

Design Optimization\MAE_HW3_Q4.py

```
1 import numpy as np
2 import sympy as sp
3
4
5 # Objective function
6 def objective_function(d,s):
7     x_1=d
8     x_2=s[0,0]
9     x_3=s[1,0]
10
11     return ((x_1**2)+(x_2**2)+(x_3**2))
12
13 # We have a function for each matrix that needs to be calculated
14 def dh_ds(h,s,s_val):
15     dh1_ds1=sp.diff(h[0,0],s[0,0])
16     dh1_ds1=dh1_ds1.subs(s[0,0],s_val[0,0])
17     dh1_ds2=sp.diff(h[0,0],s[1,0])
18     dh1_ds2=dh1_ds2.subs(s[1,0],s_val[1,0])
19     dh2_ds1=sp.diff(h[1,0],s[0,0])
20     dh2_ds1=dh2_ds1.subs(s[0,0],s_val[0,0])
21     dh2_ds2=sp.diff(h[1,0],s[1,0])
22     dh2_ds2=dh2_ds2.subs(s[1,0],s_val[1,0])
23     dh_ds_final=np.array([[dh1_ds1,dh1_ds2],[dh2_ds1,dh2_ds2]])
24
25     return dh_ds_final
26
27 def df_ds(f,s,s_val):
28     df_ds1=sp.diff(f,s[0,0])
29     df_ds1=df_ds1.subs(s[0,0],s_val[0,0])
30     df_ds2=sp.diff(f,s[1,0])
31     df_ds2=df_ds2.subs(s[1,0],s_val[1,0])
32     df_ds_final=np.array([[df_ds1,df_ds2]])
33
34     return df_ds_final
35
36 def dh_dd(h,d,d_val):
37     dh1_dd=sp.diff(h[0,0],d)
38     dh1_dd=dh1_dd.subs(d,d_val)
```

```
39     dh2_dd=sp.diff(h[1,0],d)
40     dh2_dd=dh2_dd.subs(d,d_val)
41     dh_dd_final=np.array([[dh1_dd],[dh2_dd]])
42
43     return dh_dd_final
44
45 def df_dd(f,d,d_val):
46     df_dd1=sp.diff(f,d)
47     df_dd1=df_dd1.subs(d,d_val)
48
49     return df_dd1
50
51
52 # Linesearch algorithm is defined
53
54 def linesearch(dfdd,s_val,d_val,h,s,d):
55     alp=1
56     b=0.5
57     t=0.3
58     d_val_new=d_val-(alp*dfdd)
59     par_dhds=dh_ds(h,s,s_val)
60     par_dhdd=dh_dd(h,d,d_val)
61     s_val_new=s_val+(alp*(np.matmul(np.linalg.inv(np.float64(par_dhds)),par_dhdd))*dfdd)
62     f_alp=objective_function(d_val_new,s_val_new)
63     phi_alp=objective_function(d_val,s_val) - (alp*t*(dfdd**2))
64     j=0
65     while f_alp > phi_alp:
66         alp=alp*b
67         d_val_new=d_val-(alp*dfdd)
68         s_val_new=s_val+(alp*(np.matmul(np.linalg.inv(np.float64(par_dhds)),par_dhdd))*dfdd)
69         f_alp=objective_function(d_val_new,s_val_new)
70         phi_alp=objective_function(d_val,s_val) - (alp*t*(dfdd**2))
71         j=j+1
72
73     return alp
74
75
76 # Newton ralphson algorithm defined
77
78 def newton_ralpson(s_val,d_val,h_1,s):
79     x1=d_val
```

```

80     x2=s_val[0,0]
81     x3=s_val[1,0]
82     h1=((x1**2)/4)+((x2**2)/5)+((x3**2)/25)-1
83     h2=x1+x2-x3
84     h=np.array([[h1[0,0]], [h2[0,0]]], dtype=np.float64)
85     j=0
86     s_old=s_val
87     h_norm=np.linalg.norm(h)
88     while h_norm > 0.001 or j < 10:
89         dhds=dh_ds(h_1,s,s_old)
90         s_new=s_old-(np.matmul(np.linalg.inv(np.float64(dhds)),h))
91         h= h + (np.matmul(dhds,(s_new-s_old)))
92         s_old=s_new
93         j=j+1
94         h_norm=h[0,0]+ h[1,0]
95
96     return s_new
97
98 # Using sympy we create the variables of our function
99
100 x1= sp.symbols('x1')
101 x2= sp.symbols('x2')
102 x3= sp.symbols('x3')
103
104 # Objective function and the constraint equations
105
106 f=(x1**2)+(x2**2)+(x3**2)
107 h1=((x1**2)/4)+((x2**2)/5)+((x3**2)/25)-1
108 h2=x1+x2-x3
109
110 # we create a vector of the constraint equations and the dependant variables
111
112 h=np.array([[h1],[h2]])
113 s=np.array([[x2],[x3]])
114
115 # Initial variables have been assigned ensuring that they satisfy the constraints
116
117 x0_d=1
118 x0_s=np.array([[1.56137],[2.56137]])
119
120 ephsilon=0.0001

```

```
121
122 # finding the different partial derivatives
123
124 par_dhds=dh_ds(h,s,x0_s)
125 par_dfds=df_ds(f,s,x0_s)
126 par_dfdd=df_dd(f,x1,x0_d)
127 par_dhdd=dh_dd(h,x1,x0_d)
128
129 df_dd_main=par_dfdd-(np.matmul(np.matmul(par_dfds,np.linalg.inv(np.float64(par_dhds))),par_dhdd))
130
131 k=0
132 while df_dd_main > ephsilon or k < 50:
133
134     alpha = linesearch(df_dd_main,x0_s,x0_d,h,s,x1)
135     d_next=x0_d-(alpha*df_dd_main)
136     s_next_temp=x0_s+(alpha*(np.matmul(np.linalg.inv(np.float64(par_dhds)),par_dhdd))*df_dd_main)
137     s_next_real=newton_ralpson(s_next_temp,d_next,h,s)
138
139     par_dhds=dh_ds(h,s,s_next_real)
140     par_dfds=df_ds(f,s,s_next_real)
141     par_dfdd=df_dd(f,x1,d_next[0,0])
142     par_dhdd=dh_dd(h,x1,d_next[0,0])
143
144     df_dd_main=par_dfdd-(np.matmul(np.matmul(par_dfds,np.linalg.inv(np.float64(par_dhds))),par_dhdd))
145
146     k=k+1
147
148 print ('Dependant variable D')
149 print (d_next)
150 print ('State variable S')
151 print (s_next_real)
152
153
154
155
156
```