Name: Amr Mohamed Gaber
ID : 4818

Assignment 1

We will use a class to define the two methods (Newton's method & Golden Section method) because they both will solve the same problem.

Our Class Function Code:

```

import sympy as sp


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

    ...

    Attributes
    ----------
    fx : 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)
    prec : int
        the number of decimal places after the decimal point (default 3)
    """
```

To initialize our Class we will require this parameters:

```
def __init__(self, fx, itr_no=1, prec=3):
    """
    The constructor for opt_tech class.

    ...

    Parameters
    ----------
    fx : 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)
    prec : int
        the number of decimal places after the decimal point (default 3)
    """

    self.fx = fx
    self.itr_no = itr_no
    self.prec = prec
    self.x = sp.Symbol("x")
```

Name: Amr Mohamed Gaber
ID : 4818

Initializing our Class:

```python
# the objective function
fx = 2*sp.sin(sp.Symbol("x")) - .1*(sp.Symbol("x")**2)
# the precision value (number of decimal places)
prec = 3
# the number of iterations
itr_no = 3

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

Name: Amr Mohamed Gaber
ID : 4818

Newton’s Method:

It’s an iterative direct root finding technique, which requires an initial guess (xo) of the optimum point.

Required Input:

```python

# our initial guess
xo = 2.5

# Calling our Newton's Method
opt_tech.newtons_method(xo)

```

Newton’s method code (part 1):

```python

    def newtons_method(self, xo=0):
        """
        A direct root finding Optimization Technique for 1-D constrained problem.

        ...

        Parameters
        ----------
        xo : int
            the initial guess needed to compute the optimum point (default 0)
        """

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

        # to get the 1st derivative for the objective function
        df = sp.diff(self.fx)
        # to get the 2nd derivative for the objective function
        ddf = sp.diff(df)
        # set the initial value to the initial guess
        xi_old = xo
```

Name: Amr Mohamed Gaber
ID : 4818

Newton's method code (part 2):

```python
        # find the optimal value for the given number of iterations
        for i in range(self.itr_no):
            # calculate the imporved approximation of xi_old
            xi_new = xi_old - df.subs(self.x, xi_old) / \
                ddf.subs(self.x, xi_old)
            # calculate the relative error = |xi - xi+1| / |xi+1|
            relative_error = abs(xi_new-xi_old)/abs(xi_new)
            print("x{} = {:.{prec}f}, relative error = {:.{prec}f}".format(
                i+1, xi_new, relative_error, prec=self.prec))
            # update the old approximation to be equal the new one
            xi_old = xi_new
        print()

        # calculate the 1st derivative value with our final approximation
        f_dash = df.subs(self.x, xi_old)
        # find if our final approximation is close to zero (accepted) or not
        note = "(acceptably small)" if abs(f_dash) <= 0.01 else ""
        if note:
            print("f'({:.{prec}f}) = {:.{prec}f}  {}".format(
                xi_old, f_dash, note, prec=self.prec))
            print()
            print("fmax = f({:.{prec}f}) = {:.{prec}f}".format(
                xi_old, self.fx.subs(self.x, xi_old), prec=self.prec))
        else:
            print("{:.{prec}f} is rejected".format(xi_old, prec=self.prec))
```

Newton's method output:

```
Newton's Method :
_________________________


x1 = 0.995, relative error = 1.512
x2 = 1.469, relative error = 0.323
x3 = 1.428, relative error = 0.029

f'(1.428) = -0.000  (acceptably small)

fmax = f(1.428) = 1.776
_________________________
```

Name: Amr Mohamed Gaber
ID : 4818

Golden Section Method:

It’s an iterative Elimination Technique, which requires an interval [xl, xu] where the objective function is unimodal.

We will use the same objective function from Newton's method but we will change the decimal precision point and the iteration number.

Required Input:

```
# the lower bound
xl = 0
# the upper bound
xu = 4
# the precision value (number of decimal places)
opt_tech.prec = 4
# the number of iterations
opt_tech.itr_no = 8

# Calling our Golden Section Method
opt_tech.golden_section(xl, xu)
```

Name: Amr Mohamed Gaber
ID : 4818

Golden Section method code (part 1):

```
def golden_section(self, xl, xu):
    """
    An Elimination Optimization Technique for 1-D constrained problem.

    ...

    Parameters
    ----------
    xl : int
        the lower bound for the unimodal objective function interval
    xu : int
        the upper bound for the unimodal objective function interval
    """

    print()
    print()
    print("Golden Section Method :")
    print("_______________________")
    print()
    # creating the table format for printing
    table_format = ["i", "xl", "x2", "x1", "xu",
                    "fx2", "fx1", "xu-xl", "errBound"]
    print("|{:>{tab}}|{:>{tab}}|{:>{tab}}|{:>{tab}}|{:>{tab}}|{:>{tab}}|{:>{tab}}|{:>{tab}}|{:>{tab}}|".format(
        *table_format, tab=2*self.prec))
```

Golden Section method code (part 2):

We can calculate all the lengths and error bounds for each iteration before doing our loop.

```
# we can calculate all the lengths directly using the formula
# xu(i)-xl(i) = ( xu(0)-xl(0) ) * .618^(i-1)
length = [round((xu-xl)*.618**i, self.prec)
          for i in range(self.itr_no)]

# we can calculate all the lengths directly using the formula
# xu(i)-xl(i) = ( xu(0)-xl(0) ) * .618^(i+1)
error_bound = [round((xu-xl)*.618**(i+2), self.prec)
               for i in range(self.itr_no)]
```

Name: Amr Mohamed Gaber
ID : 4818

Golden Section method code (part 3):

```python
# find the optimal value for the given number of iterations
for i in range(self.itr_no):
    # calculate our comparison points and their values
    x2 = xu - .618*(xu-xl)
    x1 = xl + .618*(xu-xl)

    fx2 = self.fx.subs(self.x, x2)
    fx1 = self.fx.subs(self.x, x1)

    # creating the data format for printing
    data = [i+1, xl, x2, x1, xu, fx2, fx1, length[i], error_bound[i]]
    print("|{:>{tab}}|{:>{tab}.{prec}f}|{:>{tab}.{prec}f}|{:>{tab}.{prec}f}|{:>{tab
        *data, tab=2*self.prec, prec=self.prec))

    # find which interval to eliminate
    # if f(x1) < f(x2), then [xl, x2] doesn't contain the maximum point
    if fx1 > fx2:
        xl, x2 = x2, x1
    # if f(x2) > f(x1), then [x1, xu] doesn't contain the maximum point
    # in case, f(x1) == f(x2), then both [xl, x2] and [x1, xu] doesn't
    # contain the maximum point so we can safely do either options
    # without losing the maximum point
    else:
        xu, x1 = x1, x2
print("                 ")
```

Golden Section method output:

```
Golden Section Method :

_______________________

|       i|      xl|      x2|      x1|      xu|     fx2|     fx1|   xu-xl|errBound|
|       1|  0.0000|  1.5280|  2.4720|  4.0000|  1.7647|  0.6303|  4.0000|  1.5277|
|       2|  0.0000|  0.9443|  1.5277|  2.4720|  1.5310|  1.7648|  2.4720|  0.9441|
|       3|  0.9443|  1.5279|  1.8884|  2.4720|  1.7647|  1.5433|  1.5277|  0.5835|
|       4|  0.9443|  1.3050|  1.5278|  1.8884|  1.7595|  1.7647|  0.9441|  0.3606|
|       5|  1.3050|  1.5278|  1.6655|  1.8884|  1.7647|  1.7136|  0.5835|  0.2228|
|       6|  1.3050|  1.4427|  1.5278|  1.6655|  1.7755|  1.7647|  0.3606|  0.1377|
|       7|  1.3050|  1.3901|  1.4427|  1.5278|  1.7742|  1.7755|  0.2228|  0.0851|
|       8|  1.3901|  1.4427|  1.4752|  1.5278|  1.7755|  1.7732|  0.1377|  0.0526|

_________________
```