Midterm2

CPE608: Applied Modelling and Optimization Techniques

Team Members:

1. Aayush Gavande CWID:20010868

2. Shivani Bhavsar CWID:20013020

3. Chinmay Bhagwat CWID:20015512

Question 1

1. Solve the following using steepest descent algorithm. Start with $x_0 = [1\ 1\ 1]^T$ and use stopping threshold $\epsilon = 10^{-6}$.
   (a) Verify that the final solution satisfies the second order necessary conditions for a minimum.
   (b) Plot the value of the objective function with respect to the number of iterations and
   (c) Comment on the convergence speed of the algorithm.

$$\text{minimize } f(\mathbf{x}) = (x_1+5)^2 + (x_2+8)^2 + (x_3+7)^2 \\ +2x_1^2x_2^2 + 4x_1^2x_3^2$$

Import Libraries

In [1]:
```
!pip install sympy
```

```
Requirement already satisfied: sympy in c:\users\aayush\anaconda3\envs\machinelearning\lib\site-packages (1.11.1)
Requirement already satisfied: mpmath>=0.19 in c:\users\aayush\anaconda3\envs\machinelearning\lib\site-packages (from sympy) (1.2.1)
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")

import sympy
from sympy import *
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from numpy.linalg import norm
import time
```

Steepest Descent Algorithm(Line Search)

Steepest Descent Algorithm(Line Search)

In [3]:
```
#Steepet Descent Algorithm
def steepestDescent(df, func, xk, threshold = 10**(-6)):

    f_list = []
    counter = 0

    while (true):
        #Hessian Matrix
        hessian_x = hessian(f(x), [x1, x2, x3]).subs([(x1,xk[0][0]), (x2,xk[0][1]), (x3,xk[0][2])])

        #Gradient
        grad = Matrix([f(x)]).jacobian(Matrix([x1,x2,x3])).subs([(x1,xk[0][0]), (x2,xk[0][1]), (x3,xk[0][2])])

        #Alpha Calculation
        num = grad * grad.T;
        den = (grad * hessian_x * grad.T);
        num = np.asarray(num).flatten();
        den = np.asarray(den).flatten();
        num = num[0];
        den = den[0];
        alpha = float(num)/float(den);

        #Second order Condition (d.T * H(x) * d >= 0)
        second_order = ((-1 * grad) * hessian_x * (-1 * grad.T))[0]

        #xk calculation
        xk = xk - (alpha * grad)
        xk = np.asarray(xk);
        grad = np.asarray(grad);
        grad = grad.astype(float)

        #Threshold condition
        check = norm(alpha * grad);

        #Stopping Threshold condition
        if(check <= threshold):
            break;

        #Objective Function at updated xk
        fkVal = funcVal(xk);
        f_list.append(fkVal)

        #Increment Iteration counter
        counter = counter + 1;

        #Round Values
        x1_round = round(xk[0][0],4)
        x2_round = round(xk[0][1],4)
        x3_round = round(xk[0][2],4)
        fkVal_round = round(fkVal,4);
        #second_order_round = round(second_order,3)

        #Output Data
        df = df.append({'Iteration': counter, 'alpha': alpha, 'x1': x1_round, 'x2': x2_round, 'x3': x3_round, 'f(x)': fkVal_round, 'Second order': second_order}, ignore_index=True)



    return f_list, df
```

In [4]:
```python
x = sympy.symbols('x')
x1 = sympy.symbols('x1')
x2 = sympy.symbols('x2')
x3 = sympy.symbols('x3')
f = sympy.Function('f')
xk = np.array([[1,1,1]]); # Initial Value of vector x
t = time.process_time()
df = pd.DataFrame(columns=['Iteration', 'alpha', 'x1', 'x2','x3', 'f(x)', 'Second order'])


#Objective Function
def f(x):
    return (x1+5)**2 + (x2+8)**2 + (x3+7)**2 + 2*x1**2*x2**2 + 4*x1**2*x3**2

#Function Value
def funcVal(xk):
    return f(x).subs(x1, xk[0][0]).subs(x2, xk[0][1]).subs(x3, xk[0][2]);

#Call Steepest Descent Algorithm
f_list, df = steepestDescent(df, f(x), xk);

#Total time to reach solution
total_time = time.process_time() - t
```

Output

In [5]:
```
df.to_csv('Output - Steepest Descent Algorithm.csv', sep='\t')
df
```

Out[5]:

	Iteration	alpha	x1	x2	x3	f(x)	Second order
0	1	0.037516	0.0996	0.1746	0.0996	143.2365	43608
1	2	0.445530	-4.4534	-7.1125	-6.2301	5087.4350	1287.90946389077
2	3	0.001179	-1.7627	-6.4496	-5.0669	594.2034	5516650807.68812
3	4	0.002331	-0.2505	-6.2700	-4.7823	41.1396	189587186.976034
4	5	0.002940	-0.0278	-6.2756	-4.7883	32.7197	1952129.57679636
5	6	0.216761	-0.1252	-7.0189	-5.7407	29.9245	144.245373476558
6	7	0.002173	-0.0211	-7.0222	-5.7446	27.4242	1060735.84977134
7	8	0.200240	-0.0692	-7.4113	-6.2433	26.5033	50.1161035136894
8	9	0.001885	-0.0185	-7.4133	-6.2457	25.8196	385232.835751911
9	10	0.187848	-0.0452	-7.6318	-6.5259	25.4970	19.1570066855414
10	11	0.001747	-0.0172	-7.6330	-6.5273	25.2713	147948.172523890
11	12	0.180485	-0.0332	-7.7638	-6.6952	25.1485	7.74635348612877
12	13	0.001672	-0.0165	-7.7646	-6.6961	25.0647	59981.0319777242
13	14	0.176032	-0.0265	-7.8460	-6.8005	25.0155	3.23180531614535
14	15	0.001627	-0.0161	-7.8464	-6.8011	24.9823	25076.0933240587
15	16	0.173261	-0.0225	-7.8982	-6.8676	24.9619	1.37334953671611
16	17	0.001600	-0.0159	-7.8985	-6.8680	24.9482	10672.4956139331
17	18	0.171501	-0.0200	-7.9320	-6.9109	24.9396	0.590227789287354
18	19	0.001582	-0.0157	-7.9322	-6.9111	24.9339	4591.77428649228
19	20	0.170367	-0.0184	-7.9539	-6.9391	24.9302	0.255475927855710
20	21	0.001571	-0.0156	-7.9541	-6.9392	24.9277	1989.02160589501
21	22	0.169629	-0.0173	-7.9683	-6.9576	24.9261	0.111087176160230
22	23	0.001564	-0.0155	-7.9684	-6.9577	24.9251	865.319981211772
23	24	0.169146	-0.0167	-7.9778	-6.9697	24.9244	0.0484467475229225
24	25	0.001559	-0.0155	-7.9779	-6.9698	24.9239	377.509348108614
25	26	0.168829	-0.0162	-7.9840	-6.9777	24.9236	0.0211692764623791
26	27	0.001556	-0.0155	-7.9841	-6.9778	24.9234	164.994572050721
27	28	0.168620	-0.0160	-7.9882	-6.9830	24.9233	0.00926188151836604
28	29	0.001554	-0.0154	-7.9882	-6.9831	24.9232	72.1987882134966
29	30	0.168482	-0.0158	-7.9909	-6.9865	24.9231	0.00405560583954801
30	31	0.001553	-0.0154	-7.9909	-6.9865	24.9231	31.6177547064263
31	32	0.168391	-0.0157	-7.9927	-6.9888	24.9231	0.00177685480668614
32	33	0.001552	-0.0154	-7.9927	-6.9888	24.9231	13.8534152455464
33	34	0.168331	-0.0156	-7.9939	-6.9904	24.9230	0.000778765092725125
34	35	0.001551	-0.0154	-7.9939	-6.9904	24.9230	6.07198999536167
35	36	0.168291	-0.0155	-7.9947	-6.9914	24.9230	0.000341401715052414
36	37	0.001551	-0.0154	-7.9947	-6.9914	24.9230	2.66197093846245
37	38	0.168264	-0.0155	-7.9952	-6.9920	24.9230	0.000149690479279960
38	39	0.001551	-0.0154	-7.9952	-6.9920	24.9230	1.16718698942562

	Iteration	alpha	x1	x2	x3	f(x)	Second order
39	40	0.168247	-0.0155	-7.9955	-6.9925	24.9230	6.56399887917465e-5
40	41	0.001551	-0.0154	-7.9955	-6.9925	24.9230	0.511823817223594
41	42	0.168235	-0.0154	-7.9958	-6.9928	24.9230	2.87854594283627e-5
42	43	0.001550	-0.0154	-7.9958	-6.9928	24.9230	0.224454833393194
43	44	0.168228	-0.0154	-7.9959	-6.9930	24.9230	1.26240274269339e-5
44	45	0.001550	-0.0154	-7.9959	-6.9930	24.9230	0.0984365194921377
45	46	0.168223	-0.0154	-7.9960	-6.9931	24.9230	5.53650893411947e-6
46	47	0.001550	-0.0154	-7.9960	-6.9931	24.9230	0.0431713869241133
47	48	0.168219	-0.0154	-7.9961	-6.9932	24.9230	2.42819130805915e-6
48	49	0.001550	-0.0154	-7.9961	-6.9932	24.9230	0.0189340713933867
49	50	0.168217	-0.0154	-7.9961	-6.9932	24.9230	1.06496574914525e-6
50	51	0.001550	-0.0154	-7.9961	-6.9932	24.9230	0.00830419412079091
51	52	0.168215	-0.0154	-7.9961	-6.9933	24.9230	4.67081059117273e-7
52	53	0.001550	-0.0154	-7.9961	-6.9933	24.9230	0.00364212296605273
53	54	0.168214	-0.0154	-7.9962	-6.9933	24.9230	2.04857299584690e-7
54	55	0.001550	-0.0154	-7.9962	-6.9933	24.9230	0.00159740163352533
55	56	0.168214	-0.0154	-7.9962	-6.9933	24.9230	8.98488118484210e-8
56	57	0.001550	-0.0154	-7.9962	-6.9933	24.9230	0.000700608244385900
57	58	0.168213	-0.0154	-7.9962	-6.9933	24.9230	3.94070902602977e-8
58	59	0.001550	-0.0154	-7.9962	-6.9933	24.9230	0.000307282216590546
59	60	0.168213	-0.0154	-7.9962	-6.9933	24.9230	1.72837173584467e-8

(a) Verify that the final solution satisfies the second order necessary conditions for a minimum.

Second order necessary conditions -

1) grad(xmin) = 0

2) d.T * H(x) * d >= 0 where d = -(grad(xmin))

In our scenario,

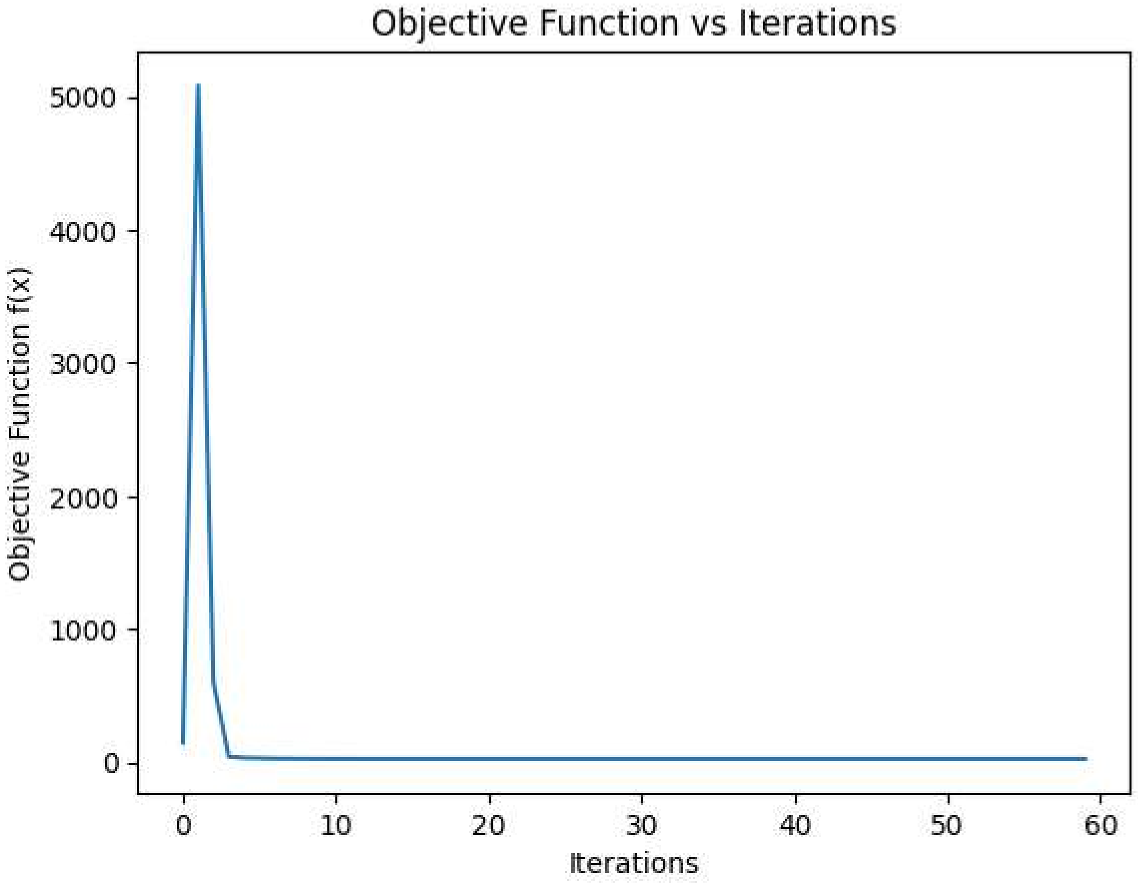
xmin = [-0.0154, -7.9962, -6.9933]

f(xmin) = 24.9230

second_order = d.T * H(xmin) * d = 1.72837173584467e-8 >= 0 (satisfied)

(b) Plot the value of the objective function with respect to the number of iterations

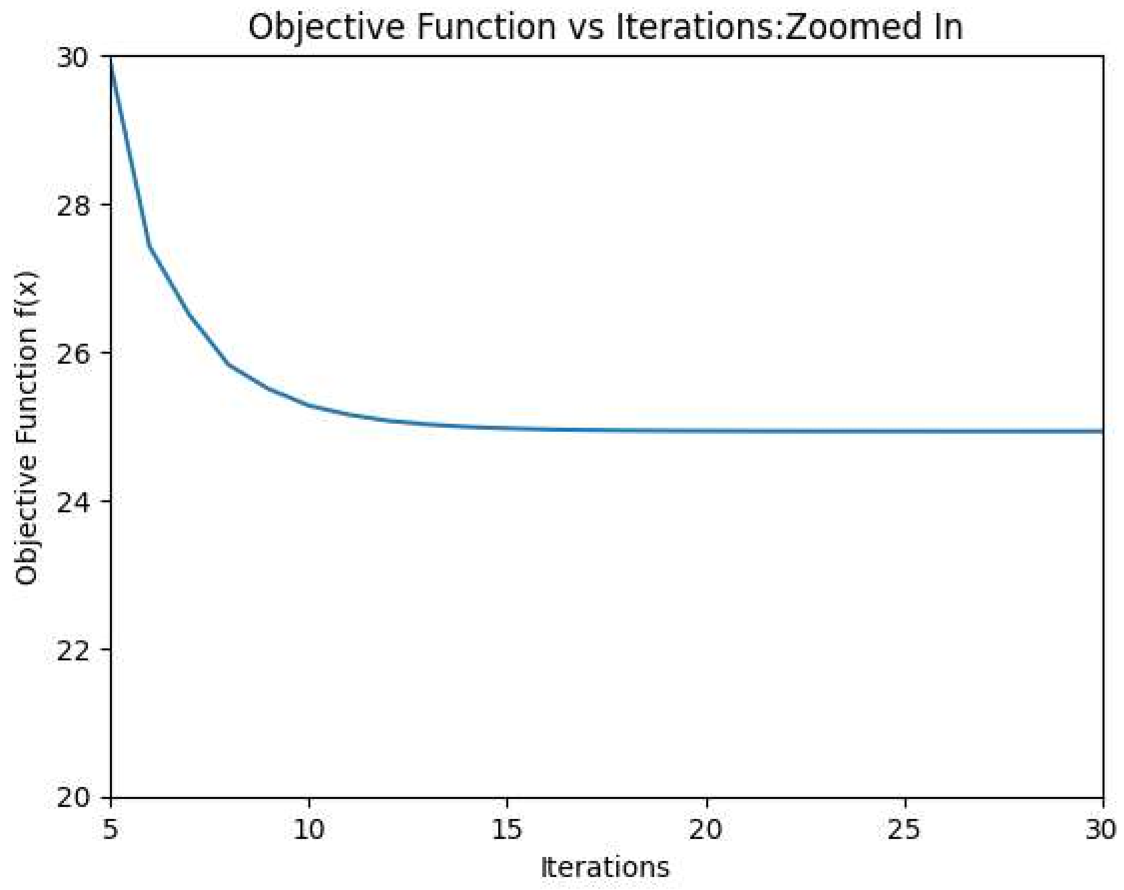
In [6]:
```
plt.plot(range(len(f_list)),f_list);
plt.xlabel('Iterations')
plt.ylabel('Objective Function f(x)')
plt.title("Objective Function vs Iterations")
plt.savefig('Objective Function vs Iterations Plot.png')
```



Zoom out plot to check minima

In [7]:
```
plt.plot(range(len(f_list)),f_list);
plt.xlim(5, 30)
plt.ylim(20, 30)
plt.xlabel('Iterations')
plt.ylabel('Objective Function f(x)')
plt.title("Objective Function vs Iterations:Zoomed In")
```

Out[7]:
```
Text(0.5, 1.0, 'Objective Function vs Iterations:Zoomed In')
```



(c) Comment on the convergence speed of the algorithm.

In [8]:
```
print("Total time to reach the solution: " +str(total_time) + ' Seconds')
```

```
Total time to reach the solution: 3.34375 Seconds
```

Optimum Solution

xmin = [-0.0154, -7.9962, -6.9933]

f(xmin) = 24.9230

Total Time = 2.96875 Seconds

As we can see in the plot(function vs Iterations), we reached the optimum solution at 25th iteration which shows that our algorithm is faster.