Successive Quadratic Optimization Writeup

This writeup with step through the code used to create a sequential quadractic approprimation for minimizing an equation

Complete code can be found here link to code:

Equation Definition:

- to utilize simply pass the x-variable to the function. Function will return y value.

```
def obj(x):
return 5 - x + 0.45 * x**2 -0.08**3 +0.005*x**4
```

Next Guess Function:

- After you complete the first iteration you calculate x*, you need to define new values for x1,x2,or x3 to complete the next ierations. The text in Engineering Optimization offers several options for generating which value to swap out for x*. I chose to utilize the two points which are closest to x* value, replacing the value that is furthest away with x*.
- The code below determines which value is furthest from x*, and based on the result provides a series of options for selecting the next x1,x2,x3 set.

```
def nextguess(x1, x2, x3, xstar):
    x = np.matrix([[x1],[x2],[x3]])
    max_pos = np.argmax(np.asarray(abs(xstar[0] - x)))

    if max_pos == 0:
        return x2,x3, xstar[0]
    if max_pos == 1:
        return x1,xstar[0],x3
    if max_pos == 2:
return xstar[0],x1,x2
```

Initial Guesses:

- see below for the initial guesses for x1,x2,x3
- I also initialize an empty array for xstar, and eq. The intent is to append these arrays as I iterate. Tracking these values lets me analyze how the algorithm is progressing after the fact.

```
x1 = 25
x2 = 50
x3 = 75
```

```
xstar=[]
eq = []
```

Matrix Vector for F and X

- F is the result for the fuction for x1,x2,x3 in matrix form
- X is the matrix result for the parabolic equation.
- Matrix multiplication is then performed on the inverse of X and F

```
F = np.matrix([[obj(x1)],
               [obj(x2)],
               [obj(x3)]])

X = np.matrix([[x1**2, x1, 1],
               [x2**2, x2, 1],
               [x3**2, x3, 1]])

C = np.matmul(X.I,F)
```

Matrix Vector Results for initial guesses:

```
F = np.matrix([[obj(x1)],
               [obj(x2)],
               [obj(x3)]])
F
Out[173]:
matrix([[  2214.374488],
        [ 32329.999488],
        [160664.374488]])

X = np.matrix([[x1 ** 2, x1, 1],
               [x2 ** 2, x2, 1],
               [x3 ** 2, x3, 1]])
X
Out[175]:
matrix([[ 625,   25,    1],
        [2500,   50,    1],
        [5625,   75,    1]])

C = np.matmul(X.I, F)
C
Out[177]:
matrix([[   78.575   ],
        [-4688.5     ],
        [70317.499488]])
```

Xstar and next guess:

- Previously "C" was used to return a,b,c variables for the quadratic ax^2+bx+C
- Below I use the relationship that -b/2c to determine what x* is.
- After finding x* I use the nextguess function to assign new x1,x2,x3 values.

```
xstar.insert(0,-C.item(1)/2/C.item(0))

x1,x2,x3 = nextguess(x1,x2,x3,xstar)
```

Stop Criteria

- First the stop criteria ensures the program has interated >1 time

- Second it looks to determine if the difference between the current and laster iteration of xstar produced a signifiantly different value. If the difference between the two values is >0.5 it continues to iterate.

```
    if len(xstar) > 1:
        if abs(obj(xstar[0]) - obj(xstar[1])) < 0.5:
          break
```

Full Implementation of Iteration loop:

- below is the code in the ieration loop.
- this takes the above steps and combines them so they are visible in one frame

```
x1 = 25
x2 = 50
x3 = 75


xstar=[]
eq = []


for i in range (1,20):


    F = np.matrix([[obj(x1)],
                   [obj(x2)],
                   [obj(x3)]])

    X = np.matrix([[x1**2, x1, 1],
                   [x2**2, x2, 1],
                   [x3**2, x3, 1]])

    C = np.matmul(X.I,F)

    eq.insert(0,C)

    xstar.insert(0,C.item(1)/2/C.item(0))

    x1,x2,x3 = nextguess(x1,x2,x3,xstar)

    if len(xstar) > 1:
        if abs(obj(xstar[0]) - obj(xstar[1])) < 0.1:
          break
```

Results

- with stop criteria of <0.1 the loop iterated 13 times util the exit criteria was satisfied
- x* result was 1.10, which when plugged into the objective function: 4.4514
- Utilized scipy.optimize.minimize_scalar to check the result. x* was 1.08, OF result was 4.4511
- Graph OF vs. DV

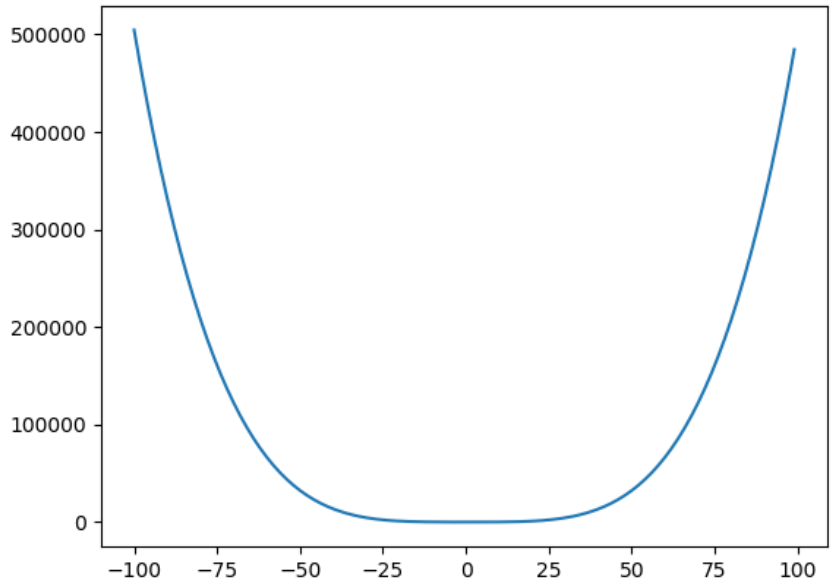


Image: Iterations of X, X-axis Iteration #, Y-Axis X* Value

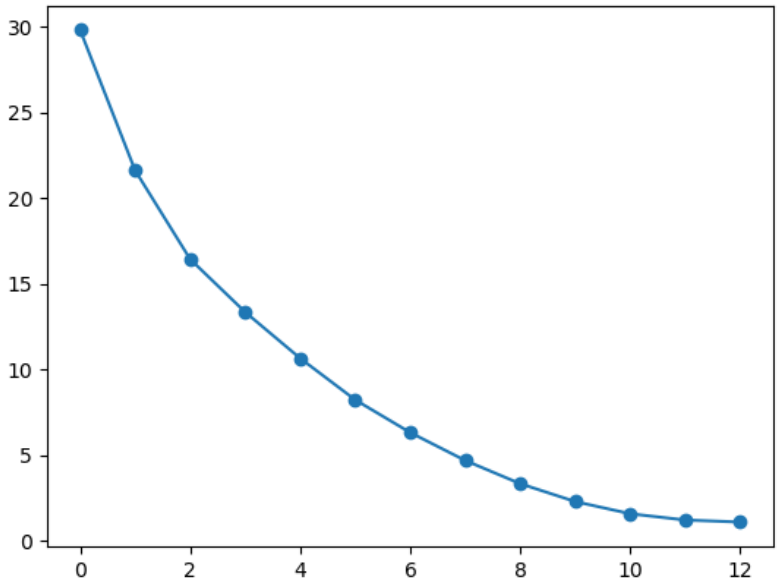
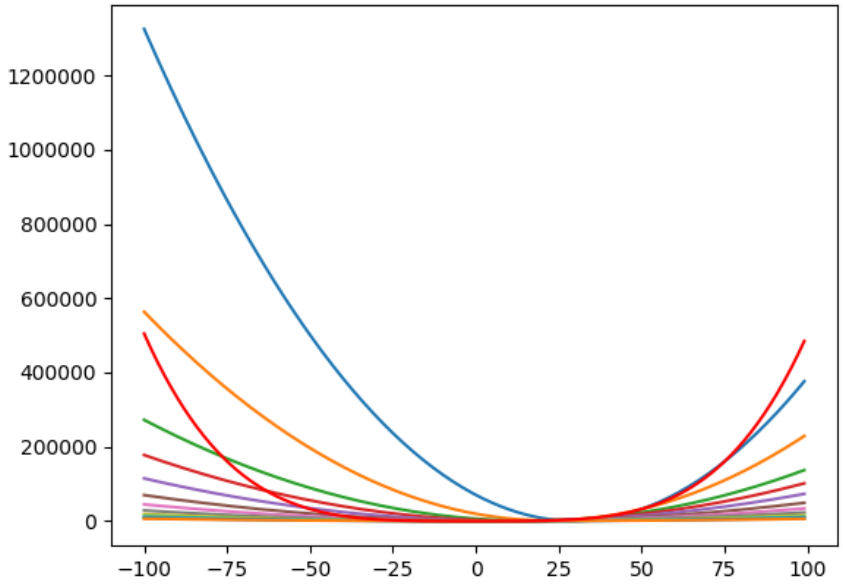


Image: Overally of all the Iterations



- The red plot is the original function