HW4 ECE 719 Optimal systems

Spring 2016
Electrical engineering department
University of Wisconsin, Madison

Instructor: Professor B Ross Barmish

By

Nasser M. Abbasi

December 30, 2019

Contents

List of Figures

List of Tables

0.1 Problem 1

Barmish

ECE 717 – Homework Amplifier

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

$$J(u) = (11 - u_1 - u_2)^2 + (1 + u_1 + 10u_2 - u_1u_2)^2$$

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

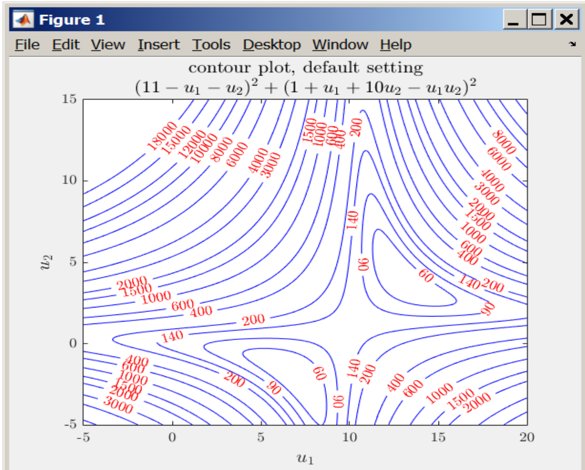
$$u^0 = \begin{bmatrix} 7 \\ -2 \end{bmatrix}$$

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.

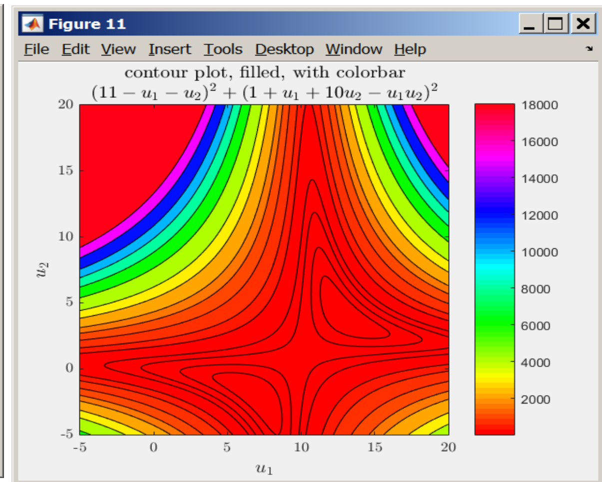
Figure 1: problem 1 description

0.1.1 part(a)

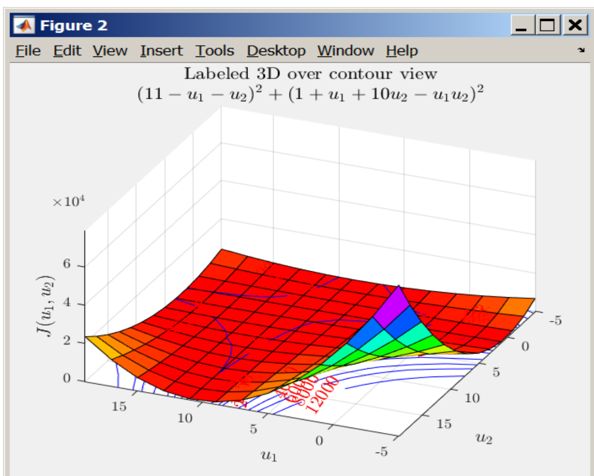
Matlab was used to generate the contour plots. The plots generated are given below and the source code used is listed in the appendix.



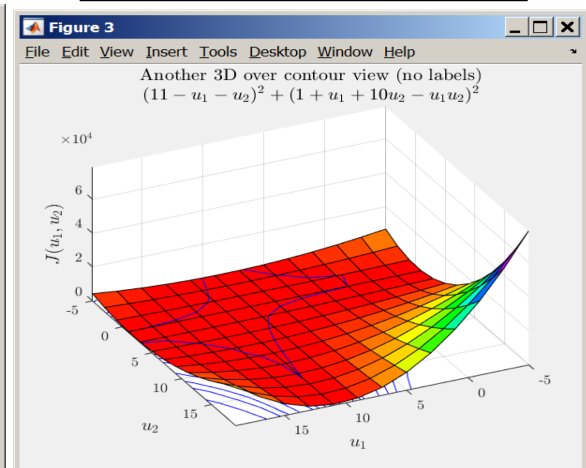
contour plot, problem 1



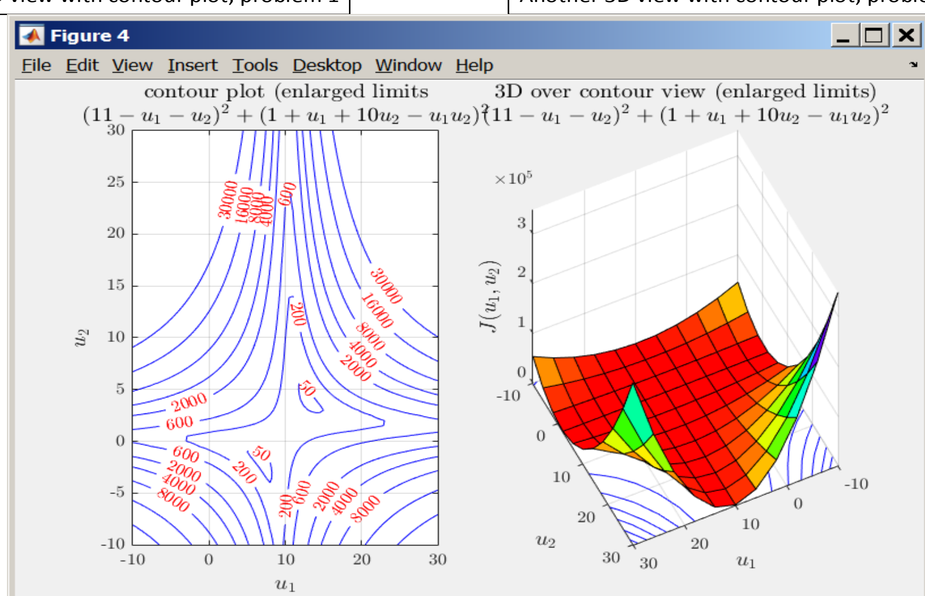
contour plot, filled, with colorbar problem 1



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

procedure STEEPEST_DESCENT
    ▷ Initialization
    $h \leftarrow$ step size
    $\epsilon \leftarrow$ minimum convergence limit on $\|\nabla J(u)\|$
    $k \leftarrow 0$
    $u^k \leftarrow u^0$
    $max\_iterations \leftarrow$ max iterations allowed
    while $\|\nabla J(u^k)\| > \epsilon$ **do**
        $u^k \leftarrow u^k - h\frac{\nabla J(u^k)}{\|\nabla J(u^k)\|}$
        $k \leftarrow k + 1$
        ▷ check for oscillation
        if $k \geq$ max_iterations **or** $J(u_k) > J(u_{k-1})$ **then**
            exit loop
        end if
    end while
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`

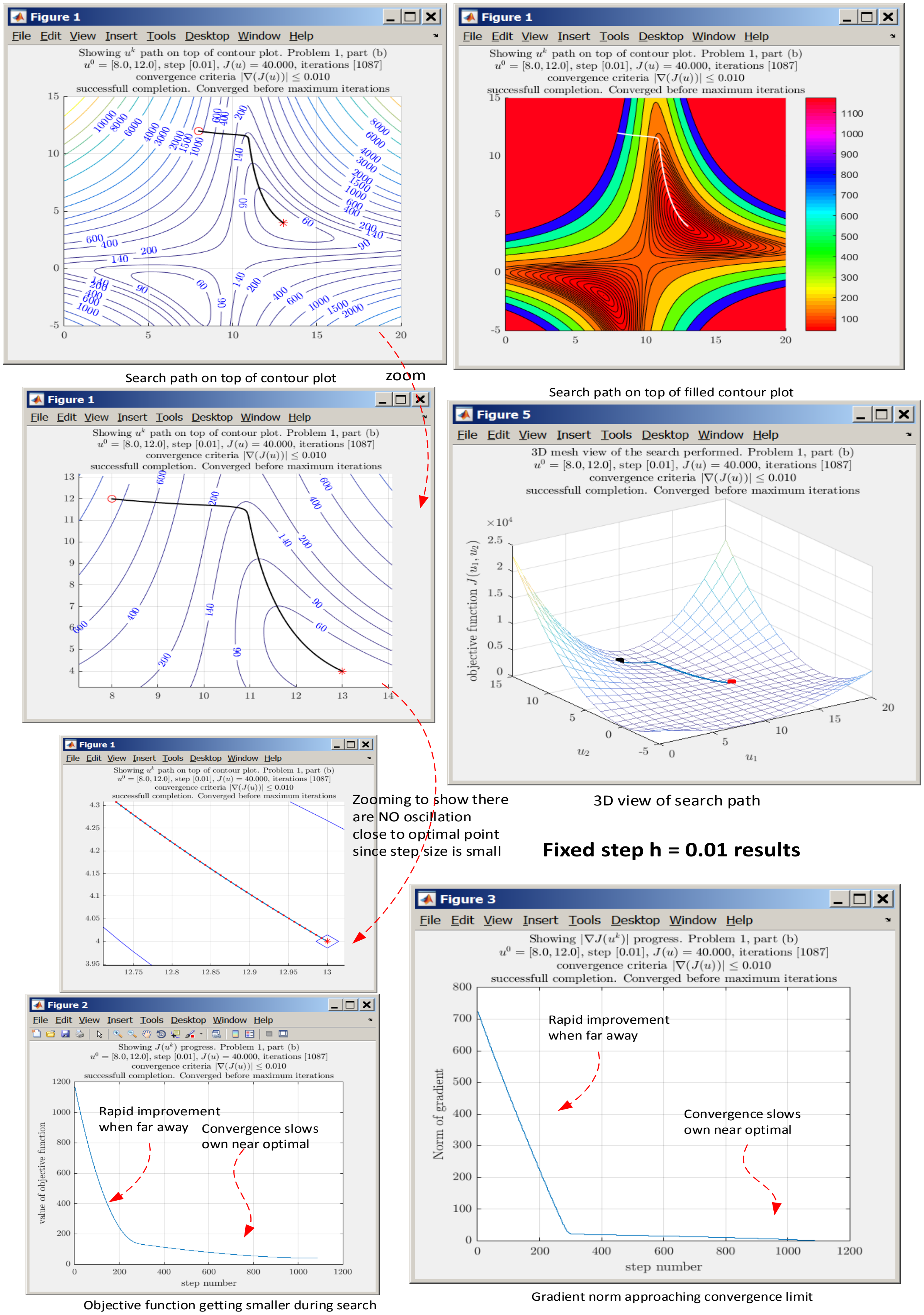


Figure 4: Result for using step 0.01 starting from (8,12)

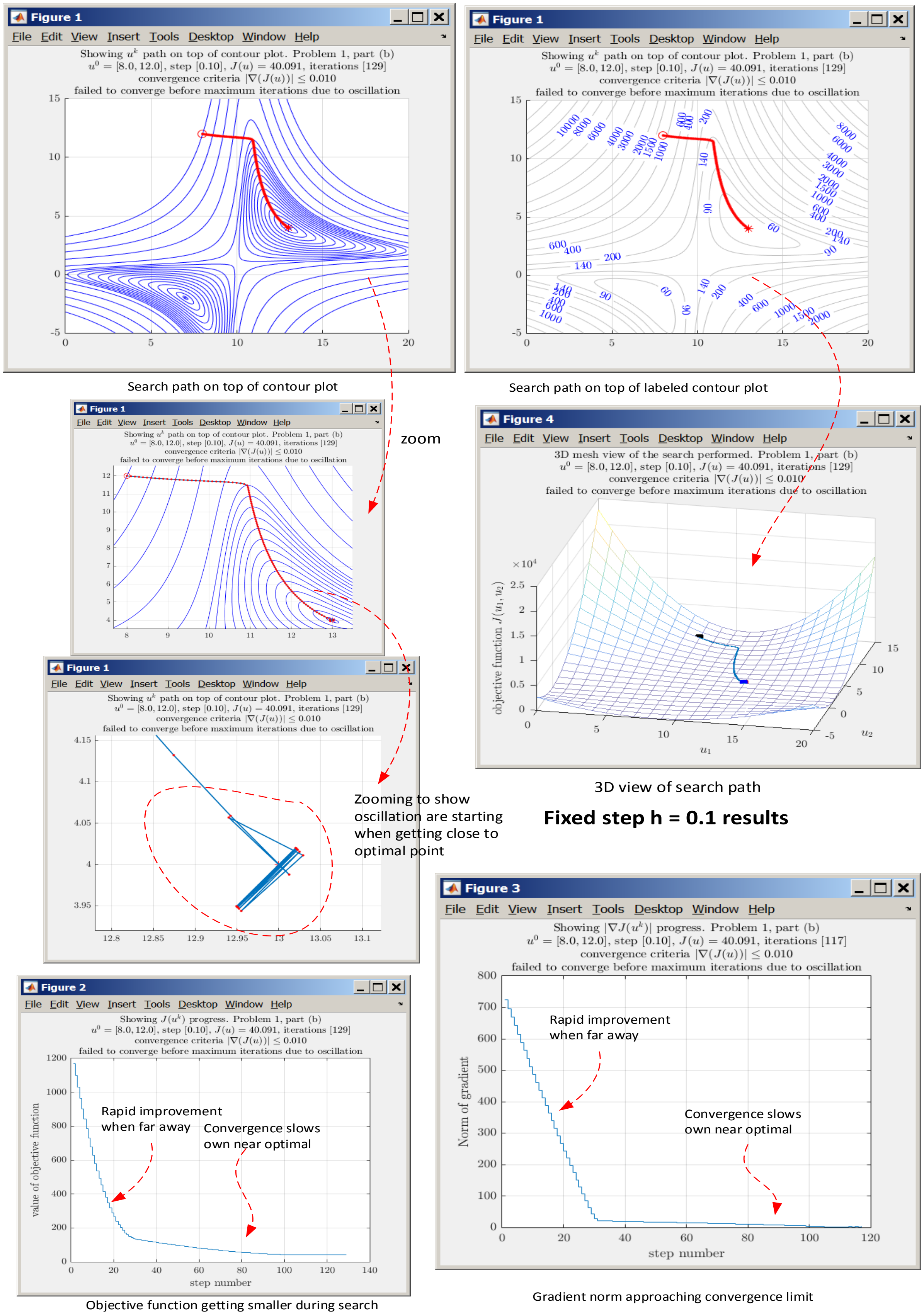
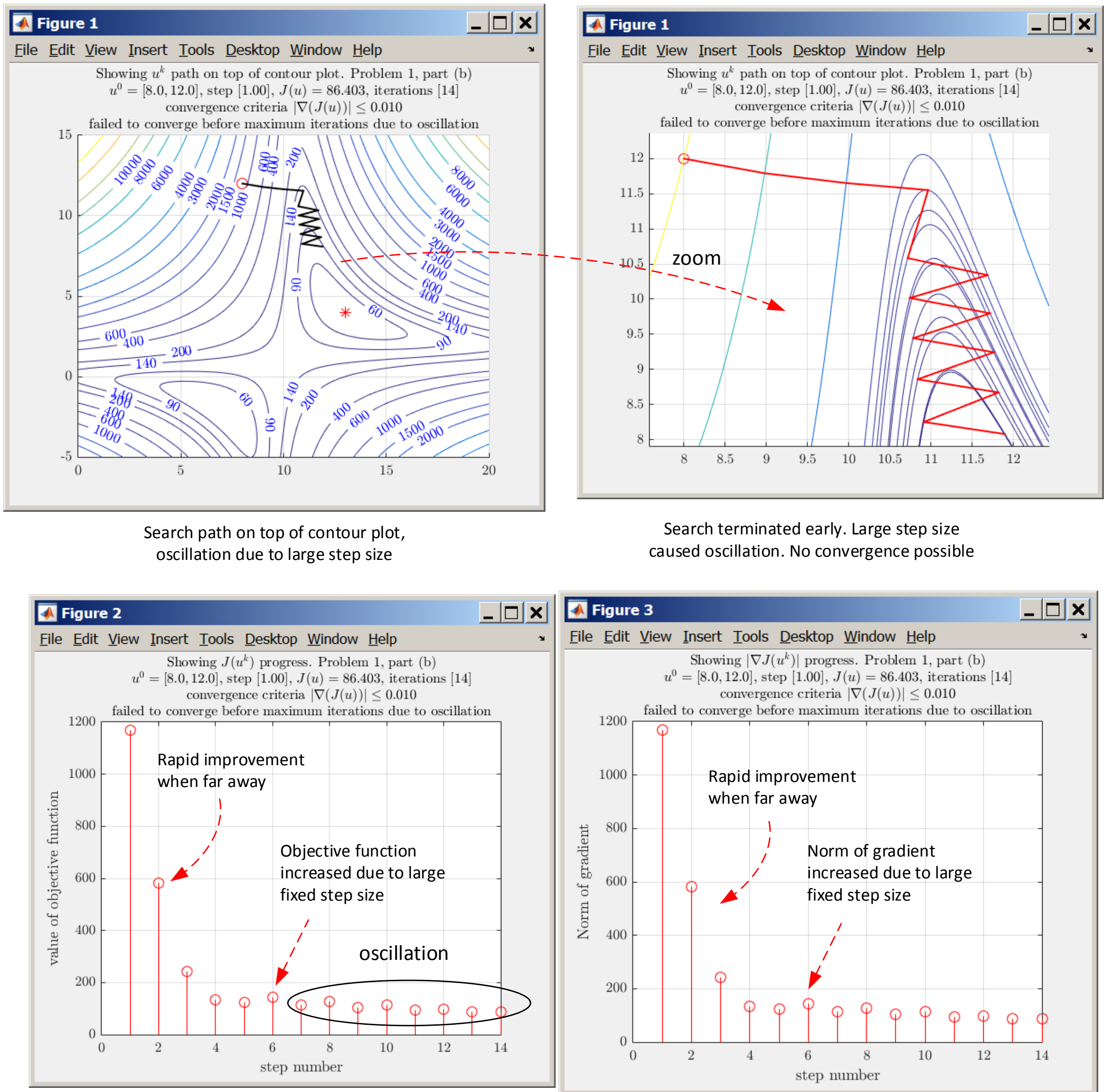


Figure 5: Result for using step 0.1 starting from (8,12)



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.

Continued on next page

Table 2 – continued from previous page

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`

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 $\lvert\nabla(J(u))\rvert$
		`K>> gradientNormTol(end-6:end)`
		`....`
		`16.1440020280613`
		`17.487837406306`
		`16.092991548592`
		`17.4442963174089`

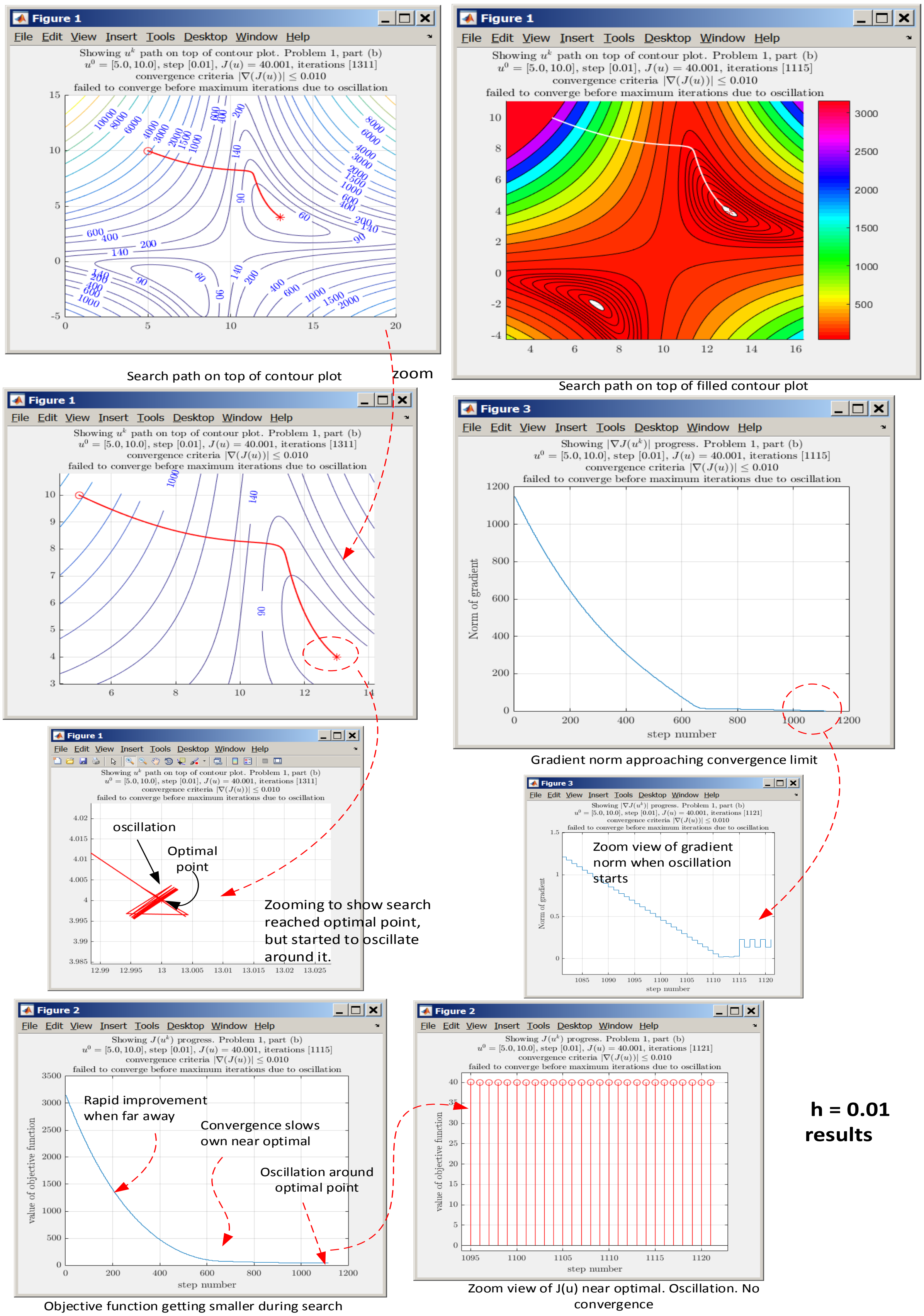


Figure 7: Result for using step 0.01 starting from (5,10)

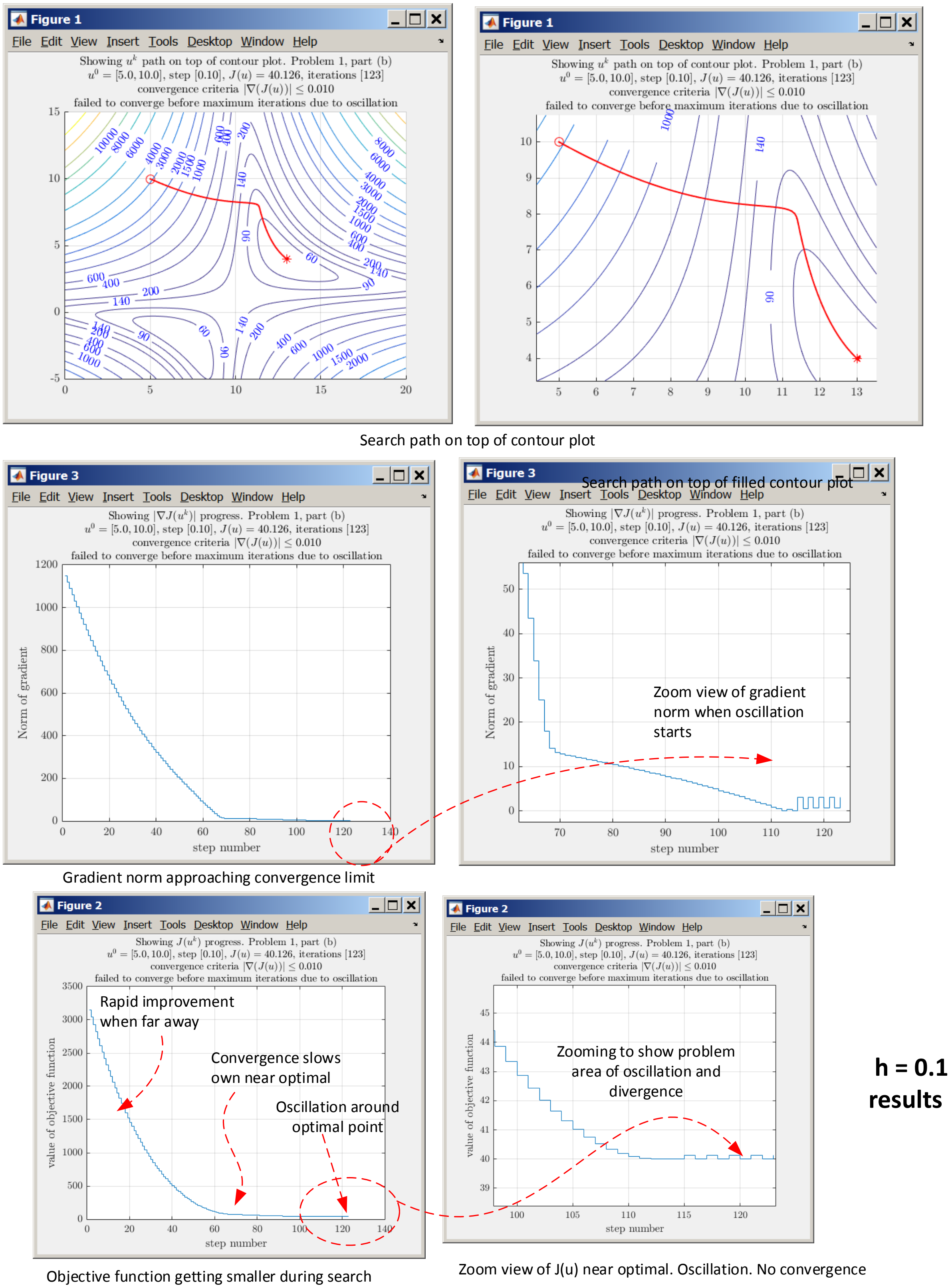


Figure 8: Result for using step 0.1 starting from (5,10)

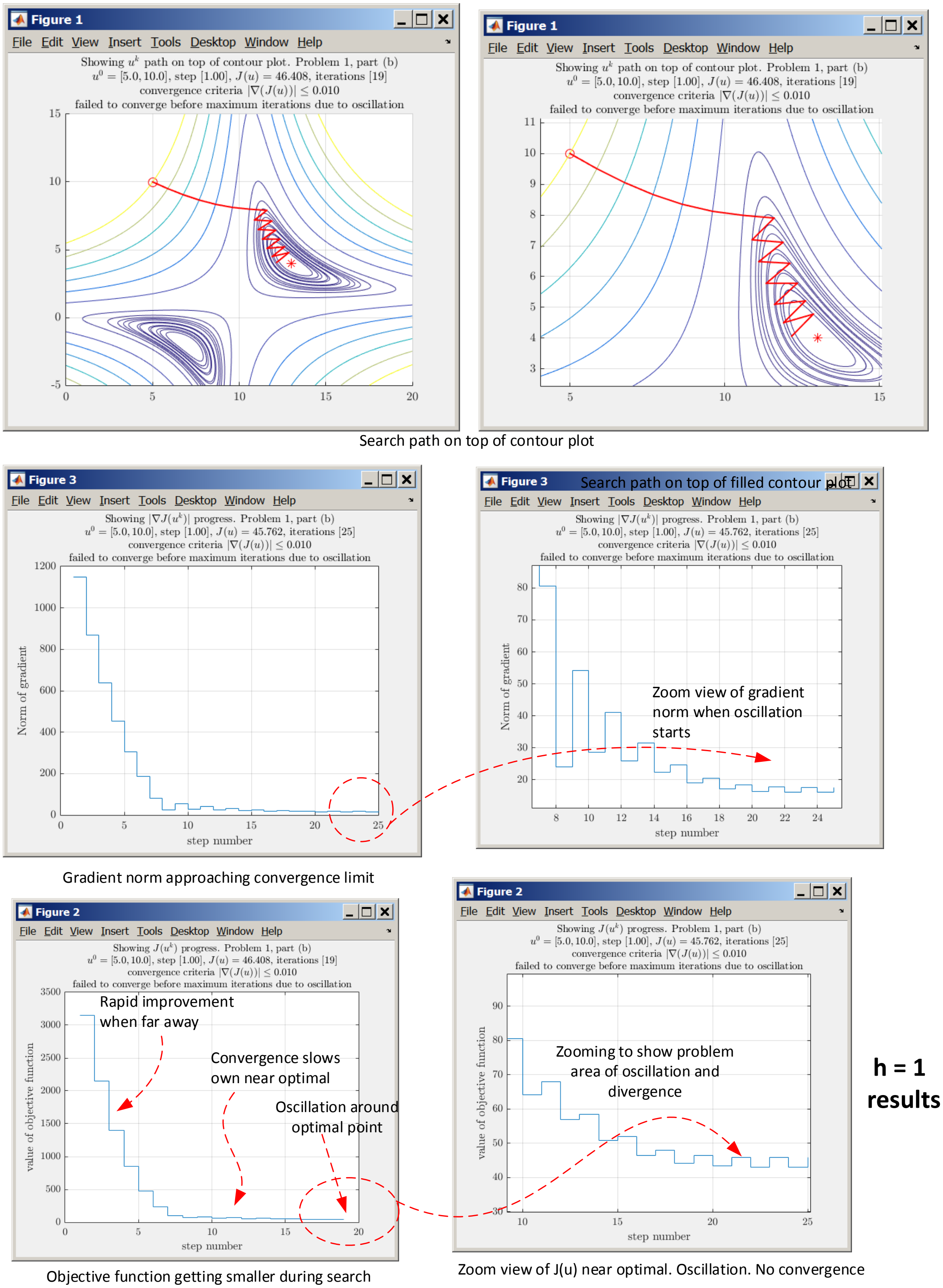


Figure 9: Result for using step 1 starting from (5,10)

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 $\lvert\nabla(J(u))\rvert$ 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 $\lvert\nabla(J(u))\rvert$
		`K>> gradientNormTol(end-6:end)`
		`....`
		`2.80005566566667`
		`0.865917403257339`
		`2.80004081985152`
		`0.865928703656839`
		`2.8000377762892`

Continued on next page

Table 3 – continued from previous page

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`

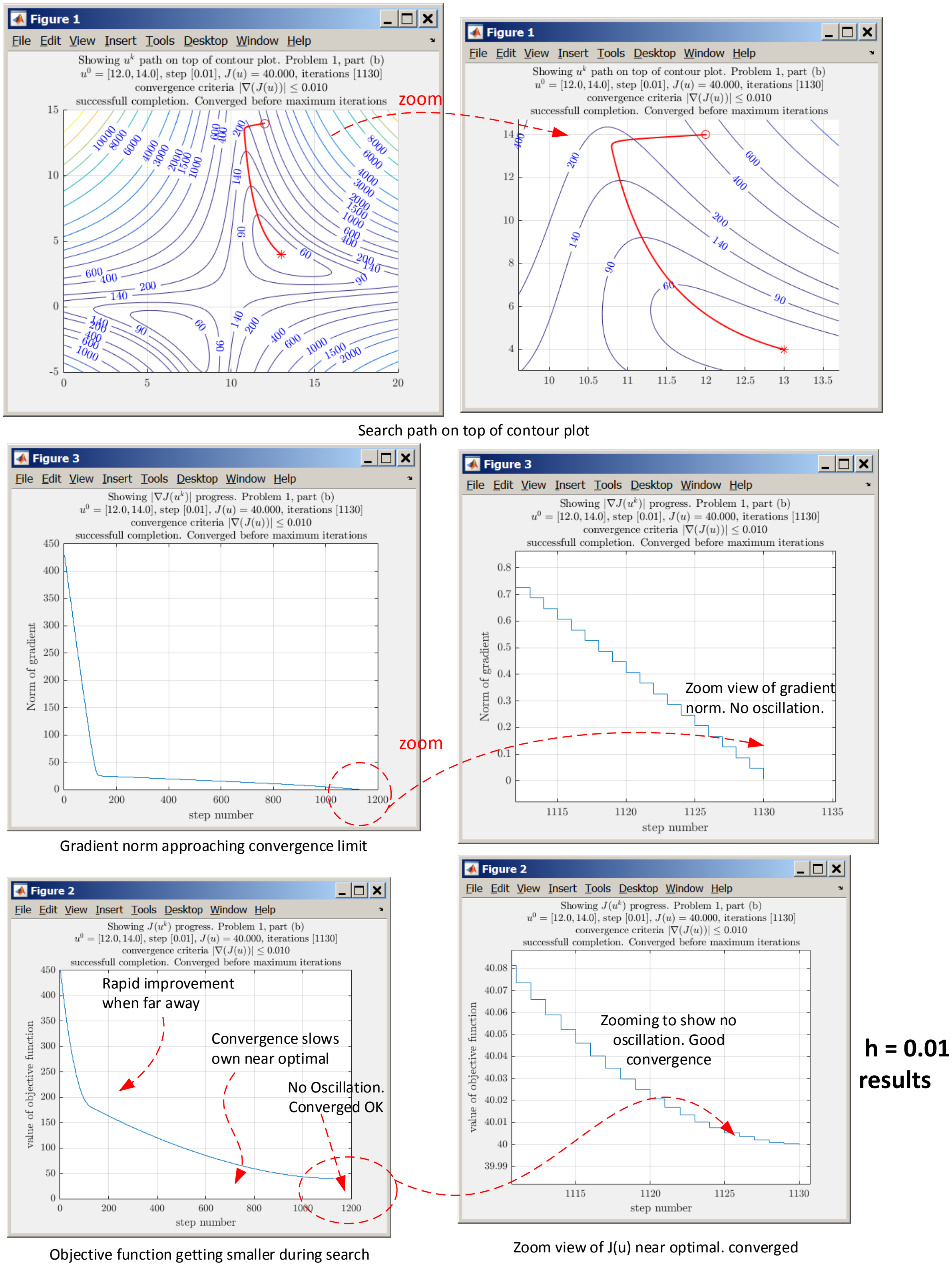


Figure 10: Result for using step 0.01 starting from (12,14)

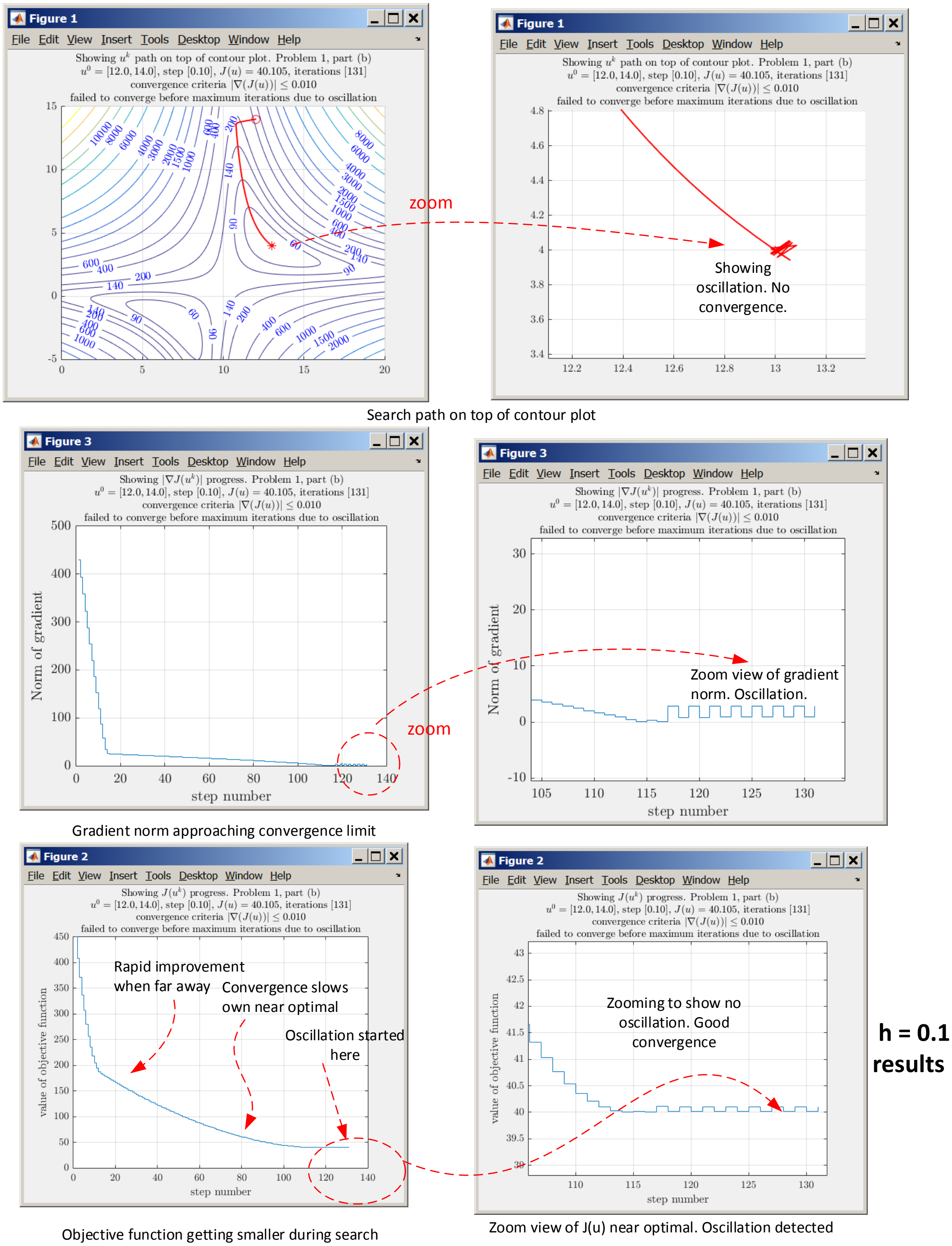
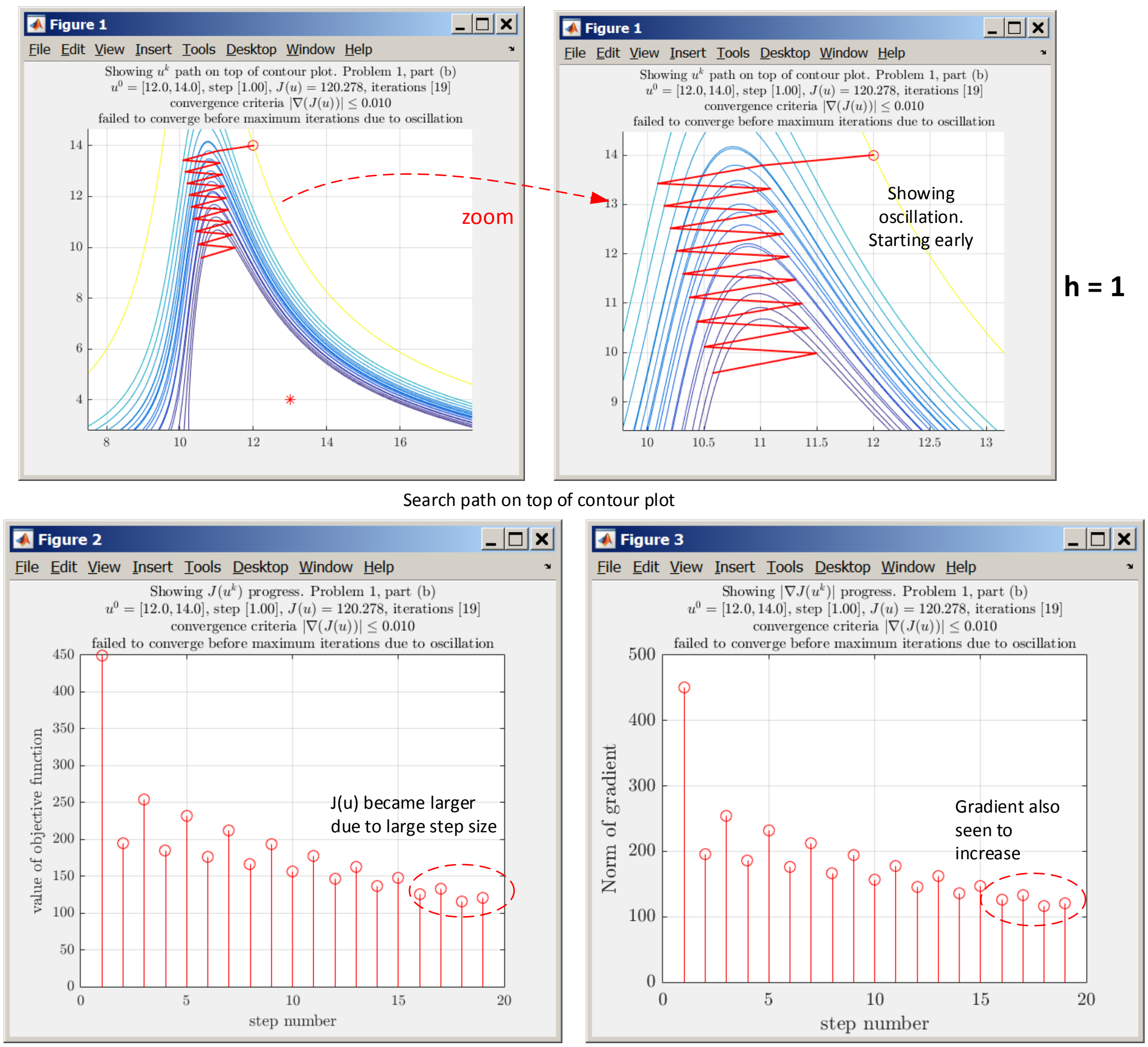


Figure 11: Result for using step 0.1 starting from (12,14)



Search path on top of contour plot

Objective function. Oscillation detected

Norm of gradient.

Figure 12: Result for using step 1 starting from (12,14)

Table 4: Starting point [12,10]

h	steps to converge	comments

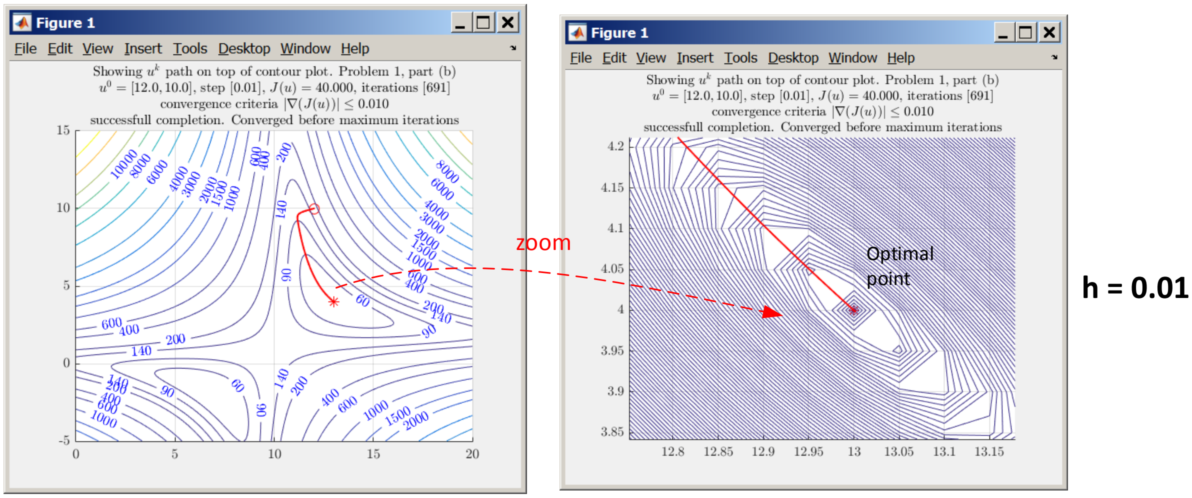
0.01	691	Converged with no oscillation. Here are the last few values of $J(u)$ recorded confirming this.
		`K>> levelSets(end-10:end)`
		`.....`
		`40.0046068598544`
		`40.0028871867126`
		`40.0015674398797`
		`40.0006476523458`
		`40.0001278473181`
		`40.0000080382134`
		Below are the corresponding 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`

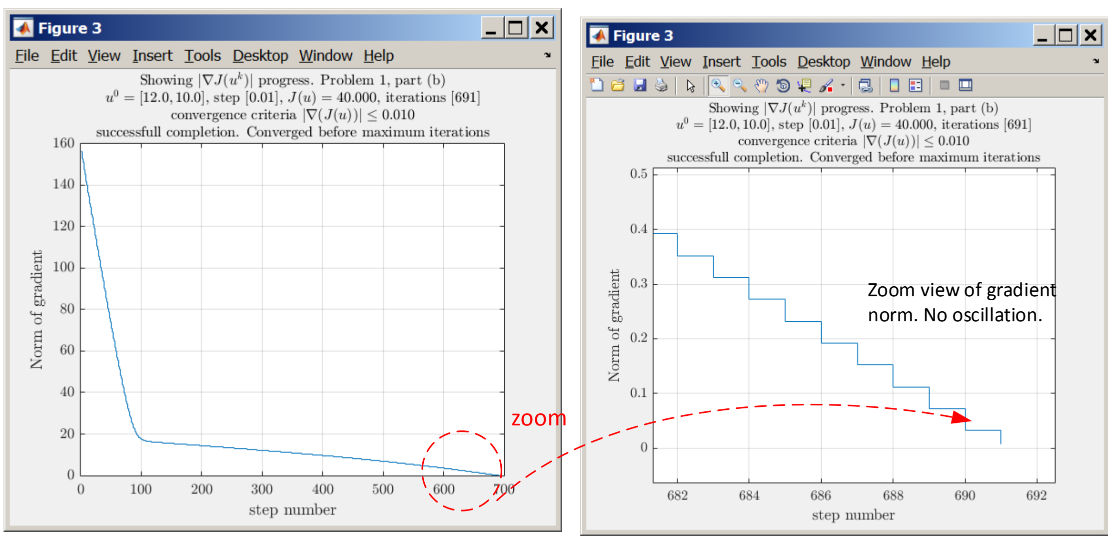
Continued on next page

Table 4 – continued from previous page

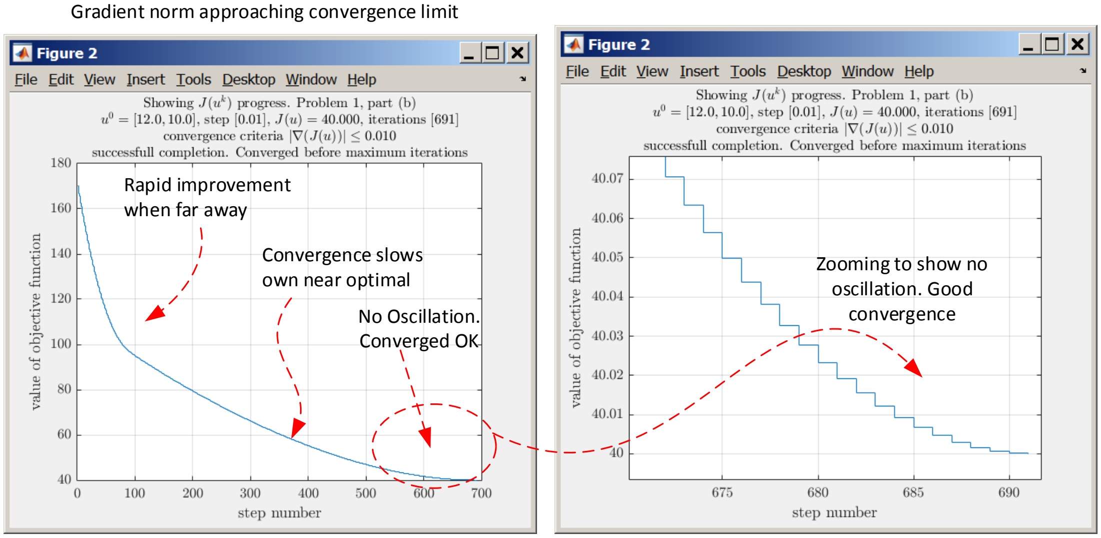
1	24	Did not converge. Oscillation was detected early in the search due to using large step size. Below are the last few values of $J(u)$ recorded confirming this.
		`K>> levelSets(end-10:end)`
		`.....`
		`45.2261295001543`
		`43.5283233241446`
		`45.2260318140989`
		`43.5282741210766`
		`45.2260091586802`
		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`



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 13: Result for using step 0.01 starting from (12,10)

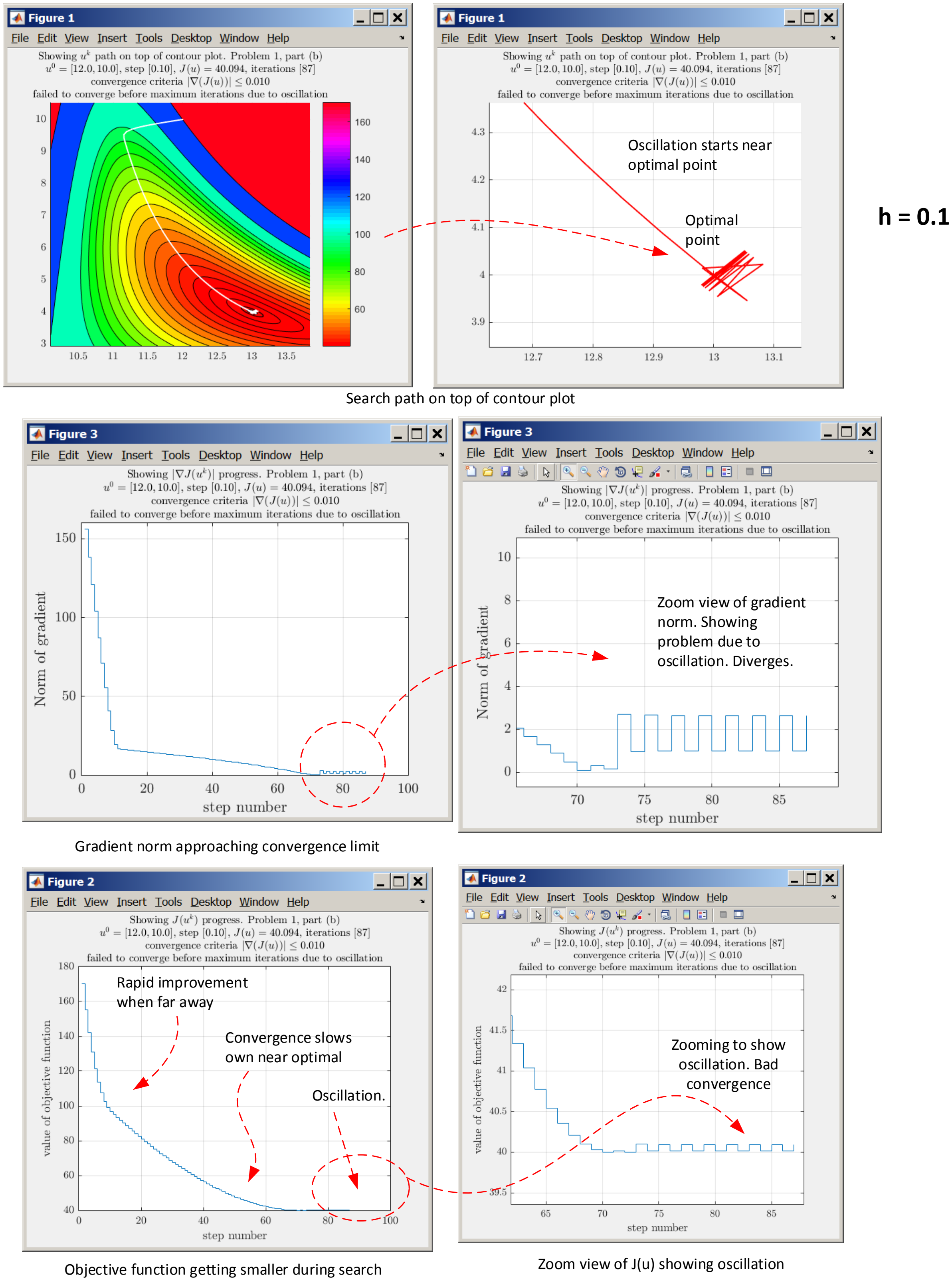


Figure 14: Result for using step 0.1 starting from (12,10)

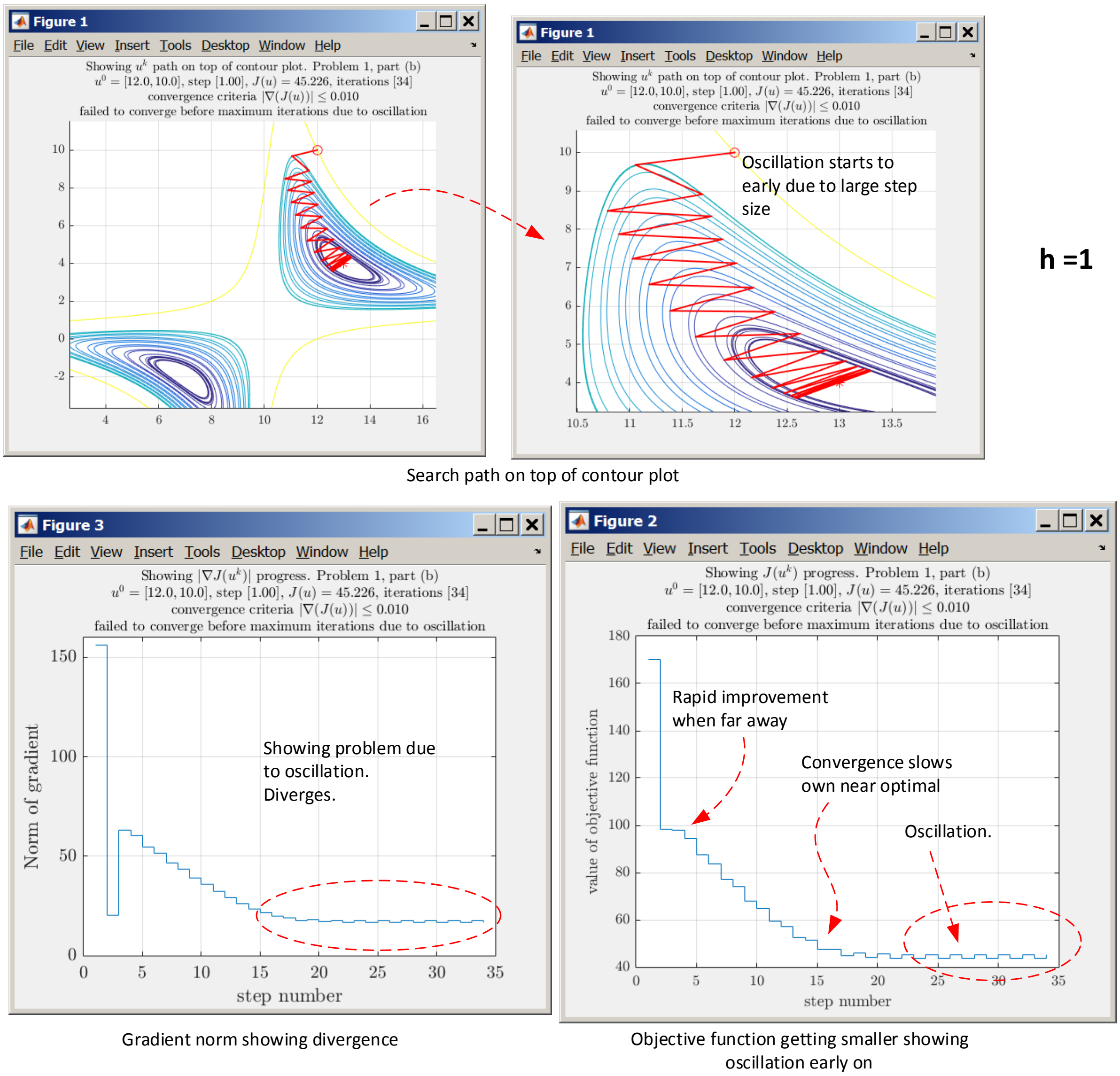


Figure 15: Result for using step 0.1 starting from (12,10)

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

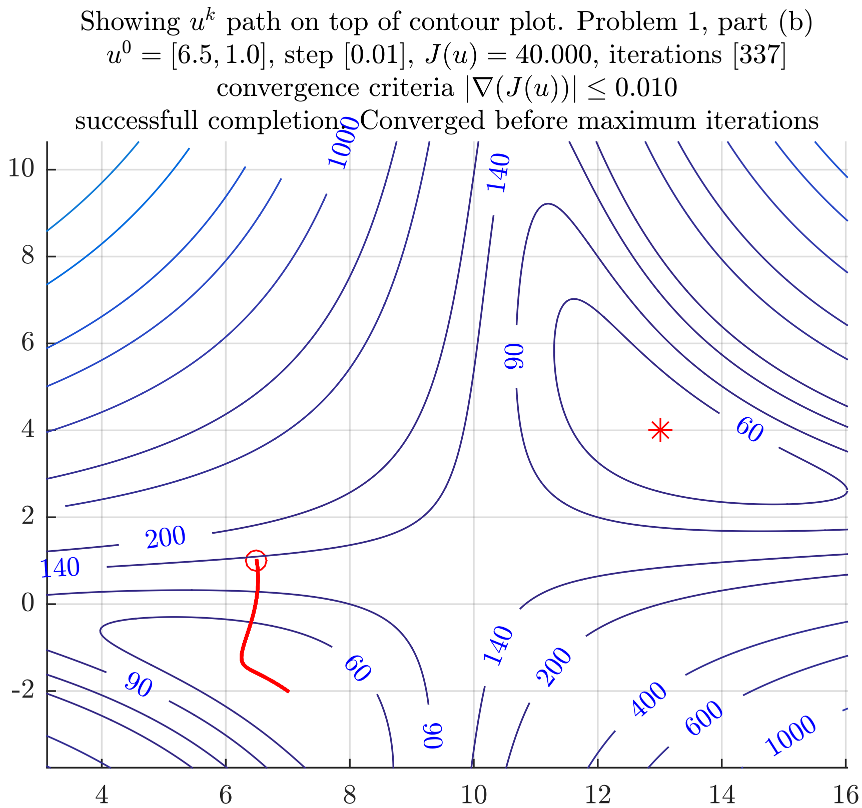


Figure 16: Converging to $(7, -2)$ using step size 0.01 starting from $(6.5, 1)$

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.

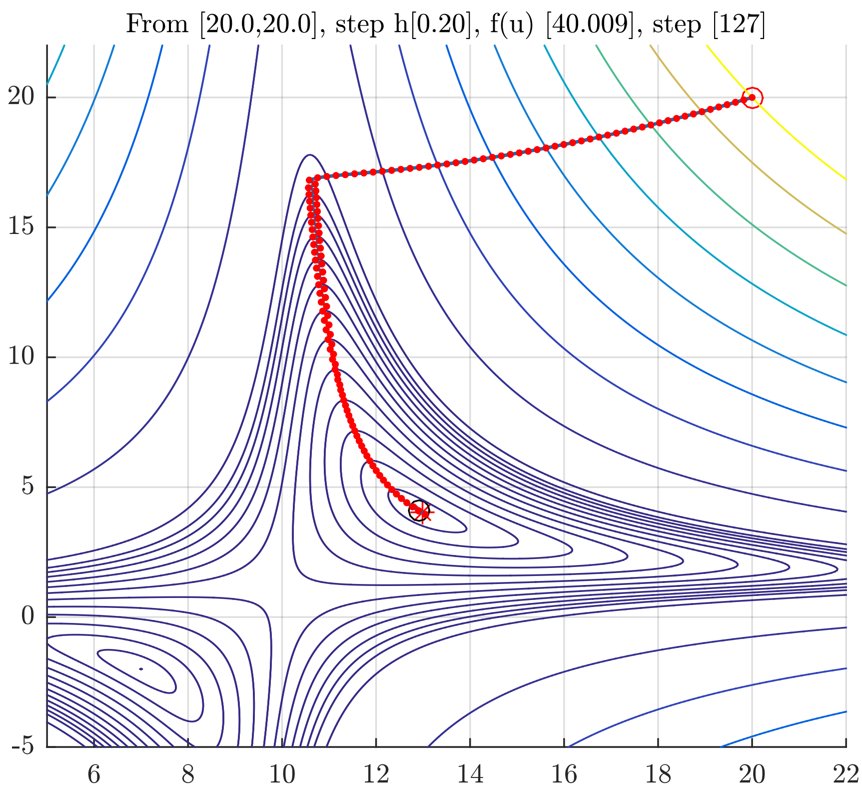


Figure 17: Missing $u^* = [7,-2]$ when starting too far it. Starting from $(20,20)$ using step size 0.01

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

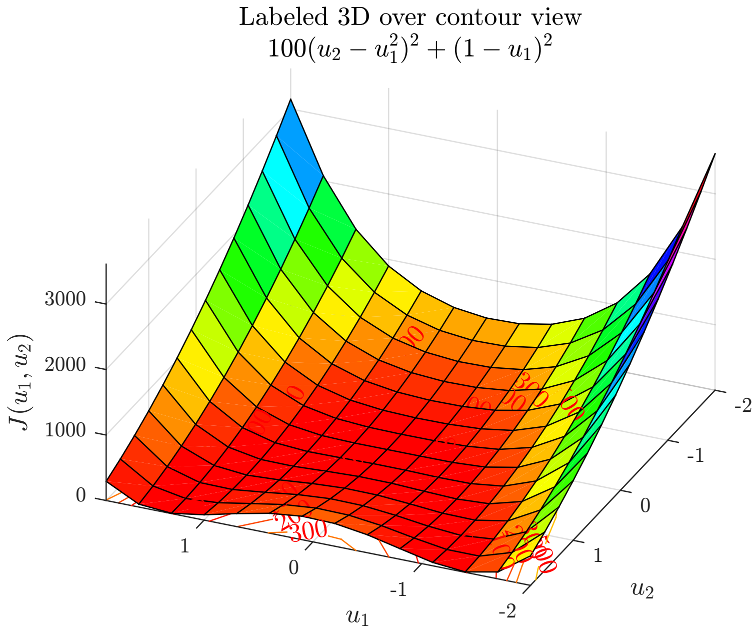


Figure 19: 3D view of $J(u)$

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).

```
procedure STEEPEST_DESCENT_OPTIMAL
    ▷ Initialization
    H ← maximum step size
    max_iterations ← max iterations allowed
    ϵ ← minimum convergence limit on ‖∇J(u)‖
    k ← 0
    u^k ← u^0
    while ‖∇J(u^k)‖ > ϵ do
        ▷ do line search
        h* ← call golden_section(H, J(u)) to find optimal h* of function J̃(h*) = J(u^k − h*∇J(u^k))
        u^k ← u^k − h* ∇J(u^k)/‖∇J(u^k)‖
        k ← k + 1
        ▷ detect oscillation
        if k ≥ max_iterations or J(u_k) > J(u_{k−1}) then
            exit loop
        end if
    end while
end procedure
```

Figure 20: Steepest descent, optimal step size algorithm

For $n = 2$, the Rosenbrock function is

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

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

For

$$-2.5 \le u_i \le 2.5$$

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 $J^* = 0$.

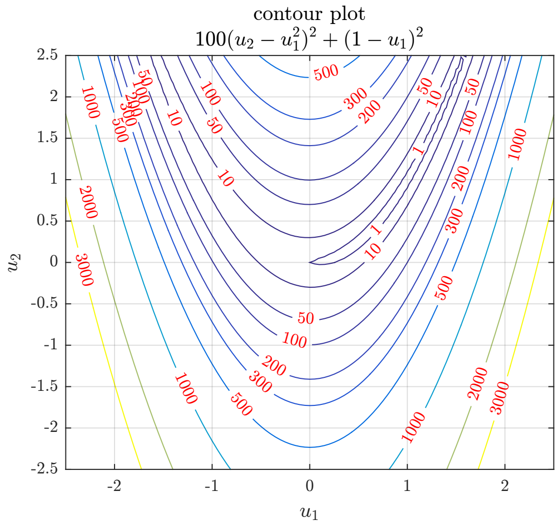


Figure 21: Contour $J(u)$

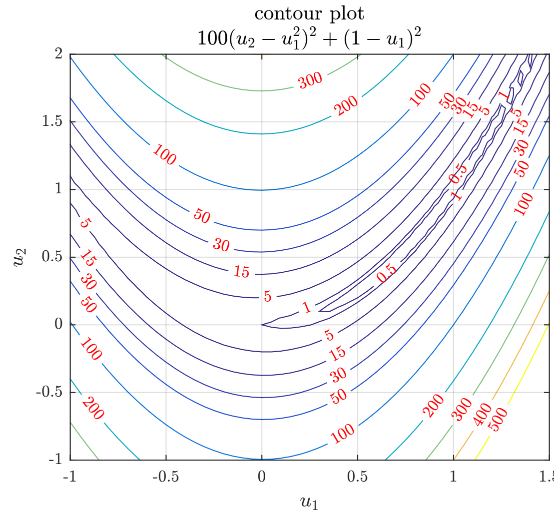


Figure 22: Zooming on Contour $J(u)$

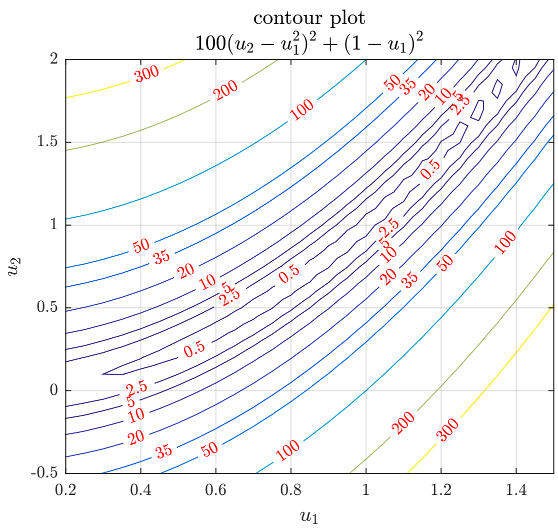


Figure 23: More zooming. Contour $J(u)$

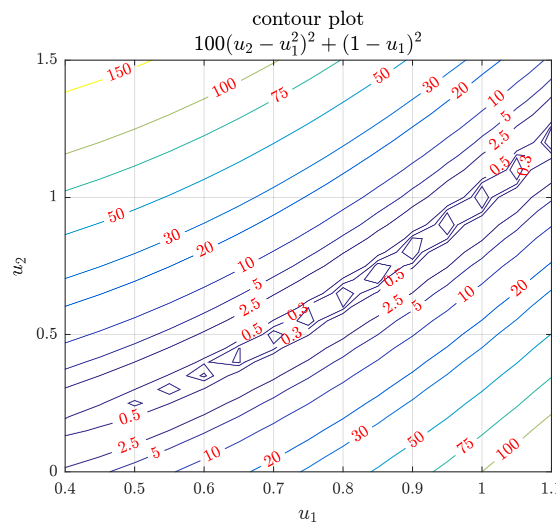


Figure 24: More zooming. Contour $J(u)$

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.

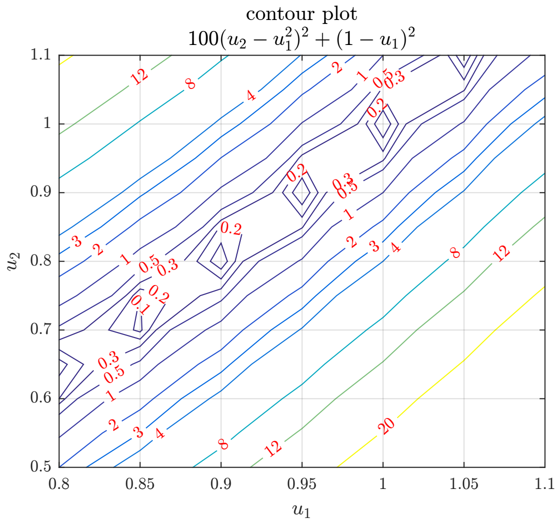


Figure 25: More zooming on Contour $J(u)$

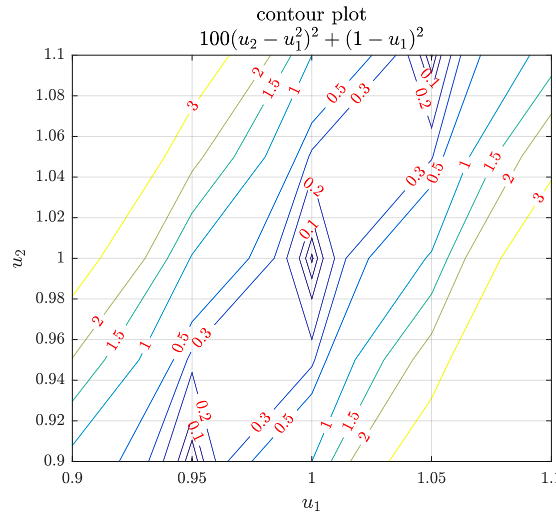


Figure 26: More zooming on Contour $J(u)$

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.

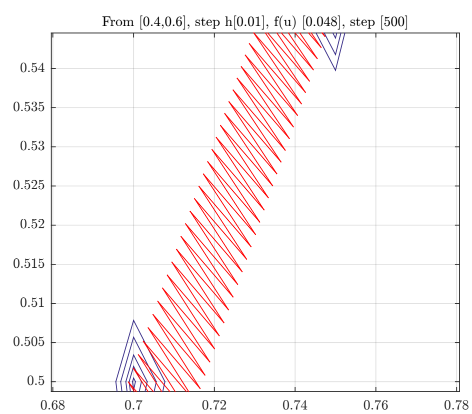


Figure 27: Zoom view of search when inside valley, showing the small steps and zig-zag pattern

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.

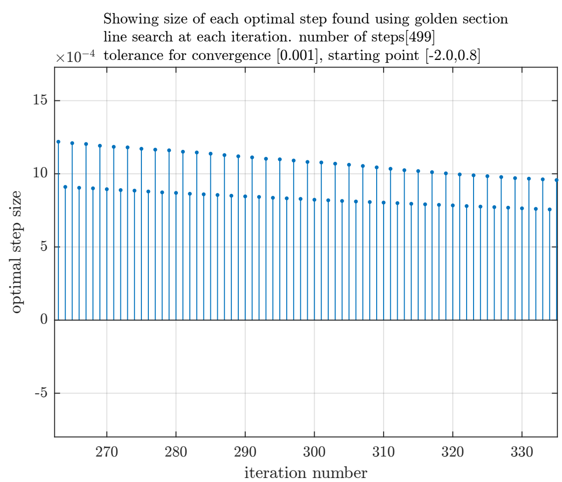


Figure 28: Showing how optimal step size changes at each iteration during typical search

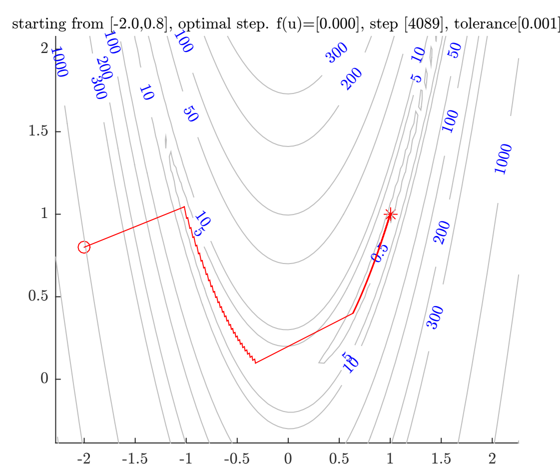


Figure 29: Typical search pattern using optimal step size from arbitrary starting point

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.

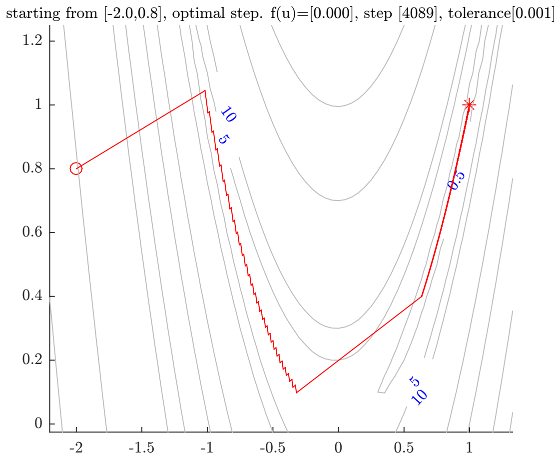


Figure 30: path of u^k using optimal step starting from $(-2, 0.8)$

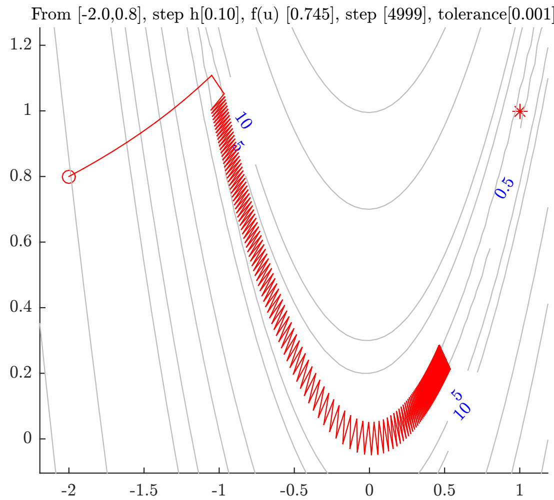


Figure 31: path of u^k using fixed step $h = 0.1$ starting from $(-2, 0.8)$

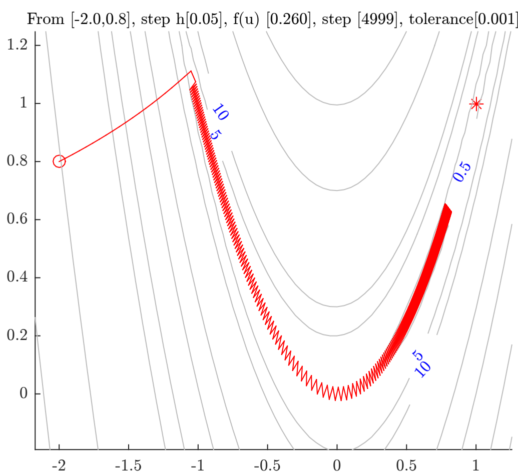


Figure 32: u^k path, fixed step $h = 0.05$ from $(-2, 0.8)$

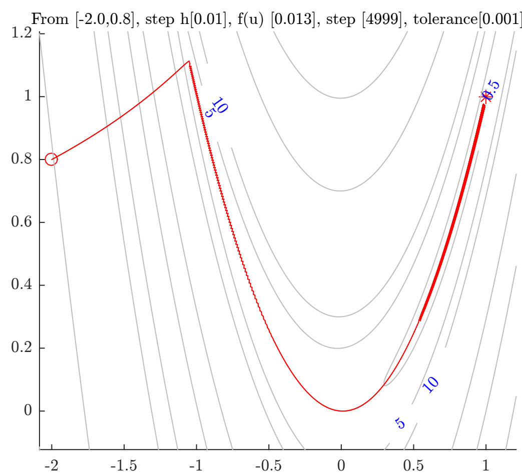


Figure 33: u^k path, fixed step $h = 0.01$ from $(-2, 0.8)$

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.

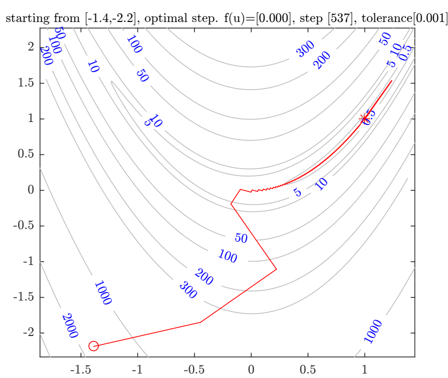


Figure 34: u^k path, optimal step from at $(-1.4, -2)$

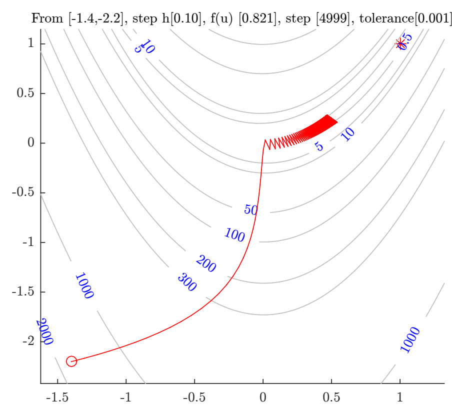


Figure 35: u^k path, fixed step $h = 0.1$ from $(-1.4, -2)$

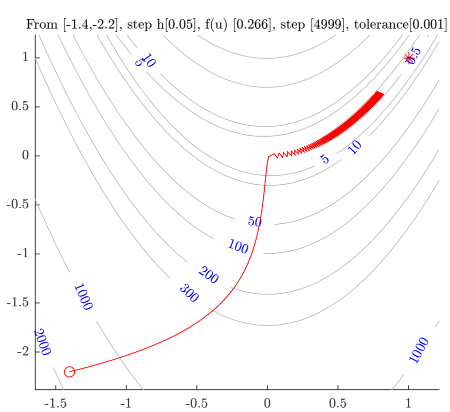


Figure 36: u^k path, fixed step $h = 0.05$ from $(-1.4, -2)$

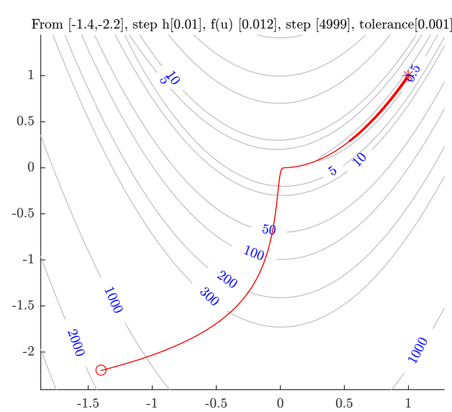


Figure 37: u^k path, fixed step $h = 0.01$ from $(-1.4, -2.0)$

0.2.2 Part(b)

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

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

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

And for $n = 4$

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

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

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

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

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

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

.

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

```
%---------------------
%Evaluate grad(J(u)) at u
```

```
function g = gradientFunc(u)
 u = u(:);
 N = size(u,1);
 g = zeros(N,1);
 for i = 1:N
    if i==1 || i==N
       if i==1
          g(i)=-400*(u(i+1)-u(i)^2)*u(i)-2*(1-u(i));
       else
          g(i)=200*(u(i)-u(i-1)^2);
       end
    else
       g(i) = 200*(u(i)-u(i-1)^2)-...
                 400*(u(i+1)-u(i)^2)*u(i)-2*(1-u(i));
    end
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 $(-2, -2, \dots, -2)$. 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.

```
....
tic;
[status,pts,levelSets, gradientNormTol,steps] = ...
             nma_steepest_descent(opt);
time_used = toc;
.....
```

.

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.

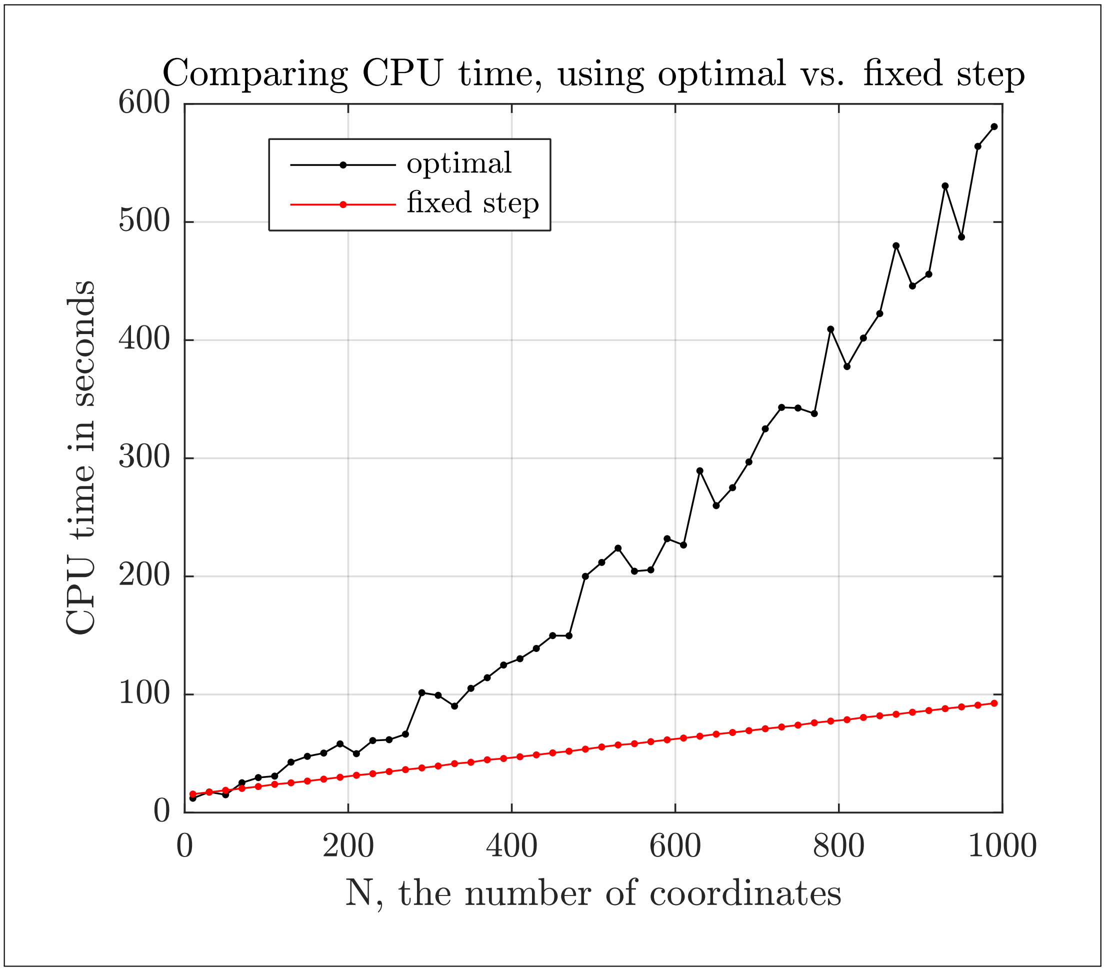


Figure 38: Comparing CPU time, optimal step and fixed step

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

```
function [status,pointsFound,levelSets,gradientNormTol,steps]= ...
                      nma_steepest_descent(opt)
% This function performs steepest descent search starting from
% a point looking for point which minimizes a function. Supports
% multi-variable function. It needs handle of the funtion and
% hand to the gradient. It reurns all points visited in the
% search. The points can then be used by client for plotting.
% Below is description of input and output.
%
% Typical use of this function is as follows:
%
%  opt.field = ...%fill in each field of the struct.
%  [pointsFound,levelSets,gradientNormTol,steps]  = ...
%                                    nma_steepest_descent(opt);
%
%  [C,h]    = contour(.....,levelSets);
%
% INPUT fields in opt struct are:
% ======
% u                vector of coordinates starting guess
% MAX_ITER         an integer, which is the maximium iteration
%                  allowed before giving up the search.
%                  Example 500
% gradientNormTol  small floating point number. The tolerance
%                  to use to decide when to stop the search.
%                  Example 0.001
% stepSize         A floating point number, which is the step
%                  size to take. If stepSize=-1 then an optimal
%                  step size is found and used at each step
%                  using golden section line search.
% objectiveFunc    handle to the objective function which accepts
%                  a row vector, that contain [x y] coordinate
%                  of the point and returns the numerical value
%                  of objectiveFunc at this point.
% gradientFunc     handle to the gradiant of f. Same input and
%                  output as objectiveFunc
% accumulate       flag. If true, then all points u^k and J(u)
```

```
%                    at each are collected during search. Else
%                    they are not.
%
% OUTPUT:
% =======
% status             can be 0,1 or 2.
%                    0 means success, It converged before MAX_ITER
%                    was reached.
%                    1 means failed, did not converge due to
%                    oscillation, which can happen when step size
%                    is too large. When oscillation detected, the
%                    search will stop.
%                    2 means failed: did not oscillate but also
%                    did not converge before hitting MAX_ITER.
%                    Caller can try with larger MAX_ITER
% pointsFound        n by 2 matrix, as in [x1 y1; x2 y2; .....]
%                    which contains coordinates of each point
%                    visited during steepestDescent the length is
%                    the same as number of points visited. This
%                    will be last point only if opt.accumlate=false
% levelSets          vector, contains the value of the objective
%                    function at each point. Last value of J(u) if
%                    opt.accumlate=false
% gradientNormTol    vector, contains the norm of gradient after
%                    each step. This will be last value only if
%                    opt.accumlate=false
% steps              vector. The optimal step used at each
%                    iteration, used golden section to find optimal
%                    step size. This will be last value only
%                    if opt.accumlate=false
%
% by Nasser M. Abbasi  ECE 719, UW Madison

%pre-allocate data for use in the main loop below
N                = size(opt.u,1);

%collect data only if user asked for it.
if opt.accumulate
    pointsFound     = zeros(opt.MAX_ITER,N);
    levelSets       = zeros(opt.MAX_ITER,1);
    gradientNormTol = zeros(opt.MAX_ITER,1);
    steps           = zeros(opt.MAX_ITER,1);
end

%function to find optimal step size at each step,
%This is used only if client asked for optimal
%step size, which is set when opt.setSize=-1
```

```
%This is same J tilde(u) function from class lecture notes
fLambda = @(lam,u,s) opt.objectiveFunc(u-lam*s);

% initialize counters before main loop
k                     = 1;
currentPoint          = opt.u;
keepRunning           = true;
status                = 0;
steps_in_oscillation = 0;
last_level            = 0;

while keepRunning
    if k>1
        last_level = current_level;
    end
    current_level     = norm(opt.objectiveFunc(currentPoint));
    current_grad      = opt.gradientFunc(currentPoint);
    current_grad_norm = norm(current_grad);

    if opt.accumulate
        pointsFound(k,:)   = currentPoint;
        levelSets(k)       = current_level;
        gradientNormTol(k) = current_grad_norm;
    end

    if k>1 && current_level>last_level% check for oscillation
        if opt.stop_on_oscillation
           steps_in_oscillation = steps_in_oscillation + 1;
        end
    end

    % check if we converged or not
    % Last check below can lead to termination too early for the
    % banana function. Since at one point, J(u(k+1)) will get
    % larger than J(u(k)) using bad step size. So it is
    %commented out for now.
    if k == opt.MAX_ITER || ...
              current_grad_norm <=opt.gradientNormTol || ...
           steps_in_oscillation>4 %let it run for 2 more steps
             %to see the oscillation stop loop and set the
             %status to correct reason why loop stopped.
        keepRunning = false;
        if steps_in_oscillation>0
            status = 1;
        else
            if k == opt.MAX_ITER
                status= 2;
```

```
            end
        end
    else
        if current_grad_norm  > eps('double') %direction vector
            s = current_grad / current_grad_norm;
            if opt.STEP_SIZE == -1 %are we using optimal size?
                lam = nma_golden_section(...
                 fLambda,currentPoint,s,0,1,sqrt(eps('double')));

             %below for verification of golden section result
             %using matlab fminbd. I get similar results. so
             %this is good.

             %lam=fminbnd(@(lam) fLambda(lam,currentPoint,s),0,1);
            else
                lam  = opt.STEP_SIZE; %using the fixed step size.
            end

            %protect aginst long step,just in case?
            %lam = min([1,lam]);

            % make step towards minumum
            currentPoint = currentPoint - lam*s;

            if opt.accumulate
                steps(k) = lam;
            end

            k = k + 1;
        else
            keepRunning = false; % |grad| < eps, stop.
        end
    end

end

%done. Chop data to correct number of steps used before returning
if opt.accumulate
    pointsFound     = pointsFound(1:k,:);
    levelSets       = levelSets(1:k);
    gradientNormTol = gradientNormTol(1:k);
    steps           = steps(1:k);
else
    pointsFound     = currentPoint ;
    levelSets       = current_level;
    gradientNormTol = current_grad_norm;
    steps           = k;
```

```
end

end
```

0.3.2 golden section line search

```
function h_optimal = nma_golden_section(f,u,s,a,b,tol)
% standard golden section function (see numerical recipes)
%converted to Matlab to use for HW 4. This finds the optimal
%step size to use with the steepest descent algorithm.
%
%Nasser M. Abbasi, ECE 719 spring 2016
%
%
%INPUT:
% f: The function to minimize
% u and s: These are specific parameters for f() used only
%          for HW4 problem and not part of the general algorithm
%          itself. These are used in calling f(). u is the
%          current point and "s" is the gradiant vector. in the
%          direction we want to minimize J(u)
% a: Starting search point
% b: ending search point.
% tol: tolerance to use to stop the line search. Such as 10^(-6)
%
% OUTPUT:
%  h_optimal: This is the optimal step size h to use
%
golden_ratio = (sqrt(5)-1)/2;
c            = b-golden_ratio*(b-a);
d            = a+golden_ratio*(b-a);

while abs(c-d)>tol
    fc = f(c,u,s);
    fd = f(d,u,s);
    if fc<fd
        b = d;
        d = c;
        c = b-golden_ratio*(b-a);
    else
        a = c;
        c = d;
        d = a+golden_ratio*(b-a);
    end
end
%done. Return the optimal step size to use.
h_optimal = (b+a)/2;
```

```
end
```

0.3.3 Problem 1 part a

```
function nma_HW4_problem_1_part_a()
%Plots a contour of
%
%    f(u) = (11-u1-u2)^2 + (1+u1+10*u2-u1*u2)^2
%
% over range u1=0..20 and u2=0..15
% Matlab 2015a
% by Nasser M. Abbasi

close all; clc;
cd(fileparts(mfilename('fullpath')));

%reset(0);
xlimits   = [-5 20];  %x limits, for plotting, change as needed
ylimits   = [-5 15]; %y limits, for plotting, change as needed
myTitle   = '$$(11 - u_1 - u_2)^2 +(1+ u_1+10 u_2-u_1 u_2)^2$$';
[u1,u2,z] = makeContourData(0.05,xlimits,ylimits);

figure(1);
v        =[40 60 90 140 200 400 600 1000 1500 2000 3000 ...
                       4000 6000 8000 10000 12000 15000 18000];
[C,h]    = contour(u1,u2,z,v,'Linecolor',[0 0 1]);

clabel(C,h,v,'Fontsize',8,'interpreter','Latex','Color','red');
setMyLabels('$$u_1$$','$$u_2$$',...
    {'\makebox[4in][c]{contour plot, default setting}',...
    sprintf('\\makebox[4in][c]{%s}',myTitle)});
saveas(gcf, 'problem_1/part_a/fig1', 'pdf');

figure(11);
xlimits   = [-5 20];  %x limits, for plotting, change as needed
ylimits   = [-5 20]; %y limits, for plotting, change as needed
myTitle   = '$$(11 - u_1 - u_2)^2 +(1+ u_1+10 u_2-u_1 u_2)^2$$';
[u1,u2,z] = makeContourData(0.1,xlimits,ylimits);
[C,h]     = contourf(u1,u2,z,v);
%colorDepth = 10000;
%colormap(jet(colorDepth));

%colormap(parula(300));
colormap(hsv);
colorbar;
setMyLabels('$$u_1$$','$$u_2$$',...
    {'\makebox[4in][c]{contour plot, filled, with colorbar}',...
```

```
    sprintf('\\makebox[4in][c]{%s}',myTitle)});
%saveas(gcf, 'problem_1/part_a/fig11', 'pdf');
%print -painters -dpdf -r600 'problem_1/part_a/fig11.pdf'

figure(12);
contour3(u1,u2,z,v);


figure(2);
[u1,u2,z] = makeContourData(2,xlimits,ylimits);
surf(u1,u2,z);
colormap(hsv);
view([-156.5,42]);

hold on;
v        = [200 600 1000 1500 2000 4000 6000 8000 12000];
[C,h]    = contour(u1,u2,z,v,'Linecolor',[0 0 1]);
clabel(C,h,v,'Fontsize',10,'interpreter','Latex','Color','red');

setMyLabels('$$u_1$$','$$u_2$$','$$J(u_1,u_2)$$',...
    {'\makebox[4in][c]{Labeled 3D over contour view}',...
    sprintf('\\makebox[4in][c]{%s}',myTitle)})
%saveas(gcf, 'problem_1/part_a/fig2', 'pdf');

figure(3);
surf(u1,u2,z);
colormap(hsv);
view([154,46]);
hold on;
contour(u1,u2,z,v,'Linecolor',[0 0 1]);
clabel(C,h,v,'Fontsize',10,'interpreter','Latex','Color','red');
setMyLabels('$$u_1$$','$$u_2$$','$$J(u_1,u_2)$$',...
{'\makebox[4in][c]{Another 3D over contour view (no labels)}',...
    sprintf('\\makebox[4in][c]{%s}',myTitle)})
%saveas(gcf, 'problem_1/part_a/fig3', 'pdf');

figure(4);
xlimits   = [-10 30];  %x limits, for plotting, change as needed
ylimits   = [-10 30];  %y limits, for plotting, change as needed
[u1,u2,z] = makeContourData(.5,xlimits,ylimits);

subplot(1,2,1);
v        =[50 200 600 2000 4000 8000 16000 30000];
[C,h]    = contour(u1,u2,z,v,'Linecolor',[0 0 1]);
grid;   %get(h,'LevelList')

clabel(C,h,v,'Fontsize',8,'interpreter','Latex','Color','red');
```

```
setMyLabels('$$u_1$$','$$u_2$$',...
    {'\makebox[4in][c]{contour plot (enlarged limits}',...
    sprintf('\\makebox[4in][c]{%s}',myTitle)});

subplot(1,2,2);
[u1,u2,z] = makeContourData(4,xlimits,ylimits);
surf(u1,u2,z);
colormap(hsv);
view([154,46]);
hold on;
contour(u1,u2,z,'Linecolor',[0 0 1]);
setMyLabels('$$u_1$$','$$u_2$$','$$J(u_1,u_2)$$',...
 {'\makebox[4in][c]{3D over contour view (enlarged limits)}',...
    sprintf('\\makebox[4in][c]{%s}',myTitle)});
%saveas(gcf, 'problem_1/part_a/fig4', 'pdf');
end

%------------
%helper function to set plot attributes.
function setMyLabels(varargin)

myXlabel = varargin{1};
myYlabel = varargin{2};
if nargin ==4
    myZlabel = varargin{3};
end
myTitle  = varargin{end};
h        = get(gca,'xlabel');
set(h,'string',myXlabel,'fontsize',10,'interpreter','Latex') ;

h = get(gca,'ylabel');
set(h,'string',myYlabel,'fontsize',10,'interpreter','Latex') ;

if nargin ==4
    h = get(gca,'zlabel');
    set(h,'string',myZlabel,'fontsize',10,'interpreter','Latex');
end

h = get(gca,'title');
set(h,'string',myTitle,'fontsize',10,'interpreter','Latex',...
    'HorizontalAlignment','center') ;

set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
end

%------------------
%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       = (11-u1-u2).^2 + (1+u1+10.*u2-u1.*u2).^2;
end
```

0.3.4 Problem 1 part b

```
function nma_HW4_problem_1_part_b()
%finds the min value of
%
%    f(u) = (11-u1-u2)^2 + (1+u1+10*u2-u1*u2)^2
%
% over range u1=0..20 and u2=0..15 using steepest descent
%
%This file is only the driver for function nma_steepestDescent.m
%ECE 719, Spring 2016
%Matlab 2015a
%Nasser M. Abbasi   Nov 25, 2016

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
%                      from GUI
DO_ANIMATE   = true;   %set to true to see animation
DO_GIF       = false;  %set to true to make animation gif
DO_3D        = false;  %if we want to show 3D search path. Set to true
xlimits      = [-20 20];   %x limits, for plotting
ylimits      = [-15 15];   %y limits, for plotting
del          = 0.05;       %grid size, used for making meshgrid
fixed_levels = [40 60 90 140 200 400 600 1000 1500 2000 ...
```

```
                 3000 4000 6000 8000,10000 12000 15000 18000];
CONTOUR_LINES_AUTO = 'fix'; %set to 'auto', to see matlab contour
%             lines, set to 'full' to see each step level set
%             set to 'limited' to see every other level
%             set to 'fix' to use pre-specificed
%

%-------------------------------------------------------
%These are the options struct used by call to
%                                 nma_steepestDescentPoints()
opt.u                = [16.805;13.245]; %starting guess x-coord
opt.MAX_ITER         = 10^3; %maximum iterations allowed

%step size. set to -1 to use an optimal step
opt.STEP_SIZE        = -1;

%see function definition at end of file
opt.objectiveFunc    = @objectiveFunc;

%see function definition at end of file
opt.gradientFunc     = @gradientFunc;
opt.gradientNormTol = 0.001;   %used to determine when converged
opt.hessian          = @hessian_func;  %see function definition
opt.accumulate       = true;
opt.stop_on_oscillation = false;

%-------------------------------------------
%data
u1       = xlimits(1):del:xlimits(2);
u2       = ylimits(1):del:ylimits(2);
[u1,u2]  = meshgrid(u1,u2);
%z        = 3 + (u1 - 1.5*u2).^2 + (u2 - 2).^2;
z         = (11-u1-u2).^2 + (1+u1+10.*u2-u1.*u2).^2;

figure('Units','pixels','position',[from_pix from_pix 600 500]);
pix_count = pix_count+1;
if DO_GUI  %check if GUI input is asked for, if so, wait for user
    plot(0,0);
    xlim(xlimits); ylim(ylimits);
    hold on;
    [x,y] = ginput(1);
    opt.u=[x;y];
end

%mark location of starting point
%t          = text(0.8*opt.u(1),1.1*opt.u(2),...
%                    sprintf('[%2.1f,%2.1f]',opt.u(1),opt.u(2)));
```

```
%t.FontSize = 8;
%t.Color    = 'red';

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

%mark location of minimum found by fminsearch on plot
%This min, can be different that one converged to by
%steepest descent! so we also plot the converged to value found
hold on;
%plot(optimalValue(1),optimalValue(2),'*r');

plot(opt.u(1),opt.u(2),'or');  %starting point
xlim(xlimits); ylim(ylimits);
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
%make the call to implement steepest descent, different m file.
if strcmp(METHOD,'steepest descent')
   [status , pts,levelSets, gradientNormTol,~] = ...
                                      nma_steepest_descent(opt);
else
   [status,pts,levelSets, gradientNormTol,~] = ...
                                       nma_fletcher_reeves(opt);
end
plot(13,4,'*r');  %known u* at top location.
switch status
    case 0, status = ...
  'successfull completion. Converged before maximum iterations';
    case 1, status = ...
'failed to converge before maximum iterations due to oscillation';
    case 2, status = ....
               'failed to converge before maximum iterations';
end

%plot the value found by steepest descent
%plot(pts(end,1),pts(end,2),'ok');

%use output from above call to make the plots
switch CONTOUR_LINES_AUTO
    case 'auto',
    [C,h] = contour(u1,u2,z,'Linecolor',[0 0 1],'LineWidth',0.1);
    case 'limited',
      lev   = round(length(levelSets)/20);
      %[C,h] = contour(u1,u2,z,levelSets(1:lev:end),'Fill','off');
      %[C,h]     = contourf(u1,u2,z,levelSets(1:lev:end));
```

```
      [C,h]     = contour(u1,u2,z,levelSets(1:lev:end));
      %colormap(hsv);
      %colorbar;
      %'Linecolor',[0 0 1],'LineWidth',.2);
    case 'full'
        [C,h] = contour(u1,u2,z,levelSets,'LineWidth',.2);
    case 'fix'
        [C,h] = contour(u1,u2,z,fixed_levels);
        h.LineWidth = .1;
        %h.LineColor  = [190/255 190/255 190/255];
        clabel(C,h,fixed_levels,'Fontsize',8,...
                          'interpreter','Latex','Color','blue');
end

%animate the steepest descent search
if length(pts(:,1))>1
    filename = 'anim.gif';
    for k=1:length(pts)-1
        %draw line between each step
        %skip case if 'full' mode or if too many points.
        %if (opt.STEP_SIZE == -1 ||...
 %                  strcmp(CONTOUR_LINES_AUTO,'limited') || ...
        %    strcmp(CONTOUR_LINES_AUTO,'auto')||length(pts)<100 )
        if strcmp(CONTOUR_LINES_AUTO,'full')||...
                           strcmp(CONTOUR_LINES_AUTO,'limited')
            line([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],...
                                   'LineWidth',1,'Color','red');
        else
            line([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],...
                                   'LineWidth',1,'Color','red');
        end
        %end
        %plot([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],'.r');
        if DO_ANIMATE
            drawnow;
            if DO_GIF
               frame = getframe(1);
               im = frame2im(frame);
               [imind,cm] = rgb2ind(im,256);
               if k ==1
                  imwrite(imind,cm,filename,'gif', 'Loopcount',0);
               else
                  if mod(k,4)==0
                     imwrite(imind,cm,filename,...
                                    'gif','WriteMode','append');
                  end
               end
```

```
            end
        end
        title(format_plot_title(...
            ['Showing $u^k$ path on top of contour plot.' ...
             'Problem 1, part (b)'],...
           opt,pts,k,status),'FontSize', 8);
    end
end
title(format_plot_title(['Showing $u^k$ path on top of'...
                         'contour plot. Problem 1, part (b)'],...
    opt,pts,size(pts,1),status),'FontSize', 8);


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

stairs(levelSets);
%stem(levelSets,'ro');
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
title(format_plot_title(...
            'Showing $J(u^k)$ progress. Problem 1, part (b)',...
    opt,pts,size(pts,1),status),'FontSize', 8);
xlabel('step number');
ylabel('value of objective function');

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

stairs(gradientNormTol);
%stem(levelSets,'ro');
grid;
title(format_plot_title(...
   'Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)',...
    opt,pts,size(pts,1),status),'FontSize', 8);

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

if DO_3D
    figure('Units','pixels','position',...
                                   [from_pix from_pix 400 300]);
    pix_count = pix_count+1;

    del      = 1;
    u1       = xlimits(1):del:xlimits(2);
    u2       = ylimits(1):del:ylimits(2);
```

```
    [u1,u2]  = meshgrid(u1,u2);
    z        = (11-u1-u2).^2 + (1+u1+10.*u2-u1.*u2).^2;
    h        = mesh(u1,u2,z);

    view(gca,-13.5,42);
    set(h,'LineWidth',.25,'LineStyle','-','EdgeAlpha',.5);
    shading(gca,'flat');
    hold on;

    %plot the optimal point found by Matlab
    plot3(optimalValue(1),optimalValue(2),objectiveAtOptimal,...
       'ws--', 'MarkerEdgeColor', 'r', 'MarkerFaceColor', 'r');

    %plot the optimal point found by steepest descent
    plot3(pts(end,1),pts(end,2),levelSets(end),...
       'ws--', 'MarkerEdgeColor', 'b', 'MarkerFaceColor', 'b');

    %plot the starting point
    plot3(pts(1,1),pts(1,2),levelSets(1),...
       'ws--', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'k');

    if size(pts,1)>1
        for k = 1:length(pts)-1
            %draw line between each step
          line([pts(k,1),pts(k+1,1)],[pts(k+1,2),pts(k+1,2)],...
                [levelSets(k),levelSets(k+1)],'LineWidth',1);
            drawnow;
        end
    end
    xlabel('$u_1$');ylabel('$u_2$');
    zlabel('objective function $J(u_1,u_2)$');

    title(format_plot_title(...
  '3D mesh view of the search performed. Problem 1, part (b)',...
               opt,pts,size(pts,1),status),'FontSize', 8);

    set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
end
end

%-------------------------
%Evaluate J(u) at u
function f = objectiveFunc(u)
x  = u(1);
y  = u(2);
%f  = 3 + (x - 1.5*y)^2 + (y - 2)^2;
f  = (11-x-y)^2 + (1+x+10*y-x*y)^2;
```

```
end

%---------------------
%Evaluate grad(J(u)) at u
function g = gradientFunc(u)
x  = u(1);
y  = u(2);
%g  =[2*(x-1.5*y);2*(x-1.5*y)*(-1.5)+2*(y-2)];
g  = [-2*(11-x-y)+2*(1+x+10*y-x*y)*(1-y); ...
      -2*(11-x-y)+2*(1+x+10*y-x*y)*(10-x)];
end
%---------------------------
%set title
function formatted_title = format_plot_title(main_title,opt,pts,k,status)

formatted_title = {sprintf('\\makebox[5in][c]{%s}',main_title),...
  sprintf('\\makebox[5in][c]{$u^0=[%4.3f,%4.3f]$, step [$%2.2f$], $J(u)=%3.3f$, iterations [$
  opt.u(1),opt.u(2),opt.STEP_SIZE,...
                        norm(opt.objectiveFunc(pts(k,:))),k),...
  sprintf('\\makebox[5in][c]{convergence criteria $| \\nabla(J(u)) | \\leq  %1.3f $}',...
  opt.gradientNormTol),...
  sprintf('\\makebox[5in][c]{%s}',status)};
end

%---------------------
%Evaluate Hessian(J(u)) at u
function g = hessian_func(u)
x  = u(1);
y  = u(2);
%g  = [2,-3;-3,13/2];

g  =[2*(y - 1)^2 + 2, 2*(x - 10)*(y - 1) - 20*y - 2*x + 2*x*y;
    2*(x - 10)*(y - 1) - 20*y - 2*x + 2*x*y,2*(x - 10)^2 + 2];
end
```

0.3.5 Problem 2 contour

```
function nma_HW4_problem_2_contour()
%Plots a contour of
%
%     f(u) = 100(u2-u1^2)^2 + (1-u1)^2
%
% over range  u1=-2.5..2.5
% Matlab 2015a
% by Nasser M. Abbasi


```

```
reset(0); close all; clear;
k=0;
myTitle   = '$$100(u_2 - u_1^2)^2 +(1- u_1)^2$$';
makeContour(-2.5,2.5,-2.5,2.5,[1 10 50 100 200 300 ...
                              500 1000 2000 3000],myTitle);
k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
makeContour(-1,1.5,-1,2,[0.5 1 5 15 30 50 100 200 300 400 500],...
                                                     myTitle);
k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
makeContour(0.2,1.5,-0.5,2,[0.5 2.5 5 10 20 35 50 100 200 300],...
                                                      myTitle);
k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
makeContour(0.4,1.1,0,1.5,[0.2 0.3 0.5 2.5 5 10 20 30 50 ...
                                    75 100 150 200],myTitle);
k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
makeContour(0.8,1.1,0.5,1.1,[0.1 0.2 0.3 0.5 1 2 3 ...
                                        4 8 12 20],myTitle);
k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
makeContour(0.9,1.1,0.9,1.1,[0.01 0.05 0.1 0.2 ...
                                0.3 0.5 1 1.5 2 3],myTitle);
k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');

figure;
[u1,u2,z] = makeContourData(0.3,[-2,2],[-2,2]);
surf(u1,u2,z);
colormap(hsv);
view([-156.5,42]);

hold on;
v=[10 100 200 300 500];
[C,h]    = contour(u1,u2,z,v);
clabel(C,h,v,'Fontsize',10,'interpreter','Latex','Color','red');
setMyLabels('$$u_1$$','$$u_2$$','$$J(u_1,u_2)$$',...
           {'\makebox[4in][c]{Labeled 3D over contour view}',...
                  sprintf('\\makebox[4in][c]{%s}',myTitle)})
k=k+1; saveas(gcf, sprintf('%d',k), 'pdf');
end

%------------
%helper function to set plot attributes.
function setMyLabels(varargin)

myXlabel = varargin{1};
myYlabel = varargin{2};
if nargin ==4
    myZlabel = varargin{3};
end
```

```
myTitle  = varargin{end};
h        = get(gca,'xlabel');
set(h,'string',myXlabel,'fontsize',10,'interpreter','Latex') ;

h = get(gca,'ylabel');
set(h,'string',myYlabel,'fontsize',10,'interpreter','Latex');

if nargin ==4
    h = get(gca,'zlabel');
    set(h,'string',myZlabel,'fontsize',10,'interpreter','Latex');
end

h = get(gca,'title');
set(h,'string',myTitle,'fontsize',10,'interpreter','Latex',...
    'HorizontalAlignment','center') ;

set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
end

%------------------
%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

```
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
CONTOUR_LINES_AUTO =  'fix';
%         set to 'auto', to see matlab contour lines
%         set to 'full' to see each step level set
%         set to 'limited' to see every other level
%         set to 'fix' to use pre-specificed
%         set to 'full0', to see each level, no labels

%level set for 'fix' option
vFixed = [.5 10 50 100 200 300 1000 2000 3000];
%-----------------------------------------------------
%These are the options struct used by call to
%nma_steepestDescentPoints() try [-2,.8]
```

```
opt.u               = [1.828;-1.878]; %starting guess x-coord
opt.MAX_ITER        = 10^6; %maximum iterations allowed
opt.STEP_SIZE       = -1; %step size. set to -1 for optimal step
opt.objectiveFunc   = @objectiveFunc; %see function definition
opt.gradientFunc    = @gradientFunc;  %see function definition
opt.hessian         = @hessian_func;  %see function definition
opt.gradientNormTol = 0.01; %used to determine when converged
opt.accumulate      = true;
opt.stop_on_oscillation = false;

%------------------------------------------
%data
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;

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

if DO_GUI  %check if GUI input is asked for, if so, wait for user
    plot(0,0);
    xlim(xlimits); ylim(ylimits);
    hold on;
    [x,y] = ginput(1);
    opt.u=[x;y];
end

%mark location of starting point
%t = text(0.8*opt.x,1.1*opt.y,sprintf('[%2.1f,%2.1f]',opt.x,opt.y));
%t.FontSize = 8;
%t.Color    = 'red';

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

%mark location of minimum found by fminsearch on plot
%This min, can be different that one converged to by steepest
%descent! so we also plot the converged to value found
hold on;
plot(optimalValue(1),optimalValue(2),'*r')
plot(opt.u(1),opt.u(2),'or');  %starting point
xlim(xlimits); ylim(ylimits);
%grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
if strcmp(METHOD,'steepest descent')
```

```
   [status , pts,levelSets, gradientNormTol,steps] = ...
                                          nma_steepest_descent(opt);
else
   [status,pts,levelSets, gradientNormTol,steps] = ...
                                           nma_fletcher_reeves(opt);
end

%plot the value found by steepest descent
%plot(pts(end,1),pts(end,2),'ok');

%use output from above call to make the plots
switch CONTOUR_LINES_AUTO
    case 'auto',
    [C,h] = contour(u1,u2,z); %,'ShowText','on');
    clabel(C,h,'Fontsize',8,'interpreter','Latex','Color','red');

    case 'limited',
        lev   = round(length(levelSets)/20);
        [C,h] = contour(u1,u2,z,levelSets(1:lev:end),...
                                  'Fill','off','ShowText','off');
        %clabel(C,h); %,'Fontsize',8,'interpreter',...
        %'Latex','Color','red');
    case 'full'
        [C,h] = contour(u1,u2,z,levelSets); %,'ShowText','on');
        clabel(C,h,levelSets,'Fontsize',8,...
                          'interpreter','Latex','Color','red');
    case 'full0'
        [C,h] = contour(u1,u2,z,levelSets); %,'ShowText','on');
    case 'fix'
        [C,h] = contour(u1,u2,z,vFixed);
        h.LineWidth = .1;
        h.LineColor  = [190/255 190/255 190/255];
        h.Fill='off';
        clabel(C,h,vFixed,'Fontsize',8,...
                          'interpreter','Latex','Color','blue');
end
%animate the steepest descent search
hold on;
if length(pts(:,1))>1
    filename = 'anim.gif';
    for k=1:length(pts)-1
        %draw line between each step
        %if (opt.STEP_SIZE == -1 || ...
                  %strcmp(CONTOUR_LINES_AUTO,'limited') || ...
        % strcmp(CONTOUR_LINES_AUTO,'auto')||length(pts)<100 )
        %   line([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],...
                                              %'LineWidth',1');
```

```
        %end
        plot([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],'-r');
        %plot(pts(k,1),pts(k,2),'.');
        if DO_ANIMATE
           drawnow;
           if DO_GIF
              frame = getframe(1);
              im = frame2im(frame);
              [imind,cm] = rgb2ind(im,256);
              if k ==1
                  imwrite(imind,cm,filename,'gif', 'Loopcount',0);
              else
                 if mod(k,2)==0
                    imwrite(imind,cm,filename,'gif',...
                                          'WriteMode','append');
                 end
              end
           end
        end
        if opt.STEP_SIZE==-1,
           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

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

%-------------------------
%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

```
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);

format long g;
tic;
[status,pts,levelSets, gradientNormTol,steps] = ...
                                     nma_steepest_descent(opt);
time_used = toc;
fprintf('\nCPU time %3.6f\n',time_used);

switch status
    case 0, status = ...
  'successfull completion. Converged before maximum iterations';
    case 1, status = ...
'failed to converge before maximum iterations due to oscillation';
    case 2, status = ...
              'failed to converge before maximum iterations';
end

fprintf('%s\n',status);

figure();
stem(levelSets,'.'); title('J(u)');
figure();
stem(steps,'.'); title('step size');
format short;
fprintf('Number of coordinates used %d\n',size(opt.u,1));
fprintf('optimal point found is\n'); disp(pts(end,:));
if opt.accumulate
    fprintf('\nNumber of steps used [%d]',length(steps));
else
    fprintf('\nNumber of steps used [%d]',steps);
end
fprintf('\nJ(u) at optimal [%3.6f]',levelSets(end));
fprintf('\n**** done *******\n');


end

%-------------------------
%Evaluate J(u) at u
function f = objectiveFunc(u)
u=u(:);
N = size(u,1);
f = 0;
```

```
for i = 1:N-1
    f  = f + 100*(u(i+1)-u(i)^2)^2 + (1-u(i))^2;
end
end

%---------------------
%Evaluate grad(J(u)) at u
function g = gradientFunc(u)
u = u(:);
N = size(u,1);
g = zeros(N,1);
for i = 1:N
    if i==1 || i==N
        if i==1
            g(i)=-400*(u(i+1)-u(i)^2)*u(i) - 2*(1-u(i));
        else
            g(i)=200*(u(i)-u(i-1)^2);
        end
    else
        g(i) = 200*(u(i)-u(i-1)^2)-...
                           400*(u(i+1)-u(i)^2)*u(i)-2*(1-u(i));
    end
end
end
```

0.3.8 Problem 2 part b CPU time program

```
function nma_HW4_problem_2_part_b_CPU()
%Does CPU testing on problem 2 by calling
%nma_HW4_problem_2_part_b() on larger and larger N and
%recording the CPU time used.
%
% ECE 719, Spring 2016
% Matlab 2015a
%Nasser M. Abbasi
clear; close all;

opt.STEP_SIZE       = 0.01; %step size. set to -1 to use optimal
save_file           = 'fixed.mat';
N                   = 10:20:1000;
data                = zeros(length(N),4);
opt.MAX_ITER        = 1*10^6; %maximum iterations allowed

opt.objectiveFunc   = @objectiveFunc; %see function definition
opt.gradientFunc    = @gradientFunc;  %see function definition
opt.gradientNormTol = 0.0001; %used to determine when converged
opt.accumulate      = false;
```

```
for i=1:length(N)
    opt.u = repmat(-2,N(i),1);    %starting guess x-coordinate
    tic;
    [status,~,levelSets, ~,number_of_steps_used] = ...
                                      nma_steepest_descent(opt);
    time_used = toc;
    switch status
        case 0, status = ...
'successfull completion. Converged before maximum iterations';
        case 1, status = ...
'failed to converge before maximum iterations due to oscillation';
        case 2, status = ...
               'failed to converge before maximum iterations';
    end
    fprintf('%s\n',status);

    data(i,1) = N(i);
    data(i,2) = time_used;
    data(i,3) = levelSets;
    data(i,4) = number_of_steps_used;
    fprintf('\n****Number of coordinates used %d\n',...
                                                 size(opt.u,1));
    fprintf('\nCPU time %3.6f\n',time_used);
    fprintf('\nJ(u) at optimal [%3.6f]\n',levelSets(end));
end

close all;
reset(0);
set(groot,'defaulttextinterpreter','Latex');
set(groot, 'defaultAxesTickLabelInterpreter','Latex');
set(groot, 'defaultLegendInterpreter','Latex');

figure();
plot(N,data(:,2),'ro',N,data(:,2),'-');
title('CPU time as N changes for fix step steepest descent');
xlabel('N'); ylabel('CPU time (sec)');
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

save(save_file,'data');

%-----------------------------
figure;
load('optimal');
optimal=data;
load('fixed')
fixed=data;
```

```
plot(optimal(:,1),optimal(:,2),'k.-')
hold on;
plot(fixed(:,1),fixed(:,2),'r.-')
title('Comparing CPU time, using optimal vs. fixed step')
xlabel('N, the number of coordinates');
ylabel('CPU time in seconds');
grid
end

%-------------------------
%Evaluate J(u) at u
function f = objectiveFunc(u)
u=u(:);
N = size(u,1);
f = 0;
for i = 1:N-1
    f  = f + 100*(u(i+1)-u(i)^2)^2 + (1-u(i))^2;
end
end

%---------------------
%Evaluate grad(J(u)) at u
function g = gradientFunc(u)
u = u(:);
N = size(u,1);
g = zeros(N,1);
for i = 1:N
    if i==1 || i==N
        if i==1
            g(i)=-400*(u(i+1)-u(i)^2)*u(i) - 2*(1-u(i));
        else
            g(i)=200*(u(i)-u(i-1)^2);
        end
    else
        g(i) = 200*(u(i)-u(i-1)^2)-400*(u(i+1)- ...
                                        u(i)^2)*u(i)-2*(1-u(i));
    end
end
end
```