

Optimization Methods the unconstrained extremization problem

Narayanan Krishnamurthy^{1*}

for ECE-2671 taught by Dr. Simaan A. Marwan^{2*}

Abstract. The subject of this technical report is the evaluation of different unconstrained optimization methods; that determine the extremum points of the given objective function(s). The analysis is done using Matlab, and assumes the same starting point and termination criteria for comparing the different iterative algorithms.

Key words. Gradient Methods, Method of Steepest Descent, Conjugate Direction Method, Newton-Raphson Method, Region Elimination Methods, Bisection Method, Equal Spacing Method, Fibonacci Search, Golden Search, Polynomial Approximation,

^{*1} Department of Electrical and Computer Engg. ^{2*}Professor ECE Department University of Pittsburgh, Pittsburgh, PA 15261

Contents

1	Introduction	3
2	Necessary and Sufficiency Conditions for Extremum points and Taylor's Approximation	3
3	Iso-line plots of given objective functions	4
4	Gradient Methods	5
4.1	Constant and Variable Step-size Steepest Descent Methods	5
4.2	Newton-Raphson method	8
4.3	Conjugate Gradient Method	10
5	Direct Search Methods	12
5.1	Golden Search Method	12
5.2	Polynomial Approximation -Quadratic fit method	14
6	Observations and interpretation of results of different methods	16
7	Conclusion	17
A	Matlab Code for the Project	17
A.1	Code for Contour Plots of functions	17
A.2	Code for Constant step size Steepest Descent method	18
A.3	Code for Variable step size Steepest Descent Method	19
A.4	Code for Newton-Raphson method	21
A.5	Code for Conjugate Gradient Method	23
A.6	Code for Direct Search - Quadratic Approximation method	25
A.7	Code for Direct Search - Golden Search Method	27

1 Introduction

The study of methods for extremizing a function $f(x)$ over the entire R^n is an unconstrained optimization problem. An important assumption for using most of the algorithms discussed here to minimize the function $f(x)$ is that the requirement that the function be differentiable over the chosen domain, and that partial derivatives of at-least first order exist, namely the *Gradient* of the function. Some algorithms require the existence of second order derivatives and positive definiteness of the *Hessian* - the second order derivative matrix. These assumptions on the behavior of the function $f(x)$ and its *Taylor approximation* is taken up next [3].

2 Necessary and Sufficiency Conditions for Extremum points and Taylor's Approximation

The First-Order Necessary condition for a relative minimum point \mathbf{x}^* of $f(x)$, over R^n , for any *feasible direction* \mathbf{d} (the feasible directions are the directions used in the neighborhood of a point, to evaluate the functions behavior in that neighborhood); satisfies the following condition:

$$\nabla f(\mathbf{x}^*)^T \mathbf{d} \geq 0 \quad (2.1)$$

where $\nabla f(\mathbf{x}^*)$ is the *Gradient or first variant* of f and is given by:

$$\nabla f(\mathbf{x}^*) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]_{\mathbf{x}^*}^T \quad (2.2)$$

The Second-Order Necessary condition for a relative minimum point \mathbf{x}^* of a twice differentiable function $f(x)$ is:

$$\nabla f(\mathbf{x}^*)^T \mathbf{d} \geq 0 \quad (2.3)$$

$$\text{if } \nabla f(\mathbf{x}^*)^T \mathbf{d} = 0, \quad \text{then } \mathbf{d}^T \nabla^2 f(\mathbf{x}^*) \mathbf{d} \geq 0 \quad (2.4)$$

where $\nabla f(\mathbf{x}^*)^T \mathbf{d} = 0$ condition occurs if the extremum point \mathbf{x}^* is an internal point with all possible *feasible directions* \mathbf{d} in the functions domain R^n . $\nabla^2 f(\mathbf{x}^*)$ is the *Hessian or second*

variant of the function f and is given by the second-order derivative matrix:

$$\nabla^2 f(\mathbf{x}^*) = \left[\frac{\partial^2 f}{\partial x_i \partial x_j} \Big|_{\mathbf{x}^*} \right]_{i,j}, \quad \text{where } 1 \leq i, j \leq n \quad (2.5)$$

The Second-Order Sufficient conditions for a relative minimum point \mathbf{x}^* of a twice differentiable function $f(x)$ is:

$$\nabla f(\mathbf{x}^*) = 0 \text{ and } \nabla^2 f(\mathbf{x}^*) \text{ is positive definite} \quad (2.6)$$

The Taylor's approximation of the function f based on the first and second variants of f is given by:

$$\begin{aligned} f(x) = f(\mathbf{x}^0) + \langle \nabla f(\mathbf{x}^0), (\mathbf{x} - \mathbf{x}^0) \rangle + \frac{1}{2} \langle (\mathbf{x} - \mathbf{x}^0), \nabla^2 f(\mathbf{x}^0)(\mathbf{x} - \mathbf{x}^0) \rangle \\ + \text{Higher Order Terms} \end{aligned} \quad (2.7)$$

where $\langle \cdot, \cdot \rangle$ is the notation for *inner product* between vectors in R^n , for more details on vector spaces and FONC/SONC and sufficiency conditions refer [2, 4].

3 Iso-line plots of given objective functions

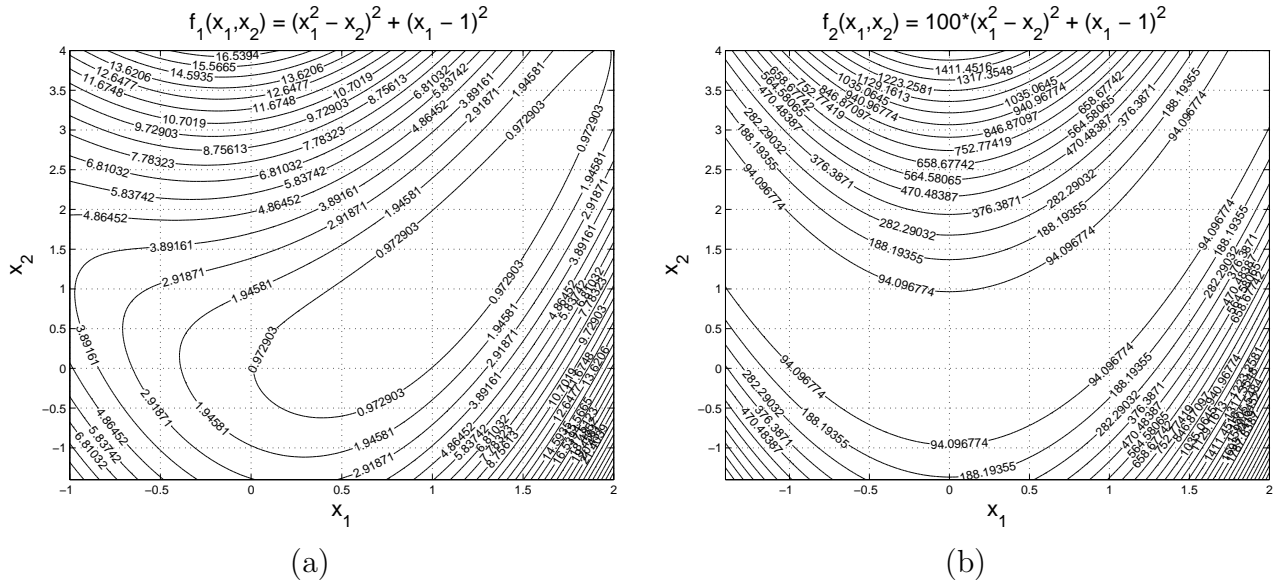
Two objective functions were provided to evaluate the performance of the various algorithms, the initial iteration guess x_0, y_0 is chosen as $[2 \ 2]^T$ and termination condition of the iterative search for the relative minimum is $\|\nabla f(x_k, y_k)\| < \epsilon$ where $\epsilon = 0.05$.

$$f_1(x_1, x_2) = (x_1^2 - x_2)^2 + (x_1 - 1)^2 \quad (3.1)$$

$$f_2(x_1, x_2) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 \quad (3.2)$$

The objective functions f_1 and f_2 are common test functions used to test convergence properties of any new algorithm. f_1 was introduced by Witte and Holst (1964), while f_2 was proposed by Rosenbrock (1960).

Iso-lines or contours are good to visualize the gradient or descent/ascent direction of the functions, and to locate the minima or maximum, especially if the function is dependent on just two variables.

Figure 1: (a)Contour Plot of $f_1(x_1, x_2)$ (b)Contour Plot of $f_2(x_1, x_2)$

4 Gradient Methods

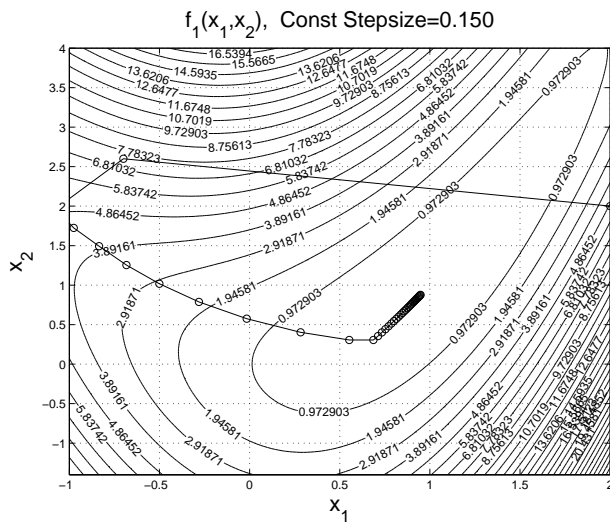
The objective of the gradient methods is to generate a sequence of vectors, $\mathbf{x}^0, \mathbf{x}^1, \dots, \mathbf{x}^n$, in R^n , such that: $f(\mathbf{x}^0) > f(\mathbf{x}^1) \dots > f(\mathbf{x}^n)$. Thus the iterative method of generating such sequences can be mathematically expressed as:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - t_k A_k \nabla f(\mathbf{x}^k) \quad (4.1)$$

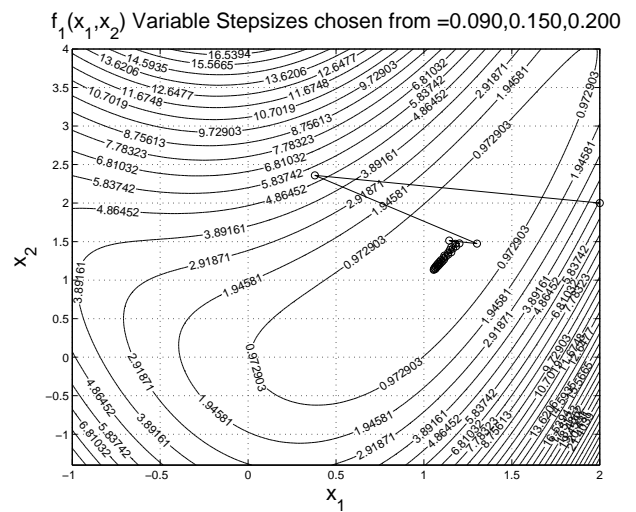
where t_k is the step-size for the 'kth' iteration. Note the process of determining the minimum involves traversing along the negative direction of the gradient vector given by $\nabla f(\mathbf{x}^k)$. A_k is a linear transformation of the gradient vector, which provides more flexibility in exploring for the minima that just the line search provided by the gradient. A_k in case of the Newton-Raphson is $\nabla^2 f(\mathbf{x}^k)^{-1}$.

4.1 Constant and Variable Step-size Steepest Descent Methods

Constant and variable step size steepest descent methods involves choosing a constant t_k or multiple t_k 's in the equation 4.1. In the variable step size method, at every step of generation of next exploration point in the function, that step from the chosen array that gives a minimum value of f evaluated at that point is chosen. Optimal Step-size method may be thought of as an extension of the variable step-size method taking the step-size to



(a)



(b)

Figure 2: (a)Convergence of f_1 using constant step-size steepest descent method
 (b)Convergence of f_1 using variable step-size steepest descent method

the limit of a continuum.

Convergence of f_1 using constant step size of 0.15, and the steepest descent algorithm:

iteration	x1	x2	f1(x1,x2)	Norm(gradient)
2	-1.0762	1.9670	4.9648	1.7511
3	-0.9756	1.7244	4.4998	1.8067
4	-0.8351	1.4926	4.0000	1.8861
5	-0.6830	1.2541	3.4528	1.9889
6	-0.5009	1.0178	2.8408	2.1212
7	-0.2811	0.7877	2.1434	2.2640
8	-0.0163	0.5751	1.3633	2.3027
9	0.2830	0.4027	0.6182	1.9113
10	0.5529	0.3059	0.1999	0.8948
11	0.6871	0.3058	0.1256	0.3730
12	0.7124	0.3557	0.1058	0.3354
13	0.7338	0.4012	0.0897	0.3035
14	0.7532	0.4424	0.0765	0.2760
15	0.7708	0.4799	0.0656	0.2519
16	0.7867	0.5142	0.0565	0.2308
17	0.8013	0.5456	0.0488	0.2122
18	0.8145	0.5745	0.0423	0.1955
19	0.8267	0.6012	0.0368	0.1806
20	0.8379	0.6259	0.0321	0.1672

21	0.8482	0.6487	0.0280	0.1551
22	0.8578	0.6700	0.0246	0.1441
23	0.8666	0.6897	0.0216	0.1341
24	0.8748	0.7081	0.0190	0.1250
25	0.8823	0.7252	0.0167	0.1166
26	0.8894	0.7412	0.0147	0.1089
27	0.8960	0.7562	0.0130	0.1018
28	0.9021	0.7701	0.0115	0.0953
29	0.9078	0.7832	0.0102	0.0893
30	0.9132	0.7955	0.0090	0.0837
31	0.9182	0.8070	0.0080	0.0786
32	0.9229	0.8179	0.0071	0.0738
33	0.9273	0.8280	0.0063	0.0693
34	0.9314	0.8376	0.0056	0.0651
35	0.9352	0.8465	0.0050	0.0613
36	0.9389	0.8550	0.0044	0.0577
37	0.9423	0.8629	0.0040	0.0543
38	0.9455	0.8704	0.0035	0.0511
39	0.9485	0.8775	0.0031	0.0482

The minima of f1 found after 39 iterations,
using constant step of 0.150000, gradient methods are x1=0.948498 x2=0.877480,
and the min value of function is =0.003144

Convergence of f1 using variable steps chosen from 0.05,0.15 and 0.25,
and negative direction of gradient is used as the line of exploration.

iteration	x1	x2	f1(x1,x2)	Norm(gradient)
2	1.3015	1.4738	0.1394	1.8044
3	1.1441	1.5134	0.0626	0.7662
4	1.2024	1.4766	0.0419	0.2635
5	1.1793	1.4710	0.0386	0.1617
6	1.1833	1.4389	0.0351	0.1989
7	1.1558	1.4273	0.0326	0.2138
8	1.1658	1.4109	0.0302	0.1373
9	1.1478	1.3901	0.0271	0.1506
10	1.1555	1.3610	0.0248	0.1983
11	1.1383	1.3564	0.0228	0.1215
12	1.1383	1.3321	0.0204	0.1324
13	1.1161	1.3175	0.0186	0.1682
14	1.1241	1.3046	0.0171	0.1038

15	1.1114	1.2882	0.0152	0.1067
16	1.1139	1.2670	0.0137	0.1232
17	1.0972	1.2591	0.0125	0.1206
18	1.1044	1.2426	0.0114	0.1175
19	1.0882	1.2357	0.0104	0.1137
20	1.0954	1.2203	0.0095	0.1094
21	1.0802	1.2141	0.0087	0.1047
22	1.0868	1.1999	0.0079	0.0996
23	1.0730	1.1943	0.0072	0.0942
24	1.0788	1.1814	0.0065	0.0887
25	1.0666	1.1761	0.0059	0.0832
26	1.0713	1.1646	0.0054	0.0777
27	1.0608	1.1595	0.0049	0.0724
28	1.0655	1.1458	0.0044	0.0889
29	1.0577	1.1439	0.0040	0.0510
30	1.0559	1.1339	0.0035	0.0494

The minima of f1 found after 30 iterations,
using Variable Stepsizes chosen from =0.090,0.150,0.200, gradient
method are x1=1.055874 x2=1.133855,
and the min value of function is =0.003482

Convergence of f2:

The function f2 did not converge at all for constant and variable
stepsize methods, various intial points ranging from [-1 1], [-1.4
0.5] [2 2] and various step sizes ranging from 0.05,0.09,0.15 were
tried in vain the steepest descent algorithm couldnt locate the
descent direction to iterate on and hence a failed output:

```
iteration    x1          x2      f2(x1,x2)    Norm(gradient)
not a descent direction ... exiting
The extremum could not be located after 2 iterations
```

4.2 Newton-Raphson method

The Newton-Raphson method, uses not only the first variant, but also the second variant information in its search for the minima. The Taylor's approximation of a function given by the expression 2.7, can be re-written (ignoring the higher order terms) as:

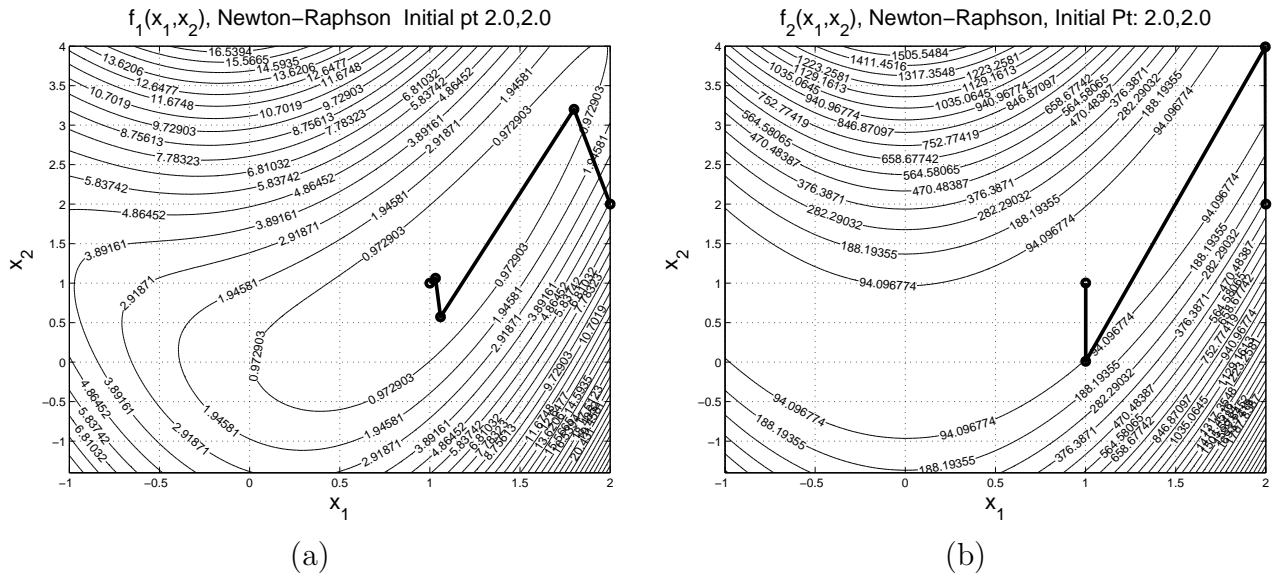


Figure 3: (a)Convergence of f_1 and (b)Convergence of f_2 using Newton-Raphson

$$\nabla f(\mathbf{x}^{k+1}) \approx \nabla f(\mathbf{x}^k) + \nabla^2 f(\mathbf{x}^k)(\mathbf{x}^{k+1} - \mathbf{x}^k) \quad (4.2)$$

which is also clear from fundamental definition of derivatives in calculus. Now finding the root of $\nabla f(\mathbf{x})$ we get:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \nabla^2 f(\mathbf{x}^k)^{-1} \nabla f(\mathbf{x}^k) \quad (4.3)$$

Convergence of f1:

iteration	x1	x2	f1(x1,x2)	Norm(gradient)
2	1.0593	0.5733	0.3046	2.6785
3	1.0310	1.0622	0.0010	0.0653
4	1.0000	0.9991	0.0000	0.0044

 The minima of f1 found after 4 iterations,
 with initial pt 2.000000 2.000000, using Newton-Raphson(NR) method are x1=1.0, x2=1.0,
 and the min value of function is =0.000001

Convergence of f2:

iteration	x1	x2	f2(x1,x2)	Norm(gradient)
2	1.0012	0.0099	98.5152	444.3233
3	1.0012	1.0025	0.0000	0.0025

The minima of f2 found after 3 iterations,
with initial pt 2.0,2.0,using Newton-Raphson(NR) method are x1=1.001233 x2=1.002467,
and the min value of function is =0.000002

4.3 Conjugate Gradient Method

The conjugate gradient method uses a method similar to *Gram-Schmidt's* orthogonalization procedure to generate *Q-Conjugate* vectors [1]. That is \mathbf{d}^k is constructed as a linear combination of \mathbf{d}^{k-1} vectors, with the coefficient β_{k-1} chosen such that $\langle \mathbf{d}^k, Q\mathbf{d}^{k-1} \rangle = 0$.

The conjugate gradient algorithm can be expressed as the following sequence of steps:

```

epsilon = 0.05           %termination condition
r0      = gradient_f1(x0_y0)
H       = hessian_f1(x0_y0)
d0      = -r0
t0      = - <d0,r0> / <d0,(H d0)>
x1_y1   = x0_y0 + t0*d0
r1      = gradient_f1(x1_y1)
b0      = <r1,r1>/<r0,r0>
d1      = -r1 + b0*d0

while (norm(gradient_f1(xk_yk)) < epsilon) {

H       = hessian_f1(x0_y0)
xk_yk   = xk-1_yk-1 + tk-1 * dk-1
rk      = gradient_f1(xk_yk)
bk-1    = <rk,rk> / <rk-1,rk-1>
dk      = -rk + bk-1 * dk-1
tk      = - <dk,rk> / <dk,(H dk)>
}

```

Results of Convergence of f1 using the conjugate gradient algorithm
whose pseudo-code is given above is:

iteration	x1	x2	f1(x1,x2)	Norm(gradient)
2	1.4975	2.1115	0.2646	1.7980
3	1.4275	2.1236	0.1901	0.4035
4	1.4032	2.1165	0.1843	0.2955
5	1.3242	1.9980	0.1649	0.8101

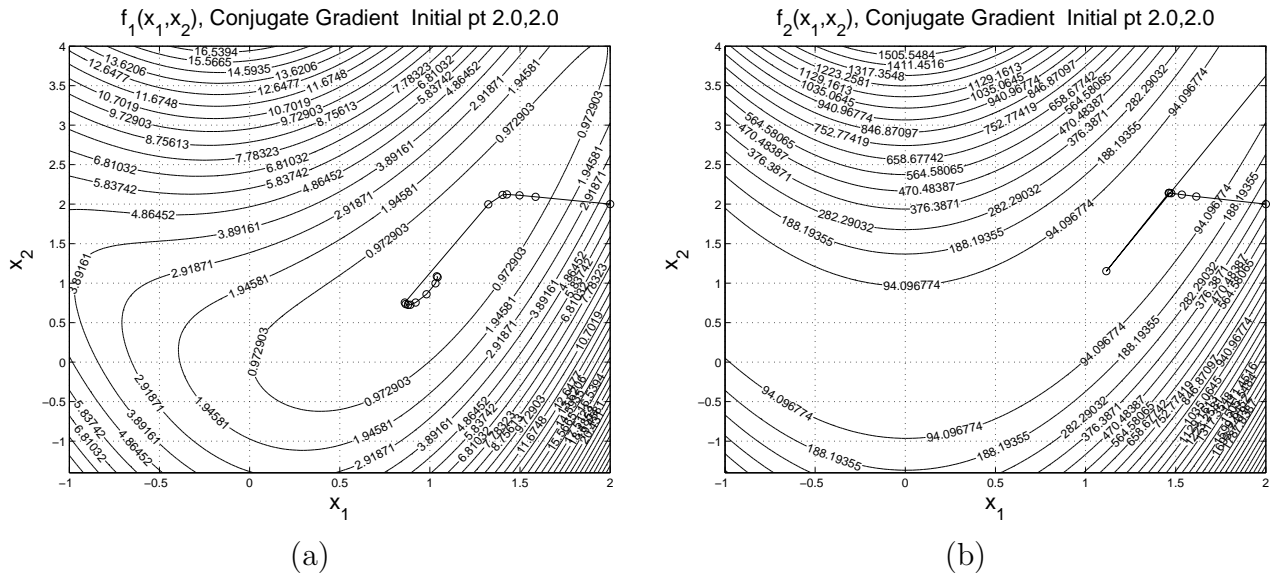


Figure 4: (a)Convergence of f_1 and (b)Convergence of f_2 using conjugate gradient method

6	0.8612	0.7543	0.0194	0.3224
7	0.8656	0.7353	0.0183	0.2226
8	0.8788	0.7243	0.0170	0.1211
9	0.8918	0.7265	0.0164	0.1405
10	0.9207	0.7539	0.0151	0.2646
11	0.9814	0.8592	0.0112	0.4254
12	1.0314	0.9974	0.0054	0.3617
13	1.0428	1.0756	0.0020	0.1371
14	1.0388	1.0870	0.0016	0.0475

 The minima of f_1 found after 14 iterations,
 with initial pt 2.000000 2.000000, using Conjugate Gradient method are
 $x_1=1.0$, $x_2=1.1$, and the min value of function is =0.001568

Convergence of f_2 :

The function f_2 does not converge using current implementation of conjugate gradient method. The Gradient vector/Hessian becomes abnormally large after a few iterations, due to precision errors while calculating the gradient when the itinerant approaches the minima. Note the direction of approach of itinerant before the abnormal gradient value is headed in the right direction i.e. towards the minima. A better approach would be to use direct methods to calculate next estimate like the polynomial fit thereby circumventing precision error in calculating first and second variants. The failed output:

iteration	x1	x2	f2(x1,x2)	Norm(gradient)
2	1.4813	2.0134	3.4993	113.955
3	1.4337	2.0290	0.2583	16.914
4	1.4257	2.0316	0.1814	1.5335
5	1.4249	2.0317	0.1807	0.2822
6	1.4203	2.0223	0.1792	2.3043
7	0.8654	0.4497	8.9710	119.390
8	4.4958	10.6666	9124.72	17279

The extremum could not be located after 8 iterations

5 Direct Search Methods

The direct search methods precludes the calculation of the computation intensive gradient vectors or Hessian matrices. But like most methods they work best for some functions while their performance and convergence is dismal for others.

5.1 Golden Search Method

The Golden Search method uses the limiting properties of the Fibonacci sequence F_k when $k \rightarrow \infty$, the sequence and the limiting properties are:

$$F_0 = F_1 = 1 \quad (5.1)$$

$$F_k = F_{k-1} + F_{k-2}, \quad \text{when } k \rightarrow \infty \quad (5.2)$$

$$\frac{F_{k-1}}{F_{k+1}} \rightarrow 0.382 \quad (5.3)$$

$$\frac{F_k}{F_{k+1}} \rightarrow 0.618 \quad (5.4)$$

Thus at every iteration in golden search the uncertainty interval (region of locating a minima) is reduced by 38.2 percent till you converge to a sufficiently small region where the minima is found.

Results of convergence of f1 using Golden Search method:

iteration	x1	x2	f1(x1,x2)	Norm(gradient)
1	2.0000	2.0000	5.0000	18.4391
2	1.0968	1.0968	0.0207	0.6930
3	1.0968	1.0968	0.0207	0.6930
4	1.0968	1.0968	0.0207	0.6930

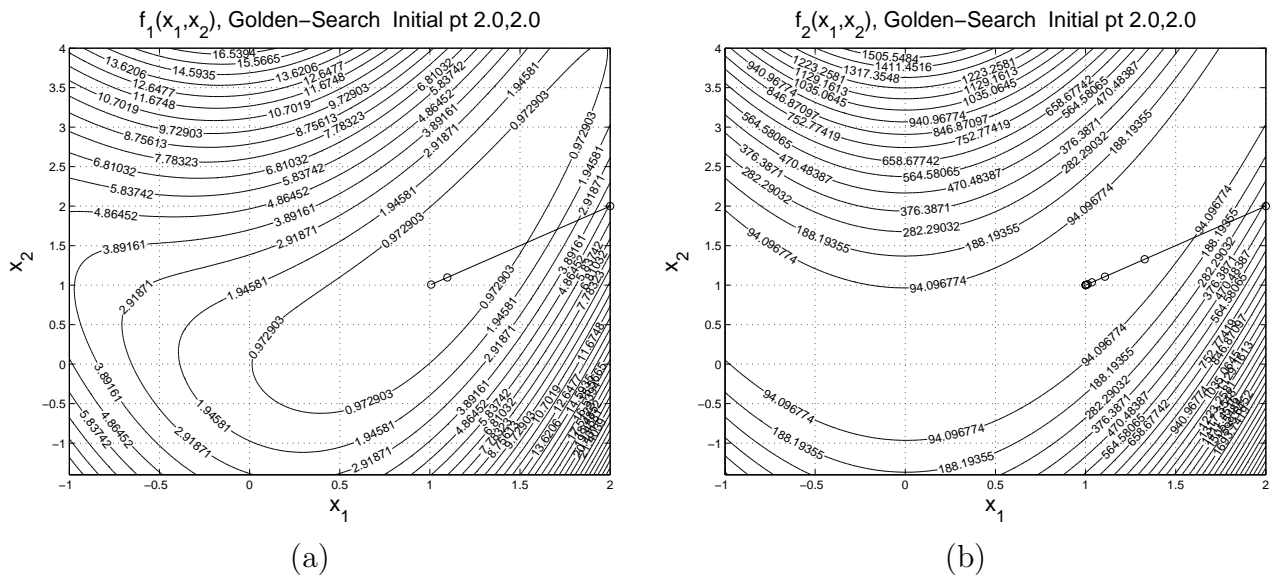


Figure 5: (a)Convergence of f_1 and (b)Convergence of f_2 using golden search

5	1.0968	1.0968	0.0207	0.6930
6	1.0069	1.0069	0.0001	0.0443

The minima of f1 found after 6 iterations,
 with initial pt 2.000000 2.000000, using Golden Search method are x1=1.0, x2=1.0,
 and the min value of function is =0.000097

Results of convergence of f2 using Golden Search method:

iteration	x1	x2	f2(x1,x2)	Norm(gradient)
1	1.3280	1.3280	19.0809	247.8529
2	1.1076	1.1076	1.4314	58.1173
3	1.0353	1.0353	0.1347	16.8643
4	1.0116	1.0116	0.0138	5.3054
5	1.0038	1.0038	0.0015	1.7162
6	1.0012	1.0012	0.0002	0.5604
7	1.0004	1.0004	0.0000	0.1835
8	1.0001	1.0001	0.0000	0.0602
9	1.0000	1.0000	0.0000	0.0197

The minima of f2 found after 9 iterations,
 with initial pt 2.000000 2.000000, using Golden Search method are x1=1.0, x2=1.0,
 and the min value of function is =0.000000

5.2 Polynomial Approximation -Quadratic fit method

The algorithm for fitting a quadratic polynomial to known three points a,b,c and function values fa,fb,fc at these points is as follows[1]:

```

d0 is any direction vector that is normalized such that ||d0|| = 1
d0      = [1/sqrt(2) 1/sqrt(2)]^T %(say)
x0_y0   = [2 2]^T                %initial point
t0      = 0.15                    %(say) assumed initial step size
fa      = f(x0_y0)
fb      = f(x0_y0 + t0.* d0)
if  fa < fb
fc      = f(x0_y0 - t0.* d0)
else
fc      = f(x0_y0 + 2*t0.*d0)
while norm(gradient_f1(xk_yk)) < e {

```

$$t_k = -\frac{1}{2} \frac{\begin{vmatrix} 1 & fa & a^2 \\ 1 & fb & b^2 \\ 1 & fc & c^2 \end{vmatrix}}{\begin{vmatrix} 1 & a & fa \\ 1 & b & fb \\ 1 & c & fc \end{vmatrix}} = -\frac{1}{2} \frac{(b^2 - a^2)fa + (c^2 - a^2)fb + (a^2 - b^2)fc}{(b - c)fa + (c - a)fb + (a - b)fc} = -\frac{\alpha_1}{2\alpha_2}$$

Note: For the algorithm to converge:

$$\alpha_2 = \frac{(c - b)fa + (a - c)fb + (b - c)fc}{(a - b)(b - c)(c - a)} > 0$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + t_k d_0$$

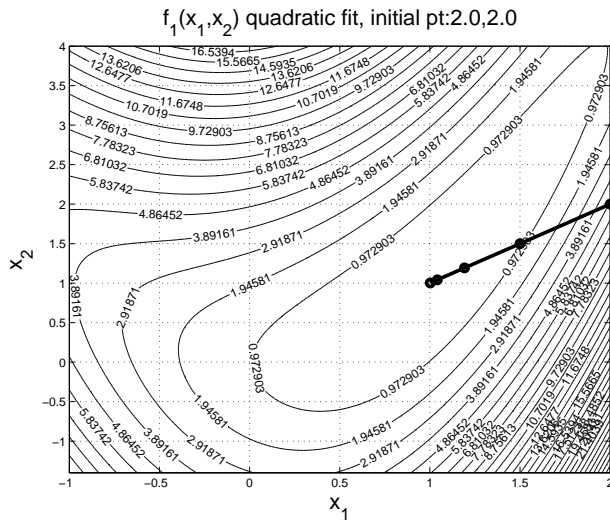
}

The essence of the above algorithm is in approximating the neighbourhood of $f(x)$ with a quadratic polynomial:

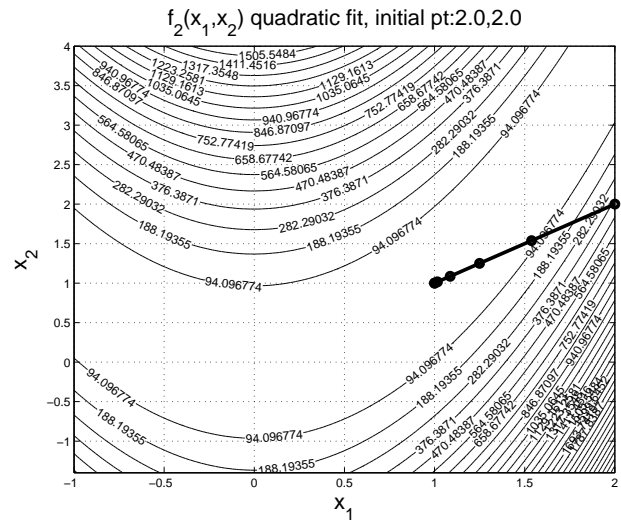
$$f(x + t_k d_0) \approx \alpha_0 + \alpha_1 t_k + \alpha_2 t_k^2$$

Results of convergence of f1 using quadratic approximation:

iteration	x1	x2	f1(x1,x2)	Norm(gradient)
2	1.1923	1.1923	0.0896	1.5476
3	1.0413	1.0413	0.0036	0.2754
4	1.0024	1.0024	0.0000	0.0151



(a)



(b)

Figure 6: (a)Convergence of f_1 and (b)Convergence of f_2 using quadratic fit method

The minima of f_1 found after 4 iterations, using quadratic polynomial approximation to determine successive stepsize with initial pt:2.0,2.0 and minima are $x_1=1.002384$ $x_2=1.002384$, and the min value of function is =0.000011

Results of convergence of f_2 using quadratic approximation:

iteration	x_1	x_2	$f_2(x_1, x_2)$	Norm(gradient)
2	1.2496	1.2496	9.7917	168.3929
3	1.0865	1.0865	0.8897	45.0909
4	1.0159	1.0159	0.0263	7.3363
5	1.0007	1.0007	0.0000	0.3087
6	1.0000	1.0000	0.0000	0.0049

The minima of f_2 found after 6 iterations, using quadratic polynomial approximation to determine successive stepsize with initial pt:2.0,2.0 and minima are $x_1=0.999989$ $x_2=0.999989$, and the min value of function is =0.000000

6 Observations and interpretation of results of different methods

The programming experience and visually seeing the convergence of different algorithms from program output or through plots helped in understanding the different methods viz.; the gradient and direct search methods. Here are the observations of the exercise:

- The contour plots (figure 1) of f_1 and f_2 indicate they have a relative minima at $[1 \ 1]^T$. It is also apparent from the expressions for the functions given by equations 3.1.
- The constant step-size steepest descent method on f_1 converged to the minima in 39 iterations, while the variable step-size method though more complex than the first was a clear improvement with convergence in 30 iterations. Note: If the sequence of steps for the variable step-size was taken to the limit, it would be the optimal step-size method (ideally the best method but computationally expensive)
- The Rosenbrock function f_2 did not converge for steepest descent and conjugate gradient methods. The method was unable to locate a descent direction to iterate upon for the former, see results of section 4.1. While in the later, though the algorithm heads in the right direction i.e. towards the minima, precision errors in calculating the gradient vector close to zero makes the algorithm go awry, resulting in abnormally high gradients in the opposite direction - see figure 4 for the partial success of the current implementation.
- The Newton-Raphson method like conjugate gradient method, though computationally expensive involving the calculation of the first and second variants of the functions at every iteration, converged to the minima with least number of iterations of all the methods. f_1 converged in 4 iterations, while f_2 converged in 3 iterations.
- The direct search methods were the computationally least expensive methods, as they did not involve computations of *Gradients/Hessians*. The complexity of the methods increase from Golden-search to Quadratic fit, While the Golden search relies on the limiting behavior of the Fibonacci sequence in determining the next itinerant, the Quadratic fit locates the next itinerant by minimizing a fitted quadratic polynomial through the known values of the function evaluated at the three known points .
- The Quadratic fit method converged to the minima in 4 iterations on f_1 test function, while it converged to the minima in 6 iterations on the f_2 test function

7 Conclusion

The unconstrained optimization problem of finding the relative minima \mathbf{x}^* , of $f(\mathbf{x})$ in R^n was studied in this report. Various methods including gradient methods and direct search methods were evaluated for the given test functions f_1 and f_2 , see previous section 6 for a summary of the interpretation on the results.

References

- [1] M. AOKI, *Introduction to Optimization Techniques*, The Macmillan Company, 1971.
- [2] E. K. P. CHONG AND S. H. ZAK, *An Introduction to Optimization Second Edition*, Wiley-Intersciences Series, 2001.
- [3] D. G. LUENBERGER, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, 1972.
- [4] S. A. MARWAN, *Lecture notes. ECE-2671 fall 2005*.

A Matlab Code for the Project

The code for testing the various algorithms on f_1 and f_2 functions are identical, the only changes are in the call to appropriate functions, for example Newton-Raphson algorithm tested on f_2 , would call '*gradient_f2*' instead of '*gradient_f1*', and similarly call appropriate functions depending on whether the test is on f_1 or f_2 . Since code is identical, the code for various algorithms tested on f_1 is provided here in the annex:

A.1 Code for Contour Plots of functions

```
%contour plots By Narayanan Krishnamurthy
%For Dr.Simaan ece 2671 class
clear,clf,clc
n = 30; %# of contours
format long g
[X,Y] = meshgrid(-1:.02:2,-1.4:.02:4);
Z = (X.^2 - Y).^2 + (X - ones(size(X))).^2;
%Then, generate a contour plot of Z.
[C,h] = contour(X,Y,Z,n);
```

```

xlabel('x_1','FontSize',18),ylabel('x_2','FontSize',18),title('f(x_1,x_2)
= (x_1^2 - x_2)^2 + (x_1 - 1)^2 ','FontSize',18);
grid on

```

A.2 Code for Constant step size Steepest Descent method

%Functions used by the Algorithm:

```

function r = f1(xk_yk)
format long g
r = (xk_yk(1)^2 - xk_yk(2))^2 + (xk_yk(1) -1)^2;

function p = gradient_f1(xk_yk)
format long g;
p(1) = 4*(xk_yk(1)^2 - xk_yk(2))*xk_yk(1) + 2*(xk_yk(1) -1) ;
p(2) = -2*(xk_yk(1)^2 - xk_yk(2));

function idx = check_min_t_f1(t1,t2,t3,xk_yk)
[m,idx] = min([f1(xk_yk -t1.*gradient_f1(xk_yk))
              f1(xk_yk -t2.*gradient_f1(xk_yk)) f1(xk_yk -t3.*gradient_f1(xk_yk))]);

%Constant step size -Steepest Descent Algorithm By Narayanan Krishnamurthy
%for ee2671 fall 2005
clear,clf,clc

%Gradient methods f1
clear; echo off;
format long g;
% initial point
x0_y0 = [2 2];
% termination point
e = 0.05; N =1000;
% constant step size
t = 0.15;
xk_yk(1,:) = x0_y0 - t.*gradient_f1(x0_y0);
disp('iteration      x1            x2      f1(x1,x2)      Norm(gradient)');
for i = 2:N
    xk_yk(i,:) = xk_yk(i-1,:) - t.*gradient_f1(xk_yk(i-1,:));

    if f1(xk_yk(i,:)) > f1(xk_yk(i-1,:))
        disp('not a descent direction ... exiting');
        disp(sprintf('The extremum could not be located after %d iterations',i));
    end
end

```

```

        break;
    end

    disp(sprintf('%-4d\t\t%3.4f\t\t%3.4f\t\t%3.4f\t\t%3.4f',i,xk_yk(i,1),xk_yk(i,2),
        f1(xk_yk(i,:)),norm(gradient_f1(xk_yk(i,:)))));
    if norm(gradient_f1(xk_yk(i,:))) < e
        disp('-----');
        disp(sprintf('The minima of f1 found after %d iterations,',i));
        disp(sprintf('using constant step of %f, gradient methods are
            x1=%f x2=%f,',t, xk_yk(i,1),xk_yk(i,2)));
        disp(sprintf('and the min value of function is =%f', f1(xk_yk(i,:))));
        disp('-----');
        break;
    end
end
end

n = 30; %# of contours
format long g
[X,Y] = meshgrid(-1:.02:2,-1.4:.02:4);
Z = (X.^2 - Y).^2 + (X - ones(size(X))).^2;
%Then, generate a contour plot of Z.
[C,h] = contour(X,Y,Z,n);
clabel(C,h),xlabel('x_1','FontSize',18),ylabel('x_2','FontSize',18),
title(sprintf('f_1(x_1,x_2),Const Stepsize=%1.3f',t),'FontSize',18);
grid on

hold on;
convergence = [x0_y0' xk_yk'];
%scatter(convergence(1,:),convergence(2,:));
plot(convergence(1,:),convergence(2,:),'-ro');

```

A.3 Code for Variable step size Steepest Descent Method

```

%Variable Step Size Steepest Descent Method, Narayanan Krishnamurthy
%for ee2671 fall 2005
clear,clf,clc

%Gradient methods f1
clear; echo off;
format long g;
% initial point

```

```

x0_y0 = [2 2];
% termination point
e = 0.05; N =1000;
% constant step size
tf1(1) = 0.09; tf1(2) =0.15; tf1(3) =0.2;
t = check_min_t_f1(tf1(1),tf1(2),tf1(3),x0_y0);
xk_yk(1,:) = x0_y0 - tf1(t).*gradient_f1(x0_y0);
disp('iteration      x1          x2      f1(x1,x2)      Norm(gradient)');
for i = 2:N
    t = check_min_t_f1(tf1(1),tf1(2),tf1(3),xk_yk(i-1,:));
    xk_yk(i,:) = xk_yk(i-1,:) - tf1(t).*gradient_f1(xk_yk(i-1,:));

    if f1(xk_yk(i,:)) > f1(xk_yk(i-1,:))
        disp('not a descent direction ... exiting');
        disp(sprintf('The extremum could not be located after %d iterations',i));
        break;
    end

    disp(sprintf('%-4d\t\t%3.4f\t\t\t%3.4f\t\t\t%3.4f\t\t\t%3.4f',i,xk_yk(i,1),
        xk_yk(i,2),f1(xk_yk(i,:)),norm(gradient_f1(xk_yk(i,:))))));

    if norm(gradient_f1(xk_yk(i,:))) < e
        disp('-----');
        disp(sprintf('The minima of f1 found after %d iterations,',i));
        disp(sprintf('using Variable Stepsizes chosen from
            =%1.3f,%1.3f,%1.3f, gradient method are x1=%f x2=%f,',
            tf1(1),tf1(2),tf1(3),xk_yk(i,1),xk_yk(i,2)));
        disp(sprintf('and the min value of function is =%f', f1(xk_yk(i,:))));
        disp('-----');
        break;
    end
end

end

n = 30; %# of contours
format long g
[X,Y] = meshgrid(-1:.02:2,-1.4:.02:4);
Z = (X.^2 - Y).^2 + (X - ones(size(X))).^2;
%Then, generate a contour plot of Z.
[C,h] = contour(X,Y,Z,n);
clabel(C,h),xlabel('x_1','FontSize',18),ylabel('x_2','FontSize',18),
title(sprintf('f_1(x_1,x_2) Variable Stepsizes chosen from =

```

```
    %1.3f,%1.3f,%1.3f',tf1(1),tf1(2),tf1(3)), 'FontSize',18);
grid on

hold on;
convergence = [x0_y0' xk_yk'];
%scatter(convergence(1,:),convergence(2,:));
plot(convergence(1,:),convergence(2,:),'-ro');
```

A.4 Code for Newton-Raphson method

%New Functions used in the algorithm:

```
function h = hessian_f1(xk_yk)
format long g
h(1,1) = 4*(xk_yk(1)^2 - xk_yk(2)) + 8*xk_yk(1)^2 + 2;
h(1,2) = -4*xk_yk(1);
h(2,1) = -4*xk_yk(1);
h(2,2) = 2;
```

```
function [ispos,posh] = is_positive_h(h)
ispos = 1; posh = h; % assume given hessian in pd
N = 5000; %try making hessian positive
if min(eig(h)) < 0
    % make hessian positive
    for i = 2 : N
        if min(eig(h)) > 0
            posh = h;
            ispos = 1;
            break;
        else
            if i == N
                ispos = -1;
                disp('couldnt find positive hessian');
            end
        end
    end
    h = h + i.* eye(size(h));
end
end
```

```
%Newton-Raphson method by Narayanan Krishnamurthy
%for ee2671 fall 2005
```

```

%Gradient methods f1
clear;clc; echo off;
format long g;
% initial point
x0_y0 = [2 2];
% termination point
e = 0.05; N =1000;
% constant step size
t = 1;
[T, H ] = is_positive_h(hessian_f1(x0_y0));
if T < 0
    disp('Unable to proceed with non positive hessian')
    break;
end
xk_yk(1,:) = x0_y0 - t.*gradient_f1(x0_y0)*inv(H);
disp('iteration      x1          x2          f1(x1,x2)      Norm(gradient)');
for i = 2:N
    [T, H ] = is_positive_h(hessian_f1(xk_yk(i-1,:)));
    if T < 0
        disp('Unable to proceed with non positive hessian')
        break;
    end
    xk_yk(i,:) = xk_yk(i-1,:) - t.*gradient_f1(xk_yk(i-1,:))*inv(H);

%   if f1(xk_yk(i,:)) > f1(xk_yk(i-1,:))
%       disp('not a descent direction ... exiting');
%       disp(sprintf('The extremum could not be located after %d iterations',i));
%       break;
%   end

disp(sprintf('%-4d\t\t%3.4f\t\t\t%3.4f\t\t%3.4f\t\t%3.4f',i,xk_yk(i,1),xk_yk(i,2),
    f1(xk_yk(i,:)),norm(gradient_f1(xk_yk(i,:)))));

if norm(gradient_f1(xk_yk(i,:))) < e
    disp('-----');
    disp(sprintf('The minima of f1 found after %d iterations,',i));
    disp(sprintf('with initial pt %f %f, using Newton-Raphson(NR)
method are x1=%1.1f, x2=%1.1f,',x0_y0(1),x0_y0(2),xk_yk(i,1),xk_yk(i,2)));
    disp(sprintf('and the min value of function is =%f', f1(xk_yk(i,:))));
    disp('-----');
    break;

```

```

    end
end

n = 30; %# of contours
format long g
[X,Y] = meshgrid(-1:.02:2,-1.4:.02:4);
Z = (X.^2 - Y).^2 + (X - ones(size(X))).^2;
%Then, generate a contour plot of Z.
[C,h] = contour(X,Y,Z,n);
clabel(C,h),xlabel('x_1','FontSize',18),ylabel('x_2','FontSize',18),
title(sprintf('f_1(x_1,x_2), Newton-Raphson Initial pt %1.1f,%1.1f',
    x0_y0(1),x0_y0(2)),'FontSize',18);
grid on

hold on;
convergence = [x0_y0' xk_yk'];
%scatter(convergence(1,:),convergence(2,:));
plot(convergence(1,:),convergence(2,:),'-ro');

```

A.5 Code for Conjugate Gradient Method

```

%Conjugate Gradient method, Narayanan Krishnamurthy
%for ee2671 fall 2005

%Gradient methods f1
clear;clc; echo off;
format long g;
% initial point
x0_y0 = [2 2];
% termination point
e = 0.05; N =1000;
% constant step size
%t = 1;
[T, H ] = is_positive_h(hessian_f1(x0_y0));
if T < 0
    disp('Unable to proceed with non positive hessian')
    break;
end
r0 = gradient_f1(x0_y0);
d0 = -1.*r0;
[T, H ] = is_positive_h(hessian_f1(x0_y0));
if T < 0

```

```

    disp('Unable to proceed with non positive hessian')
    break;
end
t0 = - d0*r0'/(d0*(H*d0'));
xk_yk(1,:) = x0_y0 + t0.*d0;
r(1,:) = gradient_f1(xk_yk(1,:));
b0 = r(1,:)*r(1,:)'/(r0*r0');
%b0 = r(1,)*(H*d0')/(d0*(H*d0'));
d(1,:) = -r(1,:) + b0.*d0;
t(1) = - d(1,:)*r(1,:)'/(d(1,)*(H*d(1,:)));
disp('iteration      x1              x2      f1(x1,x2)      Norm(gradient)');
for i = 2:N

    [T, H ] = is_positive_h(hessian_f1(xk_yk(i-1,:)));
    if T < 0
        disp('Unable to proceed with non positive hessian')
        break;
    end

    xk_yk(i,:) = xk_yk(i-1,:) + t(i-1).*d(i-1,:);
    r(i,:) = gradient_f1(xk_yk(i,:));
    %b(i-1) = r(i,)*(H*d(i-1,:))'/(d(i-1,)*(H*d(i-1,:)));
    b(i-1) = r(i,:)*r(i,:)'/(r(i-1,:)*r(i-1,:));
    d(i,:) = -r(i,:) + b(i-1).*d(i-1,:);
    t(i) = - d(i,:)*r(i,:)'/(d(i,)*(H*d(i,:)));

%    if f1(xk_yk(i,:)) > f1(xk_yk(i-1,:))
%        disp('not a descent direction ... exiting');
%        disp(sprintf('The extremum could not be located after %d iterations',i));
%        break;
%    end

disp(sprintf('%-4d\t\t%3.4f\t\t\t%3.4f\t\t\t%3.4f\t\t\t%3.4f',i,xk_yk(i,1),xk_yk(i,2),
    f1(xk_yk(i,:)),norm(gradient_f1(xk_yk(i,:)))));

if norm(gradient_f1(xk_yk(i,:))) < e
    disp('-----');
    disp(sprintf('The minima of f1 found after %d iterations,',i));
    disp(sprintf('with initial pt %f %f, using Conjugate Gradient
        method are x1=%1.1f, x2=%1.1f,',x0_y0(1),x0_y0(2),xk_yk(i,1),xk_yk(i,2)));
    disp(sprintf('and the min value of function is =%f', f1(xk_yk(i,:))));
    disp('-----');

```

```

        break;

    end
end

n = 30; %# of contours
format long g
[X,Y] = meshgrid(-1:.02:2,-1.4:.02:4);
Z = (X.^2 - Y).^2 + (X - ones(size(X))).^2;
%Then, generate a contour plot of Z.
[C,h] = contour(X,Y,Z,n);
clabel(C,h),xlabel('x_1','FontSize',18),ylabel('x_2','FontSize',18),
title(sprintf('f_1(x_1,x_2), Conjugate Gradient Initial pt
    %1.1f,%1.1f',x0_y0(1),x0_y0(2)),'FontSize',18);
grid on

hold on;
convergence = [x0_y0' xk_yk'];
%scatter(convergence(1,:),convergence(2,:));
plot(convergence(1,:),convergence(2:,:),'-ro');

```

A.6 Code for Direct Search - Quadratic Approximation method

```

%Functions used by algorithm
function q_fit_f1 = quad_fit_f1(xk_yk)
m =5; d0 =[(1/sqrt(2)) (1/sqrt(2))]; t0 = 0.005;
fa = f1(xk_yk); a = 0;
fb = f1(xk_yk + t0.*d0); b = t0;

if fa > fb
fc = f1(xk_yk +2*t0.*d0); c = 2*t0;
else
fc = f1(xk_yk -t0.*d0); c= -t0;
end
%a1 = -1*((b^2 - c^2)*fa + (c^2 - a^2)*fb + (a^2 - b^2)*fc)/ ((a-b)*(b-c)*(c-a));
a2 = ((c-b)*fa + (a-c)*fb + (b-a)*fc)/((a-b)*(b-c)*(c-a));
t = ((b^2 - c^2)*fa + (c^2 - a^2)*fb + (a^2 - b^2)*fc)/(2*((b-c)*fa + (c-a)*fb +(a-b)*fc));

if (a2 <= 0) %| (abs(t) > m*tk) %| (a1 > 0)
    q_fit_f1 = -t0;
    disp('quad fit failed');
else

```

```

    q_fit_f1 = t;
end

%Direct Search-Quadratic Approximation, Narayanan Krishnamurthy
%for ee2671 fall 2005
clear,clf,clc

%Gradient methods f1
clear; echo off;
format long g;
% initial point
x0_y0 = [2 2];
% termination point
e = 0.05; N =1000;
% initial step size
t0 = 0.005;
%initial gradient vector is used as the direction for successive quadratic fit
d0 = [1/sqrt(2) 1/sqrt(2)];
t = quad_fit_f1(x0_y0);
xk_yk(1,:) = x0_y0 + t.*d0;
disp('iteration      x1          x2      f1(x1,x2)      Norm(gradient)');
for i = 2:N
    t = quad_fit_f1(xk_yk(i-1,:));
    xk_yk(i,:) = xk_yk(i-1,:) + t.*d0;

%    if f1(xk_yk(i,:)) > f1(xk_yk(i-1,:))
%        disp('not a descent direction ... exiting');
%        disp(sprintf('The extremum could not be located after %d iterations',i));
%        break;
%    end
%

disp(sprintf('%-4d\t\t%3.4f\t\t\t%3.4f\t\t\t%3.4f\t\t\t%3.4f',i,xk_yk(i,1),xk_yk(i,2),
            f1(xk_yk(i,:)),norm(gradient_f1(xk_yk(i,:)))));

if norm(gradient_f1(xk_yk(i,:))) < e
    disp('-----');
    disp(sprintf('The minima of f1 found after %d iterations',i));
    disp('using quadratic polynomial approximation to determine successive stepsize');
    disp(sprintf('with initial pt:%1.1f,%1.1f and minima are x1=%f
                %x2=%f',x0_y0(1),x0_y0(2),xk_yk(i,1),xk_yk(i,2)));
    disp(sprintf('and the min value of function is =%f', f1(xk_yk(i,:))));

```

```

        disp('-----');
        break;

    end
end

n = 30; %# of contours
format long g
[X,Y] = meshgrid(-1:.02:2,-1.4:.02:4);
Z = (X.^2 - Y).^2 + (X - ones(size(X))).^2;
%Then, generate a contour plot of Z.
[C,h] = contour(X,Y,Z,n);
clabel(C,h),xlabel('x_1','FontSize',18),ylabel('x_2','FontSize',18),
title(sprintf('f_1(x_1,x_2) quadratic fit, initial pt:%1.1f,%1.1f',
    x0_y0(1),x0_y0(2)),'FontSize',18);
grid on

hold on;
convergence = [x0_y0' xk_yk'];
%scatter(convergence(1,:),convergence(2,:));
plot(convergence(1,:),convergence(2:,:),'-ro');

```

A.7 Code for Direct Search - Golden Search Method

```

%Assume that the relative minima lies in the uncertainty region [0 0] and
%[2 2], we will use golden search method to reduce the uncertainty region
%and thus determine the minima of f1 and f2
clear;clc; echo off;
format long g;
% initial point
xk_yk(1,:) = [0 0];
xk_yk(2,:) = [2 2];
i =2; e = 0.05;
disp('iteration      x1          x2          f1(x1,x2)      Norm(gradient)');
j = 1;
while i < 1000

    a1 = 0.382.*(xk_yk(i,:)-xk_yk(i-1,:)) + xk_yk(i-1,:);
    b1 = 0.618.*(xk_yk(i,:)-xk_yk(i-1,:)) + xk_yk(i-1,:);

    if f1(a1) > f1(b1)
        xk_yk(i+1,:) = a1; xk_yk(i+2,:) = xk_yk(i,:);
    end
end

```

```

else
    xk_yk(i+1,:) = xk_yk(i-1,:); xk_yk(i+2,:) = a1;
end

xkp_ykp(j,:) = xk_yk(i,:);
i = i + 2;
j = j + 1;
disp(sprintf('%-4d\t\t%3.4f\t\t%3.4f\t\t%3.4f\t\t%3.4f', (i-2)/2, xk_yk(i,1),
    xk_yk(i,2), f1(xk_yk(i,:)), norm(gradient_f1(xk_yk(i,:)))));

if norm(gradient_f1(xk_yk(i,:))) < e
    disp('-----');
    disp(sprintf('The minima of f1 found after %d iterations,', (i-2)/2));
    disp(sprintf('with initial pt %f %f, using Golden Search
method are x1=%1.1f, x2=%1.1f,', xk_yk(2,1), xk_yk(2,2), xk_yk(i,1), xk_yk(i,2)));
    disp(sprintf('and the min value of function is =%f', f1(xk_yk(i,:))));
    disp('-----');
    break
end
end

n = 30; %# of contours
format long g
[X,Y] = meshgrid(-1:.02:2,-1.4:.02:4);
Z = (X.^2 - Y).^2 + (X - ones(size(X))).^2;
%Then, generate a contour plot of Z.
[C,h] = contour(X,Y,Z,n);
xlabel(C,h), xlabel('x_1', 'FontSize', 18), ylabel('x_2', 'FontSize', 18),
title(sprintf('f_1(x_1,x_2), Golden-Search Initial pt
    %1.1f,%1.1f', xk_yk(2,1), xk_yk(2,2)), 'FontSize', 18);
grid on
hold on;

convergence = [xkp_ykp'];
%scatter(convergence(1,:),convergence(2,:));
plot(convergence(1,:),convergence(2,:), '-ro');

```