Name: Amr Mohamed Gaber
ID : 4818

Assignment 2

We will use a class to define the two methods (Powell's method & Steepest Descent method) because they both will solve the same problem.

Our Class Function Code:

```
import sympy as sp
import numpy as np

class opt_tech():
    """
    This is a class for the two main classes of optimization techniques for solving
    n-D unconstraind minimization problems numerically

    ...
    Attributes
    ----------
    fi : sympy class obj
        the objective function you want to find its optimum point
    itr_no : int
        the number of iterations you will go through (default 1)
    epsilon : float
        used to help find a suitable search direction Si
    """

    def __init__(self, fi, itr_no=1, epsilon=0.01):
        """
        The constructor for opt_tech class.

        ...

        Parameters
        ----------
        fi : sympy class obj
            the objective function you want to find its optimum point
        itr_no : int
            the number of iterations you will go through (default 1)
        epsilon : float
            used to help find a suitable search direction Si
        """
        self.fi = fi
        self.itr_no = itr_no
        self.epsilon = epsilon
        self.x = sp.Symbol("x")
        self.y = sp.Symbol("y")
        self.lamda = sp.Symbol("lamda")
```

Name: Amr Mohamed Gaber
ID : 4818

Initializing our Class:

```
# the objective function
x = sp.Symbol("x")
y = sp.Symbol("y")
fi = x - y + 2*x**2 + 2*x*y + y**2

# the number of iterations
itr_no = 5

# initializing our optimization techniques class
opt_tech = opt_tech(fi, itr_no)
```

Name: Amr Mohamed Gaber
ID : 4818

Powell's Method:

It's an iterative direct search optimization method, which requires an initial guess X1 vector for the optimum point.

Required Input:

```
# our initial guess
X = np.array([0, 0])

# Calling our Powell's Method
opt_tech.powells_method(X)
```

Powell's method code (part 1):

```
def powells_method(self, X=np.array([0, 0])):
    """
    A direct search Optimization method for solving n-D unconstrained problem.

    ...

    Parameters
    ----------
    X : np.array (vector)
        the initial guess needed to compute the optimum point (default for 2 variables [0, 0])
    """

    print()
    print()
    print("Powelll's Method :")
    print("_________________")

    # initializing variables for our loop
    Z = X

    # find the optimal value for the given number of iterations
    for i in range(self.itr_no):
        # for Cycle #1: S2, S1, S2;
        # we will need to manually initialize S1 & S2
        if i < 3:
            if i % 2:
                S = np.array([1, 0])  # S1
            else:
                S = np.array([0, 1])  # S2
        # otherwise, we can calculate it
        else:
            S = X - Z
        print("S{} = {}".format(i+1, S))
```

Name: Amr Mohamed Gaber
ID : 4818

Powell's method code (part 2):

```
# calculate fi(x, y) at point X
fi = self.fi.subs({self.x: X[0], self.y: X[1]})

# calculate fi_+ve & fi_-ve to find the suitable search direction
fi_postive = self.fi.subs(
    {self.x: X[0]+self.epsilon*S[0], self.y: X[1]+self.epsilon*S[1]})
fi_negative = self.fi.subs(
    {self.x: X[0]-self.epsilon*S[0], self.y: X[1]-self.epsilon*S[1]})

print("f{}={}".format(i+1, fi))
print("f{i}_+ve={:.{prec}f},  f{i}_-ve={:.{prec}f}".format(fi_postive,
                                                           fi_negative, i=i+1, prec=4))

# check if we reached our optimum point
if fi_postive > fi and fi_negative > fi:
    print("Reached Optimum Point at X{} = {}".format(i+1, X))
    break
# check which direction will make objective function decrease
elif fi_postive > fi:
    S *= -1
    print("f{} decrease along -ve S{}".format(i+1, i+1))
else:
    print("f{} decrease along +ve S{}".format(i+1, i+1))

# to find the approx step length λi* along dir Si
# sub λ in the objective function
fi_lamda = self.fi.subs(
    {self.x: X[0]+self.lamda*S[0], self.y: X[1]+self.lamda*S[1]})
print("f{}_lamda = {}".format(i+1, fi_lamda))

# get the derivative to find the appropriate step length λ*
dfi = sp.diff(fi_lamda)
print("df/d1 = {} = 0".format(dfi))
lamda = sp.solve(dfi)
print(f"λ* = {lamda}")
# calcualte the new approx point
X = X + lamda * S
print("X{} = X = {}".format(i+2, X))
```

```
# set our Z (X2) point
if i == 0:
    Z = X
    print(f"X{i+2} = X = Z = {X}")
print()
print()
```

Name: Amr Mohamed Gaber
ID : 4818

Powell’s method output:

```
Powelll's Method :
_________________
S1 = [0 1]
f1=0
f1_+ve=-0.0099,  f1_-ve=0.0101
f1 decrease along +ve S1
f1_lamda = lamda**2 - lamda
df/d1 = 2*lamda - 1 = 0
λ* = [1/2]
X2 = X = [0 1/2]
X2 = X = Z = [0 1/2]


S2 = [1 0]
f2=-1/4
f2_+ve=-0.2298,  f2_-ve=-0.2698
f2 decrease along -ve S2
f2_lamda = 2*lamda**2 - 2*lamda - 1/4
df/d1 = 4*lamda - 2 = 0
λ* = [1/2]
X3 = X = [-1/2 1/2]


S3 = [0 1]
f3=-3/4
f3_+ve=-0.7599,  f3_-ve=-0.7399
f3 decrease along +ve S3
f3_lamda = -2*lamda + (lamda + 1/2)**2 - 1
df/d1 = 2*lamda - 1 = 0
λ* = [1/2]
X4 = X = [-1/2 1]


S4 = [-1/2 1/2]
f4=-1
f4_+ve=-1.0050,  f4_-ve=-0.9950
f4 decrease along +ve S4
f4_lamda = -lamda + 2*(-lamda/2 - 1/2)**2 + 2*(-lamda/2 - 1/2)*(lamda/2 + 1) + (lamda/2 + 1)**2 - 3/2
df/d1 = lamda/2 - 1/2 = 0
λ* = [1]
X5 = X = [-1 3/2]
```

```
S5 = [-1 1]
f5=-5/4
f5_+ve=-1.2499,  f5_-ve=-1.2499
Reached Optimum Point at X5 = [-1 3/2]
_________________
```

Name: Amr Mohamed Gaber
ID : 4818

Steepest Descent Method:

It’s an iterative (Gradient) Descent search optimization method, which requires an initial guess X1 vector for the optimum point.

We will use the same objective function from Powell's method and same initial guess but we will change the iteration number.

Required Input:

```
# the number of iterations
opt_tech.itr_no = 3

# Calling our Steepest Descent Method
opt_tech.steepest_descent(X)
```

Name: Amr Mohamed Gaber
ID : 4818

Steepest Descent method code (part 1):

```
def steepest_descent(self, X=np.array([0, 0])):
    """
    A (Gradient) Descent search Optimization method for solving n-D unconstrained problem.

    ...

    Parameters
    ----------
    X : np.array (vector)
        the initial guess needed to compute the optimum point (default for 2 variables [0, 0])
    """

    print()
    print()
    print("Steepest Descent Method :")
    print("_______________________")
    print()

    # find the gradient (1st order partial derivative) for our objective function
    gradient_f = [sp.diff(self.fi, self.x), sp.diff(self.fi, self.y)]
    print(f"∇f = {gradient_f}")
    print()

    # find the optimal value for the given number of iterations
    for i in range(self.itr_no):
        # calculate the gradient values at point X
        dx = gradient_f[0].subs({self.x: X[0], self.y: X[1]})
        dy = gradient_f[1].subs({self.x: X[0], self.y: X[1]})
        neg_grad_fi = -1 * np.array([dx, dy])
        print(f"-∇f{i+1} = {neg_grad_fi}")

        # if -∇fi = [0, 0] then we reached out optimal point
        if (neg_grad_fi == np.zeros(2)).all():
            print("Reached Optimum Point at X{} = {}".format(i+1, X))
            break
```

```
        # to find the approx step length λi* along dir Si
        # sub λ in the objective function
        fi_lamda = self.fi.subs(
            {self.x: X[0]+self.lamda*neg_grad_fi[0], self.y: X[1]+self.lamda*neg_grad_fi[1]})
        print("f{}_lamda = {}".format(i+1, fi_lamda))

        # get the derivative to find the appropriate step length
        dfi = sp.diff(fi_lamda)
        print("df/dλ = {} = 0".format(dfi))
        lamda = sp.solve(dfi)
        print(f"λ* = {lamda}")

        # calcualte the new approx point
        X = X + lamda * neg_grad_fi
        print("X{} = {}".format(i+2, X))
        print()
    print("_________________")
```

Name: Amr Mohamed Gaber
ID : 4818

Steepest Descent method output:

```
Steepest Descent Method :
_______________________

∇f = [4*x + 2*y + 1, 2*x + 2*y - 1]

-∇f1 = [-1 1]
f1_lamda = lamda**2 - 2*lamda
df/dλ = 2*lamda - 2 = 0
λ* = [1]
X2 = [-1 1]

-∇f2 = [1 1]
f2_lamda = 2*(lamda - 1)**2 + 2*(lamda - 1)*(lamda + 1) + (lamda + 1)**2 - 2
df/dλ = 10*lamda - 2 = 0
λ* = [1/5]
X3 = [-4/5 6/5]

-∇f3 = [-1/5 1/5]
f3_lamda = -2*lamda/5 + 2*(-lamda/5 - 4/5)**2 + 2*(-lamda/5 - 4/5)*(lamda/5 + 6/5) + (lamda/5 + 6/5)**2 - 2
df/dλ = 2*lamda/25 - 2/25 = 0
λ* = [1]
X4 = [-1 7/5]
```