HW6

December 6, 2022

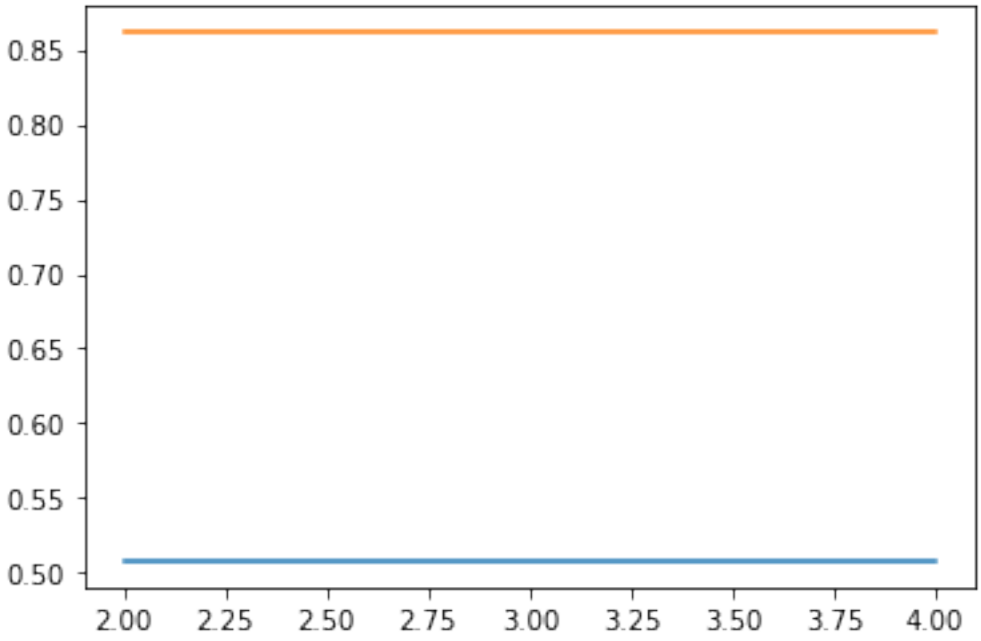
[2]:
```
import scipy.optimize
def fun(variables) :
    (x1,x2)= variables
    eqn_1 = (x1**3) - (x2**3) + x1
    eqn_2 = (x1**2) + (x2**2) - 1
    return [eqn_1,eqn_2]
result = scipy.optimize.fsolve(fun, (0, 0))
print(result)
```

```
[-0.507992   -0.86136179]
```

[3]:
```
# import scipy.optimize
import matplotlib.pyplot as plt
import numpy as np
def function(z,*args):
  x1, x2 = z
  M= args[0]
  return ((x1**3) - (x2**3) + x1,(x1**2) + (x2**2) - 1)
M= np.linspace(2,4,3)
X = []
Y =[]
for a in M:
  x1,x2 = scipy.optimize.fsolve(function,(6.0, 6.0) ,args=(a))
  X.append(x1)
  Y.append(x2)
print(x1,x2)
plt.plot(M,X)
plt.plot(M,Y)
```

```
0.5079920004084001 0.861361786662125
```

[3]:
```
[<matplotlib.lines.Line2D at 0x7f0cf0721900>]
```



[5]:
```
from IPython.display import Image

Image(url="Q3.jpg", width=500, height=500)
```

[5]: <IPython.core.display.Image object>

[6]:
```
def iter_newton(X,function,jacobian,imax = 1e6,tol = 1e-5):
    for i in range(int(imax)):
        J = jacobian(X) # calculate jacobian J = df(X)/dY(X)
        Y = function(X) # calculate function Y = f(X)
        dX = np.linalg.solve(J,Y) # solve for increment from JdX = Y
        X -= dX # step X by dX
        if np.linalg.norm(dX)<tol: # break if converged
            print('converged.')
            break
    return X
```

[ ]:
```
def iter_newton(X,function,jacobian,imax = 1e6,tol = 1e-5):
```

https://stackoverflow.com/questions/52020775/solving-a-non-linear-system-of-equations-in-python-using-newtons-method Cant Remember the link for the first functions but I am not to take credit for them