}{\sqrt{2}}(1+(x_n+-y_n)^2)^{-\frac{1}{2}}| \leq \tfrac{1}{\sqrt{2(1+1)}} \leq \tfrac{1}{2}$$

Therefore, starting at any point (x_0,y_0) in the square region $D = (x,y) \in \mathbb{R}^2 : |x|,|y| \leq \frac{1}{2}$ will converge to a unique solution for the fixed point iteration $G(x,y)$.

3 Problem 3

3.a

The idea behind Newton's method for solving a system of nonlinear equations is to approximate the function by finding the zeros of tangent lines around a point and iteratively improving the guess for the solution. In this case, we seek a solution to the equation $f(x,y) = 0$.

The iteration scheme for moving from an initial guess (x_0,y_0) to the curve $f(x,y) = 0$ is derived by approximating $f(x_{n+1},y_{n+1})$ using its first-order Taylor expansion around (x_n,y_n):

$$f(x_{n+1},y_{n+1}) \approx f(x_n,y_n) + f_x(x_n,y_n)(x_{n+1}-x_n) + f_y(x_n,y_n)(y_{n+1}-y_n)$$

Since we are finding the zeros $f(x_{n+1},y_{n+1}) = 0$, the equation becomes:

$$0 = f(x_n,y_n) + f_x(x_n,y_n)(x_{n+1}-x_n) + f_y(x_n,y_n)(y_{n+1}-y_n) \tag{17}$$

$$f(x_n,y_n) = f_x(x_n,y_n)(x_{n+1}-x_n) - f_y(x_n,y_n)(y_{n+1}-y_n) \tag{18}$$

In Newton's method, the goal is to update the current guess (x_n,y_n) in such a way that the new point (x_{n+1},y_{n+1}) moves closer to the curve $f(x,y) = 0$. However, to ensure the correct scaling of these partial derivatives, the step size d is introduced as:

$$d = \frac{f(x_n, y_n)}{f_x(x_n, y_n)^2 + f_y(x_n, y_n)^2}$$

This ensures that the movement in both the x- and y-directions is proportional to the magnitude of the gradient, allowing the new point to approach the curve optimally. Therefore, this step size is chosen to normalize movement due to the partial derivatives. Substituting d into the equation and separating the x and y variables gives:

$$\begin{aligned} x_{n+1} - x_n &= -d f_x(x_n, y_n) \\ y_{n+1} - y_n &= -d f_y(x_n, y_n) \end{aligned}$$

Thus, solving for x_{n+1} and y_{n+1}, we arrive at the iteration scheme:

$$\begin{aligned} x_{n+1} &= x_n - d f_x(x_n, y_n) \\ y_{n+1} &= y_n - d f_y(x_n, y_n) \end{aligned}$$

3.b

To generalize the iteration scheme for moving from a start location (x_0, y_0, z_0) onto a surface $f(x, y, z) = 0$, we follow a similar process as in part (a). The equation in this case is $f(x, y, z) = x^2 + 4y^2 + 4z^2 - 16 = 0$.

We will use Newton's method to iterate and approximate the roots of this equation. First, we define the function:

$$f(x, y, z) = x^2 + 4y^2 + 4z^2 - 16$$

Then, we compute the partial derivatives:

$$f_x(x, y, z) = 2x, \quad f_y(x, y, z) = 8y, \quad f_z(x, y, z) = 8z$$

We know the step size d is given by:

$$d = \frac{f(x_n, y_n, z_n)}{f_x(x_n, y_n, z_n)^2 + f_y(x_n, y_n, z_n)^2 + f_z(x_n, y_n, z_n)^2}$$

Therefore, the iteration scheme updates the values of (x, y, z) as follows:

$$x_{n+1} = x_n - d f_x(x_n, y_n, z_n)$$

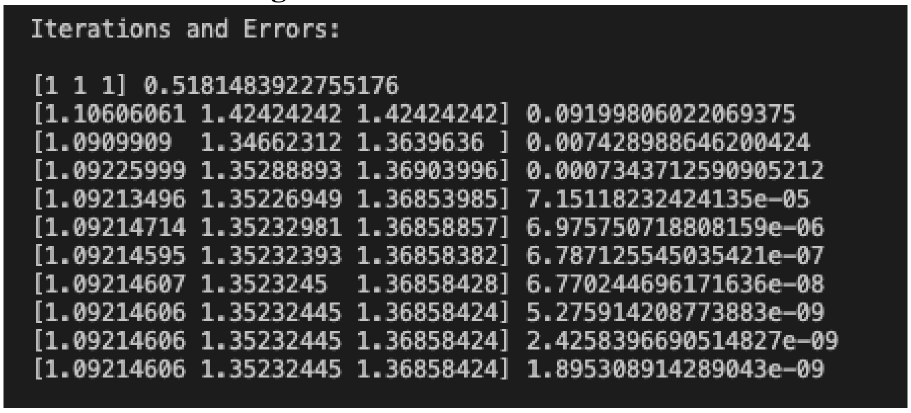
$$\begin{aligned} y_{n+1} &= y_n - d f_y(x_n, y_n, z_n) \\ z_{n+1} &= z_n - d f_z(x_n, y_n, z_n) \end{aligned}$$

For a numerical example calculated in python, we start from $(x_0, y_0, z_0) = (1, 1, 1)$ and perform several iterations using the scheme above. After 10 iterations, we obtain the point:

$$(x, y, z) \approx (1.09214606, 1.35232445, 1.36858424)$$

To show this is quadratically convergent, we display a table of the estimates and errors for each iteration:

Figure 1: Estimates and Errors

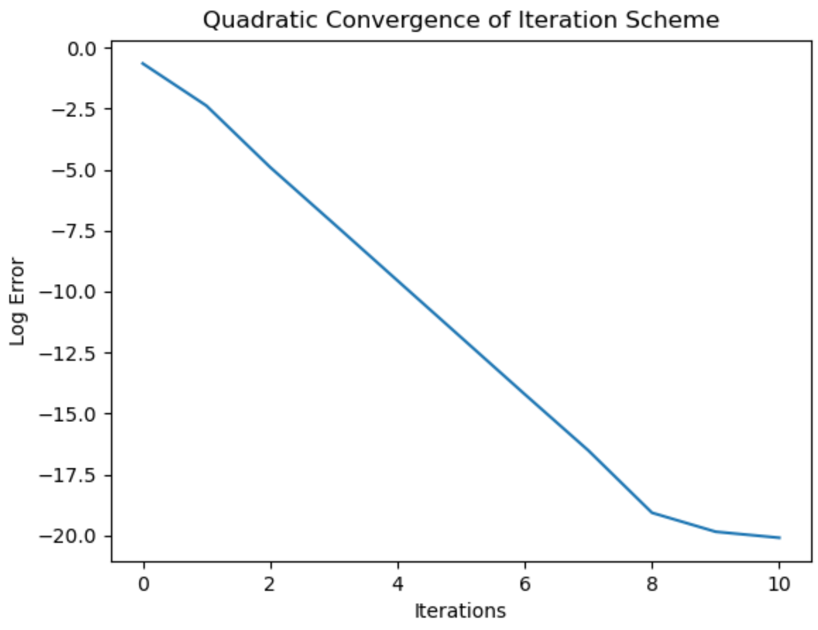


```
Iterations and Errors:

[1 1 1] 0.5181483922755176
[1.10606061 1.42424242 1.42424242] 0.09199806022069375
[1.0909909  1.34662312 1.3639636 ] 0.007428988646200424
[1.09225999 1.35288893 1.36903996] 0.0007343712590905212
[1.09213496 1.35226949 1.36853985] 7.15118232424135e-05
[1.09214714 1.35232981 1.36858857] 6.975750718808159e-06
[1.09214595 1.35232393 1.36858382] 6.787125545035421e-07
[1.09214607 1.3523245  1.36858428] 6.770244696171636e-08
[1.09214606 1.35232445 1.36858424] 5.275914208773883e-09
[1.09214606 1.35232445 1.36858424] 2.4258396690514827e-09
[1.09214606 1.35232445 1.36858424] 1.895308914289043e-09
```

It is clear that the iterations are converging rapidly. Further evidence of the quadratic convergence can be found when plotting the log error for each iteration:

Figure 2: Log Error vs. Iteration



The slope of this line is approximately -2 indicating how the log error decreases by a factor of two for each iteration. This shows the quadratic convergence of the sequence.

4 Appendix

4.1 Code

```
import numpy as np
import math
```

```python
import matplotlib.pyplot as plt

def newton_2(h,J,x0,tol,Nmax):
    """
    Newton iteration.

    Inputs:
    f,J - function array and derivative
    n - matrix dimensions
    p0   - initial guess for root
    tol  - iteration stops when p_n,p_{n+1} are within tol
    Nmax - max number of iterations
    Returns:
    p     - an array of the iterates
    pstar - the last iterate
    info  - success message
            - 0 if we met tol
            - 1 if we hit Nmax iterations (fail)

    """
    #   x = np.zeros(Nmax+1)
    #   x[0] = x0

    f = h[0]
    g = h[1]

    f_x = J[0][0]
    f_y = J[0][1]
    g_x = J[1][0]
    g_y = J[1][1]

    for it in range(Nmax):
        x,y = x0
        h_x0 = np.array([f(x,y),g(x,y)])
        J_x0 = np.array([[f_x(x,y), f_y(x,y)],[g_x(x,y),g_y(x,y)]])

        pn = np.linalg.solve(J_x0,-h_x0)
        x1 = x0 + pn

    #   x[it+1] = x1
        if (np.linalg.norm(x1-x0) < tol):
            xstar = x1
            info = 0
            return [xstar,info,it]
        x0 = x1
    xstar = x1
    info = 1
    return [xstar,info,it]

def question1():

    f = lambda x,y: 3*(x**2) - (y**2)
    g = lambda x,y: 3*x*(y**2) - (x**3) - 1

    fs = np.array([f,g])

    x_y_0 = np.array([1,1])
    J_inv = np.array([[(1/6),(1/18)],[0,(1/6)]])

    x_y_n = x_y_0

    for n in range(0,11):
        for i in range(0,n):
            x,y = x_y_n
            f_g_n = np.array([f(x,y),g(x,y)])

            x_y_n = x_y_n - np.matmul(J_inv,f_g_n)

        print(n,"iterations:", x_y_n)

    J_0 = np.array([[6,-2],[0,6]])
    J_0_inv = np.linalg.inv(J_0)
    print("J(1,1)^-1:",J_0_inv)

    print("\nNewton Method:\n")

    f_x = lambda x,y: 6*x
    f_y = lambda x,y: -2*y
    g_x = lambda x,y: 3*(y**2) - 3*(x**2)
    g_y = lambda x,y: 6*x*y
    J = np.array([[f_x, f_y],[g_x,g_y]])

    Nmax = 100
    tol = 1.e-14

    (xstar,info,it) = newton_2(fs,J,x_y_0,tol,Nmax)
    print('the approximate root is', xstar)
    print('the error message reads:', info)
    print('Number of iterations:', it)

# question1()

def question3():

    f = lambda x,y,z: x**2 + 4*y**2 + 4*z**2 - 16

    f_x = lambda x,y,z: 2*x
    f_y = lambda x,y,z: 8*y
    f_z = lambda x,y,z: 8*x

    def newton_step(x, y, z):
        fx = f_x(x, y, z)
        fy = f_y(x, y, z)
        fz = f_z(x, y, z)
        f_val = f(x, y, z)

        d = f_val / (fx**2 + fy**2 + fz**2)

        x_n = x - d * fx
        y_n = y - d * fy
        z_n = z - d * fz

        return np.array([x_n, y_n, z_n])

    # Initial Guess
    v = np.array([1, 1, 1])
    true = np.array([1.09214606,1.35232445,1.36858424])
    err = np.linalg.norm(v - true)

    n = 0
    errors = np.zeros((11,1))
    errors[n] = err

    print("\nIterations and Errors:\n")
    print(v,err)

    for i in range(10):
        n += 1
        v = newton_step(v[0], v[1], v[2])
        err = np.linalg.norm(v - true)
        errors[n] = err
        print(v,err)
    print("\n")

    x_vals = np.array([i for i in range(0,11)])

    plt.plot(x_vals,[math.log(err) for err in errors])
    plt.xlabel("Iterations")
    plt.ylabel("Log Error")
    plt.title("Quadratic Convergence of Iteration Scheme")
    plt.savefig("HW5.3.b.png")


question3()
```

NOTE: ALL CODE FOR THIS ASSIGNMENT IS ORIGINAL, WRITTEN BY JESSE HETTLEMAN