Math151B HW5

March 2, 2020

1 Math 151B Homework No. 5

- Brandon Loptman
- UID: 604105043
- March 3, 2020

[17]:
```
#Importing all the necessay libraries.
import numpy as np
import matplotlib.pyplot as plt
plt.style.use("ggplot")
```

[18]:
```
def F(f,x,n):
    """
    Evaluates an n dimensional array of functions f at a point x.
    """
    fx = np.zeros([n,1])

    for i in range(n):
        fx[i] = f[i](x)

    return fx

def J(j,x,n):
    """
    Evaluates an n x n matrix of functions j at a point x.
    """
    jx = np.zeros([n,n])

    for i in range(n):
        for k in range(n):
            jx[i,k] = j[i,k](x)

    return jx
```

1.1 10.3 Quasi-Newton Methods

[19]:
```python
def BroydenMethod(n, x0, tol = 10e-6, N = 100):
    """
    Implements Broyden's Method according to the algorithm in the book.
    """
    x = x0.reshape((n,1))

    A0 = J(j,x,n)
    v = F(f,x,n)

    A = np.linalg.inv(A0)

    s = -1*np.matmul(A,v)
    x = x + s
    k = 2

    while(k <= N):
        w = v
        v = F(f,x,n)
        y = v - w

        z = -1*np.matmul(A,y)

        p = -1*np.matmul(np.transpose(s),z)

        ut = np.matmul(np.transpose(s),A)

        A = A + (1/p)*np.matmul((s+z),ut)

        s = -1*np.matmul(A,v)
        x = x + s

        if(np.linalg.norm(s) < tol):
            print("The procedure was successful!")
            return x

        k = k + 1

    print("Maximum number of iterations exceeded!")
    return x
```

1.1.1 5.)

(a.) The given system of equations is:

$$x_1(1 - x_1) + 4x_2 - 12 = 0(x_1 - 2)^2 + (2x_2 - 3)^2 - 25 = 0 \tag{1}$$

which has Jacobian given by:

$$J(x) = \begin{pmatrix} 1-2x_1 & 4 \\ 2(x_1-2) & 4(2x_2-3) \end{pmatrix} \tag{2}$$

Performing Broyden's Method with $tol = 10^{-6}$ and $x^{(0)} = (2.5, 4)^T$ we get:

[35]:
```
def f1(x):
    return x[0]*(1-x[0]) + 4*x[1] - 12
def f2(x):
    return (x[0]-2)**2 + (2*x[1]-3)**2 - 25
def j11(x):
    return 1 - 2*x[0]
def j12(x):
    return 4
def j21(x):
    return 2*(x[0]-2)
def j22(x):
    return 4*(2*x[1]-3)

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([2.5,4])

n = 2
N = 100
tol = 10e-6

x = BroydenMethod(n,x0,tol,N)
print(x)
```

```
The procedure was successful!
[[2.54694647]
 [3.98499747]]
```

(b.) Following the same procedure as above we can approximate the solutions for the remaining systems of equations using Broyden's Method.

[30]:
```
def f1(x):
    return 5*(x[0]**2) - (x[1]**2)
def f2(x):
    return x[1] - 0.25*(np.sin(x[0]) + np.cos(x[1]))
def j11(x):
    return 10*x[0]
def j12(x):
    return -2*x[1]
def j21(x):
    return 0.25*np.cos(x[0])
def j22(x):
    return 1 + 0.25*np.sin([x[1]])
```

```
f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([.1,.1])

n = 2
N = 100
tol = 10e-6

x = BroydenMethod(n,x0,tol,N)
print(x)
```

```
The procedure was successful!
[[0.12124195]
 [0.27110513]]
```

(c.)

[37]:
```
def f1(x):
    return 15*x[0] + x[1]**2 - 4*x[2] - 13
def f2(x):
    return x[0]**2 + 10*x[1] - x[2] - 11
def f3(x):
    return x[1]**3 - 25*x[2] + 22
def j11(x):
    return 15
def j12(x):
    return 2*x[1]
def j13(x):
    return -4
def j21(x):
    return 2*x[0]
def j22(x):
    return 10
def j23(x):
    return -1
def j31(x):
    return 0
def j32(x):
    return 3*(x[1]**2)
def j33(x):
    return -25

f = np.array([f1,f2,f3])
j = np.array([[j11,j12,j13],[j21,j22,j23],[j31,j32,j33]])
x0 = np.array([0,0,0])

n = 3
```

```
N = 100
tol = 10e-6

x = BroydenMethod(n,x0,tol,N)
print(x)
```

```
The procedure was successful!
[[1.03640046]
 [1.08570658]
 [0.93119146]]
```

(d.)

[40]:
```
def f1(x):
    return 10*x[0] - 2*x[1]**2 + x[1] - 2*x[2] - 5
def f2(x):
    return 8*(x[1]**2) + 4*(x[2]**2) - 9
def f3(x):
    return 8*x[1]*x[2] + 4
def j11(x):
    return 10
def j12(x):
    return -4*x[1] + 1
def j13(x):
    return -2
def j21(x):
    return 0
def j22(x):
    return 16*x[1]
def j23(x):
    return 8*x[2]
def j31(x):
    return 0
def j32(x):
    return 8*x[2]
def j33(x):
    return 8*x[1]

f = np.array([f1,f2,f3])
j = np.array([[j11,j12,j13],[j21,j22,j23],[j31,j32,j33]])
x0 = np.array([1,-1,1])

n = 3
N = 100
tol = 10e-6

x = BroydenMethod(n,x0,tol,N)
print(x)
```

```
The procedure was successful!
[[ 0.9]
 [-1. ]
 [ 0.5]]
```

1.1.2 Initally I did the wrong problem, so that's what these cells are...

[4]:
```python
def f1(x):
    return 4*(x[0]**2) - 20*x[0] + .25*(x[1]**2) + 8
def f2(x):
    return 0.5*x[0]*(x[1]**2) + 2*x[0] - 5*x[1] + 8
def j11(x):
    return 8*x[0] - 20
def j12(x):
    return 0.5*x[1]
def j21(x):
    return 0.5*x[1]**2 + 2
def j22(x):
    return x[0]*x[1] - 5

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([0,0])

n = 2
N = 2
tol = 10e-6

x = BroydenMethod(n,x0,tol,N)
print(x)
```

```
Maximum number of iterations exceeded!
[[0.47779201]
 [1.92741123]]
```

[5]:
```python
def f1(x):
    return np.sin(4*np.pi*x[0]*x[1]) - 2*x[1] - x[0]
def f2(x):
    return ((4*np.pi-1)/(4*np.pi))*(np.exp(2*x[0])-np.exp(1)) + 4*np.
 ↪exp(1)*(x[1]**2) - 2*np.exp(1)*x[0]
def j11(x):
    return 4*np.pi*x[1]*np.cos(4*np.pi*x[0]*x[1]) - 1
def j12(x):
    return 4*np.pi*x[0]*np.cos(4*np.pi*x[0]*x[1]) - 2
def j21(x):
    return ((4*np.pi-1)/(4*np.pi))*2*np.exp(2*x[0]) - 2*np.exp(1)
def j22(x):
```

```python
    return 8*np.exp(1)*x[1]

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([0,0])

n = 2
tol = 10e-6
N = 2

x = BroydenMethod(n,x0,tol,N)
print(x)
```

```
Maximum number of iterations exceeded!
[[-0.32500698]
 [-0.08035291]]
```

[6]:

```python
def f1(x):
    return 3*(x[0]**2)-(x[1]**2)
def f2(x):
    return 3*x[0]*(x[1]**2)-(x[0]**3)-1
def j11(x):
    return 6*x[0]
def j12(x):
    return 2*x[1]
def j21(x):
    return 3*(x[1]**2)-3*(x[0]**2)
def j22(x):
    return 6*x[0]*x[1]

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([1,1])

n = 2
tol = 10e-6
N = 2

x = BroydenMethod(n,x0,tol,N)
print(x)
```

```
Maximum number of iterations exceeded!
[[0.49266557]
 [0.79785841]]
```

[7]:
```
def f1(x):
    return np.log((x[0]**2)+(x[1]**2))-np.sin(x[0]*x[1])-np.log(2)-np.log(np.pi)
def f2(x):
    return np.exp(x[0]-x[1])+np.cos(x[0]*x[1])
def j11(x):
    return 2*x[0]/((x[0]**2)+(x[1]**2))-x[0]*np.cos(x[0]*x[1])
def j12(x):
    return 2*x[1]/((x[0]**2)+(x[1]**2))-x[1]*np.cos(x[0]*x[1])
def j21(x):
    return np.exp(x[0]-x[1])-x[1]*np.sin(x[0]*x[1])
def j22(x):
    return -1*np.exp(x[0]-x[1])-x[0]*np.sin(x[0]*x[1])

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([2,2])

n = 2
tol = 10e-6
N = 2

x = BroydenMethod(n,x0,tol,N)
print(x)
```

```
Maximum number of iterations exceeded!
[[1.7794999 ]
 [1.74339606]]
```

1.1.3 10.4 Steepest Descent Techniques

[9]:
```
def G(f):
    """
    Computes the function g(x) as defined in the book.
    """
    return np.sum(f**2)

def Grad(J,F):
    """
    Computes the gradient of g(x) using the Jacobian and F.
    """
    n = F.shape[0]
    g = 2*np.matmul(np.transpose(J),F)
    return np.reshape(g,n)
```

[10]:
```
def SteepestDescent(n,x0,tol,N):
    """
    Implements the Gradient Descent Algorithm according to the algorithm in the␣
 ↪book.
```

```
    """
    k = 1
    x = x0
    while(k <= N):
        #print(k)

        g1 = G(F(f,x,n))
        #print("g1 = ", g1)

        z = Grad(J(j,x,n),F(f,x,n))

        z0 = np.linalg.norm(z)

        if(z0 == 0):
            print("Zero gradient!")
            return x

        z = z/z0
        #print("z = ", z)

        a1 = 0
        a3 = 1

        g3 = G(F(f,x-a3*z,n))
        #print("g3 = ", g3)

        while(g3 >= g1):
            a3 = 0.5*a3
            g3 = G(F(f,x-a3*z,n))

            if(a3 < tol/2):
                print("No likely imporvement...")
                return x

        a2 = 0.5*a3

        g2 = G(F(f,x-a2*z,n))
        #print("g2 = ", g2)


        h1 = (g2-g1)/a2
        h2 = (g3-g2)/(a3-a2)
        h3 = (h2-h1)/a3

        #print("h1 = ", h1)
        #print("h2 = ", h2)
        #print("h3 = ", h3)
```

```
        a0 = 0.5*(a2-h1/h3)
        g0 = G(F(f,x-a0*z,n))

        #print("a0 = ", a0)
        #print("g0 = ", g0)


        if(g0 <= g3):
            a = a0
            g = g0
        else:
            a = a3
            g = g3

        x = x -a*z

        if(np.abs(g-g1) < tol):
            print("The procedure was successful!")
            return x

        k = k+1

    print("Maximum number of iterations exceeded!")
    return x
```

1.)

(a.)

[209]:
```
def f1(x):
    return 4*(x[0]**2) - 20*x[0] + 0.25*(x[1]**2) + 8
def f2(x):
    return 0.5*x[0]*(x[1]**2) + 2*x[0] - 5*x[1] + 8
def j11(x):
    return 8*x[0] - 20
def j12(x):
    return 0.5*x[1]
def j21(x):
    return 0.5*(x[1]**2) + 2
def j22(x):
    return x[0]*x[1] - 5



f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
```

```
x0 = np.array([0,0])

n = 2
N = 20
tol = .05

x = SteepestDescent(n,x0,tol,N)
print(x)
```

```
No likely imporvement...
[0.48036371 1.93817088]
```

(b.)

[16]:
```
def f1(x):
    return 3*(x[0]**2)-(x[1]**2)
def f2(x):
    return 3*x[0]*(x[1]**2)-(x[0]**3)-1
def j11(x):
    return 6*x[0]
def j12(x):
    return 2*x[1]
def j21(x):
    return 3*(x[1]**2)-3*(x[0]**2)
def j22(x):
    return 6*x[0]*x[1]

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([1,1])

n = 2
N = 20
tol = .05

x = SteepestDescent(n,x0,tol,N)
print(x)
```

```
The procedure was successful!
[0.49981677 0.8658462 ]
```

(c.)

[211]:
```
def f1(x):
    return np.log((x[0]**2)+(x[1]**2))-np.sin(x[0]*x[1])-np.log(2)-np.log(np.pi)
def f2(x):
    return np.exp(x[0]-x[1])+np.cos(x[0]*x[1])
def j11(x):
```

```python
    return 2*x[0]/((x[0]**2)+(x[1]**2))-x[0]*np.cos(x[0]*x[1])
def j12(x):
    return 2*x[1]/((x[0]**2)+(x[1]**2))-x[1]*np.cos(x[0]*x[1])
def j21(x):
    return np.exp(x[0]-x[1])-x[1]*np.sin(x[0]*x[1])
def j22(x):
    return -1*np.exp(x[0]-x[1])-x[0]*np.sin(x[0]*x[1])

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([2,2])

n = 2
N = 20
tol = .05

x = SteepestDescent(n,x0,tol,N)
print(x)
```

```
The procedure was successful!
[1.76475919 1.79229801]
```

(d.)

[212]:
```python
def f1(x):
    return np.sin(4*np.pi*x[0]*x[1]) - 2*x[1] - x[0]
def f2(x):
    return ((4*np.pi-1)/(4*np.pi))*(np.exp(2*x[0])-np.exp(1)) + 4*np.
 ↪exp(1)*(x[1]**2) - 2*np.exp(1)*x[0]
def j11(x):
    return 4*np.pi*x[1]*np.cos(4*np.pi*x[0]*x[1]) - 1
def j12(x):
    return 4*np.pi*x[0]*np.cos(4*np.pi*x[0]*x[1]) - 2
def j21(x):
    return ((4*np.pi-1)/(4*np.pi))*2*np.exp(2*x[0]) - 2*np.exp(1)
def j22(x):
    return 8*np.exp(1)*x[1]

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([0,0])

n = 2
N = 20
tol = .05

x = SteepestDescent(n,x0,tol,N)
print(x)
```

```
No likely imporvement...
[-0.36100921  0.05788368]
```

1.1.4 10.5 Homotopy and Continuation Methods

[93]:
```python
def ContinuationMethod(n,x0,N,method = "RK4"):
    """
    Implements the Continuation Method according to the algorithm in the book.
    Has the option to use either RK4 of Euler's Method to solve the system of␣
↪ODEs.
    """
    x = x0
    h = 1.0/N
    #print(h)
    b = -1*h*F(f,x,n)
    #print(b)

    if(method == "RK4"):
        for i in range(N):
            A = J(j,x,n)
            #print("A = ", A)
            k1 = np.reshape(np.linalg.solve(A,b),n)
            #print("k1 = ", k1)

            A = J(j,(x+0.5*k1),n)
            #print("A = ", A)
            k2 = np.reshape(np.linalg.solve(A,b),n)
            #print("k2 = ", k2)


            A = J(j,x+0.5*k2,n)
            #print("A = ", A)
            k3 = np.reshape(np.linalg.solve(A,b),n)
            #print("k3 = ", k3)


            A = J(j,x+k3,n)
            #print("A = ", A)
            k4 = np.reshape(np.linalg.solve(A,b),n)
            #print("k4 = ", k4)


            x = x + (k1+2*k2+2*k3+k4)/6
            #print("iteration ", i, ": ", x)

        return x
```

```
    if(method == "Euler"):
        for i in range(N):
            A = J(j,x,n)
            k = np.reshape(np.linalg.solve(A,b),n)
            x = x + k

        return x
```

1.)

[107]:
```
def f1(x):
    return (x[0]**2) - (x[1]**2) + 2*x[1]
def f2(x):
    return 2*x[0] + (x[1]**2) - 6
def j11(x):
    return 2*x[0]
def j12(x):
    return -2*x[1] + 2
def j21(x):
    return 2
def j22(x):
    return 2*x[1]

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0_list = np.array([[0,0],[1,1],[3,-2]])

#print(F(f,x0,n))
#print(J(j,x0,n))
```

[119]:
```
n = 2
N = 2

for i in range(3):
    x0 = x0_list[i]
    x = ContinuationMethod(n,x0,N,"Euler")
    print("x0 = ", x0)
    print("x = ", x,"\n")
```

```
x0 =  [0 0]
x =  [ 3.   -2.25]

x0 =  [1 1]
x =  [0.42105263 2.61842105]

x0 =  [ 3 -2]
x =  [ 2.17310981 -1.36277308]
```

[118]:
```python
n = 2
N = 1

for i in range(3):
    x0 = x0_list[i]
    x = ContinuationMethod(n,x0,N,"RK4")
    print("x0 =", x0)
    print("x = ", x,"\n")
```

```
x0 = [0 0]
x =  [ 2.30398796 -2.00109948]

x0 = [1 1]
x =  [0.59709702 2.25796842]

x0 = [ 3 -2]
x =  [ 2.10944599 -1.33456326]
```