HW4 ECE 719 Optimal systems

BY

NASSER M. ABBASI

DECEMBER 30, 2019

Contents

List of Figures

List of Tables

0.1 Problem 1

ECE 717 − Homework Amplifier

In this homework problem, we consider the 2-stage amplifier described in class with objective function

$$J(u) = (11 - u_1 - u_2)^2 + (1 + u_1 + 10u_2 - u_1 u_2)^2$$

to be minimized.

(a) Generate a contour plot for the region in $\mathbf{R}^2$ of interest described by $0 \leq u_1 \leq 20$ and $0 \leq u_2 \leq 15$.

(b) Write your own Matlab code to implement the steepest descent algorithm with fixed step size. Include your code as an appendix.

(c) Run your algorithm from a variety of initial conditions which include

$$u^0 = \begin{bmatrix} 8 \\ 12 \end{bmatrix}, \begin{bmatrix} 5 \\ 10 \end{bmatrix}, \begin{bmatrix} 12 \\ 14 \end{bmatrix}, \begin{bmatrix} 12 \\ 10 \end{bmatrix}$$

and experiment with step sizes which include $h = 0.01, 0.10, 1.0$ and include comments about convergence, number of iterations, stopping criterion and oscillations. In each case, show the progress of your iterations by superimposing the iterative path u^k on the contour plot. Annotate your plots with relevant comments.

(d) Notice that at the point

$$u^0 = \begin{bmatrix} 7 \\ -2 \end{bmatrix}$$

we see $\nabla J(u) = 0$. Might your algorithm begin with $u_1 \geq 0, u_2 \geq 0$ and converge too this point? Discuss briefly.

Figure 1: problem 1 description

0.1.1 part(a)

Matlab was used to generate the contour plots. The plots generated are given below and the source code used is listed in the appendix.
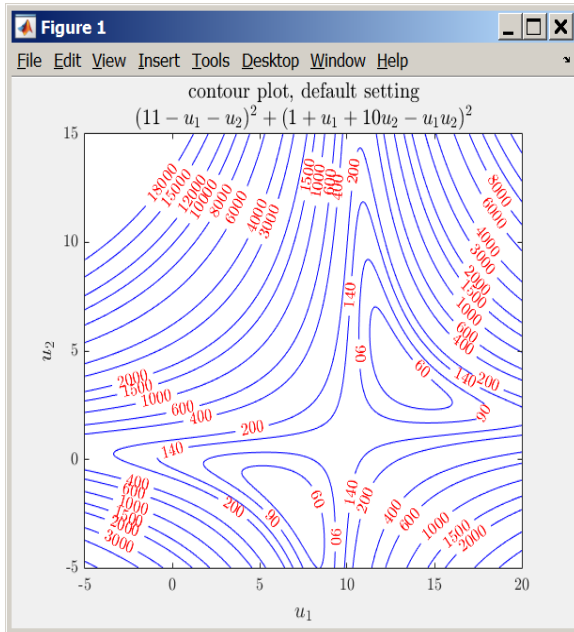
contour plot, problem 1
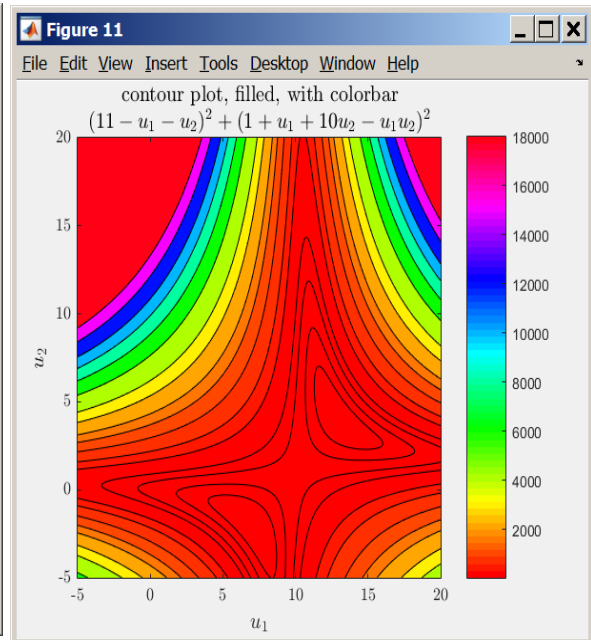
contour plot, filled, with colorbar problem 1
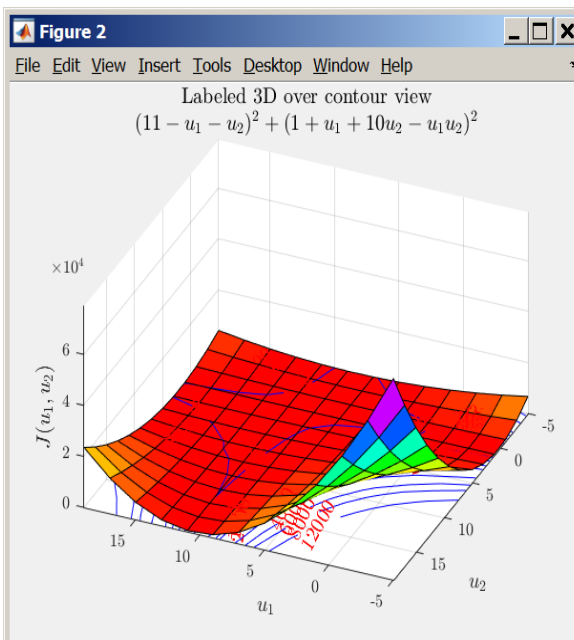
3D view with contour plot, problem 1
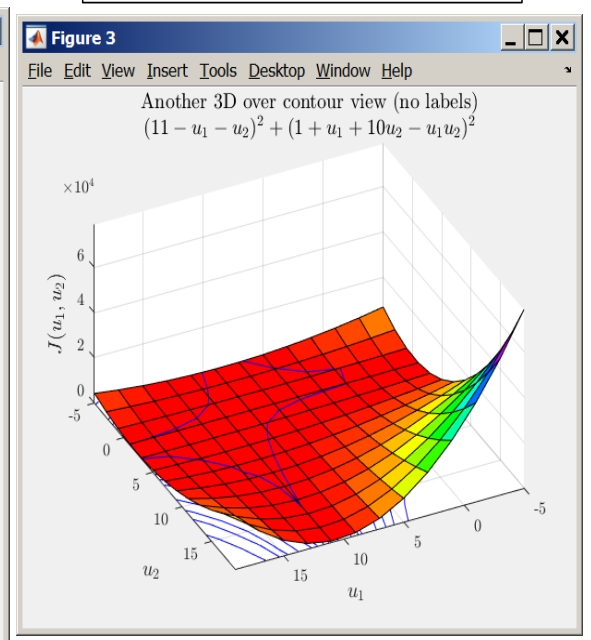
Another 3D view with contour plot, problem 1
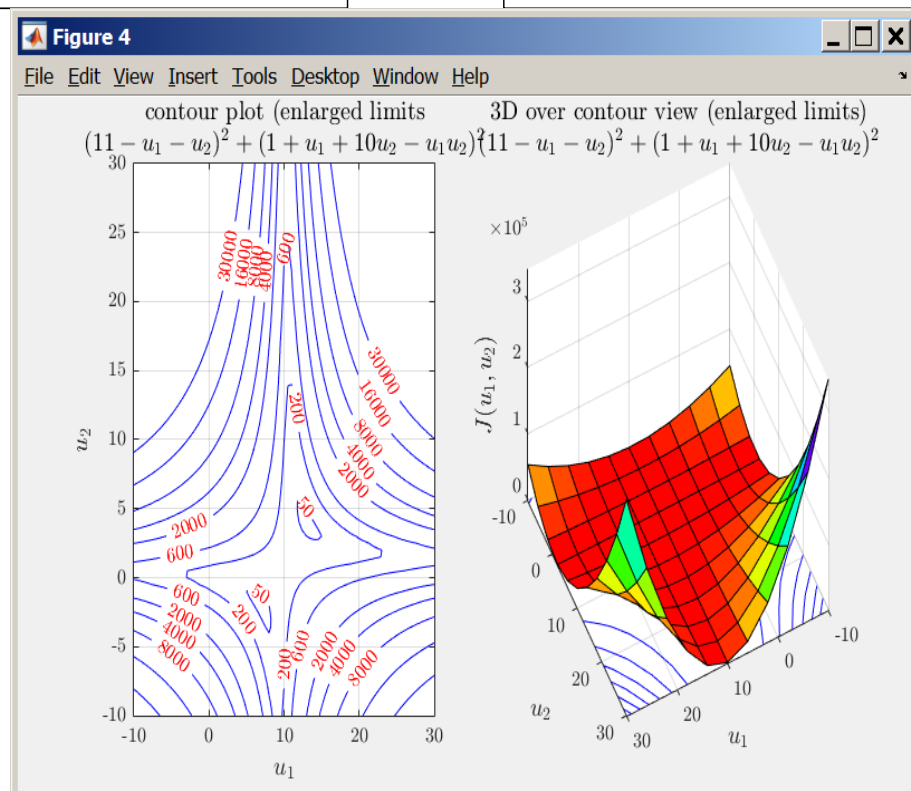
Combined 3D view and contour plot, problem 1

Figure 2: Matlab output for part(a) problem 1, HW4

0.1.2 part(b)

Matlab 2015a was used to implement steepest descent algorithm. Listing is given in the appendix. The following is the outline of general algorithm expressed as pseudo code.

Algorithm 1 Steepest descent with fixed step size search algorithm

---

1: **procedure** S T E E P E S T _ D E S C E N T
2:    ▷ Initialization
3:    $h \leftarrow$ step size
4:    $\epsilon \leftarrow$ minimum convergence limit on $\|\nabla J(u)\|$
5:    $k \leftarrow 0$
6:    $u^k \leftarrow u^0$
7:    $max\_iterations \leftarrow$ max iterations allowed

8:    **while** $\|\nabla J(u^k)\| > \epsilon$ **do**
9:       $u^k \leftarrow u^k - h\frac{\nabla J(u^k)}{\|\nabla J(u^k)\|}$
10:      $k \leftarrow k + 1$
11:      ▷ check for oscillation
12:      **if** $k \geq$ max_iterations **or** $J(u_k) > J(u_{k-1})$ **then**
13:         **exit loop**
14:      **end if**
15:    **end while**
16: **end procedure**

Figure 3: Steepest descent with fixed step size search algorithm

0.1.3 part(c)

In all of the following results, the convergence to optimal was determined as follows: First a maximum number of iterations was used to guard against slow convergence or other unforeseen cases. This is a safety measure. It is always recommended to use in any iterative method. This number was set very high at one million iterations. If convergence was not reached by this count, the iterations stop.

The second check was the main criteria, which is to check that the norm of the gradient $|\nabla(J(u))|$ has reached a minimum value. Since $|\nabla(J(u))|$ is zero at the optimal point, this check is the most common one to use to stop the iterations. The norm was calculated after each step. When $|\nabla(J(u))|$ became smaller than 0.01, the search stopped. The value 0.01 was selected arbitrarily. All cases below used the same convergence criterion.

A third check was added to verify that the value of the objective function $J(u)$ did not increase after each step. If $J(u)$ increased the search stops, as this indicates the step size taken is too large and oscillation has started. This condition happened many times when using fixed step size, but it did not happen with optimal step size.

The relevant Matlab code used to implement this convergence criteria is the following:

```
%check if we converged or not
if k>opt.MAX_ITER || gradientNormTol(k)<=opt.gradientNormTol ...
|| (k>1 && levelSets(k)>levelSets (k-1)) % check for getting worst
    keepRunning = false;
else
    ....
end
```

The result of these runs are given below in table form. For each starting point, the search path u^k is plotted on top of the contour plot. Animation of each of these is also available when running the code. The path u^k shows that search direction is along the gradient vector, which is perpendicular to the tangent line at each contour level.

Table 1: Starting point [8,12]

h	# steps	comments

0.01	1087	Converged with no oscillation detected. Below are the last few values of $J(u)$
		```K>> levelSets(end-10:end)```
		40.000847560444
		40.0002241269404
		40.0000006868238
		Below are the corresponding values of $\|\nabla(J(u))\|$
		```K>> gradientNormTol(end-6:end)```
		0.122339282346846
		0.0823426325071764
		0.042343897716672
		0.00234405713552924

0.1	129	Failed to converge. Oscillation started when near optimal point. Below are the last few values of $J(u)$ that shows this.
		```K>> levelSets(end-10:end)```
		40.0906178557323
		40.0146606611128
		40.0906176333215
		These are the corresponding values of $\|\nabla(J(u))\|$
		```K>> gradientNormTol(end-6:end)```
		1.0342875633952
		2.51122413813222
		1.03429217902894
		2.51122268542765

1	14	Early termination as the objective function started to increase as the step size was large. Oscillation started early. Below are the last few values of $J(u)$ recorded that shows this.
		```K>> levelSets(end-10:end)```
		43.8208310324077
		45.023624369293
		43.781716244717
		45.006474191836
		Below are the corresponding values of $\|\nabla(J(u))\|$
		```K>> gradientNormTol(end-6:end)```
		18.2210845193641
		16.4816791388241
		18.1783873100515
		16.4576741878144

Search path on top of contour plot

Search path on top of filled contour plot

zoom

3D view of search path

Fixed step h = 0.01 results

Zooming to show there
are NO oscillation
close to optimal point
since step size is small

Objective function getting smaller during search

Gradient norm approaching convergence limit

Figure 4: Result for using step 0.01 starting from (8,12)

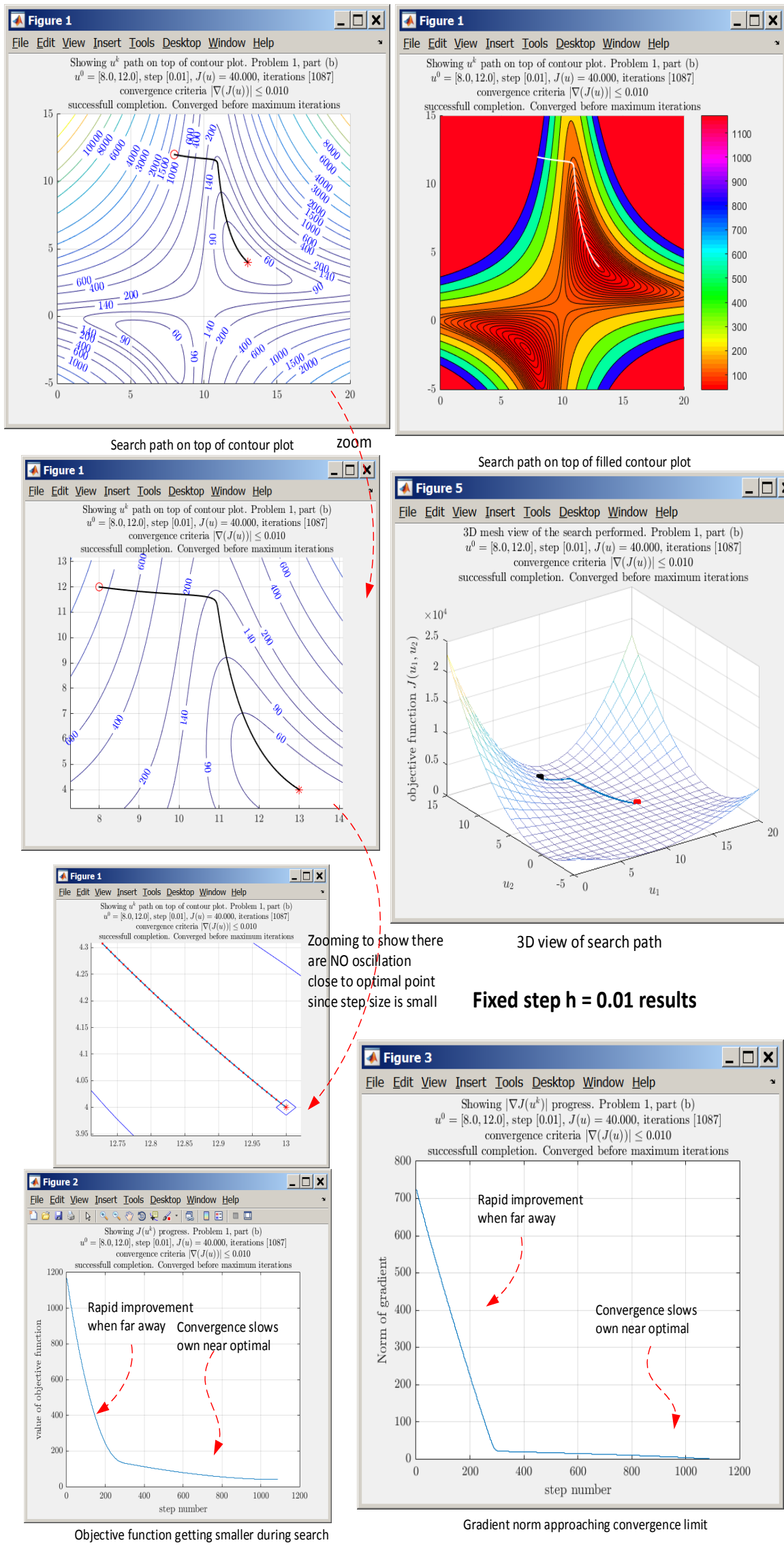Figure 5: Result for using step 0.1 starting from (8,12)

Search path on top of contour plot,
oscillation due to large step size
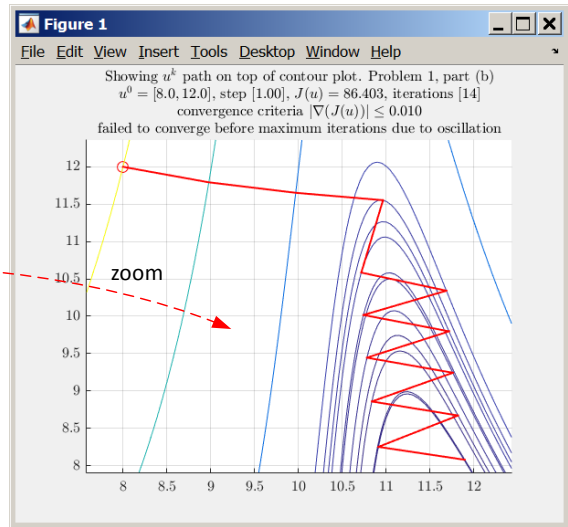
Search terminated early. Large step size
caused oscillation. No convergence possible
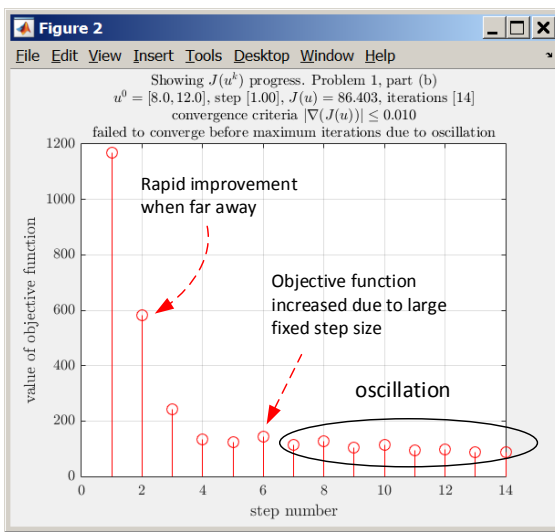
Objective function progress using large fixed size

Gradient norm approaching convergence limit

Fixed step h = 1 results

Figure 6: Result for using step 1 starting from (8,12)

Table 2: Starting point [5,10]

h	steps to converge	comments

0.01	1311	Search reached optimal point $(13,4)$, but skipped over it and started to oscillate back and forth around the optimal point due to using large fixed step size. Below are the last few values of $J(u)$ recorded showing this.
		`K>> levelSets(end-10:end)`
		`.....`
		        40.000714475783
		        40.0002484312073
		        40.0007127154844
		        40.0002478317567
		        40.0007121302667
		The above shows that $J(u)$ is oscillating around J^*, while the $\lvert\nabla(J(u))\rvert$ has not yet become small enough to stop. These are the corresponding values of $\lvert\nabla(J(u))\rvert$
		`K>> gradientNormTol(end-6:end)`
		`....`
		        0.226174843552625
		        0.133516310474324
		        0.226094172083413
		        0.133583571337061
		        0.226067390166402
		Even though this test used a small step size and the algorithm converged when starting from $(8,12)$ as shown in the earlier case, but this time it did not converge.
		This shows that the search is sensitive to the starting point when using fixed step size. One way to correct this problem is to relax the convergence criteria.

0.1	123	Failed to converge. Oscillation detected near optimal point. Below are the last few values of $J(u)$ recorded showing this.
		`K>> levelSets(end-10:end)`
		`.....`
		        40.1256594812986
		        40.0053368705834
		        40.1256594634014
		        40.0053368695271
		        40.1256594618631
		Below are the corresponding values of $\lvert\nabla(J(u))\rvert$
		`K>> gradientNormTol(end-6:end)`
		`....`
		        3.06656767477006
		        0.61736163474876
		        3.06656750731774
		        0.617361766949031
		        3.06656749292543

Table 2 – continued from previous page

1	19	Early termination due to the objective function starting to increase since the step size was too large. Oscillation started early in the search. Here are the last few values of $J(u)$ showing this

```
K>> levelSets(end-10:end)
.....
          43.0823019294829
          45.7913265189839
          43.0266791615351
          45.7622114747819
```

Below are the corresponding values of $|\nabla (J(u))|$

```
K>> gradientNormTol(end-6:end)
....
          16.1440020280613
          17.487837406306
          16.092991548592
          17.4442963174089
```
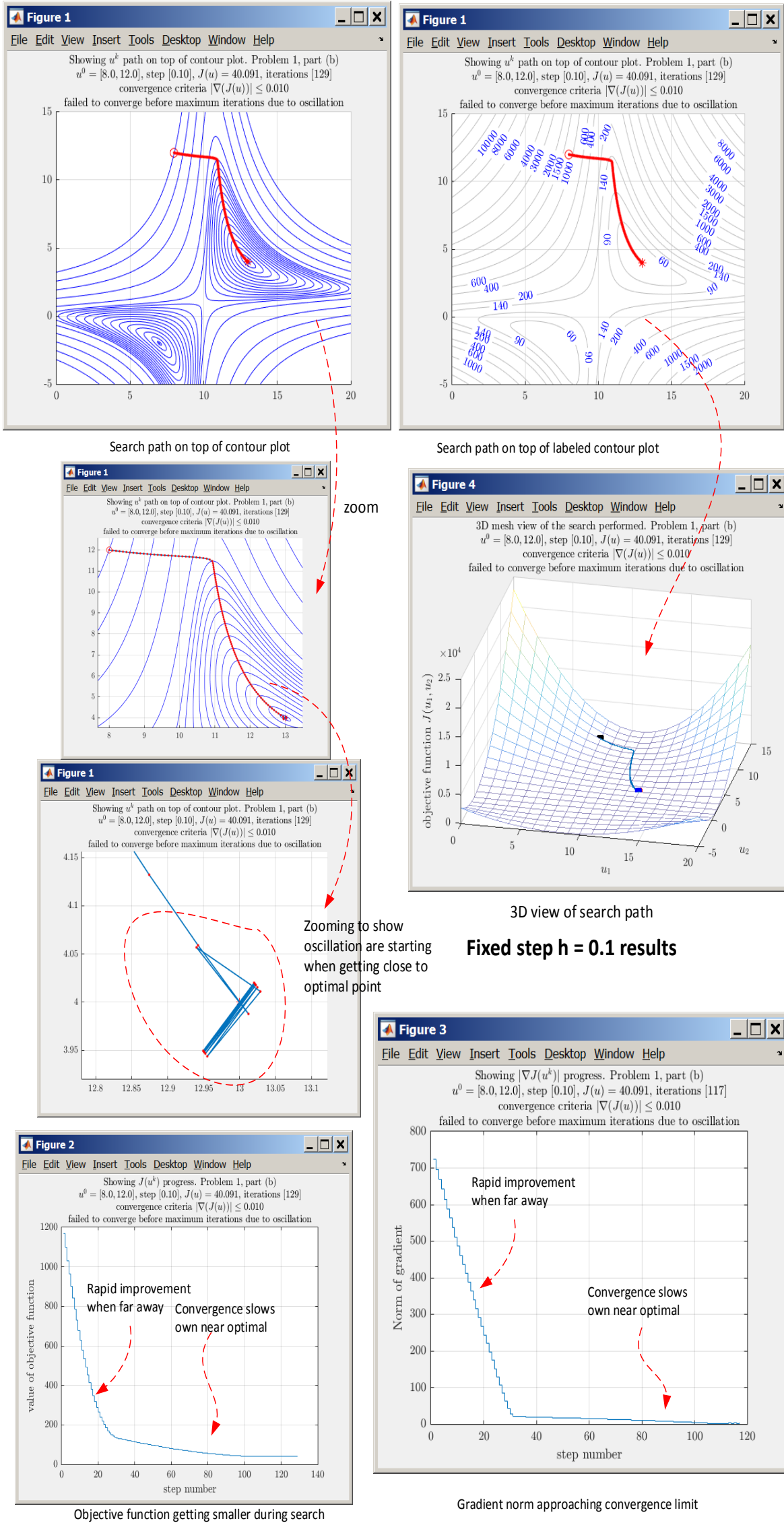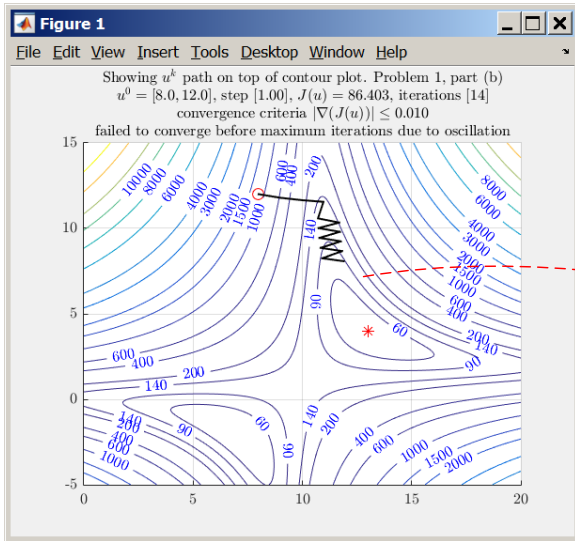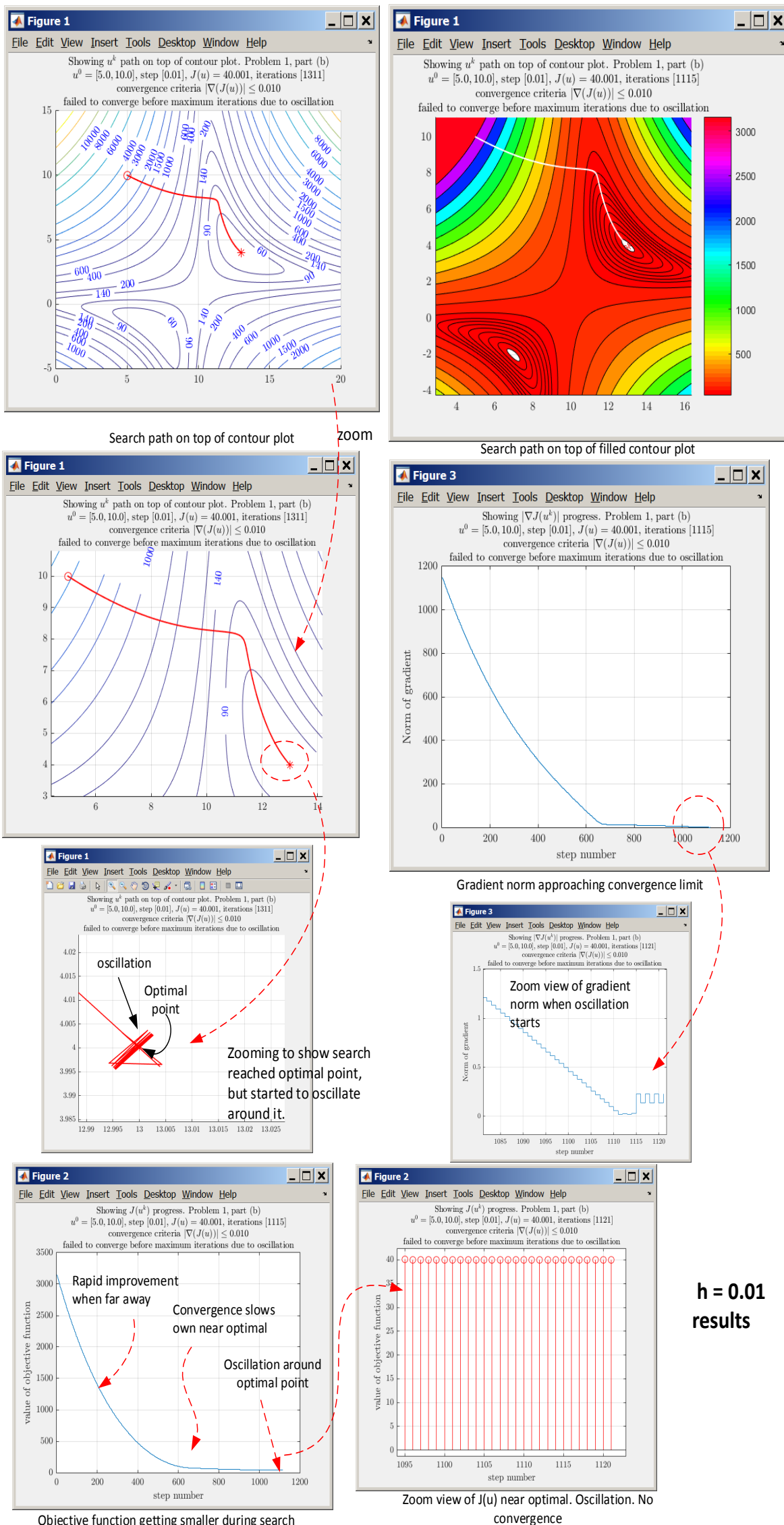
Figure 7: Result for using step 0.01 starting from (5,10)
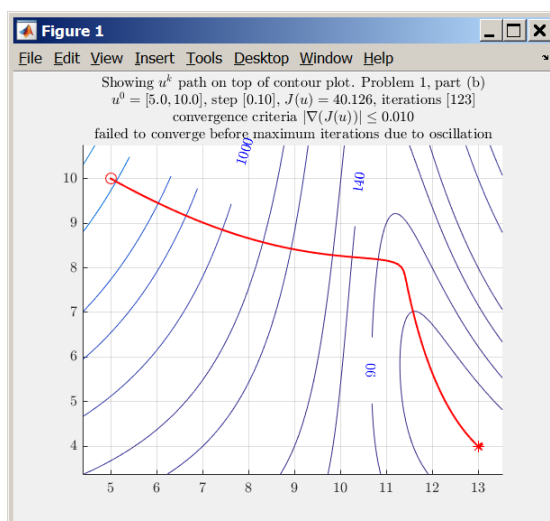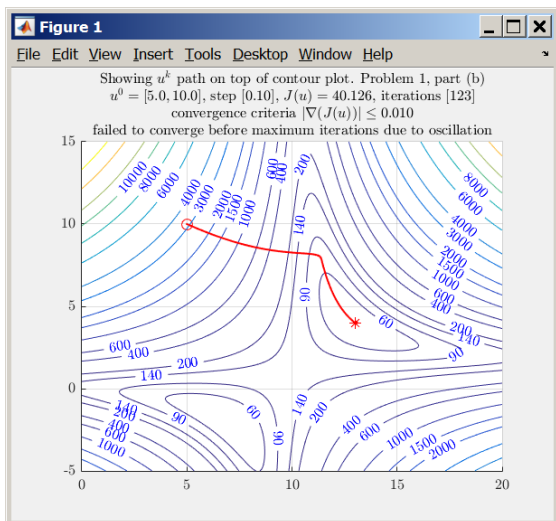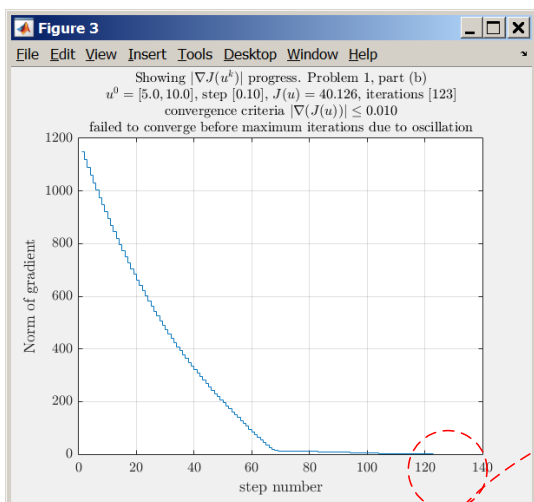
Search path on top of contour plot

Search path on top of filled contour plot

Gradient norm approaching convergence limit
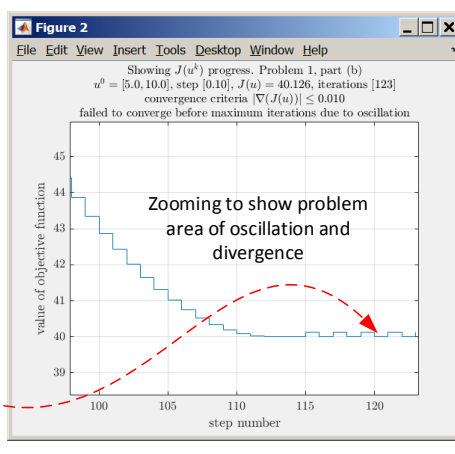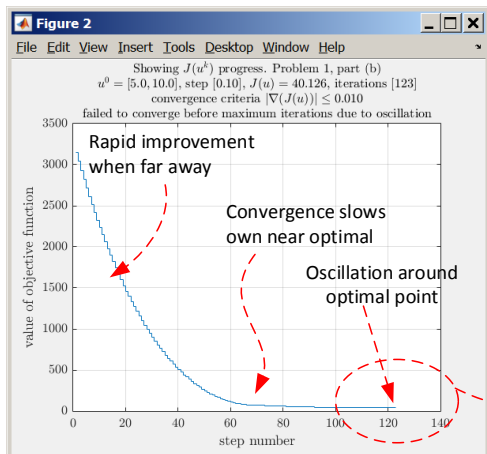
Objective function getting smaller during search

Zoom view of J(u) near optimal. Oscillation. No convergence

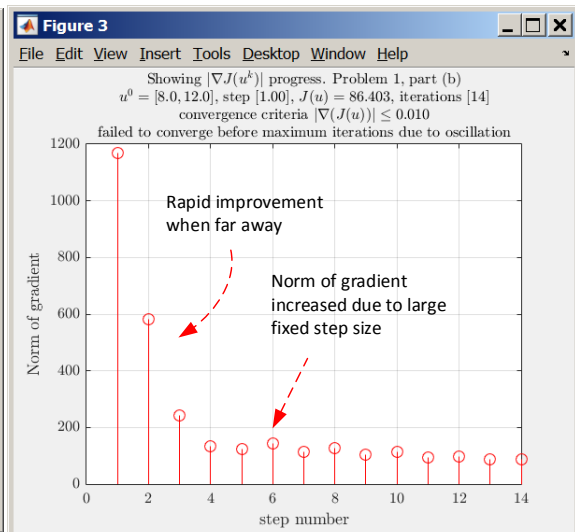h = 0.1 results

Figure 8: Result for using step 0.1 starting from (5,10)
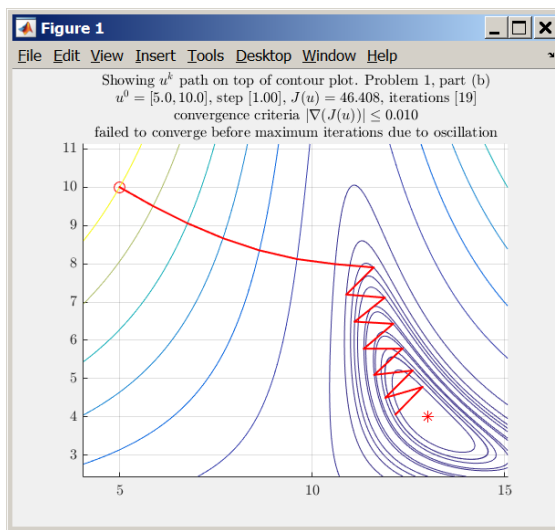
Search path on top of contour plot

Gradient norm approaching convergence limit

Objective function getting smaller during search

Zoom view of J(u) near optimal. Oscillation. No convergence

h = 1 results

Figure 9: Result for using step 1 starting from (5,10)

Table 3: Starting point [12,14]

h	steps to converge	comments

0.01	1130	Search reached optimal point $(13, 4)$ and did converge. No oscillation were detected. Here are the last few values of $J(u)$ recorded
		`K>> levelSets(end-10:end)`
		`.....`
		`          40.0034914479994`
		`          40.002020228495`
		`          40.0009489569455`
		`          40.0002776602642`
		`          40.0000063555764`
		The above shows that $J(u)$ did not oscillate and continued to become smaller with each step. These are the corresponding values of $\|\nabla(J(u))\|$ showing it reached convergence criteria and stopped.
		`K>> gradientNormTol(end-6:end)`
		`....`
		`          0.167118334256662`
		`          0.127125180003955`
		`          0.0871288452215103`
		`          0.047130308356715`
		`          0.00713054850822947`

0.1	131	Failed to converge due to oscillation Below are the last few values of $J(u)$ recorded showing that it has increased.
		`K>> levelSets(end-10:end)`
		`.....`
		`          40.1051079160718`
		`          40.0105348693244`
		`          40.1051060970453`
		`          40.0105346146167`
		`          40.1051057241206`
		The above shows that $J(u)$ started to oscillate near the optimal point since the step size was large. These are the corresponding values of $\|\nabla(J(u))\|$
		`K>> gradientNormTol(end-6:end)`
		`....`
		`          2.80005566566667`
		`          0.865917403257339`
		`          2.80004081985152`
		`          0.865928703656839`
		`          2.8000377762892`

1	19	Early termination due to the objective function increasing since the step size was too large. Below are the last few values of $J(u)$ recorded showing this
		`K>> levelSets(end-10:end)`
		`.....`
		`          136.072742913828`
		`          147.365512785727`
		`          125.964493512448`
		`          133.478776121489`
		`          115.810171973447`
		`          120.277823711614`
		Below are the corresponding values of $\|\nabla(J(u))\|$
		`K>> gradientNormTol(end-6:end)`
		`....`
		`          111.538416550055`
		`          76.4541018810368`
		`          98.2477444652928`
		`          70.7519791844584`
		`          85.8602921445108`

Search path on top of contour plot

Gradient norm approaching convergence limit

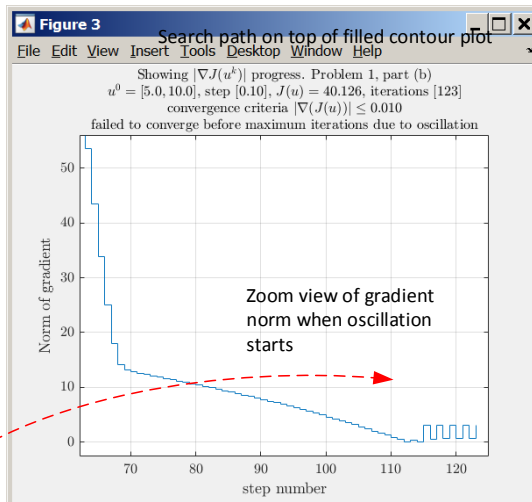Objective function getting smaller during search

Zoom view of J(u) near optimal. converged

Figure 10: Result for using step 0.01 starting from (12,14)
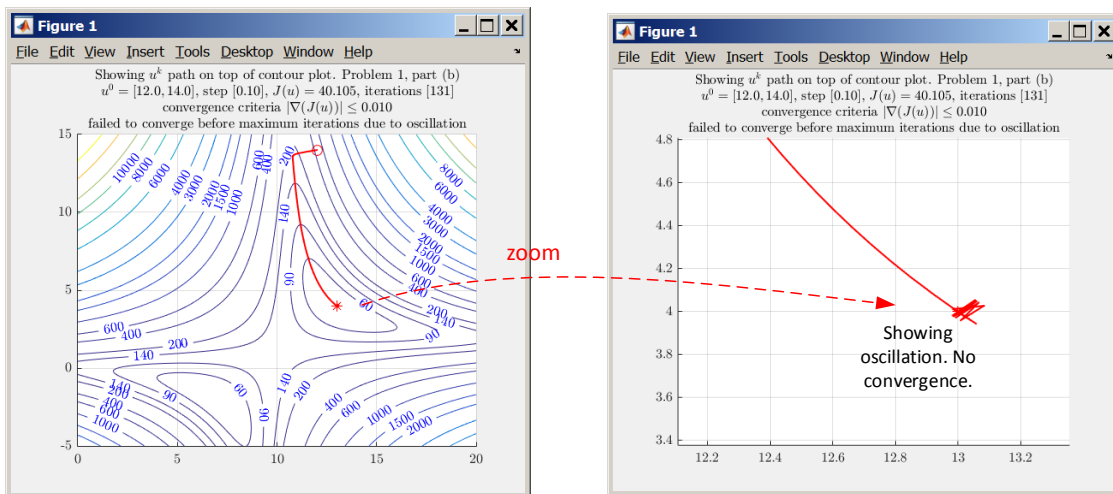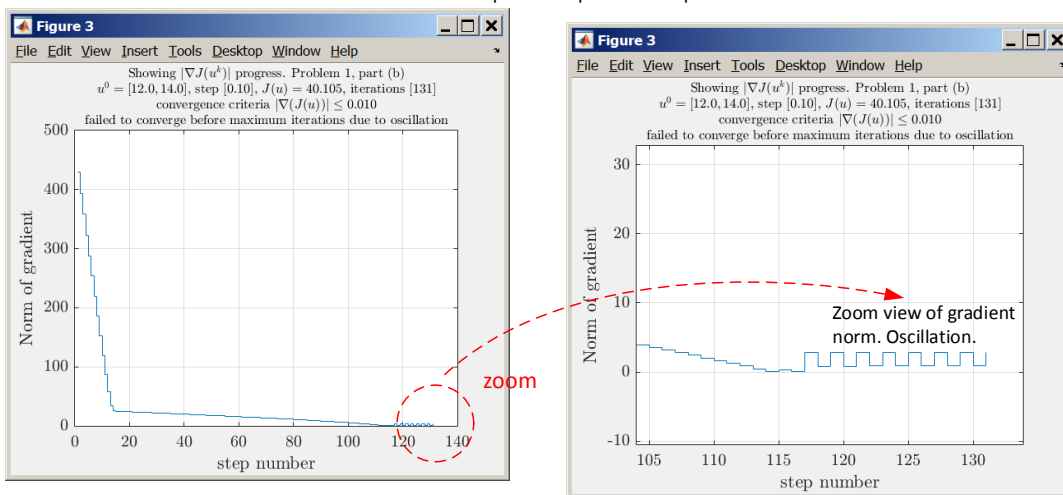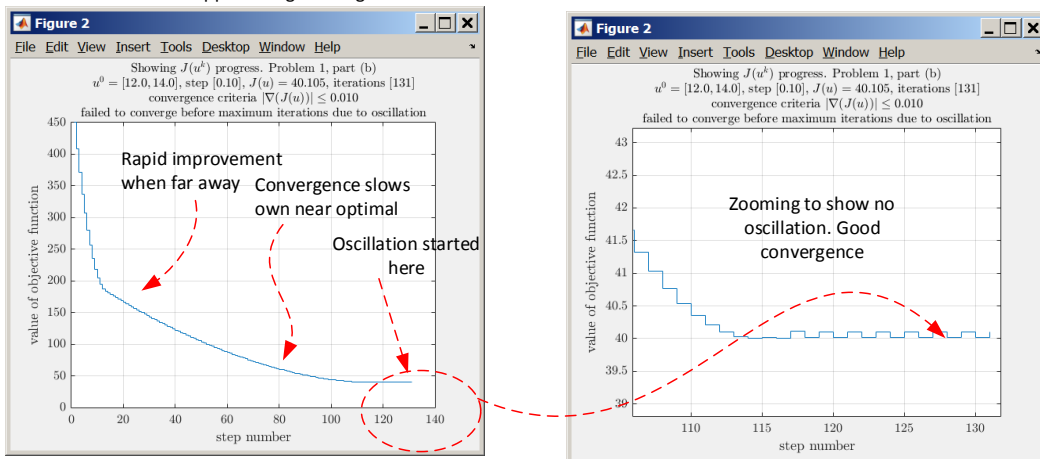
Figure 11: Result for using step 0.1 starting from (12,14)
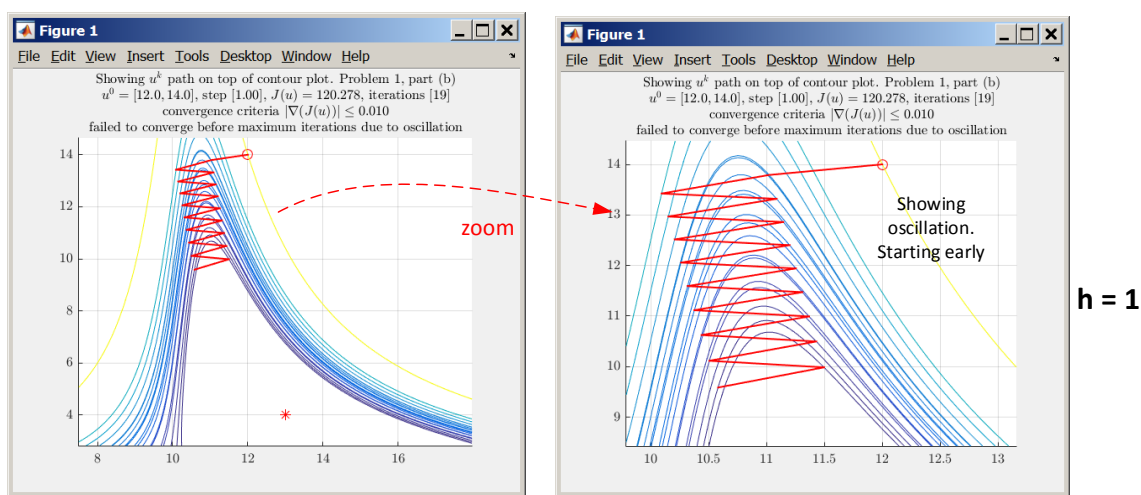
Figure 12: Result for using step 1 starting from (12,14)

Table 4: Starting point [12,10]

| h	steps to converge	comments

0.01	691	Converged with no oscillation. Here are the last few values of $J(u)$ recorded confirming this.
		`K>> levelSets(end-10:end)`
		`.....`
		`40.0046068598544`
		`40.0028871867126`
		`40.0015674398797`
		`40.0006476523458`
		`40.0001278473181`
		`40.0000080382134`
		Below are the corresponding values of $\|\nabla(J(u))\|$
		`K>> gradientNormTol(end-6:end)`
		`....`
		`0.151971746241737`
		`0.111977272332977`
		`0.0719799883799201`
		`0.0319808731053423`
		`0.00801909420920947`

0.1	87	Did not converge. Oscillation was detected. Below are the last values of $J(u)$ recorded confirming this.
		`K>> levelSets(end-10:end)`
		`.....`
		`40.0940178225724`
		`40.0143577207974`
		`40.0940127829831`
		`40.0143567476265`
		`40.0940114931914`
		Below are the corresponding values of $\|\nabla(J(u))\|$ showing it is diverging.
		`K>> gradientNormTol(end-6:end)`
		`....`
		`1.00986396810643`
		`2.64564970050157`
		`1.00989167493457`
		`2.6456402389648`

1	24	Did not converge. Oscillation was detected early in the search due to using large step size. Below are the last few values of $J(u)$ recorded confirming this.
		`K>> levelSets(end-10:end)`
		`.....`
		`45.2261295001543`
		`43.5283233241446`
		`45.2260318140989`
		`43.5282741210766`
		`45.2260091586802`
		These are the corresponding values of $\|\nabla(J(u))\|$ showing it is diverging.
		`K>> gradientNormTol(end-6:end)`
		`....`
		`16.7542019931462`
		`17.5230111072761`
		`16.7540613766743`
		`17.5229596031784`
		`16.7540287643191`

Figure 13: Result for using step 0.01 starting from (12,10)
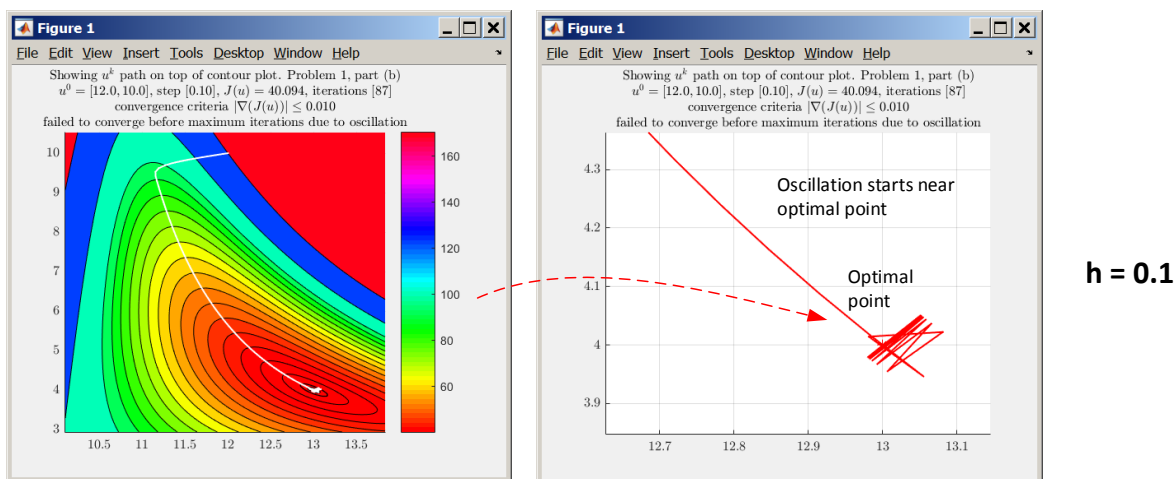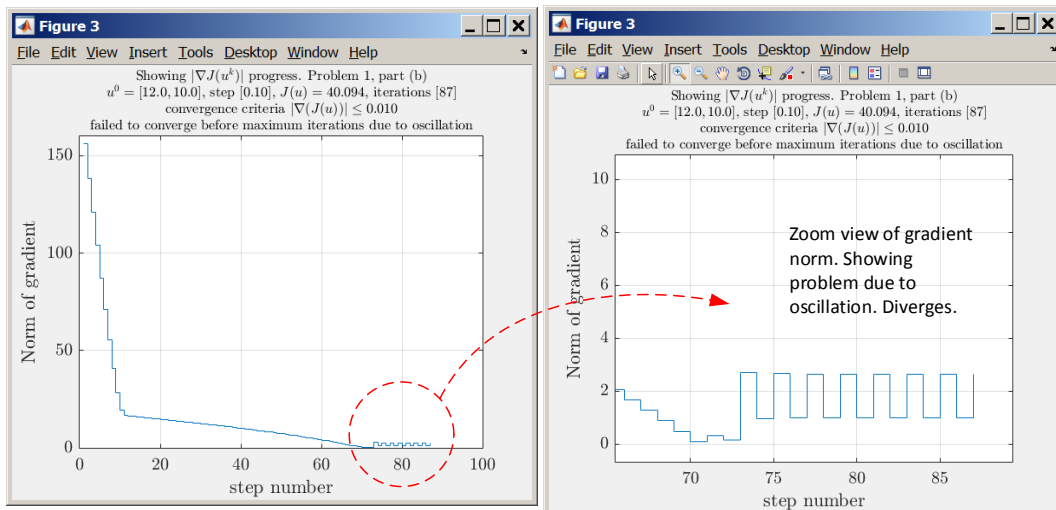
Search path on top of contour plot
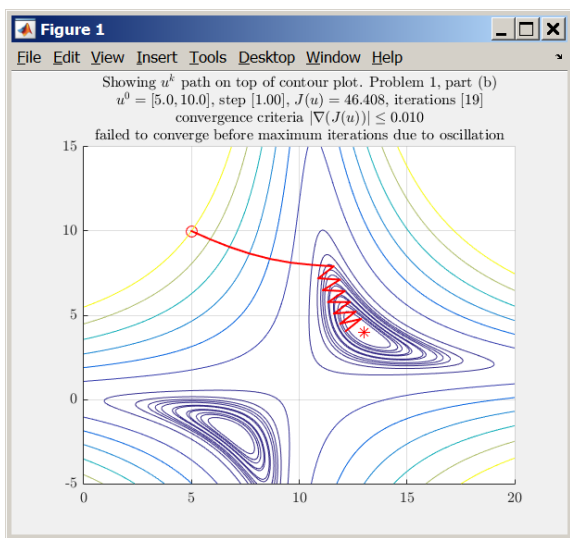

Gradient norm approaching convergence limit
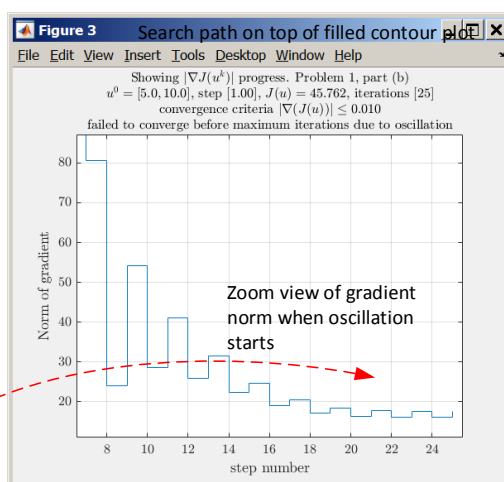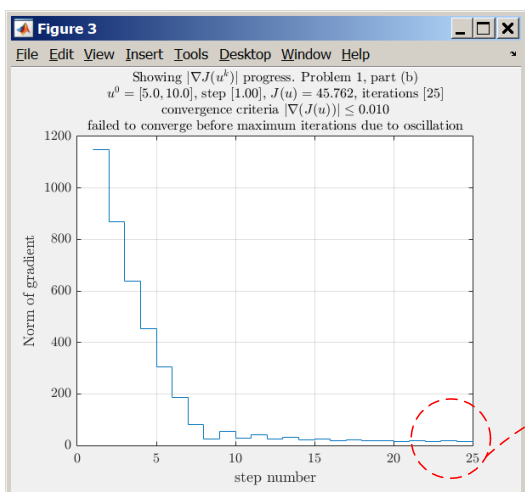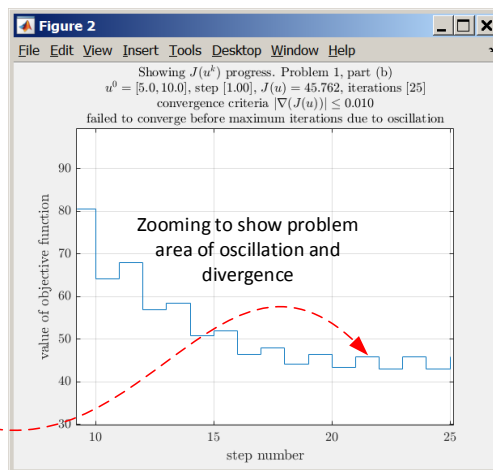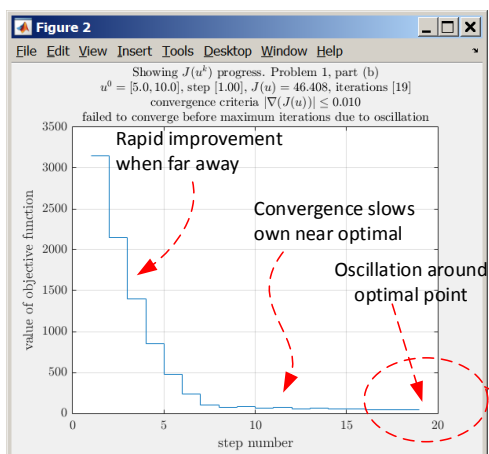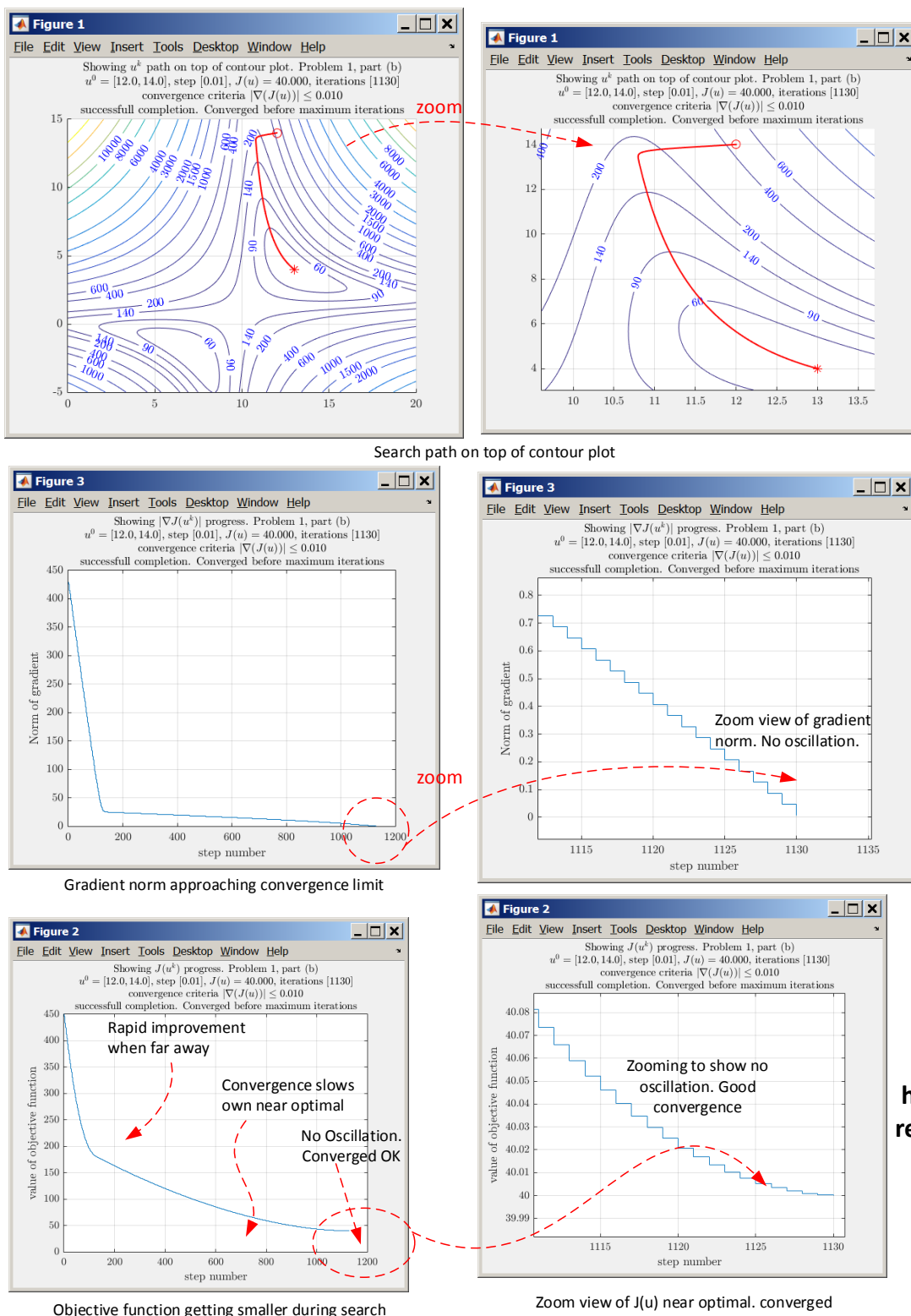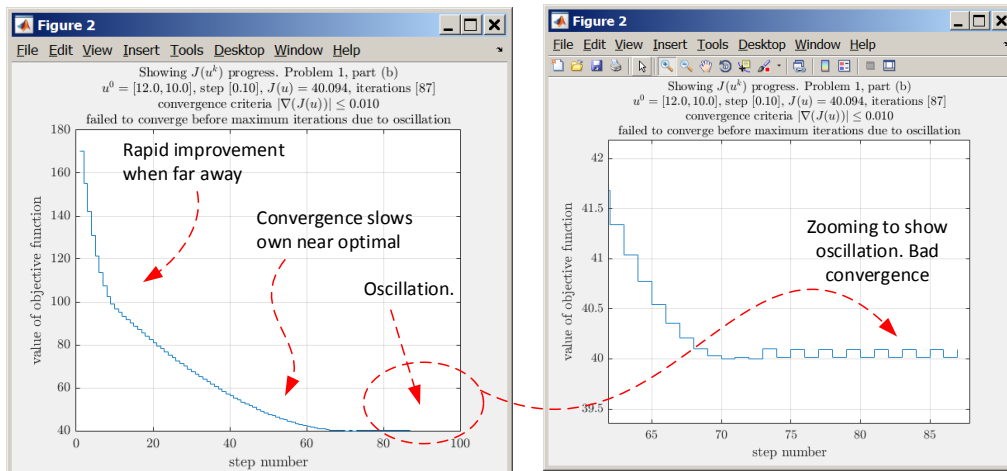

Objective function getting smaller during search

Zoom view of J(u) showing oscillation

Figure 14: Result for using step 0.1 starting from (12,10)
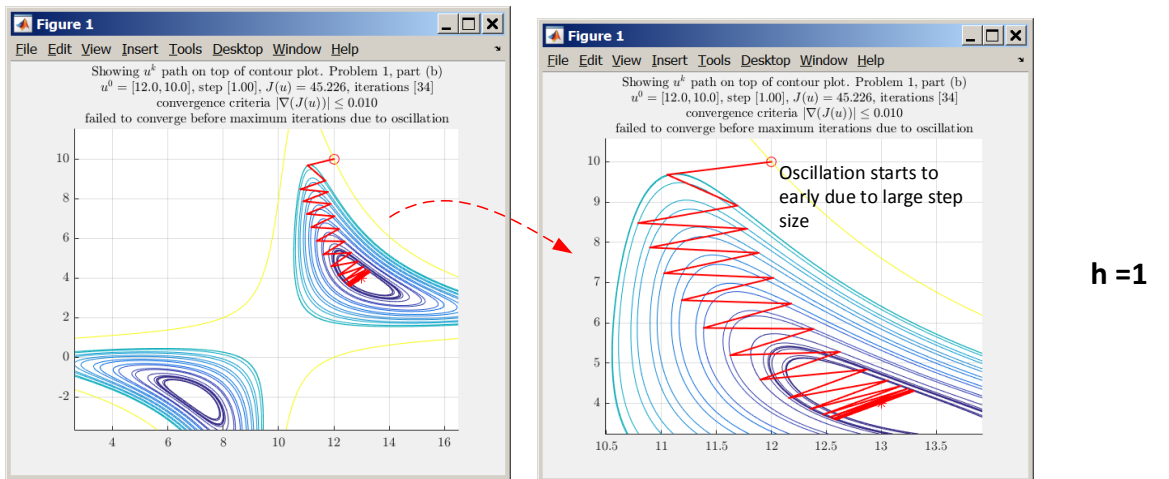
Search path on top of contour plot
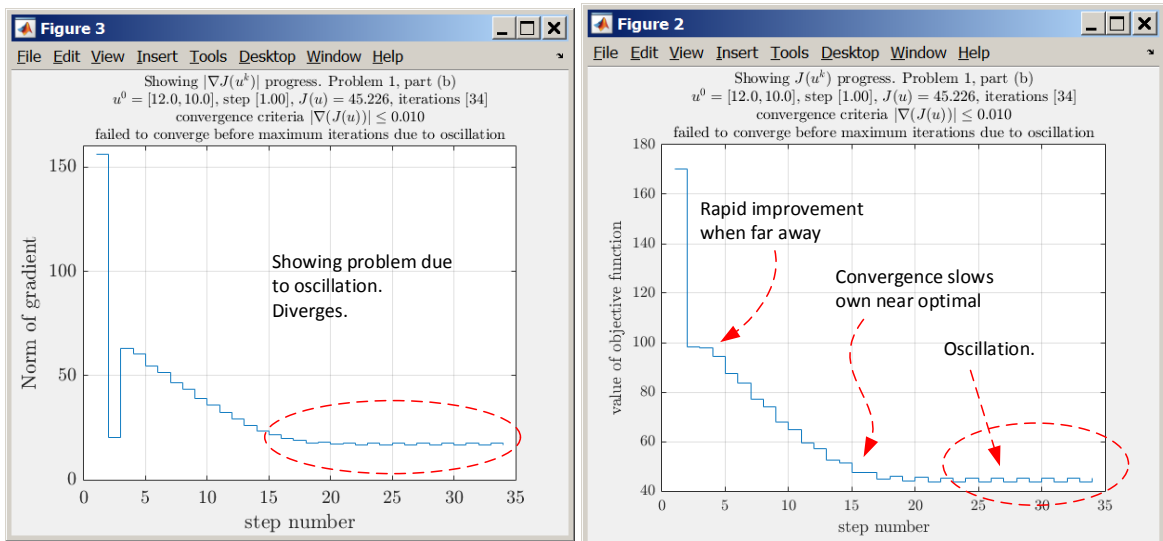
Gradient norm showing divergence

Objective function getting smaller showing
oscillation early on

Figure 15: Result for using step 0.1 starting from (12,10)

0.1.4 Part(d)

When trying different values of starting points, all with $u_1 > 1, u_2 > 0$, the search did converge to $u^* = [7, -2]$, but it also depended on where the search started from. When starting close to u^*, for example, from $u^0 = [6.5, 1]$ the search did converge using fixed step size of $h = 0.01$ with no oscillation seen. Below shows this result
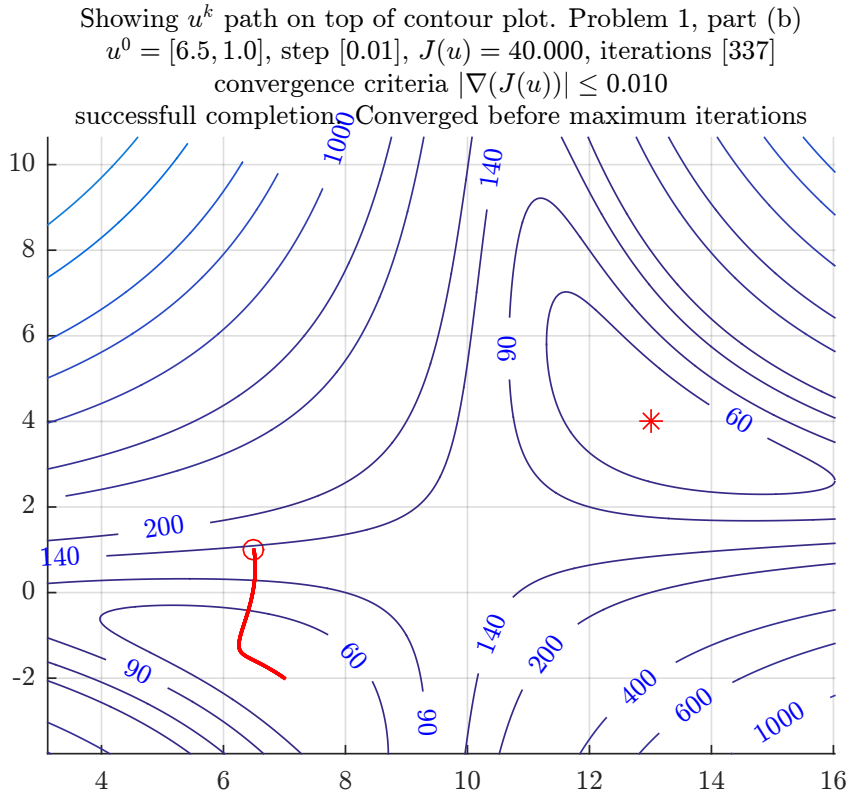
Figure 16: Converging to $(7, -2)$ using step size 0.01 starting from $(6.5, 1)$

However, when starting from a point too far away from $(7, -2)$, it did not converge to $(7, -2)$, but instead converged to the second local minimum at $u^* = [13, 4]$ as seen below. In this case the search started from $[20, 20]$.

If the starting point was relatively close to one local minimum than the other, the search will converge to nearest local minimum.
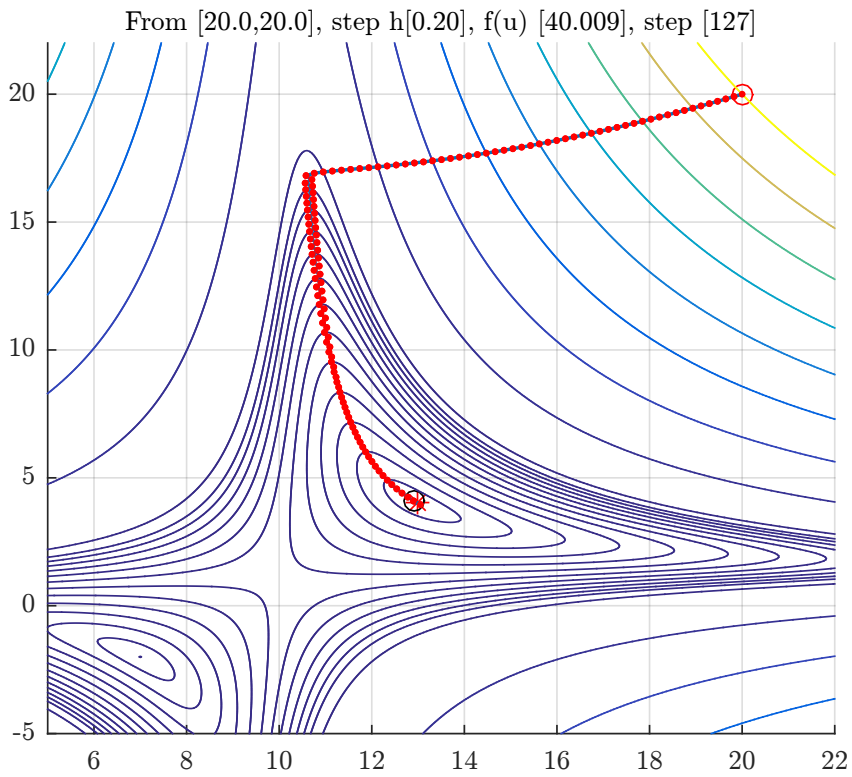
Figure 17: Missing $u^* = [7, -2]$ when starting too far it. Starting from $(20, 20)$ using step size 0.01

In this problem there are two local minimums, one at $(7, -2)$ and the other at $(4, 13)$. It depends on the location of the starting point u^0 as to which u^* the algorithm will converge to.

0.2 Problem 2

Barmish

ECE 719 – Homework Rosenbrock

For $n \geq 2$, consider Rosenbrock's Banana

$$J(u) = \sum_{i=1}^{n-1} 100(u_{i+1} - u_i^2)^2 + (1 - u_i)^2$$

with interesting domain

$$-2.5 \leq u_i \leq 2.5; \quad i = 1, 2, \ldots, n.$$

This is a commonly used *benchmark testing function* with known global minimum $J^* = 0$ which is attained with all $u_i = 1$. Note that this function also has local minima.

(a) For n = 2, use the steepest descent algorithm to study the minimization of the function above. Consider both the fixed and optimal step size cases. Provide a simple-to-read report on the performance including commentary and trajectories of the iterates u^k superimposed on the contour plot from a variety of initial conditions u^0. Also indicate the line search method which you used.

(b) Repeat the study in Part (a) for larger values of n. How large can you push n and still achieve reasonable performance? Discuss how computational effort grows as a function of n. Note that for $n > 2$, you need not display trajectories and contours.

Figure 18: problem 2 description
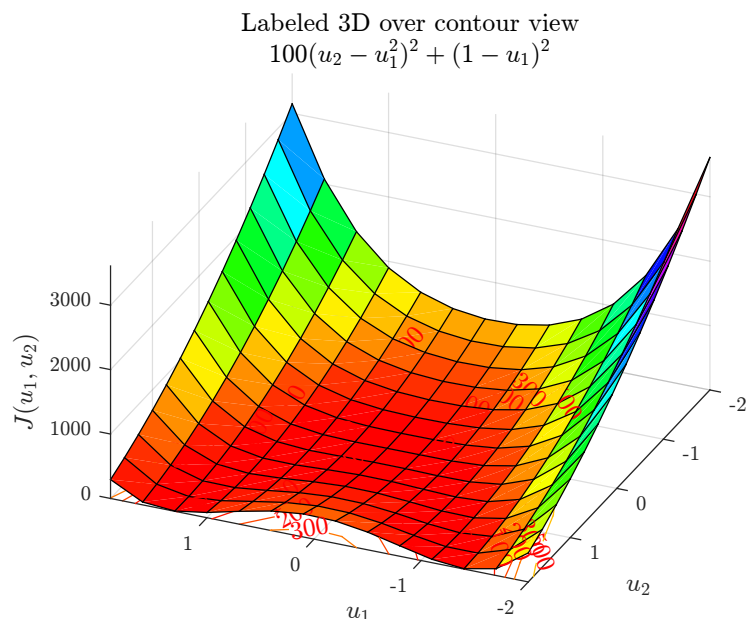
Figure 19: 3D view of $J(u)$

0.2.1 Part(a)

The steepest descent algorithm used in the first problem was modified to support an optimal step size. The following is the updated general algorithm expressed as pseudo code. The optimal step line search used was the standard golden section method. (Listing added to appendix).

```
1: procedure STEEPEST_DESCENT_OPTIMAL
2:        ▷ Initialization
3:        H ← maximum step size
4:        max_iterations ← max iterations allowed
5:        ε ← minimum convergence limit on ‖∇J(u)‖
6:        k ← 0
7:        u^k ← u^0

8:        while ‖∇J(u^k)‖ > ε do
9:               ▷ do line search
10:              h* ← call golden_section(H, J(u)) to find optimal h* of function J̃(h*) = J(u^k − h*∇J(u^k))
11:              u^k ← u^k − h* ∇J(u^k)/‖∇J(u^k)‖
12:              k ← k + 1
13:              ▷ detect oscillation
14:              if k ≥ max_iterations or J(u_k) > J(u_{k−1}) then
15:                     exit loop
16:              end if
17:        end while
18: end procedure
```

Figure 20: Steepest descent, optimal step size algorithm

For $n = 2$, the Rosenbrock function is

$$J(u) = 100\left(u_2 - u_1^2\right)^2 + (1 - u_1)^2$$

$$\nabla J(u) = \begin{bmatrix} \frac{\partial J}{\partial u_1} \\ \frac{\partial J}{\partial u_2} \end{bmatrix} = \begin{bmatrix} -400\left(u_2 - u_1^2\right)u_1 - 2\left(1 - u_1\right) \\ 200\left(u_2 - u_1^2\right) \end{bmatrix}$$

For

$$-2.5 \leq u_i \leq 2.5$$

The steepest algorithm was run on the above function. The following is the contour plot. These plots show the level set by repeated zooming around at $(1,1)$, which is the location of the optimal point. The optimal value is at $u^* = (1,1)$ where $J^* = 0$.
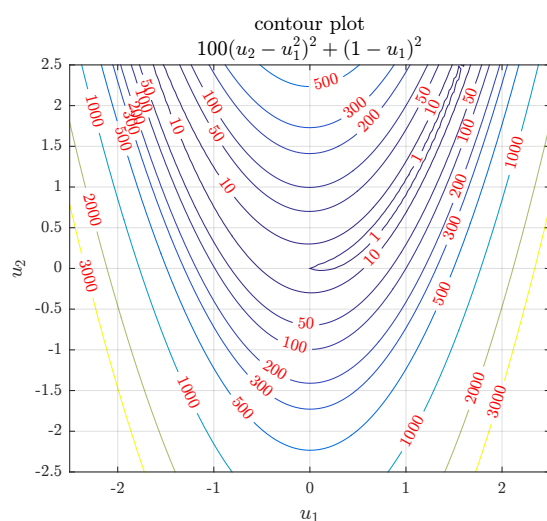
Figure 21: Contour $J(u)$
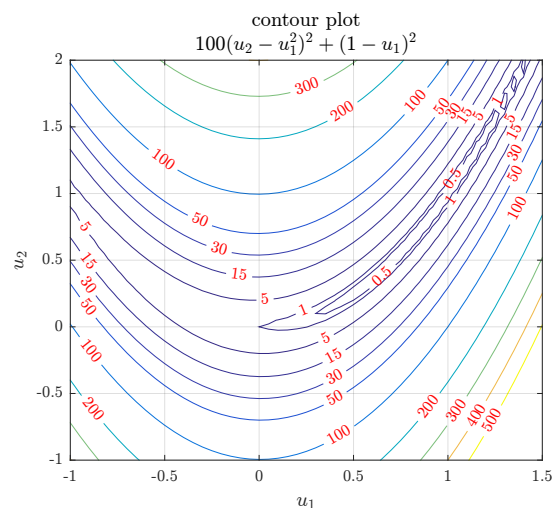
Figure 22: Zooming on Contour $J(u)$
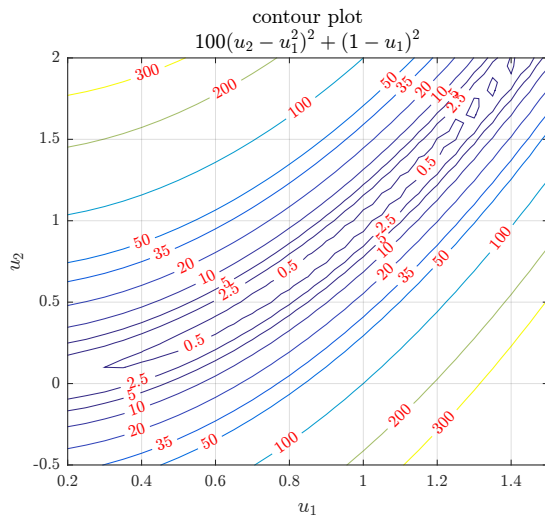
Figure 23: More zooming. Contour $J(u)$
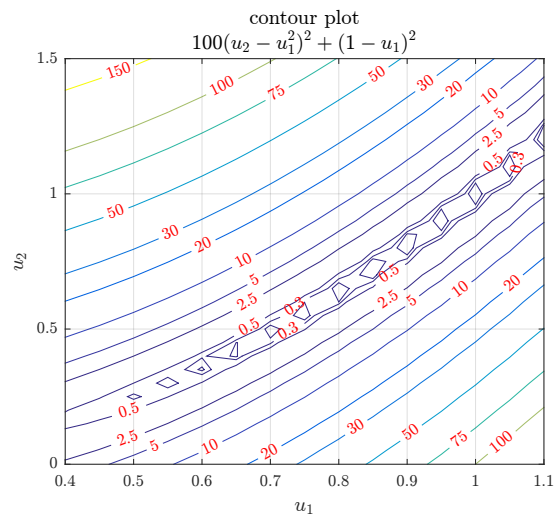
Figure 24: More zooming. Contour $J(u)$
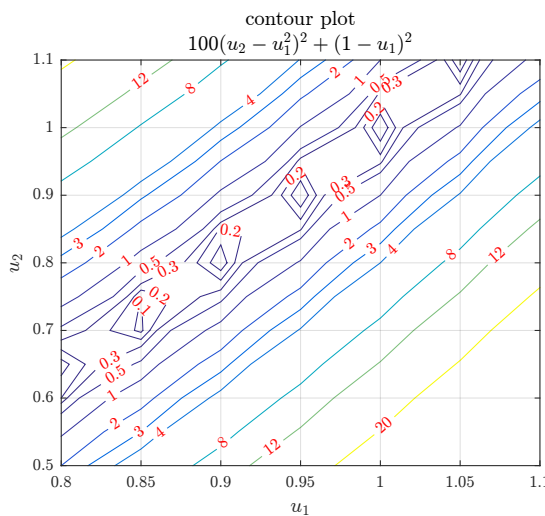
Figure 25: More zooming on Contour $J(u)$
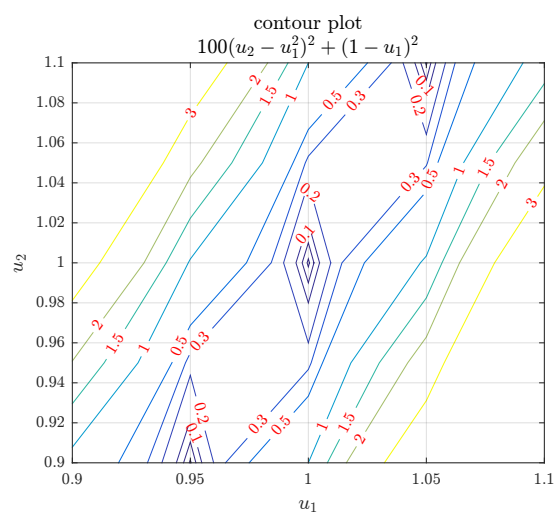
Figure 26: More zooming on Contour $J(u)$

In all of the results below, where fixed step is compared to optimal step, the convergence criteria was the same. It was to stop the search when

$$\|\nabla J(u)\| \leq 0.001$$

The search started from different locations. The first observation was that when using optimal step, the search jumps very quickly into the small valley of the function moving towards u^*. This used one or two steps only. After getting close to the optimal point, the search became very slow moving towards u^* inside the valley because the optimal step size was becoming smaller and smaller.

The closer the search was to u^*, the step size became smaller. Convergence was very slow at the end. The plot below shows the optimal step size used each time. Zooming in shows the zigzag pattern. This pattern was more clear when using small but fixed step size. Below is an example using fixed step size of $h = 0.01$ showing the pattern inside the valley of the function.
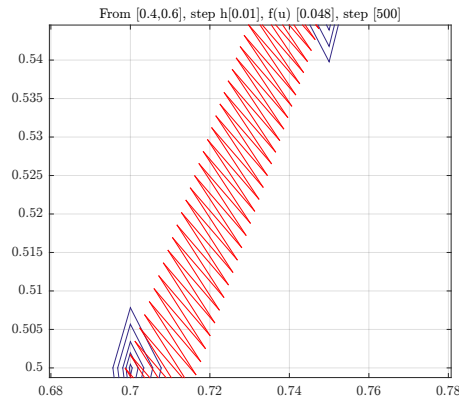
Figure 27: Zoom view of search when inside valley, showing the small steps and zig-zag pattern

Here is a partial list of the level set values, starting from arbitrary point from one run using optimal step. It shows that in one step, $J(u)$ went down from 170 to 0.00038, but after that the search became very slow and the optimal step size became smaller and the rate of reduction of $J(u)$ decreased.

```
K>> levelSets
170.581669649628
0.000381971009752197
0.000380732839496915
0.000379498903384167
0.000378228775184198
0.000376972670237551
0.000375564628332407
0.00037415586062171
....
```

Golden section line search was implemented with tolerance of $\sqrt{(eps(\text{double}))}$ and used for finding the optimal step size.

```
....
if opt.STEP_SIZE == −1    %are we using optimal  step   size ?
   h = nma_golden_section(fLambda,currentPoint,...
                    s ,0,1, sqrt(eps('double' )));
else
   h = opt.STEP_SIZE; %we are using the  fixed   step   size .
end
.....
```

The following plot shows how the optimal step size changed at each iteration in a typical part of the search, showing how the step size becomes smaller and smaller as the search approaches the optimal point u^*. The plot to the right shows the path u^k taken.
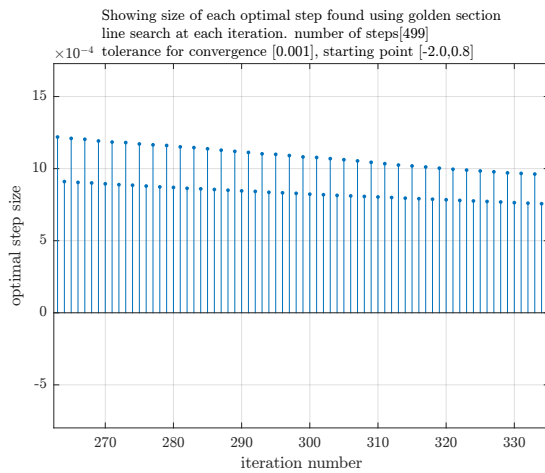
Figure 28: Showing how optimal step size changes at each iteration during typical search
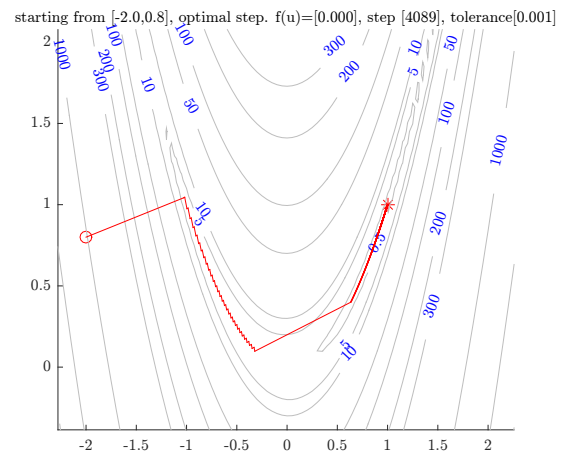
Figure 29: Typical search pattern using optimal step size from arbitrary starting point

To compare fixed size step and optimal size h, the search was started from the same point and the number of steps needed to converge was recorded.

In these runs, a maximum iteration limit was set at 10^6.

Starting from $(-2, 0.8)$

step size	number of iterations to converge
optimal	4089
0.1	Did not converge within maximum iteration limit
0.05	Did not converge within maximum iteration limit, but stopped closer to u^* than the above case using $h = 0.1$
0.01	Did not converge within maximum iteration limit, but stopped closer to u^* than the above case using $h = 0.05$

Table 5: comparing optimal and fixed step size. Starting from $(-2, 0.8)$

The following shows the path used in the above tests. The plots show that using fixed size step leads to many zigzag steps being taken which slows the search and is not efficient as using optimal step size.

Using fixed size $h = 0.1$ resulted in the search not making progress after some point due to oscillation and would be stuck in the middle of the valley.

Following is partial list of the values of $J(u)$ at each iteration using fixed size h, showing that the search fluctuating between two levels as it gets closer to optimal value u^* but it was not converging.

```
...
0.0125058920858913
0.0123566727077954
0.0125058573101063
0.0123566379524329
0.0125058226516176
0.0123566033142948
0.0125057881100252
0.0123565687929828
0.0125057536849328
0.0123565343880989
...
```

Search was terminated when oscillation is detected. Search stopped far away from u^* when the fixed step was large. As the fixed step size decreased, the final u^k that was reached was

closer to u^* but did not converge to it within the maximum iteration limit as the case with optimal step size.

The optimal step size produced the best result. It converged to u^* within the maximum iteration limit and the zigzag pattern was smaller.
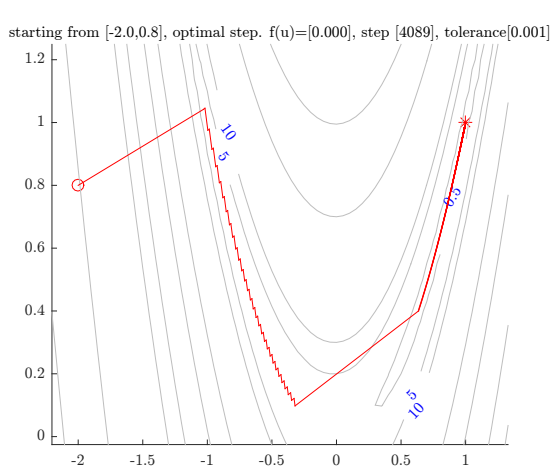
Figure 30: path of u^k using optimal step starting from $(-2, 0.8)$
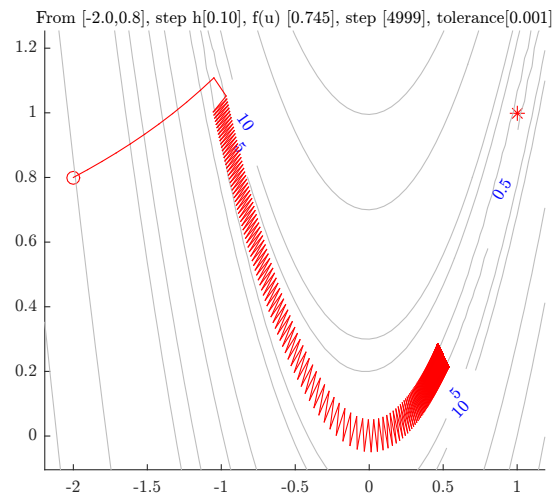
Figure 31: path of u^k using fixed step $h = 0.1$ starting from $(-2, 0.8)$
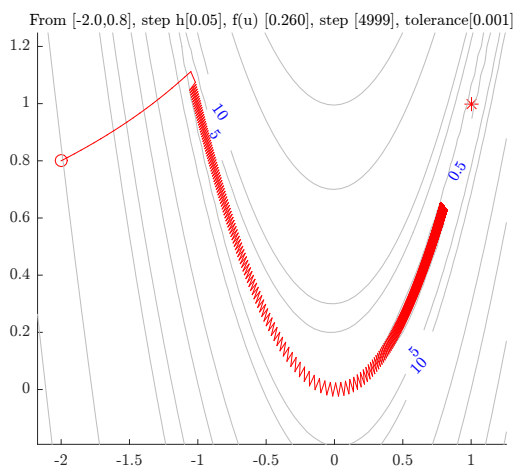
Figure 32: u^k path, fixed step $h = 0.05$ from $(-2, 0.8)$
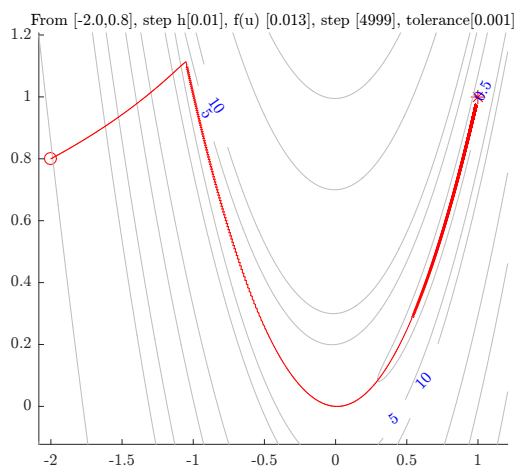
Figure 33: u^k path, fixed step $h = 0.01$ from $(-2, 0.8)$

Starting from $(-1.4, -2.2)$

step size	number of iterations to converge
optimal	537
0.1	Did not converge within maximum iteration limit
0.05	Did not converge within maximum iteration limit, but stopped closer to u^* than the above case using $h = 0.1$
0.01	Did not converge within maximum iteration limit, but stopped closer to u^* than the above case using $h = 0.05$

Table 6: comparing optimal and fixed step size. Starting from $(-1.4, -2.2)$

The following plots show the path used in the above tests. Similar observation is seen as with the last starting point. In conclusion: One should use an optimal step size even though the optimal step requires more CPU time.
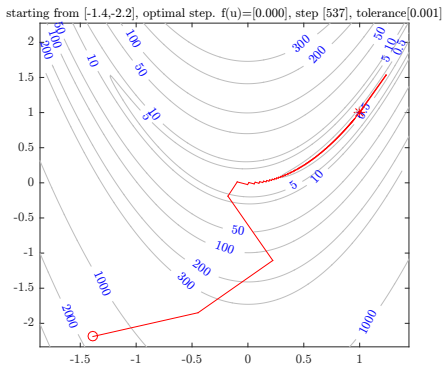
Figure 34: u^k path, optimal step from at $(-1.4, -2)$
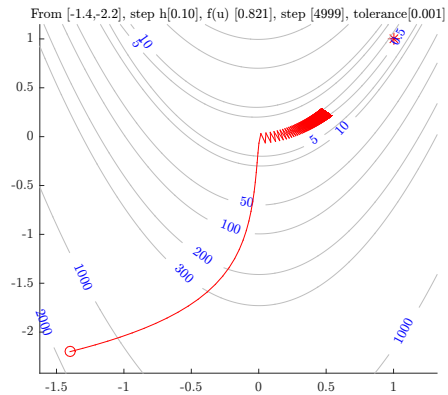
Figure 35: u^k path, fixed step $h = 0.1$ from $(-1.4, -2)$
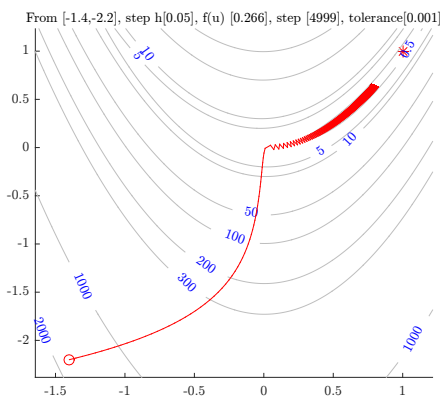
Figure 36: u^k path, fixed step $h = 0.05$ from $(-1.4, -2)$
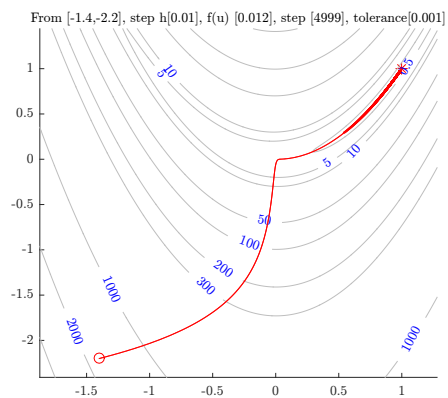
Figure 37: u^k path, fixed step $h = 0.01$ from $(-1.4, -2.0)$

0.2.2 Part(b)

A program was written to automate the search for arbitrary n. For example, for $n = 3$

$$J(u) = 100\left(u_2 - u_1^2\right)^2 + (1 - u_1)^2 + 100\left(u_3 - u_2^2\right)^2 + (1 - u_2)^2$$

$$\nabla J(u) = \begin{bmatrix} \frac{\partial J}{\partial u_1} \\ \frac{\partial J}{\partial u_2} \\ \frac{\partial J}{\partial u_3} \end{bmatrix} = \begin{bmatrix} -400\left(u_2 - u_1^2\right)u_1 - 2\left(1 - u_1\right) \\ 200\left(u_2 - u_1^2\right) - 400\left(u_3 - u_2\right)u_2 - 2\left(1 - u_2\right) \\ 200\left(u_3 - u_2^2\right) \end{bmatrix}$$

And for $n = 4$

$$J(u) = 100\left(u_2 - u_1^2\right)^2 + (1 - u_1)^2 + 100\left(u_3 - u_2^2\right)^2 + (1 - u_2)^2 + 100\left(u_4 - u_3^2\right)^2 + (1 - u_3)^2$$

$$\nabla J(u) = \begin{bmatrix} \frac{\partial J}{\partial u_1} \\ \frac{\partial J}{\partial u_2} \\ \frac{\partial J}{\partial u_3} \\ \frac{\partial J}{\partial u_4} \end{bmatrix} = \begin{bmatrix} -400\left(u_2 - u_1^2\right)u_1 - 2\left(1 - u_1\right) \\ 200\left(u_2 - u_1^2\right) - 400\left(u_3 - u_2^2\right)u_2 - 2\left(1 - u_2\right) \\ 200\left(u_3 - u_2^2\right) - 400\left(u_4 - u_3^2\right)u_3 - 2\left(1 - u_3\right) \\ 200\left(u_4 - u_3^2\right) \end{bmatrix}$$

The pattern for $\nabla J(u)$ is now clear. Let i be the row number of $\nabla J(u)$, where $i = 1 \cdots N$, then the following will generate the gradient vector for any N

$$\nabla J(u) = \begin{bmatrix} \frac{\partial J}{\partial u_i} \\ \frac{\partial J}{\partial u_i} \\ \vdots \\ \frac{\partial J}{\partial u_i} \\ \frac{\partial J}{\partial u_i} \end{bmatrix} = \begin{bmatrix} -400\left(u_{i+1} - u_i^2\right)u_i - 2\left(1 - u_i\right) \\ 200\left(u_i - u_{i-1}^2\right) - 400\left(u_{i+1} - u_i^2\right)u_i - 2\left(1 - u_i\right) \\ \vdots \\ 200\left(u_i - u_{i-1}^2\right) - 400\left(u_{i+1} - u_i^2\right)u_i - 2\left(1 - u_i\right) \\ 200\left(u_i - u_{i-1}^2\right) \end{bmatrix}$$

The program implements the above to automatically generates $\nabla J(u)$ and $J(u)$ for any N, then perform the search using steepest descent as before. The function that evaluates $J(u)$ is the following

```
1  %Evaluate J(u) at u
2  function f = objectiveFunc(u)
3    u=u(:);
4    N = size(u,1);
5    f = 0;
6    for i = 1:N-1
7        f   = f + 100*(u(i+1)-u(i)^2)^2 + (1-u(i))^2;
8    end
9  end
```

.

And the function that evaluates $\nabla J(u)$ is the following

```
1   %---------------------
2   %Evaluate grad(J(u)) at u
3   function g = gradientFunc(u)
4     u = u(:);
5     N = size(u,1);
6     g = zeros(N,1);
7     for i = 1:N
8        if i==1 || i==N
9            if i==1
10               g(i)=-400*(u(i+1)-u(i)^2)*u(i)-2*(1-u(i));
11           else
12               g(i)=200*(u(i)-u(i-1)^2);
13           end
14       else
15           g(i) = 200*(u(i)-u(i-1)^2)-...
16                     400*(u(i+1)-u(i)^2)*u(i)-2*(1-u(i));
17       end
18  end
```

.

0.2.3 Results

Two runs were made. One using fixed step size $h = 0.01$, and one using optimal step size. Both started from the same point $(-2, -2, ... , -2)$. Each time N was increased and the CPU time recorded. The same convergence criteria was used: $|\nabla J(u)| \leq 0.0001$ and a maximum iteration limit of 10^6.

Only the CPU time used for the steepest descent call was recorder.

```
1  ....
2  tic;
3  [status,pts,levelSets, gradientNormTol,steps] = ...
4              nma_steepest_descent(opt);
5  time_used = toc;
6  .....
```

.

A typical run is given below. An example of the command used for $N = 8$ is
`>> nma_HW4_problem_2_part_b([-2;-2;-2;-2;-2;-2;-2;-2])`

```
CPU time 13.180029
successful completion. Converged before maximum iterations
Number of coordinates used 8
optimal point found is
   1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  0.9999  0.9999

Number of steps used [13550]
```

The program `nma_HW4_problem_2_part_b_CPU` was run in increments of 20 up to $N = 1000$. Here is the final result.
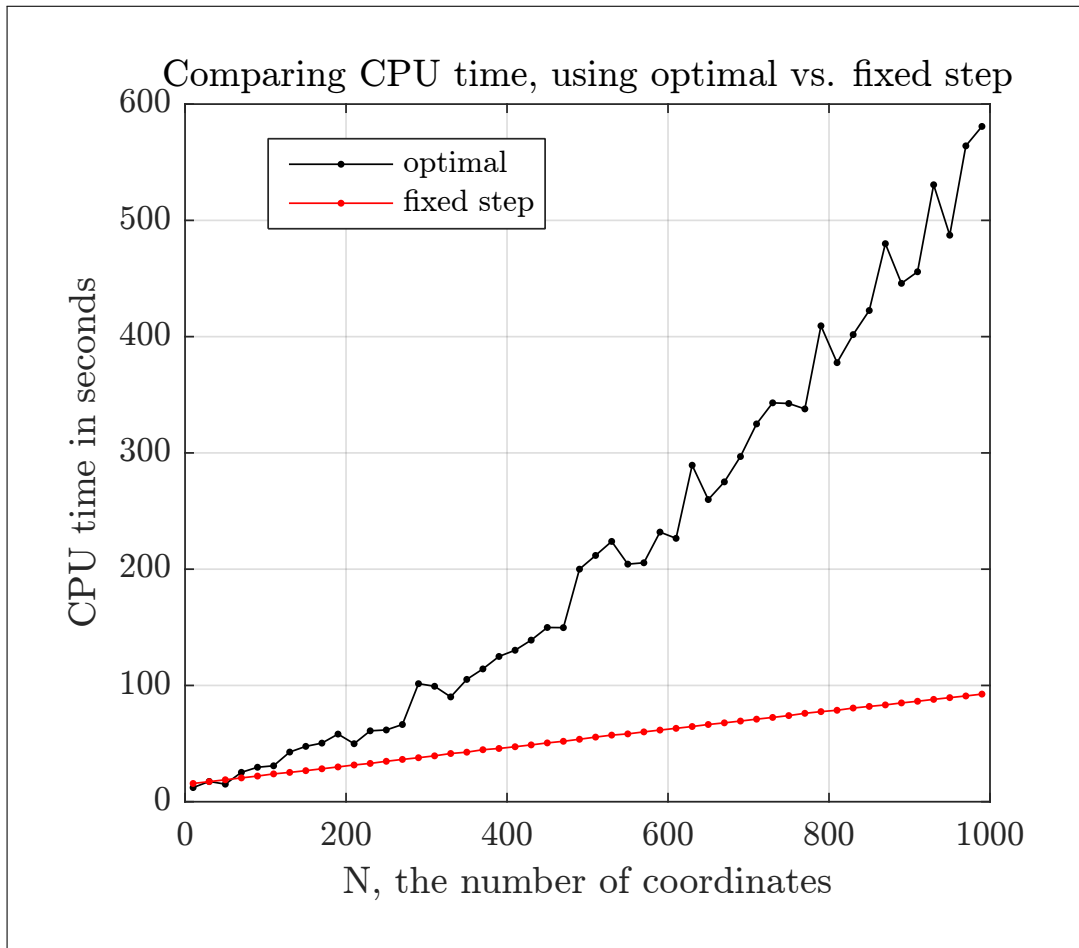
Figure 38: Comparing CPU time, optimal step and fixed step

Discussion of result The fixed step size $h = 0.01$ was selected arbitrarily to compare against. Using fixed step size almost always produced oscillation when the search was near the optimal point and the search would stop.

Using an optimal step size, the search took longer time, as can be seen from the above plot, but it was reliable in that it converged, but at a very slow rate when it was close to the optimal point.

Almost all of the CPU time used was in the line search when using optimal search. This additional computation is the main difference between the fixed and optimal step size method.

In fixed step, $|\nabla J(u)|$ was evaluated once at each step, while in optimal search, in addition to this computation, the function $J(u)$ itself was also evaluated repeated times at each step inside the golden section line search. However, even though the optimal line search took much more CPU time, it converged much better than the fixed step size search did.

Using optimal line search produces much better convergence, at the cost of using much more CPU time.

The plot above shows that with fixed step size, CPU time grows linearly with the N while with optimal step size, the CPU time grew linearly but at a much larger slope, indicating it is more CPU expensive to use.

0.3 Source code listing

0.3.1 steepest descent function

```
1  function [status,pointsFound,levelSets,gradientNormTol,steps]= ...
2                      nma_steepest_descent(opt)
3  % This function performs steepest descent search starting from
4  % a point looking for point which minimizes a function. Supports
5  % multi-variable function. It needs handle of the funtion and
```

```
 6  % hand to the gradient. It reurns all points visited in the
 7  % search. The points can then be used by client for plotting.
 8  % Below is description of input and output.
 9  %
10  % Typical use of this function is as follows:
11  %
12  %   opt.field = ...%fill in each field of the struct.
13  %   [pointsFound,levelSets,gradientNormTol,steps]  = ...
14  %                                   nma_steepest_descent(opt);
15  %
16  %   [C,h]    = contour(.....,levelSets);
17  %
18  % INPUT fields in opt struct are:
19  % ======
20  % u                 vector of coordinates starting guess
21  % MAX_ITER          an integer, which is the maximium iteration
22  %                   allowed before giving up the search.
23  %                   Example 500
24  % gradientNormTol   small floating point number. The tolerance
25  %                   to use to decide when to stop the search.
26  %                   Example 0.001
27  % stepSize          A floating point number, which is the step
28  %                   size to take. If stepSize=-1 then an optimal
29  %                   step size is found and used at each step
30  %                   using golden section line search.
31  % objectiveFunc     handle to the objective function which accepts
32  %                   a row vector, that contain [x y] coordinate
33  %                   of the point and returns the numerical value
34  %                   of objectiveFunc at this point.
35  % gradientFunc      handle to the gradiant of f. Same input and
36  %                   output as objectiveFunc
37  % accumulate        flag. If true, then all points u^k and J(u)
38  %                   at each are collected during search. Else
39  %                   they are not.
40  %
41  % OUTPUT:
42  % =======
43  % status            can be 0,1 or 2.
44  %                   0 means success, It converged before MAX_ITER
45  %                   was reached.
46  %                   1 means failed, did not converge due to
47  %                   oscillation, which can happen when step size
48  %                   is too large. When oscillation detected, the
49  %                   search will stop.
50  %                   2 means failed: did not oscillate but also
51  %                   did not converge before hitting MAX_ITER.
52  %                   Caller can try with larger MAX_ITER
53  % pointsFound       n by 2 matrix, as in [x1 y1; x2 y2; .....]
54  %                   which contains coordinates of each point
55  %                   visited during steepestDescent the length is
56  %                   the same as number of points visited. This
57  %                   will be last point only if opt.accumlate=false
58  % levelSets         vector, contains the value of the objective
59  %                   function at each point. Last value of J(u) if
60  %                   opt.accumlate=false
61  % gradientNormTol   vector, contains the norm of gradient after
62  %                   each step. This will be last value only if
63  %                   opt.accumlate=false
64  % steps             vector. The optimal step used at each
65  %                   iteration, used golden section to find optimal
66  %                   step size. This will be last value only
67  %                   if opt.accumlate=false
68  %
```

```matlab
69  % by Nasser M. Abbasi  ECE 719, UW Madison
70
71  %pre-allocate data for use in the main loop below
72  N                    = size(opt.u,1);
73
74  %collect data only if user asked for it.
75  if opt.accumulate
76      pointsFound     = zeros(opt.MAX_ITER,N);
77      levelSets       = zeros(opt.MAX_ITER,1);
78      gradientNormTol = zeros(opt.MAX_ITER,1);
79      steps           = zeros(opt.MAX_ITER,1);
80  end
81
82  %function to find optimal step size at each step,
83  %This is used only if client asked for optimal
84  %step size, which is set when opt.setSize=-1
85  %This is same J tilde(u) function from class lecture notes
86  fLambda = @(lam,u,s) opt.objectiveFunc(u-lam*s);
87
88  % initialize counters before main loop
89  k                     = 1;
90  currentPoint          = opt.u;
91  keepRunning           = true;
92  status                = 0;
93  steps_in_oscillation = 0;
94  last_level            = 0;
95
96  while keepRunning
97      if k>1
98          last_level = current_level;
99      end
100     current_level      = norm(opt.objectiveFunc(currentPoint));
101     current_grad       = opt.gradientFunc(currentPoint);
102     current_grad_norm = norm(current_grad);
103
104     if opt.accumulate
105         pointsFound(k,:)   = currentPoint;
106         levelSets(k)       = current_level;
107         gradientNormTol(k) = current_grad_norm;
108     end
109
110     if k>1 && current_level>last_level% check for oscillation
111         if opt.stop_on_oscillation
112             steps_in_oscillation = steps_in_oscillation + 1;
113         end
114     end
115
116     % check if we converged or not
117     % Last check below can lead to termination too early for the
118     % banana function. Since at one point, J(u(k+1)) will get
119     % larger than J(u(k)) using bad step size. So it is
120     %commented out for now.
121     if k == opt.MAX_ITER || ...
122                 current_grad_norm <=opt.gradientNormTol || ...
123             steps_in_oscillation>4 %let it run for 2 more steps
124                %to see the oscillation stop loop and set the
125                %status to correct reason why loop stopped.
126         keepRunning = false;
127         if steps_in_oscillation>0
128             status = 1;
129         else
130             if k == opt.MAX_ITER
131                 status= 2;
```

```matlab
132                end
133            end
134        else
135            if current_grad_norm  > eps('double') %direction vector
136                s = current_grad / current_grad_norm;
137                if opt.STEP_SIZE == -1 %are we using optimal size?
138                    lam = nma_golden_section(...
139                     fLambda,currentPoint,s,0,1,sqrt(eps('double')));
140
141                 %below for verification of golden section result
142                 %using matlab fminbd. I get similar results. so
143                 %this is good.
144
145                 %lam=fminbnd(@(lam) fLambda(lam,currentPoint,s),0,1);
146                else
147                    lam  = opt.STEP_SIZE; %using the fixed step size.
148                end
149
150                %protect aginst long step,just in case?
151                %lam = min([1,lam]);
152
153                % make step towards minumum
154                currentPoint = currentPoint - lam*s;
155
156                if opt.accumulate
157                    steps(k) = lam;
158                end
159
160                k = k + 1;
161            else
162                keepRunning = false; % |grad| < eps, stop.
163            end
164        end
165
166 end
167
168 %done. Chop data to correct number of steps used before returning
169 if opt.accumulate
170     pointsFound     = pointsFound(1:k,:);
171     levelSets       = levelSets(1:k);
172     gradientNormTol = gradientNormTol(1:k);
173     steps           = steps(1:k);
174 else
175     pointsFound     = currentPoint ;
176     levelSets       = current_level;
177     gradientNormTol = current_grad_norm;
178     steps           = k;
179 end
180
181 end
```

0.3.2 golden section line search

```matlab
1 function h_optimal = nma_golden_section(f,u,s,a,b,tol)
2 % standard golden section function (see numerical recipes)
3 %converted to Matlab to use for HW 4. This finds the optimal
4 %step size to use with the steepest descent algorithm.
5 %
6 %Nasser M. Abbasi, ECE 719 spring 2016
7 %
8 %
9 %INPUT:
10 % f: The function to minimize
```

```matlab
11  % u and s: These are specific parameters for f() used only
12  %           for HW4 problem and not part of the general algorithm
13  %           itself. These are used in calling f(). u is the
14  %           current point and "s" is the gradiant vector. in the
15  %           direction we want to minimize J(u)
16  % a: Starting search point
17  % b: ending search point.
18  % tol: tolerance to use to stop the line search. Such as 10^(-6)
19  %
20  % OUTPUT:
21  %  h_optimal: This is the optimal step size h to use
22  %
23  golden_ratio = (sqrt(5)-1)/2;
24  c            = b-golden_ratio*(b-a);
25  d            = a+golden_ratio*(b-a);
26
27  while abs(c-d)>tol
28      fc = f(c,u,s);
29      fd = f(d,u,s);
30      if fc<fd
31          b = d;
32          d = c;
33          c = b-golden_ratio*(b-a);
34      else
35          a = c;
36          c = d;
37          d = a+golden_ratio*(b-a);
38      end
39  end
40  %done. Return the optimal step size to use.
41  h_optimal = (b+a)/2;
42  end
```

0.3.3 Problem 1 part a

```matlab
1   function nma_HW4_problem_1_part_a()
2   %Plots a contour of
3   %
4   %    f(u) = (11-u1-u2)^2 + (1+u1+10*u2-u1*u2)^2
5   %
6   % over range u1=0..20 and u2=0..15
7   % Matlab 2015a
8   % by Nasser M. Abbasi
9
10  close all; clc;
11  cd(fileparts(mfilename('fullpath')));
12
13  %reset(0);
14  xlimits   = [-5 20];  %x limits, for plotting, change as needed
15  ylimits   = [-5 15]; %y limits, for plotting, change as needed
16  myTitle   = '$$(11 - u_1 - u_2)^2 +(1+ u_1+10 u_2-u_1 u_2)^2$$';
17  [u1,u2,z] = makeContourData(0.05,xlimits,ylimits);
18
19  figure(1);
20  v         =[40 60 90 140 200 400 600 1000 1500 2000 3000  ...
21                        4000 6000 8000 10000 12000 15000 18000];
22  [C,h]     = contour(u1,u2,z,v,'Linecolor',[0 0 1]);
23
24  clabel(C,h,v,'Fontsize',8,'interpreter','Latex','Color','red');
25  setMyLabels('$$u_1$$','$$u_2$$',...
26      {'\makebox[4in][c]{contour plot, default setting}',...
27      sprintf('\\makebox[4in][c]{%s}',myTitle)});
28  saveas(gcf, 'problem_1/part_a/fig1', 'pdf');
```

```matlab
29
30  figure(11);
31  xlimits   = [-5 20];  %x limits, for plotting, change as needed
32  ylimits   = [-5 20]; %y limits, for plotting, change as needed
33  myTitle   = '$$(11 - u_1 - u_2)^2 +(1+ u_1+10 u_2-u_1 u_2)^2$$';
34  [u1,u2,z] = makeContourData(0.1,xlimits,ylimits);
35  [C,h]     = contourf(u1,u2,z,v);
36  %colorDepth = 10000;
37  %colormap(jet(colorDepth));
38
39  %colormap(parula(300));
40  colormap(hsv);
41  colorbar;
42  setMyLabels('$$u_1$$','$$u_2$$',...
43      {'\makebox[4in][c]{contour plot, filled, with colorbar}',...
44      sprintf('\\makebox[4in][c]{%s}',myTitle)});
45  %saveas(gcf, 'problem_1/part_a/fig11', 'pdf');
46  %print -painters -dpdf -r600 'problem_1/part_a/fig11.pdf'
47
48  figure(12);
49  contour3(u1,u2,z,v);
50
51
52  figure(2);
53  [u1,u2,z] = makeContourData(2,xlimits,ylimits);
54  surf(u1,u2,z);
55  colormap(hsv);
56  view([-156.5,42]);
57
58  hold on;
59  v         = [200 600 1000 1500 2000 4000 6000 8000 12000];
60  [C,h]     = contour(u1,u2,z,v,'Linecolor',[0 0 1]);
61  clabel(C,h,v,'Fontsize',10,'interpreter','Latex','Color','red');
62
63  setMyLabels('$$u_1$$','$$u_2$$','$$J(u_1,u_2)$$',...
64      {'\makebox[4in][c]{Labeled 3D over contour view}',...
65      sprintf('\\makebox[4in][c]{%s}',myTitle)})
66  %saveas(gcf, 'problem_1/part_a/fig2', 'pdf');
67
68  figure(3);
69  surf(u1,u2,z);
70  colormap(hsv);
71  view([154,46]);
72  hold on;
73  contour(u1,u2,z,v,'Linecolor',[0 0 1]);
74  clabel(C,h,v,'Fontsize',10,'interpreter','Latex','Color','red');
75  setMyLabels('$$u_1$$','$$u_2$$','$$J(u_1,u_2)$$',...
76  {'\makebox[4in][c]{Another 3D over contour view (no labels)}',...
77      sprintf('\\makebox[4in][c]{%s}',myTitle)})
78  %saveas(gcf, 'problem_1/part_a/fig3', 'pdf');
79
80  figure(4);
81  xlimits   = [-10 30];  %x limits, for plotting, change as needed
82  ylimits   = [-10 30];  %y limits, for plotting, change as needed
83  [u1,u2,z] = makeContourData(.5,xlimits,ylimits);
84
85  subplot(1,2,1);
86  v         =[50 200 600 2000 4000 8000 16000 30000];
87  [C,h]     = contour(u1,u2,z,v,'Linecolor',[0 0 1]);
88  grid;    %get(h,'LevelList')
89
90  clabel(C,h,v,'Fontsize',8,'interpreter','Latex','Color','red');
91  setMyLabels('$$u_1$$','$$u_2$$',...
```

```
92        {'\makebox[4in][c]{contour plot (enlarged limits}',...
93        sprintf('\\makebox[4in][c]{%s}',myTitle)});
94
95   subplot(1,2,2);
96   [u1,u2,z] = makeContourData(4,xlimits,ylimits);
97   surf(u1,u2,z);
98   colormap(hsv);
99   view([154,46]);
100  hold on;
101  contour(u1,u2,z,'Linecolor',[0 0 1]);
102  setMyLabels('$$u_1$$','$$u_2$$','$$J(u_1,u_2)$$',...
103   {'\makebox[4in][c]{3D over contour view (enlarged limits)}',...
104       sprintf('\\makebox[4in][c]{%s}',myTitle)});
105  %saveas(gcf, 'problem_1/part_a/fig4', 'pdf');
106  end
107
108  %------------
109  %helper function to set plot attributes.
110  function setMyLabels(varargin)
111
112  myXlabel = varargin{1};
113  myYlabel = varargin{2};
114  if nargin ==4
115      myZlabel = varargin{3};
116  end
117  myTitle  = varargin{end};
118  h        = get(gca,'xlabel');
119  set(h,'string',myXlabel,'fontsize',10,'interpreter','Latex') ;
120
121  h = get(gca,'ylabel');
122  set(h,'string',myYlabel,'fontsize',10,'interpreter','Latex') ;
123
124  if nargin ==4
125      h = get(gca,'zlabel');
126      set(h,'string',myZlabel,'fontsize',10,'interpreter','Latex');
127  end
128
129  h = get(gca,'title');
130  set(h,'string',myTitle,'fontsize',10,'interpreter','Latex',...
131      'HorizontalAlignment','center') ;
132
133  set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
134  end
135
136  %------------------
137  %helper function to generate Contour data
138  function [u1,u2,z] = makeContourData(del,xlimits,ylimits)
139
140  u1      = xlimits(1):del:xlimits(2);
141  u2      = ylimits(1):del:ylimits(2);
142  [u1,u2] = meshgrid(u1,u2);
143  z       = (11-u1-u2).^2 + (1+u1+10.*u2-u1.*u2).^2;
144  end
```

0.3.4 Problem 1 part b

```
1   function nma_HW4_problem_1_part_b()
2   %finds the min value of
3   %
4   %    f(u) = (11-u1-u2)^2 + (1+u1+10*u2-u1*u2)^2
5   %
6   % over range u1=0..20 and u2=0..15 using steepest descent
7   %
```

```matlab
8   %This file is only the driver for function nma_steepestDescent.m
9   %ECE 719, Spring 2016
10  %Matlab 2015a
11  %Nasser M. Abbasi    Nov 25, 2016
12
13  if(~isdeployed)
14      baseFolder = fileparts(which(mfilename));
15      cd(baseFolder);
16  end
17
18  close all;
19  %reset(0);
20  set(groot,'defaulttextinterpreter','Latex');
21  set(groot, 'defaultAxesTickLabelInterpreter','Latex');
22  set(groot, 'defaultLegendInterpreter','Latex');
23  from_pix = 100;
24  pix_count = 1;
25
26  %paramters, change as needed
27                                   % 'conjugate gradient'
28  METHOD        = 'steepest descent'; %'steepest descent';
29  DO_GUI        = false;  %set to true to get input starting point
30  %                        from GUI
31  DO_ANIMATE    = true;   %set to true to see animation
32  DO_GIF        = false;  %set to true to make animation gif
33  DO_3D         = false;  %if we want to show 3D search path. Set to true
34  xlimits       = [-20 20];   %x limits, for plotting
35  ylimits       = [-15 15];   %y limits, for plotting
36  del           = 0.05;       %grid size, used for making meshgrid
37  fixed_levels = [40 60 90 140 200 400 600 1000 1500 2000  ...
38                  3000 4000 6000 8000,10000 12000 15000 18000];
39  CONTOUR_LINES_AUTO = 'fix'; %set to 'auto', to see matlab contour
40  %             lines, set to 'full' to see each step level set
41  %             set to 'limited' to see every other level
42  %             set to 'fix' to use pre-specified
43  %
44
45  %------------------------------------------------------------
46  %These are the options struct used by call to
47  %                         nma_steepestDescentPoints()
48  opt.u            = [16.805;13.245]; %starting guess x-coord
49  opt.MAX_ITER     = 10^3; %maximum iterations allowed
50
51  %step size. set to -1 to use an optimal step
52  opt.STEP_SIZE        = -1;
53
54  %see function definition at end of file
55  opt.objectiveFunc    = @objectiveFunc;
56
57  %see function definition at end of file
58  opt.gradientFunc     = @gradientFunc;
59  opt.gradientNormTol = 0.001;    %used to determine when converged
60  opt.hessian          = @hessian_func;  %see function definition
61  opt.accumulate       = true;
62  opt.stop_on_oscillation = false;
63
64  %------------------------------------------
65  %data
66  u1      = xlimits(1):del:xlimits(2);
67  u2      = ylimits(1):del:ylimits(2);
68  [u1,u2]  = meshgrid(u1,u2);
69  %z        = 3 + (u1 - 1.5*u2).^2 + (u2 - 2).^2;
70  z         = (11-u1-u2).^2 + (1+u1+10.*u2-u1.*u2).^2;
```

```matlab
71
72 figure('Units','pixels','position',[from_pix from_pix 600 500]);
73 pix_count = pix_count+1;
74 if DO_GUI  %check if GUI input is asked for, if so, wait for user
75     plot(0,0);
76     xlim(xlimits); ylim(ylimits);
77     hold on;
78     [x,y] = ginput(1);
79     opt.u=[x;y];
80 end
81
82 %mark location of starting point
83 %t          = text(0.8*opt.u(1),1.1*opt.u(2),...
84 %                  sprintf('[%2.1f,%2.1f]',opt.u(1),opt.u(2)));
85 %t.FontSize = 8;
86 %t.Color    = 'red';
87
88 %Find the minumum using Matlab build-in, in order
89 %to compare with in plot
90 optimalValue = fminsearch(opt.objectiveFunc, opt.u);
91 objectiveAtOptimal = objectiveFunc(optimalValue);
92
93 %mark location of minimum found by fminsearch on plot
94 %This min, can be different that one converged to by
95 %steepest descent! so we also plot the converged to value found
96 hold on;
97 %plot(optimalValue(1),optimalValue(2),'*r');
98
99 plot(opt.u(1),opt.u(2),'or');  %starting point
100 xlim(xlimits); ylim(ylimits);
101 grid;
102 set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
103 %make the call to implement steepest descent, different m file.
104 if strcmp(METHOD,'steepest descent')
105     [status , pts,levelSets, gradientNormTol,~] = ...
106                                     nma_steepest_descent(opt);
107 else
108     [status,pts,levelSets, gradientNormTol,~] = ...
109                                     nma_fletcher_reeves(opt);
110 end
111 plot(13,4,'*r');  %known u* at top location.
112 switch status
113     case 0, status = ...
114   'successfull completion. Converged before maximum iterations';
115     case 1, status = ...
116 'failed to converge before maximum iterations due to oscillation';
117     case 2, status = ....
118                 'failed to converge before maximum iterations';
119 end
120
121 %plot the value found by steepest descent
122 %plot(pts(end,1),pts(end,2),'ok');
123
124 %use output from above call to make the plots
125 switch CONTOUR_LINES_AUTO
126     case 'auto',
127     [C,h] = contour(u1,u2,z,'Linecolor',[0 0 1],'LineWidth',0.1);
128     case 'limited',
129       lev  = round(length(levelSets)/20);
130       %[C,h] = contour(u1,u2,z,levelSets(1:lev:end),'Fill','off');
131       %[C,h]    = contourf(u1,u2,z,levelSets(1:lev:end));
132       [C,h]    = contour(u1,u2,z,levelSets(1:lev:end));
133       %colormap(hsv);
```

```matlab
134        %colorbar;
135        %'Linecolor',[0 0 1],'LineWidth',.2);
136     case 'full'
137         [C,h] = contour(u1,u2,z,levelSets,'LineWidth',.2);
138     case 'fix'
139         [C,h] = contour(u1,u2,z,fixed_levels);
140         h.LineWidth = .1;
141         %h.LineColor  = [190/255 190/255 190/255];
142         clabel(C,h,fixed_levels,'Fontsize',8,...
143                       'interpreter','Latex','Color','blue');
144 end
145
146 %animate the steepest descent search
147 if length(pts(:,1))>1
148     filename = 'anim.gif';
149     for k=1:length(pts)-1
150         %draw line between each step
151         %skip case if 'full' mode or if too many points.
152         %if (opt.STEP_SIZE == -1 ||...
153 %                  strcmp(CONTOUR_LINES_AUTO,'limited') || ...
154         %    strcmp(CONTOUR_LINES_AUTO,'auto')||length(pts)<100 )
155         if strcmp(CONTOUR_LINES_AUTO,'full')||...
156                          strcmp(CONTOUR_LINES_AUTO,'limited')
157             line([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],...
158                                     'LineWidth',1,'Color','red');
159         else
160             line([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],...
161                                     'LineWidth',1,'Color','red');
162         end
163         %end
164         %plot([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],'.r');
165         if DO_ANIMATE
166             drawnow;
167             if DO_GIF
168                 frame = getframe(1);
169                 im = frame2im(frame);
170                 [imind,cm] = rgb2ind(im,256);
171                 if k ==1
172                     imwrite(imind,cm,filename,'gif', 'Loopcount',0);
173                 else
174                     if mod(k,4)==0
175                         imwrite(imind,cm,filename,...
176                                      'gif','WriteMode','append');
177                     end
178                 end
179             end
180         end
181         title(format_plot_title(...
182             ['Showing $u^k$ path on top of contour plot.' ...
183             'Problem 1, part (b)'],...
184           opt,pts,k,status),'FontSize', 8);
185     end
186 end
187 title(format_plot_title(['Showing $u^k$ path on top of'...
188                       'contour plot. Problem 1, part (b)'],...
189     opt,pts,size(pts,1),status),'FontSize', 8);
190
191
192 figure('Units','pixels','position',[from_pix from_pix 400 300]);
193 pix_count = pix_count+1;
194
195 stairs(levelSets);
196 %stem(levelSets,'ro');
```

```matlab
197  grid;
198  set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
199  title(format_plot_title(...
200              'Showing $J(u^k)$ progress. Problem 1, part (b)',...
201      opt,pts,size(pts,1),status),'FontSize', 8);
202  xlabel('step number');
203  ylabel('value of objective function');
204
205  figure('Units','pixels','position',[from_pix from_pix 400 300]);
206  pix_count = pix_count+1;
207
208  stairs(gradientNormTol);
209  %stem(levelSets,'ro');
210  grid;
211  title(format_plot_title(...
212      'Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)',...
213      opt,pts,size(pts,1),status),'FontSize', 8);
214
215  xlabel('step number'); ylabel('Norm of gradient');
216  set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
217
218  if DO_3D
219      figure('Units','pixels','position',...
220                                  [from_pix from_pix 400 300]);
221      pix_count = pix_count+1;
222
223      del       = 1;
224      u1        = xlimits(1):del:xlimits(2);
225      u2        = ylimits(1):del:ylimits(2);
226      [u1,u2]   = meshgrid(u1,u2);
227      z         = (11-u1-u2).^2 + (1+u1+10.*u2-u1.*u2).^2;
228      h         = mesh(u1,u2,z);
229
230      view(gca,-13.5,42);
231      set(h,'LineWidth',.25,'LineStyle','-','EdgeAlpha',.5);
232      shading(gca,'flat');
233      hold on;
234
235      %plot the optimal point found by Matlab
236      plot3(optimalValue(1),optimalValue(2),objectiveAtOptimal,...
237          'ws--', 'MarkerEdgeColor', 'r', 'MarkerFaceColor', 'r');
238
239      %plot the optimal point found by steepest descent
240      plot3(pts(end,1),pts(end,2),levelSets(end),...
241          'ws--', 'MarkerEdgeColor', 'b', 'MarkerFaceColor', 'b');
242
243      %plot the starting point
244      plot3(pts(1,1),pts(1,2),levelSets(1),...
245          'ws--', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'k');
246
247      if size(pts,1)>1
248          for k = 1:length(pts)-1
249              %draw line between each step
250              line([pts(k,1),pts(k+1,1)],[pts(k+1,2),pts(k+1,2)],...
251                  [levelSets(k),levelSets(k+1)],'LineWidth',1);
252              drawnow;
253          end
254      end
255      xlabel('$u_1$');ylabel('$u_2$');
256      zlabel('objective function $J(u_1,u_2)$');
257
258      title(format_plot_title(...
259    '3D mesh view of the search performed. Problem 1, part (b)',...
```

```matlab
260                       opt,pts,size(pts,1),status),'FontSize', 8);
261
262       set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
263 end
264 end
265
266 %------------------------
267 %Evaluate J(u) at u
268 function f = objectiveFunc(u)
269 x  = u(1);
270 y  = u(2);
271 %f  = 3 + (x - 1.5*y)^2 + (y - 2)^2;
272 f  = (11-x-y)^2 + (1+x+10*y-x*y)^2;
273 end
274
275 %--------------------
276 %Evaluate grad(J(u)) at u
277 function g = gradientFunc(u)
278 x  = u(1);
279 y  = u(2);
280 %g  =[2*(x-1.5*y);2*(x-1.5*y)*(-1.5)+2*(y-2)];
281 g  = [-2*(11-x-y)+2*(1+x+10*y-x*y)*(1-y); ...
282        -2*(11-x-y)+2*(1+x+10*y-x*y)*(10-x)];
283 end
284 %-------------------------
285 %set title
286 function formatted_title = format_plot_title(main_title,opt,pts,k,status)
287
288 formatted_title = {sprintf('\\makebox[5in][c]{%s}',main_title),...
289   sprintf('\\makebox[5in][c]{$u^0=[%4.3f,%4.3f]$, step [$%2.2f$], $J(u)=%3.3f$, iterations [
290   opt.u(1),opt.u(2),opt.STEP_SIZE,...
291                       norm(opt.objectiveFunc(pts(k,:))),k),...
292   sprintf('\\makebox[5in][c]{convergence criteria $| \\nabla(J(u)) | \\leq  %1.3f $}',...
293   opt.gradientNormTol),...
294   sprintf('\\makebox[5in][c]{%s}',status)};
295 end
296
297 %--------------------
298 %Evaluate Hessian(J(u)) at u
299 function g = hessian_func(u)
300 x  = u(1);
301 y  = u(2);
302 %g  = [2,-3;-3,13/2];
303
304 g  =[2*(y - 1)^2 + 2, 2*(x - 10)*(y - 1) - 20*y - 2*x + 2*x*y;
305      2*(x - 10)*(y - 1) - 20*y - 2*x + 2*x*y,2*(x - 10)^2 + 2];
306 end
```

0.3.5   Problem 2 contour

```matlab
1 function nma_HW4_problem_2_contour()
2 %Plots a contour of
3 %
4 %     f(u) = 100(u2-u1^2)^2 + (1-u1)^2
5 %
6 % over range  u1=-2.5..2.5
7 % Matlab 2015a
8 % by Nasser M. Abbasi
9
10
11 reset(0); close all; clear;
12 k=0;
13 myTitle   = '$$100(u_2 - u_1^2)^2 +(1- u_1)^2$$';
```

```matlab
14  makeContour(-2.5,2.5,-2.5,2.5,[1 10 50 100 200 300   ...
15                                 500 1000 2000 3000],myTitle);
16  k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
17  makeContour(-1,1.5,-1,2,[0.5 1 5 15 30 50 100 200 300 400 500],...
18                                                      myTitle);
19  k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
20  makeContour(0.2,1.5,-0.5,2,[0.5 2.5 5 10 20 35 50 100 200 300],...
21                                                      myTitle);
22  k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
23  makeContour(0.4,1.1,0,1.5,[0.2 0.3 0.5 2.5 5 10 20 30 50 ...
24                                 75 100 150 200],myTitle);
25  k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
26  makeContour(0.8,1.1,0.5,1.1,[0.1 0.2 0.3 0.5 1 2 3 ...
27                                      4 8 12 20],myTitle);
28  k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
29  makeContour(0.9,1.1,0.9,1.1,[0.01 0.05 0.1 0.2 ...
30                                  0.3 0.5 1 1.5 2 3],myTitle);
31  k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
32
33  figure;
34  [u1,u2,z] = makeContourData(0.3,[-2,2],[-2,2]);
35  surf(u1,u2,z);
36  colormap(hsv);
37  view([-156.5,42]);
38
39  hold on;
40  v=[10 100 200 300 500];
41  [C,h]    = contour(u1,u2,z,v);
42  clabel(C,h,v,'Fontsize',10,'interpreter','Latex','Color','red');
43  setMyLabels('$$u_1$$','$$u_2$$','$$J(u_1,u_2)$$',...
44              {'\makebox[4in][c]{Labeled 3D over contour view}',...
45                  sprintf('\\makebox[4in][c]{%s}',myTitle)})
46  k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
47  end
48
49  %-------------
50  %helper function to set plot attributes.
51  function setMyLabels(varargin)
52
53  myXlabel = varargin{1};
54  myYlabel = varargin{2};
55  if nargin ==4
56      myZlabel = varargin{3};
57  end
58  myTitle  = varargin{end};
59  h        = get(gca,'xlabel');
60  set(h,'string',myXlabel,'fontsize',10,'interpreter','Latex') ;
61
62  h = get(gca,'ylabel');
63  set(h,'string',myYlabel,'fontsize',10,'interpreter','Latex');
64
65  if nargin ==4
66      h = get(gca,'zlabel');
67      set(h,'string',myZlabel,'fontsize',10,'interpreter','Latex');
68  end
69
70  h = get(gca,'title');
71  set(h,'string',myTitle,'fontsize',10,'interpreter','Latex',...
72      'HorizontalAlignment','center') ;
73
74  set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
75  end
76
```

```matlab
%-----------------
%helper function to generate Contour data
function [u1,u2,z] = makeContourData(del,xlimits,ylimits)

u1      = xlimits(1):del:xlimits(2);
u2      = ylimits(1):del:ylimits(2);
[u1,u2] = meshgrid(u1,u2);
z       =  100*(u2-u1.^2).^2 + (1-u1).^2;
end


%------------------
%helper function to generate Contour data
function [u1,u2,z] = makeContour(xMin,xMax,yMin,yMax,v,myTitle)

figure();
[u1,u2,z] = makeContourData(0.05,[xMin,xMax],[yMin,yMax]);
[C,h]     = contour(u1,u2,z,v); grid;
clabel(C,h,v,'Fontsize',8,'interpreter','Latex','Color','red');
setMyLabels('$$u_1$$','$$u_2$$',...
          {'\makebox[4in][c]{contour plot}',...
                    sprintf('\\makebox[4in][c]{%s}',myTitle)});
end
```

0.3.6 Problem 2 part a

```matlab
function nma_HW4_problem_2_part_a()
%finds the min value of
%
%    f(u) = 100(u2-u1^2)^2 + (1-u1)^2
%
% over range u1=-2.5..2.5
%
% This file is only the driver for function
% nma_steepest_descent.m Solves part (a) of problem 2
%
% ECE 719, SPring 2016
% Matlab 2015a
%Nasser M. Abbasi

if(~isdeployed)
    baseFolder = fileparts(which(mfilename));
    cd(baseFolder);
end

close all;
reset(0);
set(groot,'defaulttextinterpreter','Latex');
set(groot, 'defaultAxesTickLabelInterpreter','Latex');
set(groot, 'defaultLegendInterpreter','Latex');
from_pix = 100;
pix_count = 1;
%paramters, change as needed
              % 'conjugate gradient'
METHOD       = 'steepest descent'; %'steepest descent';
DO_GUI     = false; %set to true to get input starting point GUI
DO_ANIMATE = true;  %set to true to see animation of the search
DO_GIF       = false;  %set to true to make animation gif
%xlimits   = [0 20];  %x limits, for plotting, change as needed
%ylimits   = [-5 15]; %y limits, for plotting, change as needed
xlimits    = [-2.5 2.5]; %x limits, for plotting, change
ylimits    = [-2.5 2.5];   %y limits, for plotting, change
del        = 0.01;      %grid size, used for making meshgrid
```

```matlab
38  CONTOUR_LINES_AUTO =  'fix';
39  %         set to 'auto', to see matlab contour lines
40  %         set to 'full' to see each step level set
41  %         set to 'limited' to see every other level
42  %         set to 'fix' to use pre-specified
43  %         set to 'full0', to see each level, no labels
44
45  %level set for 'fix' option
46  vFixed = [.5 10 50 100 200 300 1000 2000 3000];
47  %--------------------------------------------------------
48  %These are the options struct used by call to
49  %nma_steepestDescentPoints() try [-2,.8]
50  opt.u             = [1.828;-1.878]; %starting guess x-coord
51  opt.MAX_ITER      = 10^6; %maximum iterations allowed
52  opt.STEP_SIZE     = -1; %step size. set to -1 for optimal step
53  opt.objectiveFunc = @objectiveFunc; %see function definition
54  opt.gradientFunc  = @gradientFunc;  %see function definition
55  opt.hessian       = @hessian_func;  %see function definition
56  opt.gradientNormTol = 0.01; %used to determine when converged
57  opt.accumulate    = true;
58  opt.stop_on_oscillation = false;
59
60  %-------------------------------------------
61  %data
62  u1      = xlimits(1):del:xlimits(2);
63  u2      = ylimits(1):del:ylimits(2);
64  [u1,u2] = meshgrid(u1,u2);
65  z       = 100*(u2-u1.^2).^2 + (1-u1).^2;
66
67  figure('Units','pixels','position',[from_pix from_pix 400 300]);
68  pix_count = pix_count+1;
69
70  if DO_GUI  %check if GUI input is asked for, if so, wait for user
71      plot(0,0);
72      xlim(xlimits); ylim(ylimits);
73      hold on;
74      [x,y] = ginput(1);
75      opt.u=[x;y];
76  end
77
78  %mark location of starting point
79  %t = text(0.8*opt.x,1.1*opt.y,sprintf('[%2.1f,%2.1f]',opt.x,opt.y));
80  %t.FontSize = 8;
81  %t.Color    = 'red';
82
83  %Find the minumum using Matlab build-in, in order to compare with
84  optimalValue = fminsearch(opt.objectiveFunc, opt.u);
85  objectiveAtOptimal = objectiveFunc(optimalValue);
86
87  %mark location of minimum found by fminsearch on plot
88  %This min, can be different that one converged to by steepest
89  %descent! so we also plot the converged to value found
90  hold on;
91  plot(optimalValue(1),optimalValue(2),'*r')
92  plot(opt.u(1),opt.u(2),'or');  %starting point
93  xlim(xlimits); ylim(ylimits);
94  %grid;
95  set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
96  if strcmp(METHOD,'steepest descent')
97     [status , pts,levelSets, gradientNormTol,steps] =  ...
98                                     nma_steepest_descent(opt);
99  else
100     [status,pts,levelSets, gradientNormTol,steps] = ...
```

```matlab
101                                              nma_fletcher_reeves(opt);
102  end
103
104  %plot the value found by steepest descent
105  %plot(pts(end,1),pts(end,2),'ok');
106
107  %use output from above call to make the plots
108  switch CONTOUR_LINES_AUTO
109      case 'auto',
110      [C,h] = contour(u1,u2,z); %,'ShowText','on');
111      clabel(C,h,'Fontsize',8,'interpreter','Latex','Color','red');
112
113      case 'limited',
114          lev   = round(length(levelSets)/20);
115          [C,h] = contour(u1,u2,z,levelSets(1:lev:end),...
116                                  'Fill','off','ShowText','off');
117          %clabel(C,h); %,'Fontsize',8,'interpreter',...
118          %'Latex','Color','red');
119      case 'full'
120          [C,h] = contour(u1,u2,z,levelSets); %,'ShowText','on');
121          clabel(C,h,levelSets,'Fontsize',8,...
122                          'interpreter','Latex','Color','red');
123      case 'full0'
124          [C,h] = contour(u1,u2,z,levelSets); %,'ShowText','on');
125      case 'fix'
126          [C,h] = contour(u1,u2,z,vFixed);
127          h.LineWidth = .1;
128          h.LineColor  = [190/255 190/255 190/255];
129          h.Fill='off';
130          clabel(C,h,vFixed,'Fontsize',8,...
131                          'interpreter','Latex','Color','blue');
132  end
133  %animate the steepest descent search
134  hold on;
135  if length(pts(:,1))>1
136      filename = 'anim.gif';
137      for k=1:length(pts)-1
138          %draw line between each step
139          %if (opt.STEP_SIZE == -1 || ...
140                  %strcmp(CONTOUR_LINES_AUTO,'limited') || ...
141          % strcmp(CONTOUR_LINES_AUTO,'auto')||length(pts)<100 )
142          %   line([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],...
143                                          %'LineWidth',1);
144          %end
145          plot([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],'-r');
146          %plot(pts(k,1),pts(k,2),'.');
147          if DO_ANIMATE
148              drawnow;
149              if DO_GIF
150                  frame = getframe(1);
151                  im = frame2im(frame);
152                  [imind,cm] = rgb2ind(im,256);
153                  if k ==1
154                      imwrite(imind,cm,filename,'gif', 'Loopcount',0);
155                  else
156                      if mod(k,2)==0
157                          imwrite(imind,cm,filename,'gif',...
158                                              'WriteMode','append');
159                      end
160                  end
161              end
162          end
163          if opt.STEP_SIZE==-1,
```

```matlab
                title(sprintf(...
 'starting from [%4.3f,%4.3f], optimal step. f(u)=[%3.3f],  step [%d], tolerance[%2.3f]',...
              opt.u(1),opt.u(2),norm(opt.objectiveFunc(pts(k,:))),...
                                         k,opt.gradientNormTol),...
              'FontSize', 8);
           else
              title(sprintf(...
 'From [%2.1f,%2.1f], step h[%2.2f], f(u) [%3.3f],  step [%d], tolerance[%2.3f]',...
              opt.u(1),opt.u(2),opt.STEP_SIZE   ,...
       norm(opt.objectiveFunc(pts(k,:))),k,opt.gradientNormTol),...
              'FontSize', 8);
           end
       end
   end
end

figure('Units','pixels','position',...
                          [from_pix*pix_count from_pix 400 300]);
pix_count = pix_count+1;

stairs(levelSets);
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
title({'Showing value of objective function at each step',...
 sprintf('Step size [%3.3f], number of steps needed [%d]',...
 opt.STEP_SIZE,length(pts)-1),...
 sprintf('convergence tolerance [%2.3f], starting point [%2.1f,%2.1f]',...
 opt.gradientNormTol,opt.u(1),opt.u(2))},...
 'FontSize', 8);
xlabel('step number');
ylabel('value of objective function');


figure('Units','pixels','position',...
                          [from_pix*pix_count from_pix 400 300]);
pix_count = pix_count+1;

stem(gradientNormTol,'.');
grid;
title({'Showing gradient Norm at each step',...
   sprintf('Step size [%3.3f], number of steps needed [%d]',...
   opt.STEP_SIZE,length(pts)-1),...
   sprintf('tolerance for convergence [%2.3f], starting point [%2.1f,%2.1f]',...
   opt.gradientNormTol,opt.u(1),opt.u(2))},'FontSize', 8);

xlabel('step number'); ylabel('Norm of gradient');
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

if opt.STEP_SIZE == -1
    figure('Units','pixels','position',...
                          [from_pix*pix_count from_pix 400 300]);
    pix_count = pix_count+1;
    stem(steps,'.');
    grid;
    title({'Showing size of each optimal step found using golden section',...
     sprintf('line search at each iteration. number of steps[%d]',...
     length(pts)-1),...
     sprintf('tolerance for convergence [%2.3f], starting point [%2.1f,%2.1f]',...
     opt.gradientNormTol,opt.u(1),opt.u(2))},'FontSize', 8);
    xlabel('iteration number'); ylabel('optimal step size');
    set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
end

end
```

```matlab
%------------------------
%Evaluate J(u) at u
function f = objectiveFunc(u)
x  = u(1);
y  = u(2);
f  = 100*(y-x.^2).^2 + (1-x).^2;
end


%--------------------
%Evaluate grad(J(u)) at u
function g = gradientFunc(u)
x  = u(1);
y  = u(2);
g  = [200*(y-x.^2)*(-2*x)-2*(1-x);...
     200*(y-x.^2)];
end

%--------------------
%Evaluate Hessian(J(u)) at u
function g = hessian_func(u)
x  = u(1);
y  = u(2);
g  = [ 1200*x^2 - 400*y + 2, -400*x;
     -400*x,     200];
end
```

0.3.7   Problem 2 part b

```matlab
function nma_HW4_problem_2_part_b(u)
%finds the min value of
%
%     f(u) = sum i=1..N-1  100(u(i+1)-u(i)^2)^2 + (1-u(i))^2
%
% for any N.
%
% over range ui=-2.5..2.5
%
% Solves part (b) of problem 2
%
% ECE 719, SPring 2016
% Matlab 2015a
%Nasser M. Abbasi
%
% INPUT:
% u:  vector N by 1, represent starting point u_0. Example call
%    nma_HW4_problem_2_part_b([-2;-2;-2])

%These are the options struct used by call to
%nma_steepest_descent_multi()
close all;
opt.u                = u;     %starting guess x-coordinate
opt.MAX_ITER         = 1*10^6; %maximum iterations allowed
opt.STEP_SIZE        = 0.01; %set to -1 to optimal step
opt.objectiveFunc    = @objectiveFunc; %see function definition
opt.gradientFunc     = @gradientFunc;  %see function definition
opt.gradientNormTol = 0.0001; %used to determine when converged
opt.accumulate       = false;

%Find the minumum using Matlab build-in, in order
%to compare with in plot optimalValue =
%fminsearch(opt.objectiveFunc, opt.u);
```

```
35   format long g;
36   tic;
37   [status,pts,levelSets, gradientNormTol,steps] = ...
38                                    nma_steepest_descent(opt);
39   time_used = toc;
40   fprintf('\nCPU time %3.6f\n',time_used);
41
42   switch status
43       case 0, status = ...
44     'successfull completion. Converged before maximum iterations';
45       case 1, status = ...
46   'failed to converge before maximum iterations due to oscillation';
47       case 2, status = ...
48                   'failed to converge before maximum iterations';
49   end
50
51   fprintf('%s\n',status);
52
53   figure();
54   stem(levelSets,'.'); title('J(u)');
55   figure();
56   stem(steps,'.'); title('step size');
57   format short;
58   fprintf('Number of coordinates used %d\n',size(opt.u,1));
59   fprintf('optimal point found is\n'); disp(pts(end,:));
60   if opt.accumulate
61       fprintf('\nNumber of steps used [%d]',length(steps));
62   else
63       fprintf('\nNumber of steps used [%d]',steps);
64   end
65   fprintf('\nJ(u) at optimal [%3.6f]',levelSets(end));
66   fprintf('\n**** done ******\n');
67
68
69   end
70
71   %-------------------------
72   %Evaluate J(u) at u
73   function f = objectiveFunc(u)
74   u=u(:);
75   N = size(u,1);
76   f = 0;
77   for i = 1:N-1
78       f  = f + 100*(u(i+1)-u(i)^2)^2 + (1-u(i))^2;
79   end
80   end
81
82   %--------------------
83   %Evaluate grad(J(u)) at u
84   function g = gradientFunc(u)
85   u = u(:);
86   N = size(u,1);
87   g = zeros(N,1);
88   for i = 1:N
89       if i==1 || i==N
90           if i==1
91               g(i)=-400*(u(i+1)-u(i)^2)*u(i) - 2*(1-u(i));
92           else
93               g(i)=200*(u(i)-u(i-1)^2);
94           end
95       else
96           g(i) = 200*(u(i)-u(i-1)^2)-...
97                           400*(u(i+1)-u(i)^2)*u(i)-2*(1-u(i));
```

```
98        end
99    end
100   end
```

0.3.8   Problem 2 part b CPU time program

```matlab
1   function nma_HW4_problem_2_part_b_CPU()
2   %Does CPU testing on problem 2 by calling
3   %nma_HW4_problem_2_part_b() on larger and larger N and
4   %recording the CPU time used.
5   %
6   % ECE 719, Spring 2016
7   % Matlab 2015a
8   %Nasser M. Abbasi
9   clear; close all;
10
11  opt.STEP_SIZE       = 0.01; %step size. set to -1 to use optimal
12  save_file           = 'fixed.mat';
13  N                   = 10:20:1000;
14  data                = zeros(length(N),4);
15  opt.MAX_ITER        = 1*10^6; %maximum iterations allowed
16
17  opt.objectiveFunc   = @objectiveFunc; %see function definition
18  opt.gradientFunc    = @gradientFunc;  %see function definition
19  opt.gradientNormTol = 0.0001; %used to determine when converged
20  opt.accumulate      = false;
21
22  for i=1:length(N)
23      opt.u = repmat(-2,N(i),1);     %starting guess x-coordinate
24      tic;
25      [status,~,levelSets, ~,number_of_steps_used] = ...
26                              nma_steepest_descent(opt);
27      time_used = toc;
28      switch status
29          case 0, status = ...
30  'successfull completion. Converged before maximum iterations';
31          case 1, status = ...
32  'failed to converge before maximum iterations due to oscillation';
33          case 2, status = ...
34              'failed to converge before maximum iterations';
35      end
36      fprintf('%s\n',status);
37
38      data(i,1) = N(i);
39      data(i,2) = time_used;
40      data(i,3) = levelSets;
41      data(i,4) = number_of_steps_used;
42      fprintf('\n****Number of coordinates used %d\n',...
43                                      size(opt.u,1));
44      fprintf('\nCPU time %3.6f\n',time_used);
45      fprintf('\nJ(u) at optimal [%3.6f]\n',levelSets(end));
46  end
47
48  close all;
49  reset(0);
50  set(groot,'defaulttextinterpreter','Latex');
51  set(groot, 'defaultAxesTickLabelInterpreter','Latex');
52  set(groot, 'defaultLegendInterpreter','Latex');
53
54  figure();
55  plot(N,data(:,2),'ro',N,data(:,2),'-');
56  title('CPU time as N changes for fix step steepest descent');
57  xlabel('N'); ylabel('CPU time (sec)');
```

```matlab
58  set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
59
60  save(save_file,'data');
61
62  %-----------------------------
63  figure;
64  load('optimal');
65  optimal=data;
66  load('fixed')
67  fixed=data;
68  plot(optimal(:,1),optimal(:,2),'k.-')
69  hold on;
70  plot(fixed(:,1),fixed(:,2),'r.-')
71  title('Comparing CPU time, using optimal vs. fixed step')
72  xlabel('N, the number of coordinates');
73  ylabel('CPU time in seconds');
74  grid
75  end
76
77  %------------------------
78  %Evaluate J(u) at u
79  function f = objectiveFunc(u)
80  u=u(:);
81  N = size(u,1);
82  f = 0;
83  for i = 1:N-1
84      f  = f + 100*(u(i+1)-u(i)^2)^2 + (1-u(i))^2;
85  end
86  end
87
88  %--------------------
89  %Evaluate grad(J(u)) at u
90  function g = gradientFunc(u)
91  u = u(:);
92  N = size(u,1);
93  g = zeros(N,1);
94  for i = 1:N
95      if i==1 || i==N
96          if i==1
97              g(i)=-400*(u(i+1)-u(i)^2)*u(i) - 2*(1-u(i));
98          else
99              g(i)=200*(u(i)-u(i-1)^2);
100         end
101     else
102         g(i) = 200*(u(i)-u(i-1)^2)-400*(u(i+1)- ...
103                                 u(i)^2)*u(i)-2*(1-u(i));
104     end
105 end
106 end
```