APPM 4600 Lab 6

3 October 2024

The code for this lab can be seen on github here and is included below.

1 Prelab: Finite Differences

1. The implementation of the forward and central difference is given in the code below. The approximate derivatives for $f(x) = \cos x$ at $x = \pi/2$ are listed below in Table 1 and 2 below.

h	Forward Difference at $x = \frac{\pi}{2}$
0.01	-0.9999833334166673
0.005	-0.9999958333385205
0.0025	-0.9999989583336375
0.00125	-0.9999997395833322
0.000625	-0.999999934895991
0.0003125	-0.9999999837237595
0.00015625	-0.9999999959315011
7.8125e-05	-0.9999999989818379
3.90625e-05	-0.9999999997447773
1.953125e-05	-0.9999999999355124

Table 1: Forward differences of $f(x = \frac{\pi}{2})$.

h	Central Difference at $x = \frac{\pi}{2}$
0.01	-0.9999833334166673
0.005	-0.9999958333385205
0.0025	-0.9999989583336376
0.00125	-0.9999997395833324
0.000625	-0.9999999348959908
0.0003125	-0.9999999837237594
0.00015625	-0.9999999959315011
7.8125e-05	-0.9999999989818379
3.90625e-05	-0.9999999997447773
1.953125e-05	-0.9999999999355121

Table 2: Central differences of $f(x = \frac{\pi}{2})$.

2. Both methods have numerical order $\alpha \approx 1$, as expected.

2 Slacker Newton

We choose to update the Jacobian whenever the current point is sufficiently far from the last point at which we computed the Jacobian, that is, where

$$||x_0 - x_n|| > t,$$

where t is some tolerance. The implementation is provided in the code at the end of the document.

We choose $t = 10^{-5}$ and find the approximate root

$$x \approx 0.99860694, \ y \approx -0.10553049.$$

This is the same as is found by lazy newton, except that slacker newton takes 3 iterations to reach 10^{-10} tolerance whereas lazy newton takes 7 iterations.

3 Approximate Jacobian

We approximate the Jacobian with forward differences, that is,

$$J(x,y) \approx \begin{bmatrix} \frac{f(x+h,y)-f(x,y)}{h} & \frac{f(x,y+h)-f(x,y)}{h} \\ \frac{g(x+h,y)-g(x,y)}{h} & \frac{g(x,y+h)-g(x,y)}{h} \end{bmatrix}.$$

The implementation is provided in the code at the end of the document.

We pick $h = 10^{-8}$ and find the approximate root

$$x \approx 0.99860694,\ y \approx -0.10553049,$$

which is found in 3 iterations.

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

def forwardDifference(f, s, h):
    return (f(s + h) - f(s)) / h

def centralDifference(f, s, h):
    return (f(s+h) - f(s-h))/(2*h)

def compute_order(x, xstar):
    diff1 = np.abs(x[1::] - xstar)
    diff2 = np.abs(x[0:-1]-xstar)
    fit = np.polyfit(np.log(diff2.flatten()), np.log(diff1.flatten()),1)
    print('the order of the equation is')
    print("lambda = " + str(np.exp(fit[1])))
    print("alpha = " + str(fit[0]))

    alpha = fit[0]
    l = np.exp(fit[1])

    return [fit, alpha, l]

def question1_1():
    h = 0.01*2. ** (-np.arange(0, 10))
    s = math.pi / 2 * np.ones(10)
    f = lambda x: np.cos(x)

    fd = forwardDifference(f, s, h)
    cd = centralDifference(f, s, h)
    compute_order(fd, -1.0)
    compute_order(cd, -1.0)

    for i in range(len(fd)):
        print(h[i], "&", fd[i], "\\\\")

    for i in range(len(cd)):
        print(h[i], "&", cd[i], "\\\\")

question1_1()

def evalF(x):
    F = np.zeros(2)
    F[0] = 4.0*x[0]**2 + x[1]**2 - 4
    F[1] = x[0] + x[1] - np.sin(x[0] - x[1])
```

```
    return F

def evalJ(x):
    return np.array([[8*x[0], 2*x[1]],
                     [1-np.cos(x[0] - x[1]), 1+np.cos(x[0] - x[1])]])

def approxJ(x, h):
    # approximate jacobian with forward difference
    f0 = evalF(x)
    f1 = evalF(x+h*np.array([1, 0]))
    f2 = evalF(x+h*np.array([0, 1]))

    return np.array([[(f1[0] - f0[0]) / h, (f2[0] - f0[0]) / h],
                     [(f1[1] - f0[1]) / h, (f2[1] - f0[1]) / h]])

def NewtonApprox(x0,tol,h,Nmax):

    ''' inputs: x0 = initial guess, tol = tolerance, Nmax = max its'''
    ''' Outputs: xstar= approx root, ier = error message, its = num its'''

    for its in range(Nmax):
       J = approxJ(x0, h)
       Jinv = inv(J)
       F = evalF(x0)

       x1 = x0 - Jinv.dot(F)

       if (norm(x1-x0) < tol):
           xstar = x1
           ier =0
           return[xstar, ier, its]

       x0 = x1

    xstar = x1
    ier = 1
    return[xstar,ier,its]

def LazyNewton(x0,tol,Nmax):

    ''' Lazy Newton = use only the inverse of the Jacobian for initial guess'''
    ''' inputs: x0 = initial guess, tol = tolerance, Nmax = max its'''
    ''' Outputs: xstar= approx root, ier = error message, its = num its'''

    J = evalJ(x0)
    Jinv = inv(J)
    for its in range(Nmax):

       F = evalF(x0)
       x1 = x0 - Jinv.dot(F)

       if (norm(x1-x0) < tol):
           xstar = x1
           ier =0
           return[xstar, ier,its]

       x0 = x1
```

```
    xstar = x1
    ier = 1
    return[xstar,ier,its]

def SlackerNewton(x0, tol, update_tol, Nmax):
    J = evalJ(x0)
    Jinv = inv(J)
    x0_last_eval = x0
    for its in range(Nmax):

        F = evalF(x0)
        # update Jinv if we moved sufficiently far from the starting point
        if norm(x0_last_eval - x0) > update_tol:
            x0_last_eval = x0
            Jinv = inv(evalJ(x0))

        x1 = x0 - Jinv.dot(F)

        if (norm(x1-x0) < tol):
            xstar = x1
            ier =0
            return[xstar, ier,its]

        x0 = x1

    xstar = x1
    ier = 1
    return[xstar,ier,its]

def question3_2():
    print("Lazy Newton")
    [xstar, ier, its] = LazyNewton(np.array([1, 0]), 1e-10, 100)
    print("xstar=", xstar, "ier=", ier, "iters=", its)

    print("Slacker Newton")
    [xstar, ier, its] = SlackerNewton(np.array([1, 0]), 1e-10, 1e-4, 100)
    print("xstar=", xstar, "ier=", ier, "iters=", its)

    print("Approx Newton")
    [xstar, ier, its] = NewtonApprox(np.array([1, 0]), 1e-10, 1e-8, 100)
    print("xstar=", xstar, "ier=", ier, "iters=", its)

question3_2()
```