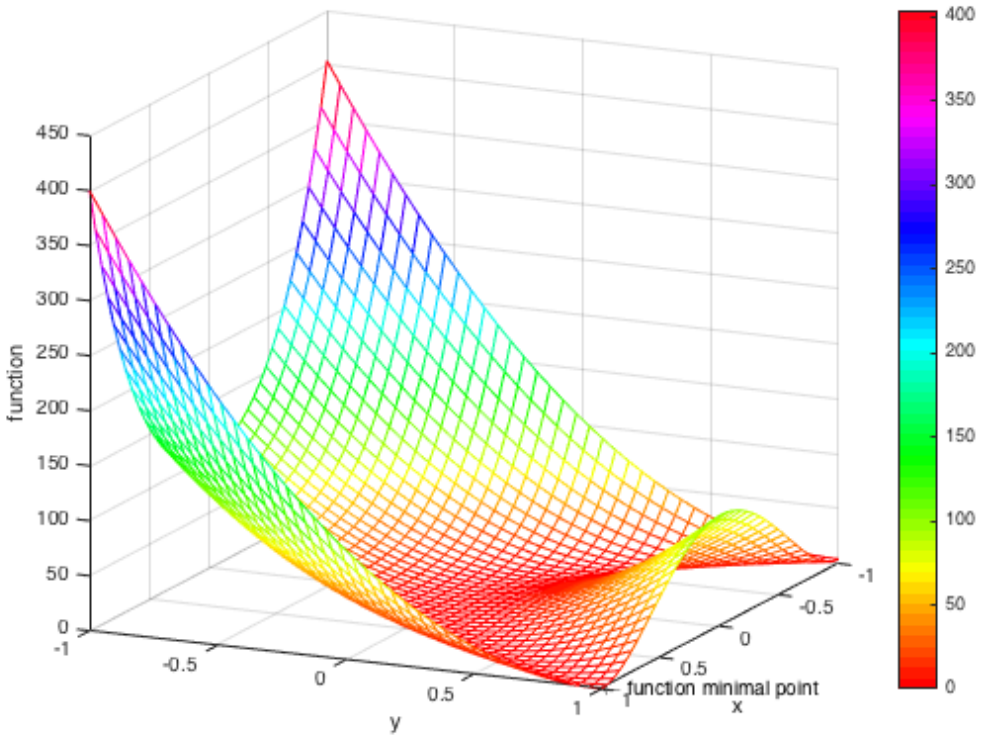
CSC466/2305 – Assignment 3
Sooa Lim, 994500731 (c5limsoo)

Question 1. Program the steepest descent, conjugate gradient (with Polka and Ribiere β_k), BFGS and Newton algorithms using the inexact line search algorithm that was summarized Feb. 3rd.
Use your program to minimize **Rosenbrock function**.

$$f(x) = 100(x_2 - {x_1}^2)^2 + (1 - x_1)^2$$

Try the initial point x_0 = (1.2, 1.2) and then x_0 = (-1.2, 1)
Print iterates and the step lengths used by each method on each iterates. (The first and last 10 iterates). In addition for each initial guess, generate a plot that shows the iterations $\{x_k\}$ generated by each algorithm. Comment on observations.




```
f = @(x, y) 100*(y - x.^2).^2 + (1-x).^2;
```

Question 2. Repeat question 1 on the Extended Rosenbrock Function.

$$f(x) = \sum_{i=1}^{n} f_i(x)^2$$

where n is a positive multiple of 2 and

$$x = (x_1, x_2, \dots, x_n)^T$$
$$f_{2i-1}(x) = 10\,(x_{2i} - {x^2}_{2i-1}), \qquad i = 1, 2, \dots, n/2$$
$$f_{2i}(x) = 1 - x_{2i-1}, \qquad i = 1, 2, \dots, n/2$$

Try n = 4, 8, 16
Try the initial conditions $x_0 = (1.2, 1.2, \dots, 1.2)^T$ and $x_0 = (-1.2, 1, -1.2, 1\ \dots, -1.2, 1)^T$
Print out the step lengths, but do not plot the iterates. Comment on your observations.

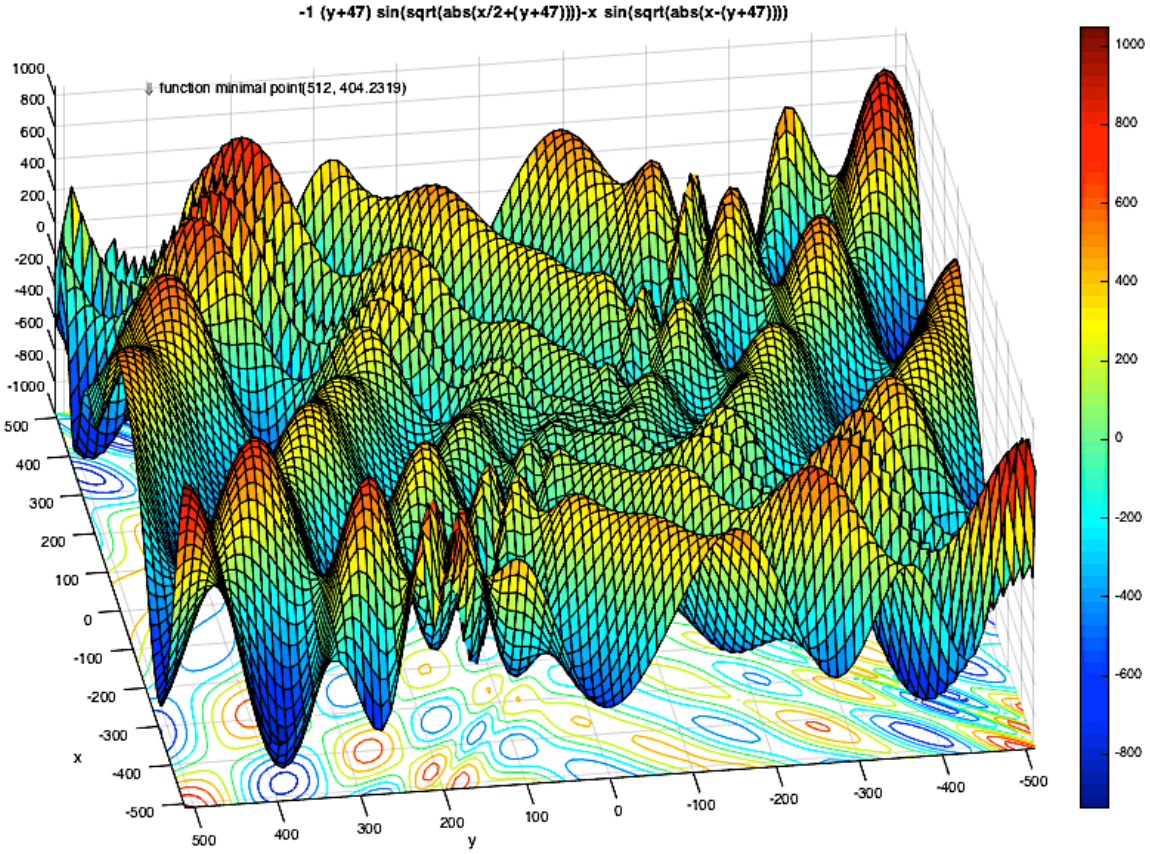
Question 3. Repeat question 1 for one of the Test functions for optimization that are listed on

the Wikipedia page https://en.wikipedia.org/wiki/Test_functions_for_optimization. Choose a function that appears interesting to you. As always, comment on your observations.

Eggholder function:

$$f(x,y) = -(y+47)\sin\sqrt{\left|\frac{x}{2}+(y+47)\right|} - x\sin\sqrt{|x-(y+47)|}$$

```
egg = @(x,y)-1*(y+47)*sin(sqrt(abs(x/2+(y+47))))-x*sin(sqrt(abs(x-(y+47))))
```



Index:

- Code presented at the end of the assignment (Last 11 pages)

- Print iterate x_k, step lengths $||x_{k+1} - x_k||_2$ (or first 10 and last 10 iterates)

- Generate a plot that shows the iterate paths (x_k) by each algorithm.

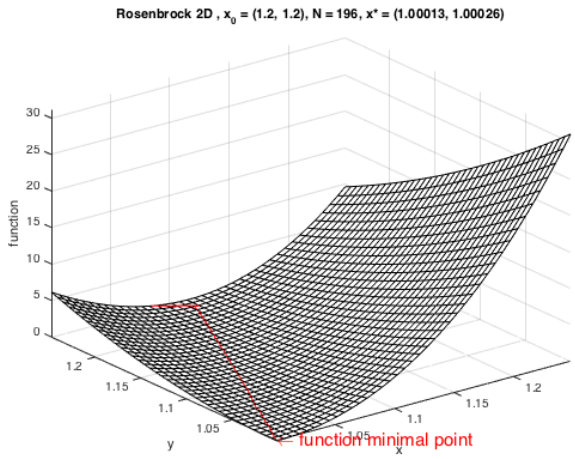
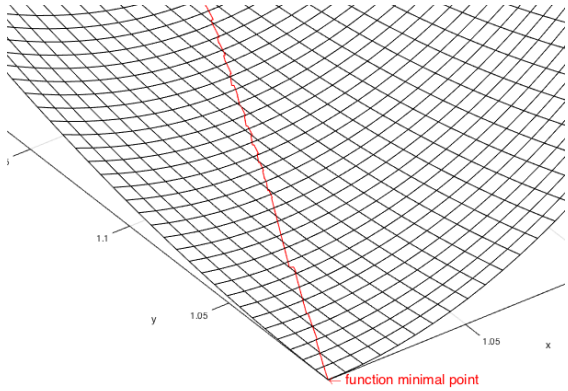
(4 algorithms x 2 initial points x 3 different functions = 24 plots)

Summary of each method

	intial point	total_iter	tol > step length	minima x*	algorithm
Question 1 Rosenbrock 2 Dimension	(1.2, 1.2)	196	1.00E-06	1.0001, 1.0002	SD
	(-1.2, 1)	872	1.00E-06	0.9994, 0.9987	SD
	(1.2, 1.2)	50	1.00E-06	1.0000, 1.0000	CG
	(-1.2, 1)	111	1.00E-06	1.0000, 1.0000	CG
	(1.2, 1.2)	31	1.00E-05	0.9999,1.0000	BFGS
	(-1.2, 1)	53*	1.00E-05	1.0001, 1.0003	BFGS
	(1.2, 1.2)	n/a	n/a		Newton IE
	(-1.2, 1)	n/a	n/a		Newton IE
Question 3 Egg Holder	(1.2, 1.2)	83	1.00E-06	8.4569, 15.6509	SD
	(-1.2, 1)	89	1.00E-06	8.4569, 15.6509	SD
	(500, 400)	11	1.00E-06	511.3985, 403.4635	SD
	(1.2, 1.2)	45	1.00E-06	8.4521, 15.6524	CG
	(-1.2, 1)	52	1.00E-06	8.4523, 15.6523	CG
	(500, 400)	12	1.00E-06	510.6847, 403.8371	CG
	(1.2, 1.2)	27	1.00E-04	8.4570, 15.6510	BFGS
	(-1.2, 1)	26	1.00E-03	8.4563, 15.6503	BFGS
	(500, 400)	42	1.00E-03	511.532, 402.2578	BFGS
	(1.2, 1.2)	n/a	n/a		Newton IE
	(-1.2, 1)	n/a	n/a		Newton IE

Question 1 – Steepest Descent

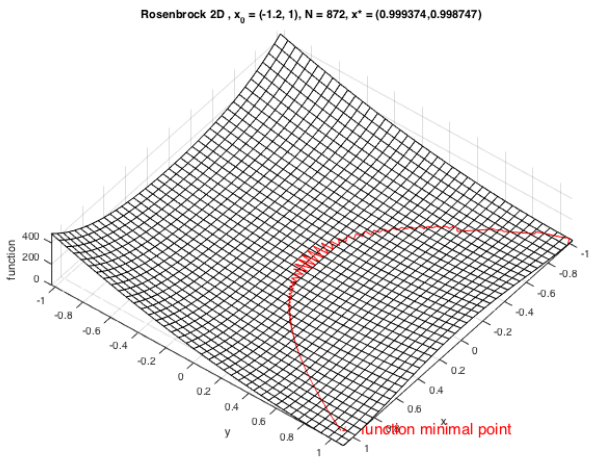
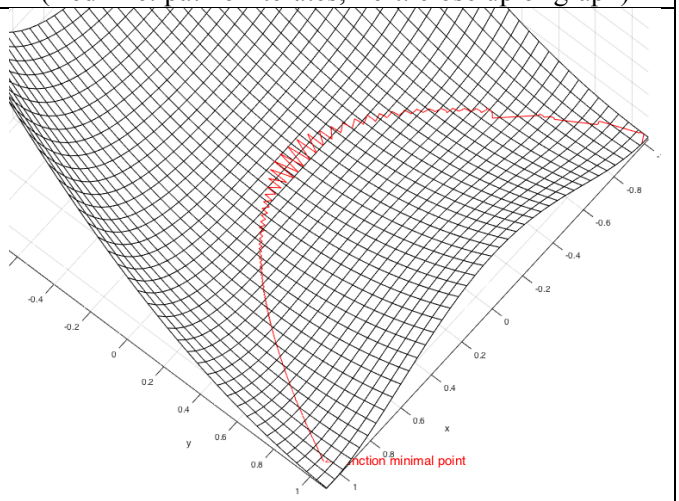
[x, f_k, x, y, step_length,z]=stepest_descent([1.2, 1.2], f, 1e-6, 1000, 0.1, 'plot')

Rosenbrock 2D graph x0 = (1.2, 1.2)	(Red line: path of iterates, Left: close up of graph)
	

Iterations First 10			Iterations Last 10		
1	1.08440000000000	1.24800000000000	187	1.00039769480783	1.00080455687087
2	1.11549516336640	1.23358467200000	188	1.00040050448898	1.00080275505206
3	1.11046986964736	1.23573362949877	189	1.00023401763815	1.00048562100210
4	1.11139950910729	1.23521556987794	190	1.00024056362851	1.00048211480979
5	1.11117968909944	1.23521422967113	191	1.00022964712490	1.00046352117170
6	1.11115146727011	1.23422637326312	192	1.00023085788772	1.00046268633488
7	1.11073750850660	1.23431261525380	193	1.00013613244157	1.00027923352175
8	1.11077141534622	1.23419765476373	194	1.00013864059777	1.00027784350043
9	1.11026443173432	1.23342861953677	195	1.00013263883734	1.00026698182674
10	1.11037321225860	1.23328031730428	196	1.00013304827274	1.00026664451494

Step length First 10		Step length Last 10	
1	0.125169325315750	187	3.74627267253802e-05
2	0.0342740552915859	188	3.33779256258825e-06
3	0.00546549131315104	189	0.000358178554841420
4	0.00106424400218498	190	7.42585849323119e-06
5	0.000219824093320096	191	2.15613874441034e-05
6	0.000988259457075136	192	1.47067982126427e-06
7	0.000422846944931804	193	0.000206465117623247
8	0.000119856531110376	194	2.86757855181766e-06
9	0.000921112133812724	195	1.24095561440205e-05
10	0.000183920511703002	196	5.30487136685475e-07

[x, f_k, x, y, step_length,z]=stepest_descent([-1.2, 1], f, 1e-6, 1000, 0.1, 'plot')

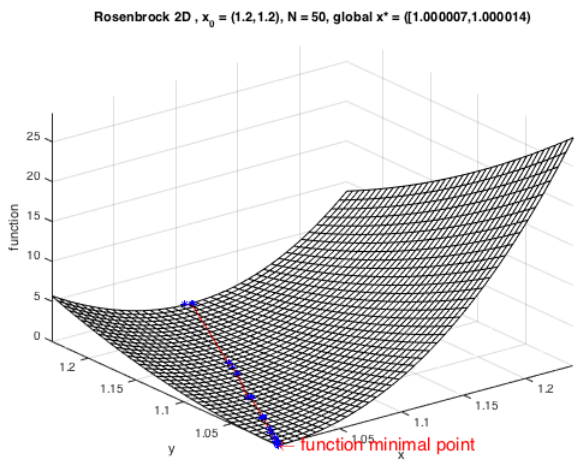
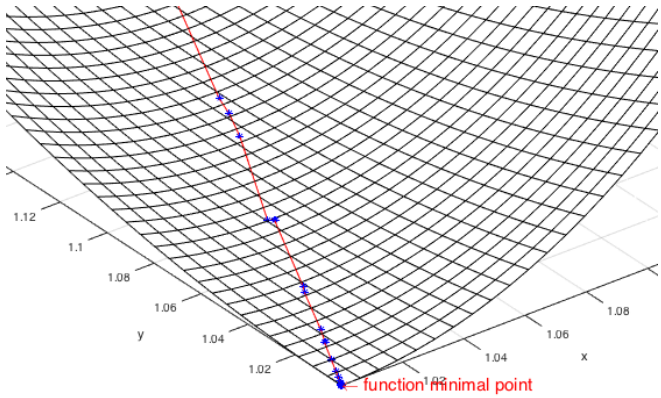
Rosenbrock 2D graph x0 = (-1.2, 1)	(Red line: path of iterates, Left: close up of graph)
	

Iterations First 10			Iterations Last 10		
1	-0.984400000000000	1.08800000000000	863	0.996923696737901	0.994236539798474
2	-1.02727156656640	1.06420867200000	864	0.997081255209162	0.994160603262291
3	-1.02338658323948	1.04636507095157	865	0.997098046922219	0.994181455716664
4	-1.01894886501691	1.04655607651217	866	0.997094653811586	0.994186067608552
5	-0.953421054488740	0.880570336653876	867	0.998246500749168	0.996522277827636
6	-0.938667571802845	0.886258610751585	868	0.998260469997768	0.996517037513701
7	-0.936728322517279	0.885226250672118	869	0.998972972313231	0.997902726806319
8	-0.840379202324189	0.729900241350455	870	0.998957335513877	0.997911581327522
9	-0.844652813801966	0.725167633820172	871	0.999373666722331	0.998746951219921
10	-0.844926368469100	0.722821782228853	872	0.999374609775683	0.998747106123545

Step length First 10		Step length Last 10	
1	0.232867687754227	863	0.00438863365987569
2	0.0490305874721165	864	0.000174902914193497
3	0.0182616317405074	865	2.67728683698055e-05
4	0.00444182689296714	866	5.72562193588882e-06
5	0.178452121839228	867	0.00260473214656924
6	0.0158120749294221	868	1.49198122165668e-05
7	0.00219691946262201	869	0.00155813810851876
8	0.182782171268907	870	1.79697534701116e-05
9	0.00637662364411406	871	0.000933367308331104
10	0.00236174762504455	872	9.55690722194905e-07

Question 1 – Conjugate Gradient

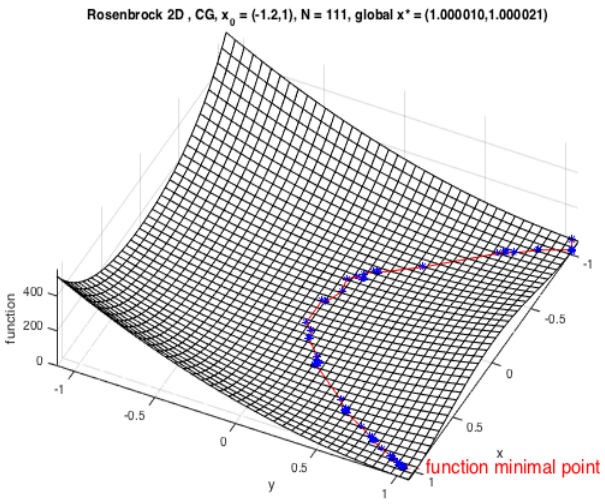
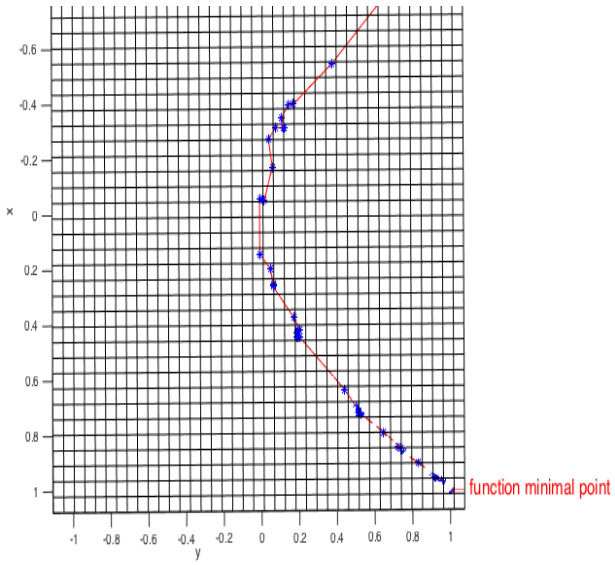
[x, f_k, x, y, step_length,z]=conjugate_gradient([1.2, 1.2], f, 1e-6, 1000, 0.1, 'plot')

Rosenbrock 2D graph x0 = (1.2, 1.2)	(Red line: path of iterates, Left: close up of graph)
	

Iterations First 10			Iterations Last 10		
1	1.10764510417517	1.23834805362969	41	1.00005611018319	1.00011260455194
2	1.11247049566517	1.23607115238022	42	1.00005613311541	1.00011249380339
3	1.11247049566517	1.23607115238022	43	1.00003365375911	1.00006754583319
4	1.11156941675734	1.23637504264927	44	1.00003366793937	1.00006747162636
5	1.11149371735471	1.23628585643904	45	1.00002018662271	1.00004051159236
6	1.11164081928022	1.23609392569169	46	1.00002019492676	1.00004047100654
7	1.06555490370160	1.13742906984025	47	1.00001210818726	1.00002430051780
8	1.06280544995174	1.12957422498279	48	1.00001211323542	1.00002427505098
9	1.05775492208749	1.11846810823699	49	1.00000726289307	1.00001457561741
10	1.03701023911782	1.07965635137625	50	1.00000726588961	1.00001456088388

Step length First 10		Step length Last 10	
1	0.100000000000000	41	8.37914043656360e-05
2	0.00533560515139008	42	1.13097873064957e-07
3	0	43	5.02557607053766e-05
4	0.000950942949786006	44	7.55495453830762e-08
5	0.000116981108109315	45	3.01428156013614e-05
6	0.000241818916281645	46	4.14266332032083e-08
7	0.108897499488764	47	1.80798247148023e-05
8	0.00832214417426431	48	2.59623338722806e-08
9	0.0122005598592487	49	1.08445761742986e-05
10	0.0440078895441410	50	1.50351672106646e-08

[x, f_k, x, y, step_length,z]=conjugate_gradient([-1.2, 1], f, 1e-6, 1000, 0.1, 'plot')

Rosenbrock 2D graph x0 = (-1.2, 1)	(Red line: path of iterates, Left: close up of graph)
	

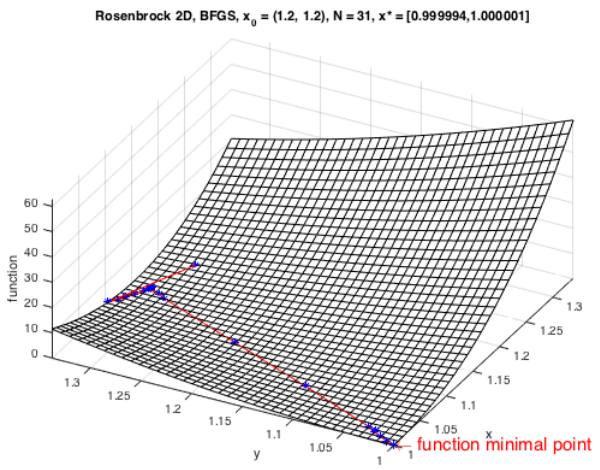
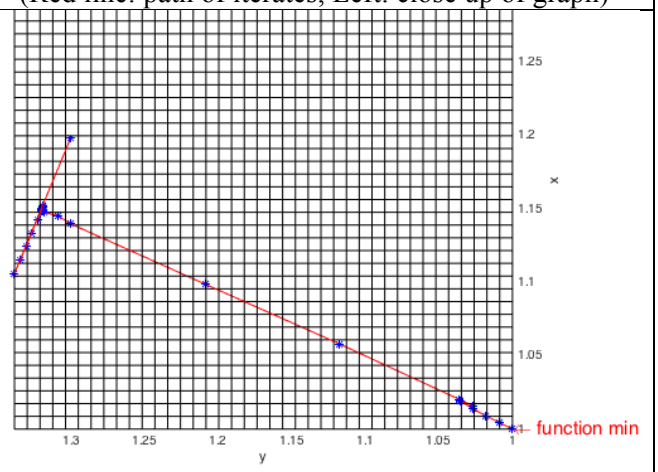
Iterations First 10			Iterations Last 10		
1	-1.10741523563048	1.03778969974266	102	1.00008430202536	1.00016866992632
2	-1.01989031202824	1.07541406335142	103	1.00008411505422	1.00016857403358
3	-1.01989031202824	1.07541406335142	104	1.00005054712297	1.00010115838751
4	-1.03022601376961	1.06836650039495	105	1.00005045519574	1.00010111500357
5	-0.949820939133800	0.903023849480304	106	1.00003032361644	1.00006067728211
6	-0.944242068425811	0.898722879417010	107	1.00003026260834	1.00006064732579
7	-0.870656303067592	0.791568991288537	108	1.00001818607415	1.00003639388816
8	-0.855673107979409	0.751491101310959	109	1.00001815244931	1.00003637792499
9	-0.855673107979410	0.751491101310960	110	1.00001090918376	1.00002183017690
10	-0.858572566785448	0.747628174592596	111	1.00001088815447	1.00002182003173

Step length First 10		Step length Last 10	
1	0.100000000000000	102	0.000125578806303252
2	0.0952691187559895	103	2.10127641964921e-07
3	3.14018491736755e-16	104	7.53105261123008e-05
4	0.0125097911298375	105	1.01650291154277e-07
5	0.183856379379865	106	4.51717810382636e-05
6	0.00704429853582390	107	6.79659372519352e-08
7	0.129987770980212	108	2.70937615519510e-05
8	0.0427870704781877	109	3.72216691341127e-08
9	8.67111901826274e-16	110	1.62512113512953e-05
10	0.00483001699783358	111	2.33485602233628e-08

Question 1 – BFGS

Note: The performance depends on the initial hessian value. When B_0 is set to the identity matrix eye(2,2) the algorithm would track the right path but would never converge and iterate infinitely back and forth.
So I had to calculate the initial hessian for the first step and it converged nicely in 31 steps.

[x, f_k, x, y, step_length,z]=BFGS([1.2,1.2], f, 1e-5, 100, 0.1, 'plot')

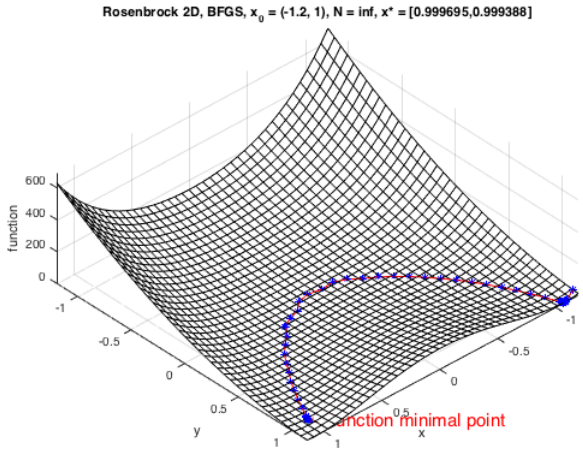
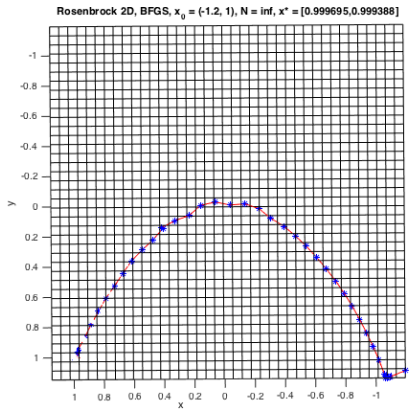
Rosenbrock 2D graph x0 = (1.2, 1.2)	(Red line: path of iterates, Left: close up of graph)
	

Iterations			Steplength	
1	1.19822722898617	1.29998428517988	1	0.100000000000000
2	1.10585700515258	1.33829540300397	2	0.100000000000000
3	1.11502649951927	1.33430536878618	3	0.00999999999999995
4	1.12423702148244	1.33041097728482	4	0.00999999999999999
5	1.13347380129037	1.32657928069194	5	0.0100000000000001
6	1.14271859486250	1.32276695963926	6	0.00999999999999992
7	1.15193023915974	1.31887522360746	7	0.0100000000000000
8	1.15099887583163	1.31923931473761	8	0.000999999999999987
9	1.15005533785257	1.31957057910647	9	0.000999999999999909
10	1.14909374418115	1.31984505608381	10	0.00100000000000000
11	1.14809832865915	1.31994070094021	11	0.00100000000000001
12	1.14819447139804	1.31991319505433	12	0.000100000000000047
13	1.14828134364819	1.31986366514262	13	9.99999999998545e-05
14	1.14835017082444	1.31979111984774	14	0.000100000000000075
15	1.14839396951638	1.31970122171325	15	0.000100000000000118
16	1.14841372403926	1.31960319233589	16	9.99999999999127e-05
17	1.14843620448197	1.31860344505298	17	0.00100000000000002
18	1.14837780751899	1.31868462260353	18	0.000100000000000031
19	1.14756344829029	1.31810426146886	19	0.00100000000000003
20	1.14493742682616	1.30845521948170	20	0.0100000000000000
21	1.13961931976233	1.29998658693225	21	0.0100000000000000
22	1.09863294491022	1.20877191115865	22	0.100000000000000
23	1.05795944385958	1.11741729270789	23	0.100000000000000
24	1.01588911627642	1.02669746280811	24	0.100000000000000
25	1.01978612732290	1.03590687673645	25	0.00999999999999990
26	1.01900556888397	1.03528179392321	26	0.000999999999999987
27	1.01377537818547	1.02675858412541	27	0.00999999999999998
28	1.00904770652062	1.01794671028216	28	0.00999999999999994
29	1.00450128164234	1.00903997009137	29	0.0100000000000001
30	1.00003103053342	1.00009475600656	30	0.00999999999999995
31	0.999994552391982	1.00000164668802	31	9.99999999998983e-05

[x, f_k, x, y, step_length,z]=BFGS([-1.2,1], f, 1e-5, 100, 0.1, 'plot')

The algorithm wouldn't converge but started oscillating between two points (1.0001, 1.0003) and (0.9997, 0.9994)

The number of iteration when this started happening was around step 53 so roughly speaking we could say the algorithm converged at **step 53.** I think tweaking the stop condition should allow the algorithm to converge, but I left it at this point.

Rosenbrock 2D graph x0 = (-1.2, 1)	(Red line: path of iterates, Left: close up of graph)
	

Iterations First 10			Iterations Last 10		
1	-1.19352014050257	1.09978983626048	91	1.00014153768733	1.00028370519946
2	-1.10091063592837	1.13751886570098	92	0.999695064688507	0.999388908093228
3	-1.09163045362584	1.14124421352793	93	1.00014153768733	1.00028370519946
4	-1.08219514638214	1.14455706384406	94	0.999695064688507	0.999388908093228
5	-1.07252338151355	1.14709811959446	95	1.00014153768733	1.00028370519946
6	-1.06252589579877	1.14732235024079	96	0.999695064688507	0.999388908093228
7	-1.06345601074673	1.14695508169656	97	1.00014153768733	1.00028370519946
8	-1.06412530619835	1.14621208533153	98	0.999695064688507	0.999388908093228
9	-1.06729555625643	1.13672791354011	99	1.00014153768733	1.00028370519946
10	-1.06642877468851	1.13722660153226	100	0.999695064688507	0.999388908093228

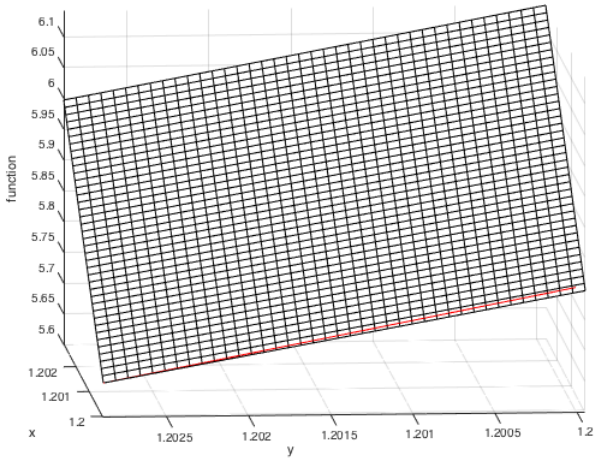
Step length First 10		Step length Last 10	
1	0.100000000000000	91	0.00100000000000003
2	0.100000000000000	92	0.00100000000000003
3	0.0100000000000000	93	0.00100000000000003
4	0.00999999999999999	94	0.00100000000000003
5	0.0100000000000000	95	0.00100000000000003
6	0.0100000000000001	96	0.00100000000000003
7	0.000999999999999997	97	0.00100000000000003
8	0.000999999999999994	98	0.000999999999999928
9	0.0100000000000001	99	0.000999999999999928
10	0.000999999999999941	100	0.00100000000000003

Question 1 – Inexact lines search with Newton method

Couldn't get the method working. Search was direction was diverging.

[x, f_k, x, y, step_length,z]=Newton_inexact([1.2,1.2], f, 1e-4, 1000, 0.1, 'plot');

Rosenbrock 2D graph x0 = (1.2, 1.2)	(Red line: path of iterates)

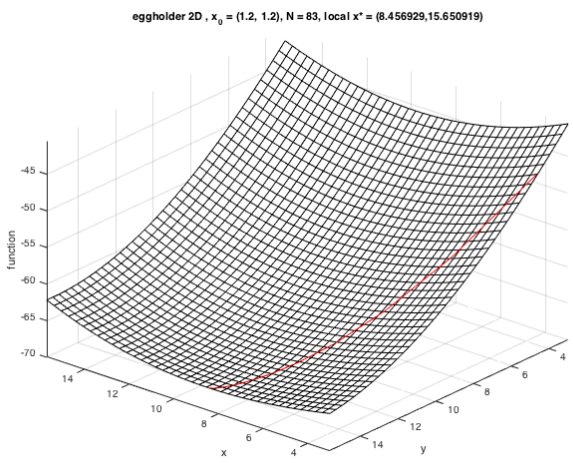
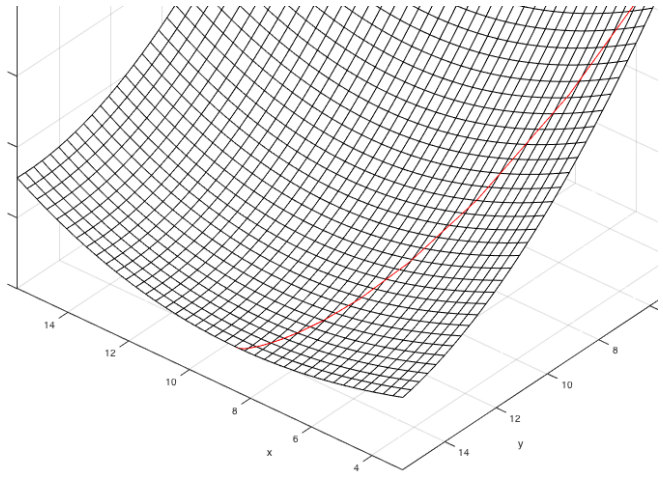


Iterations		
1	-1.19999999009042	1.00000015260752
2	-1.19999998018084	1.00000030521491
3	-1.19999997027126	1.00000045782217
4	-1.19999996036168	1.00000061042930
5	-1.19999995045210	1.00000076303631
6	-1.19999994054251	1.00000091564319
7	-1.19999993063293	1.00000106824994
8	-1.19999992072335	1.00000122085657
9	-1.19999991081376	1.00000137346307
10	-1.19999990090417	1.00000152606944
	…	
991	-1.19999017920724	1.00015117173013
992	-1.19999016929685	1.00015132421176
993	-1.19999015938647	1.00015147669326
994	-1.19999014947608	1.00015162917463
995	-1.19999013956569	1.00015178165588
996	-1.19999012965530	1.00015193413700
997	-1.19999011974491	1.00015208661800
998	-1.19999010983452	1.00015223909886
999	-1.19999009992413	1.00015239157960
1000	-1.19999009001373	1.00015254406021

Steplength	
1	1.52928917086935e-07
2	1.52928790401902e-07
3	1.52928663480903e-07
4	1.52928536795871e-07
5	1.52928410110840e-07
6	1.52928283425809e-07
7	1.52928156726390e-07
8	1.52928030041360e-07
9	1.52927903134753e-07
10	1.52927776435336e-07
	…
990	1.52803600081516e-07
991	1.52803473397017e-07
992	1.52803346698117e-07
993	1.52803220013619e-07
994	1.52803093329122e-07
995	1.52802966630223e-07
996	1.52802839945727e-07
997	1.52802713261231e-07
998	1.52802586562334e-07
999	1.52802459877839e-07
1000	1.52802333193345e-07

Question 3 – Steepest Descent

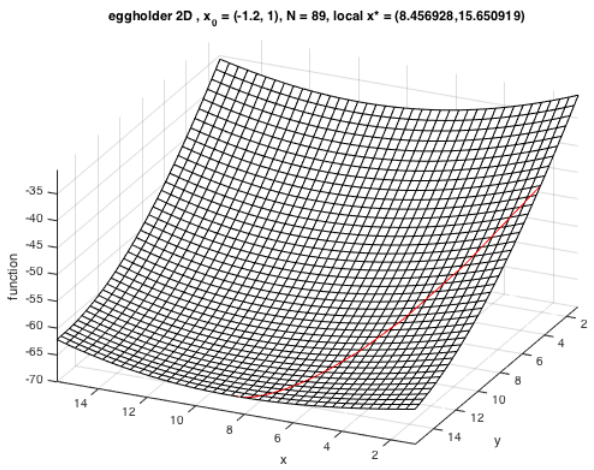
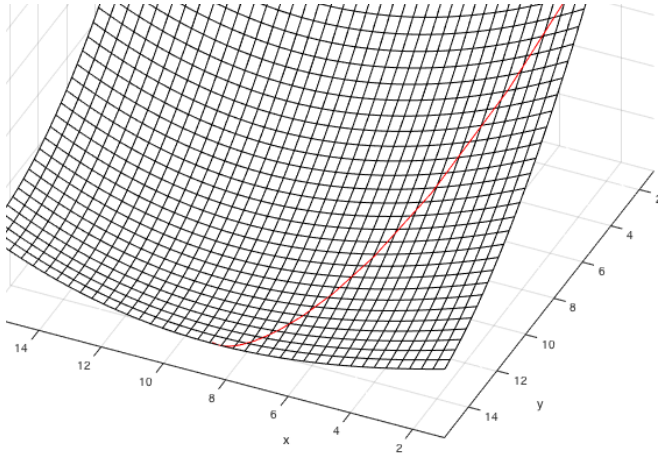
[x, f_k, x, y, step_length,z]=steepest_descent([1.2, 1.2], egg, 1e-6, 1000, 0.1, 'plot')

Egg holder 2D graph x0 = (1.2, 1.2)	(Red line: path of iterates, Left: close up of graph)
	

Iterations First 10			Iterations Last 10		
1	2.98466199858123	4.55274058786790	74	8.45691458078695	15.6509244199225
2	4.40839515040496	7.47208595312915	75	8.45691735275107	15.6509235438089
3	5.48034221002730	9.79557985224969	76	8.45691973830512	15.6509227898234
4	6.26083502941130	11.5361402800137	77	8.45692179131439	15.6509221409419
5	6.82177091397611	12.7934334727011	78	8.45692355813539	15.6509215825134
6	7.22485276669358	13.6830497694039	79	8.45692507866257	15.6509211019290
7	7.51645860020946	14.3050974110314	80	8.45692638722911	15.6509206883375
8	7.72956688606066	14.7368853705964	81	8.45692751338220	15.6509203324001
9	7.88715350029004	15.0350539642727	82	8.45692848255015	15.6509200260802
10	8.00515620292560	15.2400421342752	83	8.45692931661660	15.6509197624609

Step length First 10		Step length Last 10	
1	3.79814269067352	74	3.37801467644339e-06
2	3.24801376987142	75	2.90712230871219e-06
3	2.55884630212198	76	2.50187176258082e-06
4	1.90754282882671	77	2.15311269025855e-06
5	1.37674806663037	78	1.85297028079416e-06
6	0.976674016931635	79	1.59466744262651e-06
7	0.687005990217536	80	1.37237179953798e-06
8	0.481514260976474	81	1.18106399051006e-06
9	0.337250724594080	82	1.01642434572401e-06
10	0.236526505217228	83	8.74735352424492e-07

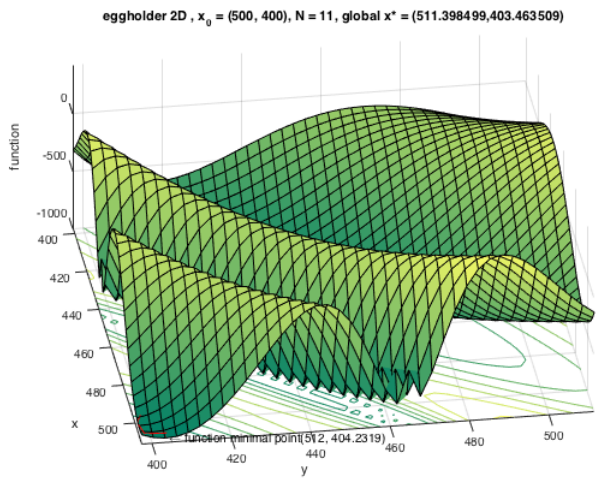
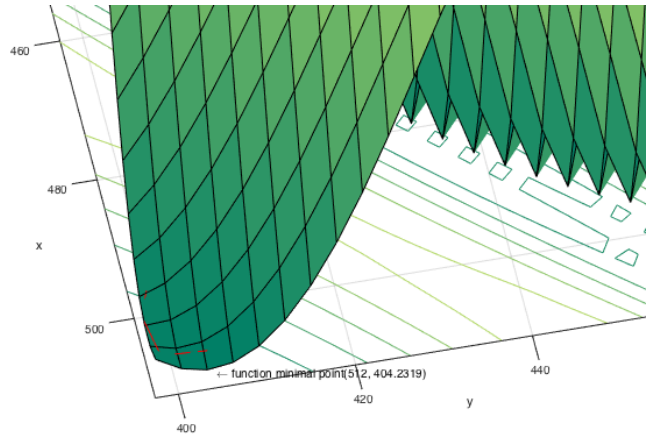
[x, f_k, x, y, step_length,z]=steepest_descent([-1.2, 1], egg, 1e-6, 1000, 0.1, 'plot')

Egg holder graph x0 = (-1.2, 1)	(Red line: path of iterates, Left: close up of graph)
	

Iterations First 10			Iterations Last 10		
1	0.968343692059938	4.37623631225415	80	8.45691295273769	15.6509249345395
2	2.73353688218167	7.38348634943849	81	8.45691595164834	15.6509239866818
3	4.08927397528734	9.80277658298020	82	8.45691853251350	15.6509231709554
4	5.09717155457630	11.6165673125587	83	8.45692075360819	15.6509224689410
5	5.83929489952764	12.9188826856236	84	8.45692266508398	15.6509218647873
6	6.38845094737415	13.8305845033261	85	8.45692431010142	15.6509213448524
7	6.79989333703428	14.4589899828167	86	8.45692572580455	15.6509208973964
8	7.11296137365851	14.8873708594036	87	8.45692694415958	15.6509205123156
9	7.35505185761298	15.1765806387857	88	8.45692799267666	15.6509201809149
10	7.54519701403775	15.3698256480207	89	8.45692889503108	15.6509198957114

Step length First 10		Step length Last 10	
1	4.01256601230180	80	3.65458487359497e-06
2	3.48704169527658	81	3.14513898405964e-06
3	2.77326311404686	82	2.70670919550719e-06
4	2.07501670861739	83	2.32939600959685e-06
5	1.49892374390525	84	2.00467987884789e-06
6	1.06431788920818	85	1.72522886135315e-06
7	0.751118024456259	86	1.48473306599049e-06
8	0.530586252159912	87	1.27776218465535e-06
9	0.377160574439501	88	1.09964288496023e-06
10	0.271106647107814	89	9.46353284170094e-07

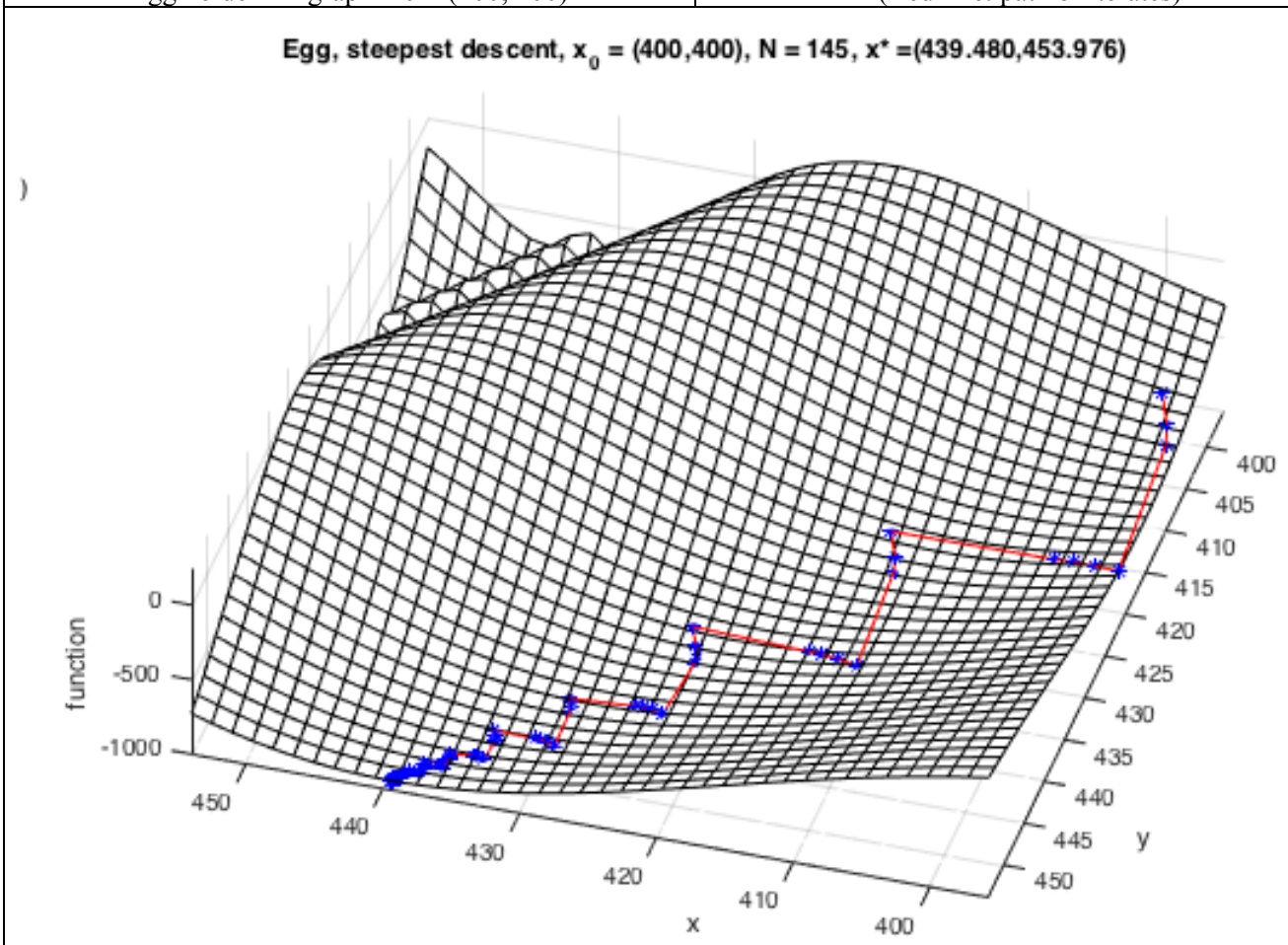
[x, f_k, x, y, step_length,z]=steepest_descent([500, 400], egg, 1e-6, 1000, 0.1, 'plot')

Egg holder 2D graph x0 = (500, 400)	(Red line: path of iterates, Left: close up of graph)
	

Iterations			Step length	
1	502.074476734689	398.483592284829	1	2.56961983207579
2	503.383943296493	397.752299489096	2	1.49983060029220
3	504.254751381207	397.464255149674	3	0.917211132660847
4	504.878008588182	397.421391405074	4	0.624729418746823
5	509.689623477378	398.357170745737	5	4.90176711159787
6	509.361726577104	399.183516267336	6	0.889023789487950
7	509.254900461777	399.775064279589	7	0.601116352893651
8	509.548009682094	404.212000116796	8	4.44660686799871
9	510.417547717771	403.708320758124	9	1.00488272541728
10	510.991276462459	403.503412704800	10	0.609222441147383
11	511.398499518855	403.463509561848	11	0.409173408811576

Example of falling into a local minimal even when the initial point is close to the global minima. x*= (512, 404.2319)
[x, f_k, x, y, step_length,z]=steepest_descent([400, 400], egg, 1e-4, 100, 0.1, 'plot')

Egg holder 2D graph x0 = (400, 400)	(Red line: path of iterates)

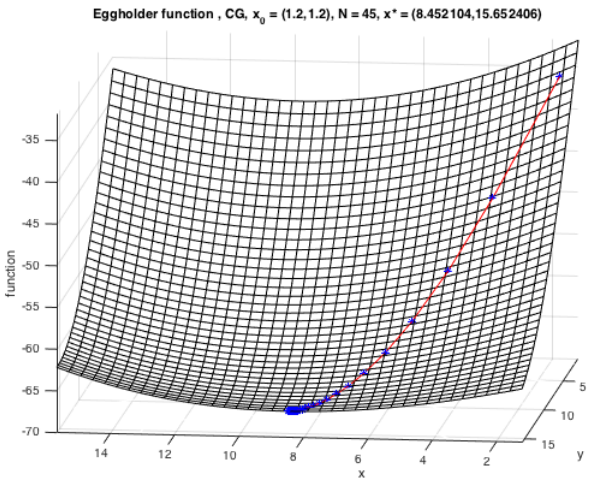
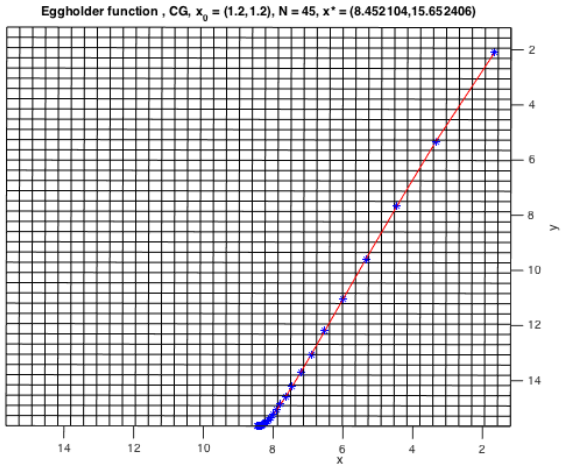


Iterations First 10			Iterations Last 10		
1	398.021249659772	403.320587496003	136	439.478144428268	453.974006604909
2	396.913891684574	405.788825597063	137	439.478083650309	453.974194572461
3	396.435217385268	407.630536235782	138	439.478066029432	453.974335241598
4	395.667826050543	421.973777949333	139	439.478161704757	453.975435422546
5	397.596485196802	421.235356340416	140	439.478341100799	453.975345485490
6	399.267791372516	420.758240287117	141	439.478472288023	453.975303778599
7	400.692713799216	420.526654942027	142	439.478572319161	453.975292611788
8	412.807308605041	420.298285710964	143	439.479370205525	453.975373255003
9	411.930673331430	422.255491137725	144	439.479317434450	453.975501356990
10	411.565146545993	423.683897397044	145	439.479296156514	453.975595291485

Step length First 10		Step length Last 10	
1	3.86545653546482	136	0.000291354391164490
2	2.70526172647975	137	0.000197549388424575
3	1.90289967197081	138	0.000141768477810509
4	14.3637555087090	139	0.00110433323126437
5	2.06518584513964	140	0.000200677885042881
6	1.73807481464215	141	0.000137657374397102
7	1.44361895740294	142	0.000100652502566273
8	12.1167470805911	143	0.000801951356129877
9	2.14456114053224	144	0.000138545679966027
10	1.47443354293567	145	9.63142767087629e-05

Question 3 – Conjugate Gradient

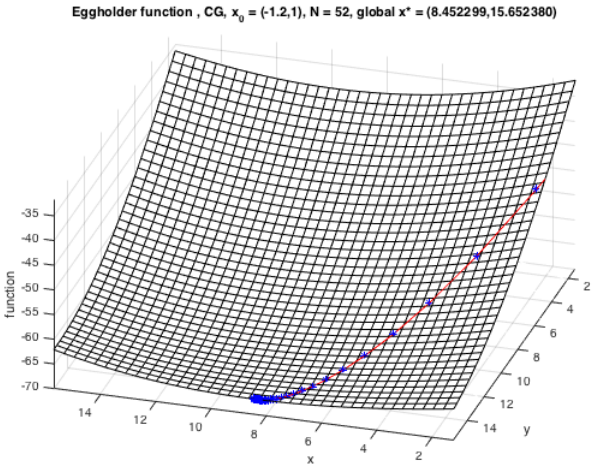
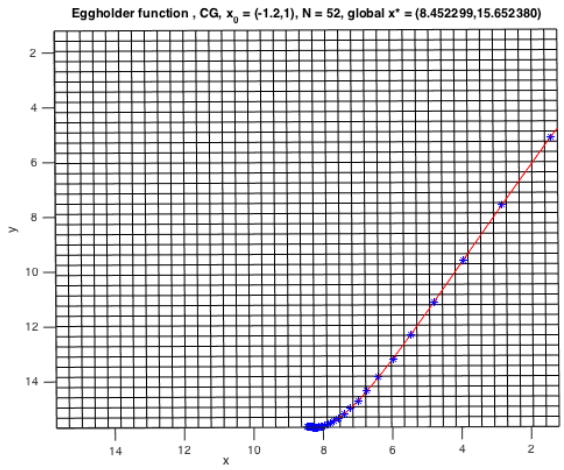
[x, f_k, x, y, step_length,z]=conjugate_gradient([1.2, 1.2], egg, 1e-6, 1000, 0.1, 'plot')

Egg holder graph x0 = (1.2, 1.2)	(Red line: path of iterates, Left: close up of graph)
	

Iterations First 10			Iterations Last 10		
1	1.66987755435401	2.08273160355472	36	8.44090265657240	15.6553607037798
2	3.35520383646928	5.32606770593951	37	8.44291329656125	15.6548909863371
3	4.46201286839494	7.66439320739494	38	8.44466873143639	15.6544579674467
4	5.34950267548582	9.59145933619121	39	8.44620211702324	15.6540628112033
5	6.01260795209067	11.0630358277087	40	8.44754211773642	15.6537050238922
6	6.52375616531179	12.2002124502018	41	8.44871355605947	15.6533830592014
7	6.91502715636819	13.0628988709794	42	8.44973796021547	15.6530947434930
8	7.21799613581182	13.7173137302707	43	8.45063402702552	15.6528375706411
9	7.45363006802989	14.2107756815810	44	8.45141801416621	15.6526089029281
10	7.63841559413904	14.5822888677776	45	8.45210407359960	15.6524061047428

Step length First 10		Step length Last 10	
1	1	36	0.00235847004077655
2	3.65507233173586	37	0.00206477786715424
3	2.58704317395536	38	0.00180805336213703
4	2.12160835745998	39	0.00158348344313381
5	1.61407743873955	40	0.00138694400438074
6	1.24677310150008	41	0.00121487818601611
7	0.947270209093655	42	0.00106420384443717
8	0.721144237005525	43	0.000932241172575732
9	0.546834570417358	44	0.000816654614696385
10	0.414931004116824	45	0.000715405234913059

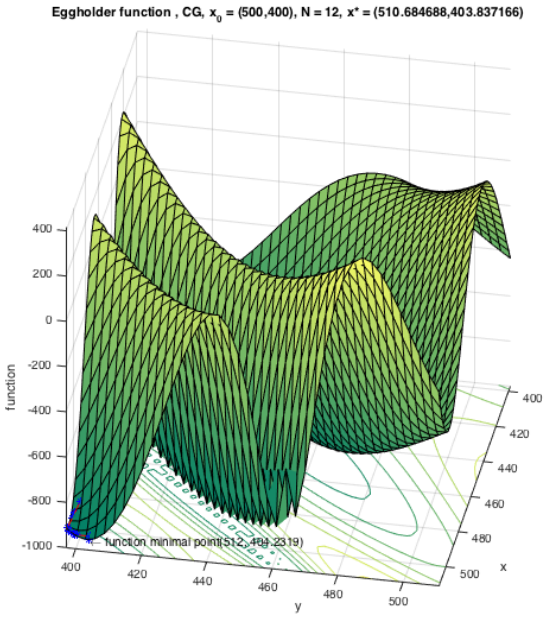
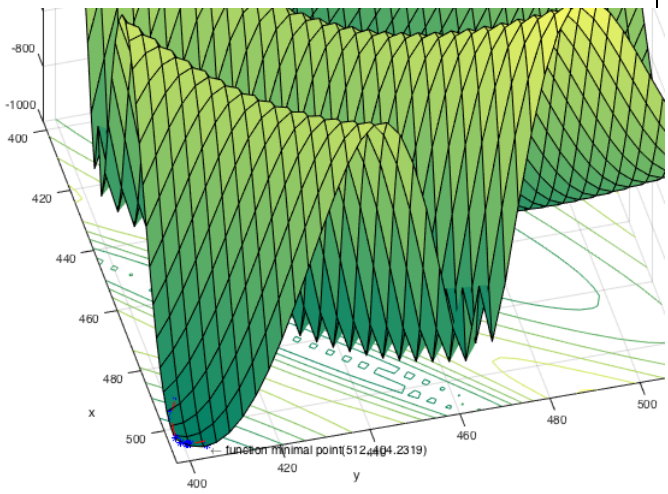
[x, f_k, x, y, step_length,z]=conjugate_gradient([-1.2, 1], egg, 1e-6, 1000, 0.1, 'plot')

Egg holder 2D graph x0 = (-1.2, 1)	(Red line: path of iterates, Left: close up of graph)
	

Iterations First 10			Iterations Last 10		
1	-0.659611708464811	1.84141576784113	43	8.44167052171134	15.6556905901552
2	1.40650280563606	5.13989289220928	44	8.44356415646518	15.6551064001817
3	2.81187281672709	7.59965673862576	45	8.44522271180892	15.6545925863775
4	3.93443331201490	9.60466067752230	46	8.44667541970760	15.6541409793657
5	4.79006073510767	11.1383583807771	47	8.44794786258314	15.6537442728062
6	5.45953044155155	12.3146480273609	48	8.44906243316778	15.6533959566452
7	5.98351900592175	13.2012470938836	49	8.45003873522758	15.6530902478453
8	6.39909970740159	13.8674114997132	50	8.45089393308336	15.6528220218019
9	6.73151992943980	14.3644155305343	51	8.45164305670884	15.6525867464780
10	7.00031644465998	14.7337907944621	52	8.45229926822474	15.6523804204701

Step length First 10		Step length Last 10	
1	1	43	0.00226168562516717
2	3.89214343586644	44	0.00198169889388221
3	2.83293188202816	45	0.00173632095351943
4	2.29786484819527	46	0.00152128535523256
5	1.75622519402192	47	0.00133284919095553
6	1.35345743209983	48	0.00116772930774028
7	1.02986500103156	49	0.00102304622687070
8	0.785163890558419	50	0.000896274836682951
9	0.597926593046604	51	0.000785201047053203
10	0.456825625590627	52	0.000687883693048154

[x, f_k, x, y, step_length,z]=conjugate_gradient([500, 400], egg, 1e-6, 1000, 0.1, 'plot')

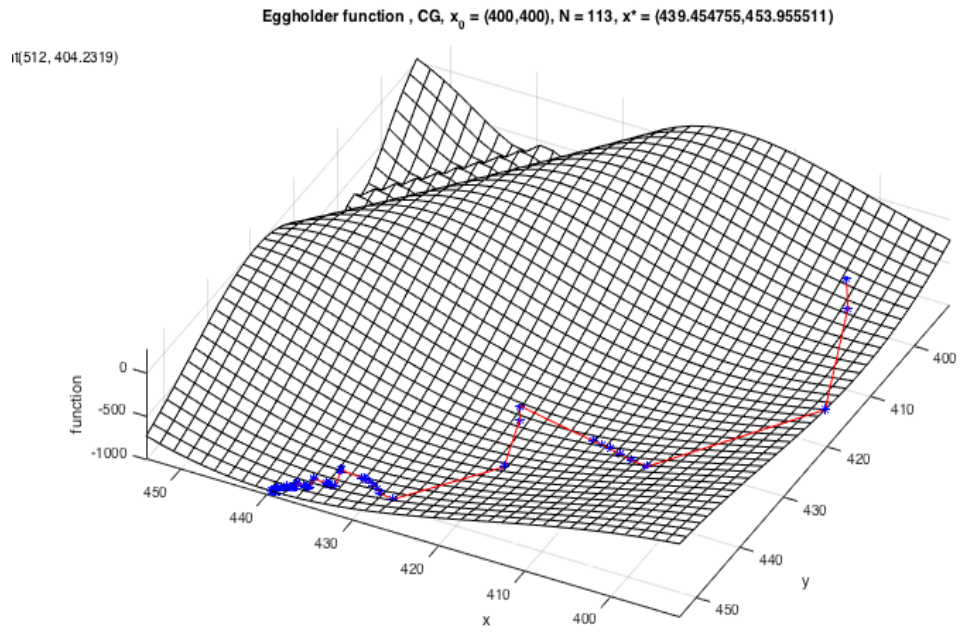
Egg holder 2D graph x0 = (500, 400)	(Red line: path of iterates, Left: close up of graph)
	

Iterations		
1	500.807308812298	399.409870792464
2	502.575970621960	398.204846869990
3	503.300034429642	397.927926165696
4	504.102808424552	397.628919555645
5	509.447569422344	397.092777569819
6	510.914934041551	397.932650819788
7	510.481682059156	399.182542241352
8	510.243927217067	399.714229905562
9	510.099715321969	400.270999873823
10	509.869075077081	404.292256279032
11	509.869075077082	404.292256279034
12	510.684688540632	403.837166322261

Step length	
1	0.999999999999990
2	2.14015122145439
3	0.775211889784032
4	0.856651060676062
5	5.37158434286237
6	1.69072351427289
7	1.32285140735721
8	0.582425220272703
9	0.575143345824198
10	4.02786519126420
11	1.55152338993005e-12
12	0.933987254024565

Example of falling into a local minimal even when the initial point is close to the global minima. x*= (512, 404.2319)
Conjugate_gradient([400, 400], egg, 1e-6, 1000, 0.1, 'plot')

Egg holder 2D graph x0 = (400, 400)	(Red line: path of iterates)



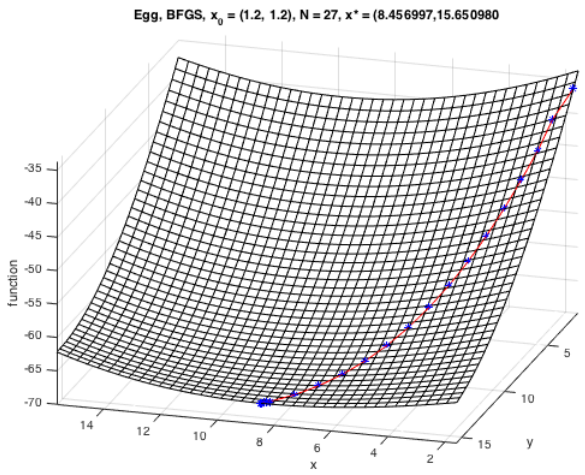
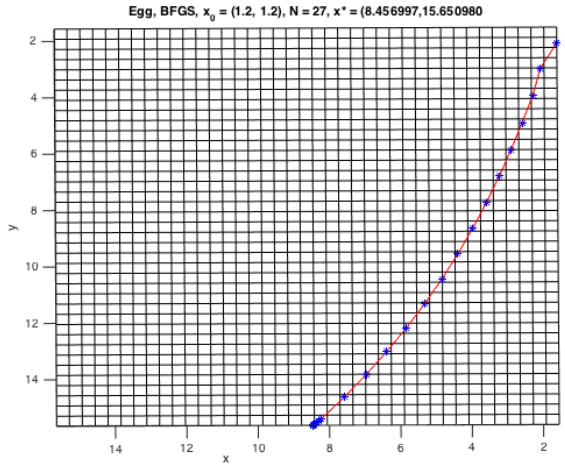
Iterations First 10			Iterations Last 10		
1	399.488094013715	400.859041478164	104	439.451879877994	453.935541326208
2	397.724060691879	403.968820616342	105	439.451879878028	453.935541326224
3	391.944180428372	420.594352335746	106	439.450083485012	453.938817087465
4	404.424634478962	436.701209022876	107	439.449562733453	453.940434994937
5	406.921853421303	435.596037226735	108	439.449106724699	453.942099704149
6	408.722783724597	434.490412996047	109	439.448175336357	453.954408914400
7	410.312734155396	433.661731679228	110	439.449970818280	453.957317950845
8	411.653388941306	433.069542978531	111	439.452470157354	453.956372482393
9	412.787963305013	432.682702529721	112	439.453584692687	453.955854181946
10	422.419697016332	430.521453785947	113	439.454755243970	453.955511269744

Step length First 10		Step length Last 10	
1	1.00000000000001	104	0.0193810920788387
2	3.57526780098958	105	3.66387915067377e-11
3	17.6015715438542	106	0.00373599244816149
4	20.3762745772777	107	0.00169964901505765
5	2.73084366908324	108	0.00172603613626121
6	2.11323328026280	109	0.0123443971609566
7	1.79294592702684	110	0.00341851552122614
8	1.46562024829604	111	0.00267219131064433
9	1.19870952261903	112	0.00122915595439062
10	9.87123550616755	113	0.00121974550054495

Question 3 – BFGS

[x, f_k, x, y, step_length,z]=BFGS([1.2, 1.2], egg, 1e-4, 100, 0.1, 'plot')

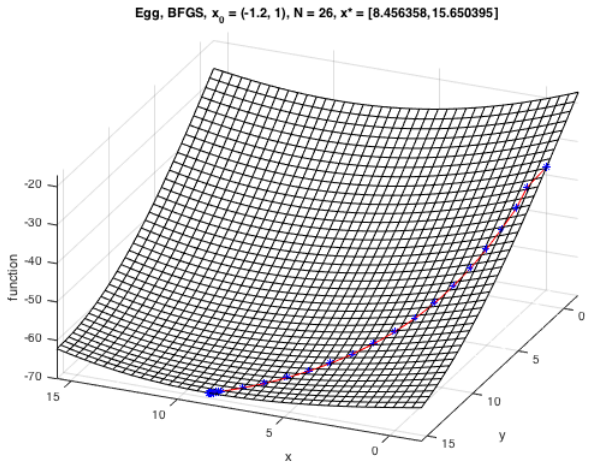
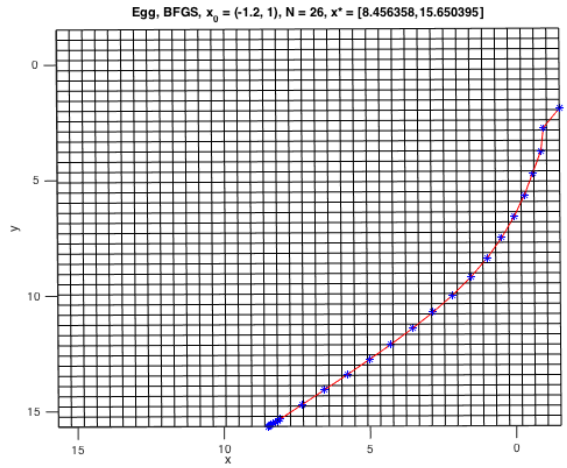
BFGS required more steps to converge for a tolerance rate of 1e-6 so I loosened the ceiling of the step_length tolerance value to 1e-4 in order to have the algorithm converge in 27 steps.

Egg holder graph x0 = (1.2, 1.2)	(Red line: path of iterates, Left:2D view of graph)
	

Iterations First 10			Step length	
1	1.56297641244963	2.13179832796866	1	1.00000000000000
2	2.02848725286293	3.01684050631821	2	1.00000000000000
3	2.24726416624129	3.99261540975272	3	1
4	2.54998552870710	4.94569451282174	4	1.00000000000000
5	2.86076402345737	5.89617687836082	5	1.00000000000000
6	3.19831093620724	6.83748558526468	6	1.00000000000000
7	3.56188149496438	7.76905224890450	7	1.00000000000000
8	3.95399224013914	8.68897026890329	8	1.00000000000000
9	4.37637455552560	9.59538804214517	9	0.999999999999999
10	4.83040567247681	10.4863738696644	10	1.00000000000000
11	5.31690715298624	11.3600536218975	11	1.00000000000000
12	5.83606625480101	12.2147312377287	12	1.00000000000000
13	6.38744064179333	13.0489891626948	13	1.00000000000000
14	6.97012213418038	13.8616897708093	14	1.00000000000000
15	7.58322746053608	14.6516909472605	15	1.00000000000000
16	8.22761145967480	15.4163930210474	16	1
17	8.29745743806997	15.4879579517189	17	0.0999999999999996
18	8.36726286470650	15.5595624376963	18	0.0999999999999990
19	8.43724738186049	15.6309918949162	19	0.100000000000000
20	8.44427482906998	15.6381063158847	20	0.00999999999999952
21	8.45130251590640	15.6452205001478	21	0.0100000000000002
22	8.45833206691810	15.6523328424184	22	0.0100000000000002
23	8.45762933602532	15.6516213866627	23	0.000999999999999489
24	8.45692645149901	15.6509100826892	24	0.000999999999999969
25	8.45699703777863	15.6509809175474	25	0.000100000000000610
26	8.45692644510046	15.6509100890659	26	0.000100000000000198
27	8.45699704610591	15.6509809092469	27	0.000100000000000872

[x, f_k, x, y, step_length,z]=BFGS([-1.2, 1], egg, 1e-3, 100, 0.1, 'plot')

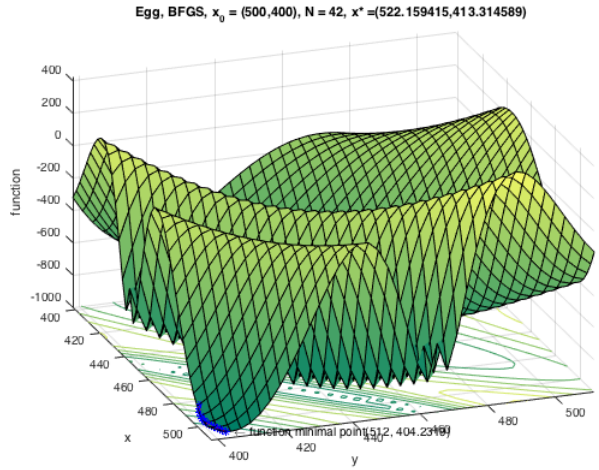
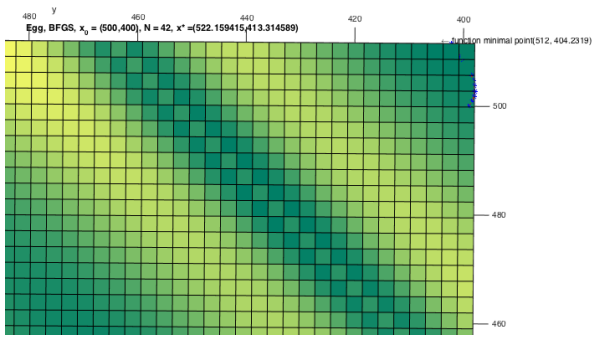
The algorithm didn't converge fast enough for step length tolerance of 1e-4 so I had to loosen it to 1e-3. The algorithm converged in 26 steps. (versus 22 steps under the same conditions for x0 = (1.2, 1.2))

Egg holder 2D graph x0 = (-1.2, 1)	(Red line: path of iterates, Left: close up of graph)
	

Iterations			Step Length	
1	-1.52535637165048	1.94559147173948	1	1.00000000000000
2	-0.973928123728205	2.77981379665855	2	1
3	-0.890020769995194	3.77628735676415	3	1
4	-0.606176848858861	4.73515785293866	4	1.00000000000000
5	-0.314749282019522	5.69175075134743	5	1.00000000000000
6	0.0442357224632586	6.62509407861586	6	1.00000000000000
7	0.468875654905882	7.53045639712438	7	1.00000000000000
8	0.964464867287491	8.39901343760645	8	1.00000000000000
9	1.52923396293965	9.22426241130848	9	1.00000000000000
10	2.15524432309964	10.0040771514695	10	1.00000000000000
11	2.83009769199845	10.7420289998213	11	1.00000000000000
12	3.54040327137900	11.4459224462327	12	1.00000000000000
13	4.27474139858195	12.1247062961703	13	1.00000000000000
14	5.02479442141101	12.7860839968414	14	1.00000000000000
15	5.78494085063471	13.4358358003313	15	1
16	6.55139757081621	14.0781317571056	16	1.00000000000000
17	7.32155241151921	14.7159887327214	17	1.00000000000000
18	8.09350592902059	15.3516677136680	18	1.00000000000000
19	8.17070542751463	15.4152305757202	19	0.100000000000000
20	8.24791411770000	15.4787822724706	20	0.100000000000001
21	8.32510420666249	15.5423565610310	21	0.0999999999999996
22	8.40231357658799	15.6059074319590	22	0.0999999999999998
23	8.47945784502597	15.6695373136749	23	0.0999999999999992
24	8.47175716602589	15.6631576943337	24	0.0100000000000001
25	8.46403590384016	15.6568030023460	25	0.00999999999999996
26	8.45635812126892	15.6503958458168	26	0.00999999999999935

[x, f_k, x, y, step_length,z]=BFGS([500, 400], egg, 1e-3, 100, 0.1, 'plot')
: Choosing an initial point where it would search for the global minima. note that tol = 1e-3
The algorithm didn't terminate because we reached the tolerance rate but simply because the x estimation hit our search boundary (-512, 512) after 15 iterations – I had programmed it to terminate the search whenever it hit the boundaries of the range where the function is defined. (range could be chosen to be anything, but I chose the one stated in Wikipedia)

If we had removed the -512<=x<= 512 conditions, it would have converged to a different local minima of point x*= (522, 413.3) in 42 steps. (versus 15) (whilst the global minima stated in wiki says its x* = (512, 404.2319) for function in range (-512, 512) – I don't understand why they restricted the function to this region. I figured that when I expanded my search range for x >512 we would hit a new global minima everytime)

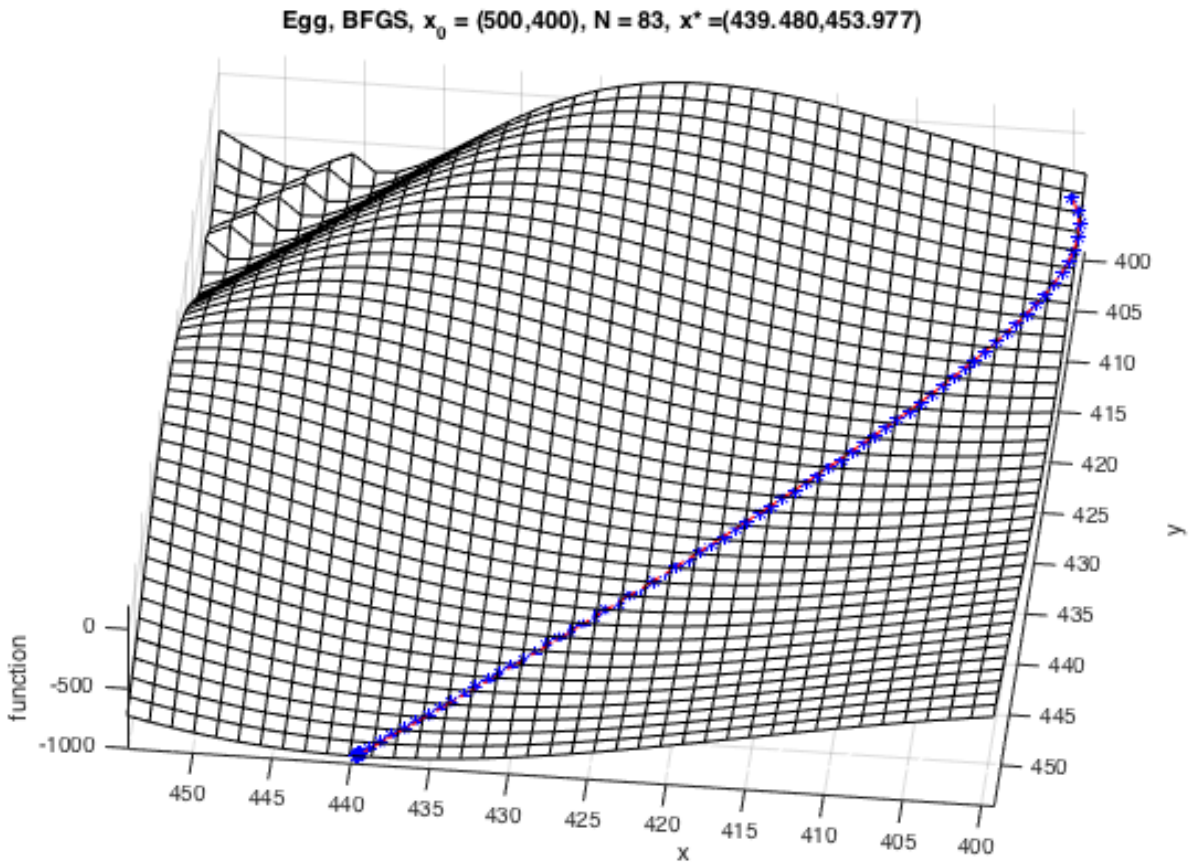
Egg holder 2D graph x0 = (500, 400)	(Red line: path of iterates, Left: close up of graph)
	

Iterations			Step length	
1	499.999333281860	399.000000222257	1	0.999999999999990
2	500.823232019739	398.433263184715	2	0.999999999999980
3	501.753760806989	398.067044430280	3	0.999999999999989
4	502.742206350290	397.915468020743	4	0.999999999999988
5	503.741224618668	397.959768127368	5	1.00000000000000
6	504.720550690851	398.162056150597	6	0.999999999999983
7	505.667653643230	398.482986045739	7	1.00000000000003
8	506.580760014302	398.890707464243	8	0.999999999999988
9	507.462657565277	399.362148562987	9	1.00000000000001
10	508.317343701970	399.881293636877	10	0.999999999999977
11	509.148655794508	400.437099542567	11	0.999999999999972
12	509.959860080390	401.021862406100	12	0.999999999999998
13	510.753617472757	401.630096902049	13	1.00000000000001
14	511.532067846174	402.257803056334	14	1.00000000000000
			15	1.00000000000002
Iteration without restriction -512< x <512			Step length without restriction -512< x <512	
1	499.999333281860	399.000000222257	1	0.999999999999990
2	500.823232019739	398.433263184715	2	0.999999999999980
3	501.753760806989	398.067044430280	3	0.999999999999989
4	502.742206350290	397.915468020743	4	0.999999999999988
5	503.741224618668	397.959768127368	5	1.00000000000000
6	504.720550690851	398.162056150597	6	0.999999999999983
7	505.667653643230	398.482986045739	7	1.00000000000003
8	506.580760014302	398.890707464243	8	0.999999999999988
9	507.462657565277	399.362148562987	9	1.00000000000001
10	508.317343701970	399.881293636877	10	0.999999999999977
	…			….
33	522.424012079539	413.526397582133	33	0.0999999999999883
34	522.351231786170	413.457818949184	34	0.100000000000013
35	522.276159255554	413.391757499406	35	0.100000000000045
36	522.198442047093	413.328828686378	36	0.0999999999999560
37	522.115496884737	413.272970830692	37	0.100000000000005
38	522.193196880402	413.335920895615	38	0.100000000000018
39	522.185106699786	413.330043057781	39	0.00999999999998735
40	522.176765853463	413.324526692799	40	0.0100000000000400
41	522.168211370613	413.319347810373	41	0.00999999999998466
42	522.159415920406	413.314589653624	42	0.0100000000000180

[x, f_k, x, y, step_length,z]=BFGS([400, 400], egg, 1e-4, 100, 0.1, 'plot')

:Example of falling into a local minimal even when the initial point is close to the global minima. x*= (512, 404.2319)/
Observe how the search path is much smoother compared to the conjugates gradient and steepest descent method, thus converging faster in 83 steps (vs. 113 for CG and 145 for SD)

Egg holder 2D graph x0 = (400, 400)	(Red line: path of iterates)



Iterations First 10			Iterations Last 10		
1	400.000833821217	400.999999652371	74	439.446438393601	453.958000307975
2	399.501325153963	401.866308541473	75	439.453770519673	453.964800302628
3	399.252842501454	402.834944887788	76	439.461408292493	453.971255101334
4	399.225016891768	403.834557680546	77	439.469402294267	453.977263090652
5	399.363482977405	404.824924856460	78	439.477865935687	453.982589137395
6	399.620249442956	405.791398333680	79	439.487149351075	453.986306421180
7	399.960986594420	406.731556933924	80	439.482993862124	453.977210712764
8	400.362587217259	407.647371838659	81	439.482064892922	453.977580869788
9	400.809544349178	408.541927215495	82	439.481069945189	453.977480475524
10	401.291254191926	409.418257990602	83	439.480984292315	453.977428865592

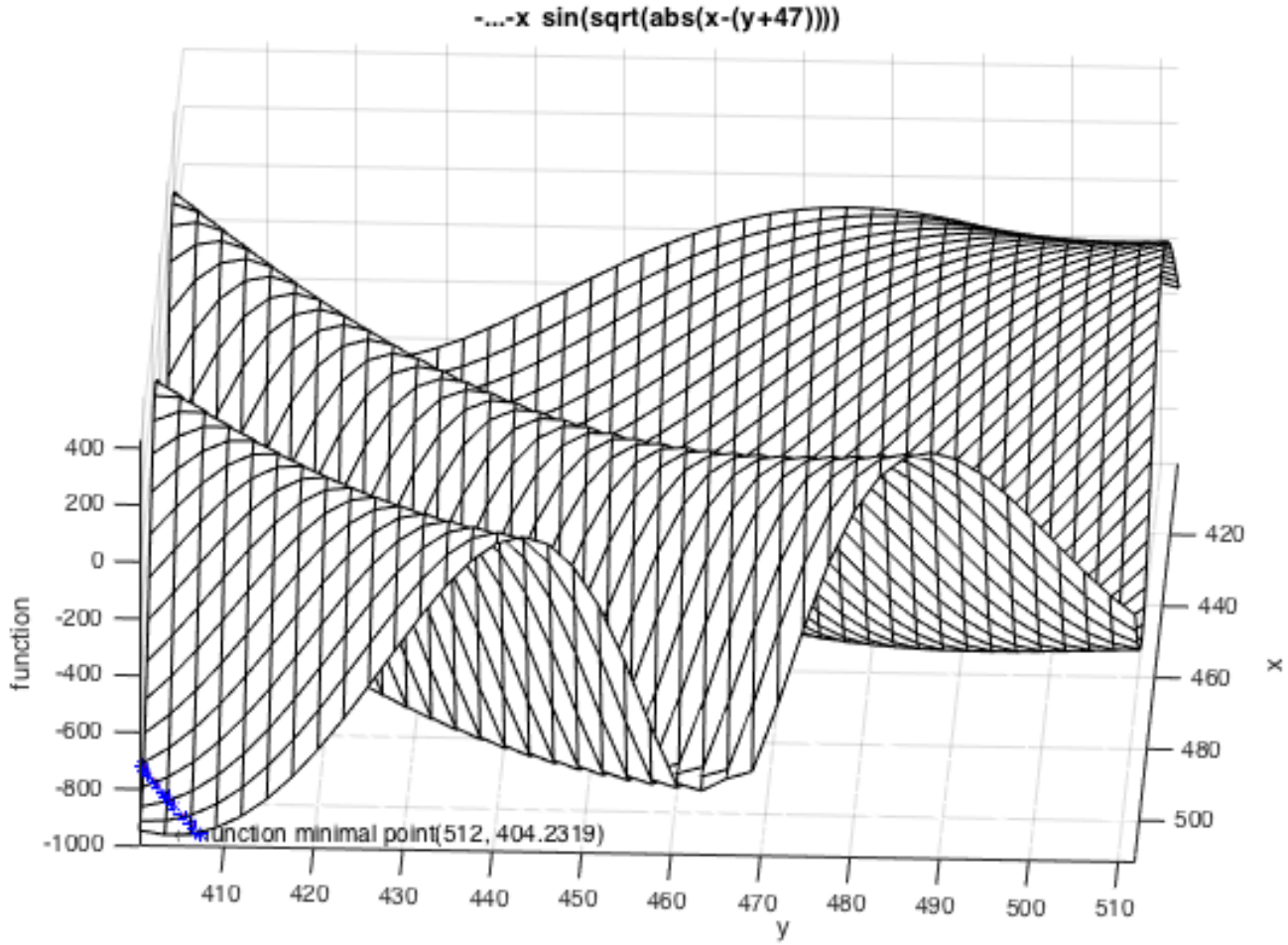
Step length First 10		Step length Last 10	
1	0.999999999999974	74	0.0100000000000333
2	1.00000000000000	75	0.00999999999999245
3	1.00000000000000	76	0.0100000000000040
4	1.00000000000002	77	0.0100000000000010
5	1.00000000000001	78	0.0100000000000273
6	0.999999999999985	79	0.00999999999999004
7	1.00000000000002	80	0.00999999999998451
8	0.999999999999980	81	0.000999999999978899
9	1.00000000000001	82	0.00100000000000866
10	1.00000000000002	83	0.000100000000027390

Question 3 – Inexact line search with Newton's method

Couldn't get the method working. Search was direction was diverging.
There is a problem in my step length computation...

[x, f_k, x, y, step_length,z]=Newton_inexact([500, 400], egg, 1e-4, 100, 0.1, 'plot');

Egg holder 2D graph x0 = (500 ,400)	(Red line: path of iterates)



Iterations			Steplength	
1	500.852229393808	400.523168290638	1	0.999999999999984
2	501.704687020306	401.045964613336	2	0.999999999999991
3	502.557392817204	401.568356060371	3	0.999999999999989
4	503.410366127451	402.090310590964	4	0.999999999999987
5	504.263625797024	402.611796867179	5	1.00000000000000
6	505.117190265562	403.132784101235	6	1.00000000000002
7	505.971077650965	403.653241912253	7	1.00000000000003
8	506.825305828954	404.173140190698	8	0.999999999999970
9	507.679892508483	404.692448968952	9	0.999999999999995
10	508.534855303801	405.211138296605	10	1.00000000000002
11	509.390211803885	405.729178119171	11	0.999999999999981
12	510.245979639910	406.246538159014	12	0.999999999999989
13	511.102176551379	406.763187797347	13	0.999999999999969
14	511.958820451505	407.279095956205	14	1.00000000000002

Code for Assignment 3

Sooa Lim #994500731

Rosenbrock 2D function
: Outputs value of function and plots a graph with a datatip indicating the global minima

rosenbrock_2d.m

```
function varargout = rosenbrock_2d(X, r_x, r_y)
% Rosenbrock function
%
%   rosenbrock([x1, x2]) returns the value of the value of the
%   two dimensional rosenbrock function at the specified points. All
[xi] may be
%   vectors.
%
%   The global minimum is
%
%                   f(x1, x2) = f(1, 0) =
%
% Please report bugs and inquiries to:

    % if no input is given, return dimensions, bounds and minimum
    f = @(x, y) 100*(y - x.^2).^2 + (1-x).^2;
    if (nargin == 0)
        varargout{1} = [1, 0];  % # dims
    end
    if(nargin > 0)

        % split input vector X into x1, x2, x3
        if size(X, 1) == 2
            x1 = X(1, :);  x2 = X(2, :);
        else
            x1 = X(:, 1);  x2 = X(:, 2);
        end

        % output function value
        varargout{1} = f(x1, x2);
    end

    if (nargin == 3)
       s = linspace(r_x,r_y,40);
       t = s;
       colormap(white);
       [x, y] = meshgrid(s, t);
       surfc(x, y, 100*(y - x.^2).^2 + (1-x).^2)
       xlabel('x'); ylabel('y'); zlabel('function'); %colorbar
       t = text(1, 1 , f(1,1), '\leftarrow function minimal point');
       t.Color = 'red';
       t.FontSize = 15;
    end

end
```

Egg holder function (Question 3 Test function for optimization)
: Outputs value of function and plots a graph with a datatip indicating the global minima

test_function.m (Egg holder function)

```
function varargout = test_function(X, r_x, r_y)
% Rosenbrock function
%
%   eggholder([x1, x2]) returns the value of the value of the
%   two dimensional eggholder function at the specified points. All
[xi] may be
%   vectors.
%
%   The global minimum is
%
%                   f(x1, x2) = ...


    % if no input is given, return dimensions, bounds and minimum
    f = @(x, y) -1*(y+47)*sin(sqrt(abs(x/2+(y+47))))-
x*sin(sqrt(abs(x-(y+47)))); % eggholder function

    if (nargin == 0)
        varargout{1} = [512, 404.2319];  % #solution
        varargout{2} = f(512, 404.2319);

    end
    if(nargin > 0)

        % split input vector X into x1, x2, x3
        if size(X, 1) == 2
            x1 = X(1, :);  x2 = X(2, :);
        else
            x1 = X(:, 1);  x2 = X(:, 2);
        end

        % output function value
        varargout{1} = f(x1, x2);
    end

    if (nargin == 3)
       syms x y
       ezsurfc(f, [r_x, r_y, r_x, r_y], 40)
       xlabel('x'); ylabel('y'); zlabel('function');
       text(512, 404.2319 ,f(512, 404.2319), '\leftarrow function
minimal point(512, 404.2319)');
       colormap(white)

    end

end
```

Egg Hodler Grid search

: Additional script that computes and stores local minima within range [400:512] and then computing the global minima x* by finding the minimum f(x) value evaluated for each local minima.

Assuming that global minima existed within this range for the egg holder function. (After plotting a 3d plot for this function)

This was necessary since everytime a local minima serach is performed, the algorithm would get stuck at a local minima point.

egg_holder_search.m

```
=========================================================================
f = @(x, y) -1*(y+47)*sin(sqrt(abs(x/2+(y+47))))-x*sin(sqrt(abs(x-
(y+47)))); % eggholder function
k = 1;
result_CG = zeros(1, 6);
for i = 400:16:512
    for j =400:16:512
        try
            %[a,f_k,x,y] = conjugate_gradient([i, j], f, 1e-6, 2000,
0.3, '');
            [a,f_k,x,y] = steepest_descent([i, j], f, 1e-6, 2000,
0.3, '');
        catch
            continue;
        end
        len = length(x);
        result_CG(k, :) = [i, j, x(len), y(len),f_k(len), len];
        k = k+1;
    end
end
T = array2table(result_CG);
T.Properties.VariableNames = {'i' 'j' 'x' 'y' 'f_k' 'N'};
rows = T.f_k == min(T.f_k);
minimum = T(rows, T.Properties.VariableNames)
=========================================================================
```

Steepest Descent line search (2D)
: Finds local minima given initial point x and out puts the local minimum x*
f_k : sequence of the function output for each iterate
x_x : sequence of x value for each iterate
x_y : sequence of x value for each iterate
where x = (x_x, x_y)

Function can be modified to for N-Dimensional line search

steepest_descent.m

```
function [x,f_k, x_x, x_y, step_length, z] = steepest_descent(X, f,
tol, N, beta, plot)

%[a,f_k,x,y] = steepest_descent([1.2, 1.2], f, 1e-6,7000, 0.1);
%(rosenbrock) (196,872 - non normalized)
%[a,f_k,x,y] = steepest_descent([500,370], f, 1e-6, 7000, 0.1);
%(egg holder, pk = pk/norm(pk)
% f = @(x, y) 100*(y - x.^2).^2 + (1-x).^2 (rosenbrock)
% f = @(x, y) -1*(y+47)*sin(sqrt(abs(x/2+(y+47))))-x*sin(sqrt(abs(x-
(y+47)))); % eggholder function
% Inputs
% X: starting value (initial guess x_0)
% f: function
% tol: tolerance
% N: max iterations
% direction p_k = -grad(f(x_k))
% initial step size a_0 = 1
% step size a_k deteremined by backtracking
% x_k+1 = x_k + a_k*p_k
format long

if (nargin < 4)
    tol = 1e-6;
end %if

k = 1; %initialize count
x = X;
MaxIter = N;
eps = 1;

while (eps > tol) && (k <= MaxIter) && (x(1)<=512) && (x(2) <=512)
    p_k = -1 * Grad(x);
    %p_k = p_k/norm(p_k)

    a_k = 1;
    a_k = linesearch(a_k, beta, p_k, f, x);
    old_x = x;
    x = x + a_k * p_k;
    step_length(k, 1:2) = [k, norm(x - old_x, 2)];
    if(x(1) < -512 || x(2) < -512 || x(1) > 512 || x(2) > 512)
       break;
    end
    x_x(k) = x(1);
    x_y(k) = x(2);
    z(k, 1:3) = [k,x];
    f_k(k) = f(x(1), x(2));

    eps = norm(p_k*a_k, 2);
    k = k+1;
end %while
```

```
if nargin > 5 && strcmp(plot,'plot')
    %plot graph of function and path
    %rosenbrock_2d([X(1), X(2)],min(min(x_x, x_y)),max(max(x_x,
x_y))) ;
    test_function([X(1), X(2)],min(min(x_x, x_y)),max(max(x_x,
x_y))) ;
    hold on
    plot3(x_x, x_y, f_k, 'r');
    scatter3(x_x, x_y, f_k, 'b*');
end



end
%end steepest_descent

%backtracking algorithm
function alpha = linesearch(a_k, beta, p_k, f, x)
 in = x + a_k * p_k;
 c1 = 1e-4;
 while feval(f, in(1), in(2)) > (feval(f, x(1), x(2)) + c1 * a_k *
(p_k' * p_k))
    a_k = beta * a_k;
    in = x + a_k * p_k;
 end
 alpha = a_k;
end

function out = Grad(x_k)
%f_grad = @(x,y) [2*x - 400*x*(- x^2 + y) - 2, - 200*x^2 + 200*y];
f_grad = @(x,y) [(x*sign(y - x + 47)*cos(abs(y - x +
47)^(1/2)))/(2*abs(y - x + 47)^(1/2)) - sin(abs(y - x + 47)^(1/2)) -
(sign(x/2 + y + 47)*cos(abs(x/2 + y + 47)^(1/2))*(y +
47))/(4*abs(x/2 + y + 47)^(1/2)), ...
    - sin(abs(x/2 + y + 47)^(1/2)) - (x*sign(y - x + 47)*cos(abs(y -
x + 47)^(1/2)))/(2*abs(y - x + 47)^(1/2)) - (sign(x/2 + y +
47)*cos(abs(x/2 + y + 47)^(1/2))*(y + 47))/(2*abs(x/2 + y +
47)^(1/2))];
out = f_grad(x_k(1), x_k(2));
end
```

==========

Conjugate Gradient optimization method
: Conjugate Gradient method with the Polak and Ribiere method for updating search direction.

conjugate_gradient.m

```
=====================================================================
function [x,f_k, x_x, x_y, step_length, z] = conjugate_gradient(X, f,
tol, N, beta, plot)

% example: [a,f_k,x,y] = conjugate_gradient([-1.2, 1], f, 1e-6, 300,
0.1);
% example: [a,f_k,x,y] = con_grad([1.2, 1.2], f, 1e-6, 300, 0.1);
% (100, 222 - normalized) (110, 236 - non normalized)
%
% f = @(x, y) 100*(y - x.^2).^2 + (1-x).^2
% Inputs
% X: starting value (initial guess x_0)
% f: function
% tol: tolerance
% N: max iterations
% direction p_k = -grad(f(x_k))
% initial step size a_0 = 1
% step size a_k deteremined by backtracking
% x_k+1 = x_k + a_k*p_k
format long

if (nargin < 4)
    tol = 1e-5;
end %if

k = 1; %initialize count
x = X;
MaxIter = N;
grad_f = Grad(x);
p_k = -1 * grad_f;
p_k = p_k/norm(p_k);
f_new = feval(f, x(1), x(2));

while (norm(p_k,2) > 1e-5*(1+abs(f_new))) && (k <= MaxIter)

    a_k = 1;
    a_k = linesearch(a_k, beta, p_k, f, x);
    old_x = x;
    x = x + a_k * p_k;

    step_length(k,1:2) = [k, norm(x - old_x, 2)];

    if(x(1) < -512 || x(2) < -512 || x(1) > 512 || x(2) > 512)
       break;
    end
    f_new = feval(f, x(1), x(2));
    grad_f_new = Grad(x);

    PolakB = grad_f_new * (grad_f_new - grad_f)' / norm(grad_f,
2)^2;
    p_k = -1 * grad_f_new + PolakB * p_k;
    grad_f = grad_f_new;

    x_x(k) = x(1);
    x_y(k) = x(2);
    f_k(k) = f(x(1), x(2));
    z(k, 1:3) = [k, x];
    eps = norm(p_k * a_k, 2);
```

```
    k = k+1;

end %while

%plot graph of function and path
if nargin > 5 && strcmp(plot,'plot')
        %rosenbrock_2d([X(1), X(2)],min(min(x_x, x_y)),max(max(x_x,
x_y))) ;
        test_function([X(1), X(2)],min(min(x_x, x_y)),max(max(x_x,
x_y))) ;
        hold on
        plot3(x_x, x_y, f_k, 'r');
        scatter3(x_x, x_y, f_k, 'b*');
end
end
%end con_grad

% Polak - Ribiere method and variants
function alpha = linesearch(a_k, beta, p_k, f, x)
 in = x + (a_k * p_k);
 c1 = 1e-4;
 c2 = 0.1;
 % 0 < c1 < c2 < 0.5 stronger wolfe condition

 while  feval(f, in(1), in(2)) > (feval(f, x(1), x(2)) + c1 * a_k *
(p_k' * p_k))
    %while Grad(in)' * p_k > c2 * grad_f' * p_k
        a_k = beta * a_k;
        in = x + a_k * p_k;
    %end
 end
 alpha = a_k;
end

function out = Grad(x_k)
%f_grad = @(x,y) [2*x - 400*x*(- x^2 + y) - 2, - 200*x^2 + 200*y];
f_grad = @(x,y) [(x*sign(y - x + 47)*cos(abs(y - x +
47)^(1/2)))/(2*abs(y - x + 47)^(1/2)) - sin(abs(y - x + 47)^(1/2)) -
(sign(x/2 + y + 47)*cos(abs(x/2 + y + 47)^(1/2))*(y +
47))/(4*abs(x/2 + y + 47)^(1/2)), ...
    - sin(abs(x/2 + y + 47)^(1/2)) - (x*sign(y - x + 47)*cos(abs(y -
x + 47)^(1/2)))/(2*abs(y - x + 47)^(1/2)) - (sign(x/2 + y +
47)*cos(abs(x/2 + y + 47)^(1/2))*(y + 47))/(2*abs(x/2 + y +
47)^(1/2))];
out = f_grad(x_k(1), x_k(2));
end
```

==========

BFGS method using backtracking method (2D)
: Quasi newton method – BFGS

Note 1:
1) Backtracking method / 2) inexact line search method (quadratic and cubic interpolation) for updating step length α
Note 2: Hessian approximation computed using SMW inverse update method.

BFGS.m

```
=====================================================================
function [x,f_k, x_x, x_y, step_length, z] = BFGS(x_0, f, tol,
MaxIter, beta, plot)
%[a,f_k,x,y] = BFGS([1.2, 1.2], f, 1e-6,7000, 0.1);
%(rosenbrock) (196,872 - non normalized)
%[a,f_k,x,y] = BFGS([500,370], f, 1e-6, 7000, 0.1);
%(egg holder, pk = pk/norm(pk)
% f = @(x, y) 100*(y - x.^2).^2 + (1-x).^2 (rosenbrock)
% f = @(x, y) -1*(y+47)*sin(sqrt(abs(x/2+(y+47))))-x*sin(sqrt(abs(x-
(y+47)))); % eggholder function
% Inputs
% X: starting value (initial guess x_0)
% f: function
% tol: tolerance
% N: max iterations
% direction p_k = -grad(f(x_k))
% initial step size a_0 = 1
% step size a_k deteremined by backtracking
% x_k+1 = x_k + a_k*p_k

format long

if (nargin < 4)
    tol = 1e-6;
end %if

k = 1; %initialize count
x=x_0;
% initialize Hessian
% B = eye(2,2);
 B = Hess(x);
 H = inv(B);
steps(k) = 1;

%f'(x)
grad1 = Grad(x);

while  (k <= MaxIter)
    p_k = -1 * H * grad1';
    p_k = p_k/norm(p_k);

    a_k = 1;
    a_k = linesearch(a_k, beta, p_k, f, x);
    %a_k = inexact(a_k, p_k, f, x, grad1);
    s_k = a_k * p_k; % x_(k+1) - x_k
    s_k = s_k';
    old_x = x;
    x = x + s_k;
    steps(k) = norm(x - old_x, 2);
    if (steps(k) < tol)
        break;
    end
    step_length(k, 1:2) = [k, steps(k)];
    norm_grad(k) = norm(grad1,2);
```

```
    %{
    if(x(1) < -512 || x(2) < -512 || x(1) > 512 || x(2) > 512)
       break;
    end
    %}

    grad2 = Grad(x);
    y_k = grad2 - grad1;
    y_k = y_k';
    grad1 = grad2;

    rho = 1/(y_k'*s_k');
    if k ==1
        H = (y_k'* s_k')/(y_k' * y_k) * eye(2);
    else
          s_k = s_k';
          rho = (y_k' * s_k)^-1;
          Id = eye(2, 2);
          x1 = Id - rho * s_k * y_k';
          x2 = H;
          x3 = Id - rho * y_k * s_k';
          x4 = rho * s_k * s_k';
          H = x1 * x2 * x3 + x4;

    end
    x_x(k) = x(1);
    x_y(k) = x(2);
    z(k, 1:3) = [k, x];
    f_k(k) = f(x(1), x(2));
    k = k+1;

end %while

if nargin > 5 && strcmp(plot,'plot')
    %plot graph of function and path
    %rosenbrock_2d([x_0(1), x_0(2)],min(min(x_x, x_y)),max(max(x_x,
x_y)));
    test_function([x_0(1), x_0(2)],min(min(x_x, x_y)),max(max(x_x,
x_y))) ;
    hold on
    plot3(x_x, x_y, f_k, 'r');
    scatter3(x_x, x_y, f_k, 'b*');
end
end
%end BFGS

%backtracking algorithm
function alpha = linesearch(a_k, beta, p_k, f, x)
 in = x + a_k * p_k';
 c1 = 1e-4;
 while feval(f, in(1), in(2)) > (feval(f, x(1), x(2)) + c1 * a_k *
(p_k' * p_k))
    a_k = beta * a_k;
    in = x + a_k * p_k';
 end
 alpha = a_k;
end

% inexact line search - cubic interpolation method
function alpha = inexact(a_k, p_k, f, x, grad)
 in = x + a_k * p_k';
 c1 = 1e-4;
 n =1;
```

```
 f0(n) = feval(f, x(1), x(2));
 f1(n)= feval(f, in(1), in(2));

 alpha(n) = 1;
 %alpha(n) = 2*(f1(n) - f0(n))/(grad*p_k);


 while feval(f, in(1), in(2)) > (feval(f, x(1), x(2)) + c1 * a_k *
(p_k' * p_k))
     n = n+1;
     if n == 2
         %quadratic interpolation find a_1
         alpha(n) = -a_k^2 * grad * p_k /(2*(f1(n-1)-f0(n-1)-grad *
p_k*a_k));
     elseif n>2
         %cubic interpolation to find a_2 ...
         ab = [alpha(n-1)^3, alpha(n-1)^2; alpha(n-2)^3, alpha(n-
2)^2];
         aa = inv(ab) * [f1(n-1)-alpha(n-1)*grad*p_k - f0(n-1);
f1(n-2)-alpha(n-2)*grad*p_k - f0(n-2)];
         a = aa(1);
         b = aa(2);
         alpha(n) = (-b + sqrt(b^2 - 3*a*grad*p_k))/(3*a);

     end %if-else
    x = in;
    in = x + alpha(n) * p_k';
    f0(n) = f1(n-1);
    f1(n) = feval(f, in(1), in(2));

    if (abs((alpha(n)-alpha(n-1))/alpha(n-1))>0.9 || abs((alpha(n)-
alpha(n-1))/alpha(n-1))<0.1)
     alpha(n) = 1/2 * alpha(n-1);
    end %if

 end % while
 alpha = alpha(n);

end %inexact search

function out = Grad(x_k)
%f_grad = @(x,y) [2*x - 400*x*(- x^2 + y) - 2, - 200*x^2 + 200*y];
f_grad = @(x,y) [(x*sign(y - x + 47)*cos(abs(y - x +
47)^(1/2)))/(2*abs(y - x + 47)^(1/2)) - sin(abs(y - x + 47)^(1/2)) -
(sign(x/2 + y + 47)*cos(abs(x/2 + y + 47)^(1/2))*(y +
47))/(4*abs(x/2 + y + 47)^(1/2)), ...
    - sin(abs(x/2 + y + 47)^(1/2)) - (x*sign(y - x + 47)*cos(abs(y -
x + 47)^(1/2)))/(2*abs(y - x + 47)^(1/2)) - (sign(x/2 + y +
47)*cos(abs(x/2 + y + 47)^(1/2))*(y + 47))/(2*abs(x/2 + y +
47)^(1/2))];
out = f_grad(x_k(1), x_k(2));
end

function out = Hess(x_k)
f_hess = @(x,y) [1200*x^2-400*y+2,-400*x;-400*x,200];
out = f_hess(x_k(1), x_k(2));
end  %for initial hessian matrix
```

==========

Newton Method with inexact lines search (using the method from notes presented in class)

newton_inexact.m

```
function [x,f_k, x_x, x_y, step_length, z] = Newton_inexact(x_0, f,
tol, MaxIter, beta, plot)

format long

if (nargin < 4)
    tol = 1e-6;
end %if

k = 1; %initialize count
x= x_0;

% initialize Hessian
% B = eye(2,2);

B = Hess(x);
H = inv(B);

%f'(x)
grad1 = Grad(x);

while  (k <= MaxIter) && (norm(grad1,2) > tol)

    p_k = -1 * H * grad1';
    p_k = p_k/norm(p_k);

    a_k = 1;
    a_k = inexact(a_k, p_k, f, x, grad1);

    s_k = a_k * p_k; % x_(k+1) - x_k
    s_k = s_k';

    old_x = x;
    x = x + s_k;
    step_length(k, 1:2) = [k, norm(x-old_x, 2)];

    B = Hess(x);
    H = inv(B);

    %storing iterates
    x_x(k) = x(1);
    x_y(k) = x(2);
    z(k, 1:3) = [k,x];
    f_k(k) = f(x(1), x(2));
    norm_grad(k) = norm(grad1, 2);

    k = k+1;

end %while

% plot graph of path and function if 'plot'
if nargin > 5 && strcmp(plot,'plot')
    %plot graph of function and path
    rosenbrock_2d([x_0(1), x_0(2)],min(min(x_x, x_y)),max(max(x_x,
x_y)));
    %test_function([X(1), X(2)],min(min(x_x, x_y)),max(max(x_x,
x_y))) ;
    hold on
```

```
    plot3(x_x, x_y, f_k, 'r');
    scatter3(x_x, x_y, f_k, 'b*');
end
end
%end Newton

% inexact line search - quadratic & cubic interpolation method
function alpha = inexact(a_k, p_k, f, x, grad)
 in = x + a_k * p_k';
 c1 = 1e-4;
 n =1;
 f0(n) = feval(f, x(1), x(2));
 f1(n)= feval(f, in(1), in(2));

 alpha(n) = 1;
 %alpha(n) = 2*(f1(n) - f0(n))/(grad*p_k);

 while feval(f, in(1), in(2)) > (feval(f, x(1), x(2)) + c1 * a_k *
(p_k' * p_k))
     n = n+1;
     if n == 2
         %quadratic interpolation find a_1
         alpha(n) = -a_k^2 * grad * p_k /(2*(f1(n-1)-f0(n-1)-grad *
p_k*a_k));
     elseif n>2
         %cubic interpolation to find a_2 ...
         ab = [alpha(n-1)^3, alpha(n-1)^2; alpha(n-2)^3, alpha(n-
2)^2];
         aa = ab^-1 * [f1(n-1)-alpha(n-1)*grad*p_k - f0(n-1); f1(n-
2)-alpha(n-2)*grad*p_k - f0(n-2)];
         a = aa(1);
         b = aa(2);
         alpha(n) = (-b + sqrt(b^2 - 3*a*grad*p_k))/(3*a);
     end %if-else
    x = in;
    in = x + alpha(n) * p_k';
    f0(n) = f1(n-1);
    f1(n) = feval(f, in(1), in(2));

    if (abs((alpha(n)-alpha(n-1))/alpha(n-1))>0.9 || abs((alpha(n)-
alpha(n-1))/alpha(n-1))<0.1)
     alpha(n) = 1/2 * alpha(n-1);
    end %if

 end % while
 alpha = alpha(n);

end %inexact search

%hard coded gradient and hessian function
function out = Grad(x_k)
f_grad = @(x,y) [2*x - 400*x*(- x^2 + y) - 2, - 200*x^2 + 200*y];
%f_grad = @(x,y) [(x*sign(y - x + 47)*cos(abs(y - x +
47)^(1/2)))/(2*abs(y - x + 47)^(1/2)) - sin(abs(y - x + 47)^(1/2)) -
(sign(x/2 + y + 47)*cos(abs(x/2 + y + 47)^(1/2))*(y +
47))/(4*abs(x/2 + y + 47)^(1/2)), ...
%    - sin(abs(x/2 + y + 47)^(1/2)) - (x*sign(y - x + 47)*cos(abs(y
- x + 47)^(1/2)))/(2*abs(y - x + 47)^(1/2)) - (sign(x/2 + y +
47)*cos(abs(x/2 + y + 47)^(1/2))*(y + 47))/(2*abs(x/2 + y +
47)^(1/2))];
out = f_grad(x_k(1), x_k(2));
end

function out = Hess(x_k)
```

```
%rosenbrock hessian
f_hess = @(x,y) [1200*x^2-400*y+2,-400*x;-400*x,200];

%egg_holder hessian
%{
f_hess = @(x,y) [(sign(y - x + 47)*cos(abs(y - x + 47)^(1/2)))/abs(y
- x + 47)^(1/2) + (sign(x/2 + y + 47)^2*sin(abs(x/2 + y +
47)^(1/2))*(y + 47))/(16*abs(x/2 + y + 47)) - (x*dirac(y - x +
47)*cos(abs(y - x + 47)^(1/2)))/abs(y - x + 47)^(1/2) - (dirac(x/2 +
y + 47)*cos(abs(x/2 + y + 47)^(1/2))*(y + 47))/(4*abs(x/2 + y +
47)^(1/2)) + (x*sign(y - x + 47)^2*cos(abs(y - x +
47)^(1/2)))/(4*abs(y - x + 47)^(3/2)) + (x*sign(y - x +
47)^2*sin(abs(y - x + 47)^(1/2)))/(4*abs(y - x + 47)) + (sign(x/2 +
y + 47)^2*cos(abs(x/2 + y + 47)^(1/2))*(y + 47))/(16*abs(x/2 + y +
47)^(3/2)), ...
(sign(x/2 + y + 47)^2*sin(abs(x/2 + y + 47)^(1/2))*(y +
47))/(8*abs(x/2 + y + 47)) - (sign(x/2 + y + 47)*cos(abs(x/2 + y +
47)^(1/2)))/(4*abs(x/2 + y + 47)^(1/2)) - (sign(y - x +
47)*cos(abs(y - x + 47)^(1/2)))/(2*abs(y - x + 47)^(1/2)) +
(x*dirac(y - x + 47)*cos(abs(y - x + 47)^(1/2)))/abs(y - x +
47)^(1/2) - (dirac(x/2 + y + 47)*cos(abs(x/2 + y + 47)^(1/2))*(y +
47))/(2*abs(x/2 + y + 47)^(1/2)) - (x*sign(y - x + 47)^2*cos(abs(y -
x + 47)^(1/2)))/(4*abs(y - x + 47)^(3/2)) - (x*sign(y - x +
47)^2*sin(abs(y - x + 47)^(1/2)))/(4*abs(y - x + 47)) + (sign(x/2 +
y + 47)^2*cos(abs(x/2 + y + 47)^(1/2))*(y + 47))/(8*abs(x/2 + y +
47)^(3/2)); ...
(sign(x/2 + y + 47)^2*sin(abs(x/2 + y + 47)^(1/2))*(y +
47))/(8*abs(x/2 + y + 47)) - (sign(x/2 + y + 47)*cos(abs(x/2 + y +
47)^(1/2)))/(4*abs(x/2 + y + 47)^(1/2)) - (sign(y - x +
47)*cos(abs(y - x + 47)^(1/2)))/(2*abs(y - x + 47)^(1/2)) +
(x*dirac(y - x + 47)*cos(abs(y - x + 47)^(1/2)))/abs(y - x +
47)^(1/2) - (dirac(x/2 + y + 47)*cos(abs(x/2 + y + 47)^(1/2))*(y +
47))/(2*abs(x/2 + y + 47)^(1/2)) - (x*sign(y - x + 47)^2*cos(abs(y -
x + 47)^(1/2)))/(4*abs(y - x + 47)^(3/2)) - (x*sign(y - x +
47)^2*sin(abs(y - x + 47)^(1/2)))/(4*abs(y - x + 47)) + (sign(x/2 +
y + 47)^2*cos(abs(x/2 + y + 47)^(1/2))*(y + 47))/(8*abs(x/2 + y +
47)^(3/2)), ...
(sign(x/2 + y + 47)^2*sin(abs(x/2 + y + 47)^(1/2))*(y +
47))/(4*abs(x/2 + y + 47)) - (sign(x/2 + y + 47)*cos(abs(x/2 + y +
47)^(1/2)))/abs(x/2 + y + 47)^(1/2) - (x*dirac(y - x + 47)*cos(abs(y
- x + 47)^(1/2)))/abs(y - x + 47)^(1/2) - (dirac(x/2 + y +
47)*cos(abs(x/2 + y + 47)^(1/2))*(y + 47))/abs(x/2 + y + 47)^(1/2) +
(x*sign(y - x + 47)^2*cos(abs(y - x + 47)^(1/2)))/(4*abs(y - x +
47)^(3/2)) + (x*sign(y - x + 47)^2*sin(abs(y - x +
47)^(1/2)))/(4*abs(y - x + 47)) + (sign(x/2 + y + 47)^2*cos(abs(x/2
+ y + 47)^(1/2))*(y + 47))/(4*abs(x/2 + y + 47)^(3/2))]
%}

out = f_hess(x_k(1), x_k(2));
end
```