University Course

ECE 719
Optimal Systems

University of Wisconsin, Madison
Spring 2016

My Class Notes

Nasser M. Abbasi

Spring 2016

Contents

Chapter 1

Introduction

Instructor web page: Professor B Ross Barmish

1.1 syllabus

Barmish

ECE 719 - Handout Syllabus

Course Description

Audience: This course is intended for graduate students interested in the systems sciences. The course concentrates on finite-dimensional parameter optimization methods with examples in the context of static and dynamic systems. Coverage of the material will be suitable for students both inside and outside ECE.

Prerequisites: ECE 334 or consent of instructor.

Topics: Preliminaries; Formulation of parameter optimization problems for static and dynamic systems; Common sense optimization; Existence and uniqueness; Convex optimization; Optimal gain control and other dynamic systems problems; Line search methods; Steepest descent, Newton-Raphson and conjugate direction algorithms; Convergence issues; Linear Programming with control applications; Discrete-time dynamic programming; Optimization of Linear Quadratic Regulators; Identification and Kalman filtering in an optimization context.

Lectures: Professor B. R. Barmish

Grading: The grade will be based on three midterm tests @ 25%, homework and a special problem (25%). The instructor may exercise discretion up to 10% in each of the grading categories.

1.2 References

Some Initial References

D. P. Bertsekas, *Convex Analysis and Optimization*, Athena Scientific, Belmont, 2003.

S. Boyd and C. Barratt, *Linear Controller Design, Limits of Performance*, Prentice-Hall, New York, 1991.

S. Boyd and L. Vandenberghe, *Introduction to Convex Optimization With Engineering Applications*, Stanford University, 1999.

E. K. P. Chong and S. H. Zak, *Introduction to Optimization*, Wiley-Interscience, New York, 2001.

R. Fletcher, *Practical Methods of Optimization*, John Wiley & Sons, New York, 1980.

D. G. Luenberger, *Optimization by Vector Space Methods*, John Wiley & Sons, New York, 1968.

J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer-Verlag, New York, 1999.

R. T. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, 1970.

1.3 Organization

Barmish

ECE 719 – Handout Organization

- *Lectures*

B. R. Barmish
Office: 3613 Engineering Hall
E-mail: barmish@engr.wisc.edu
Office Hours: Wednesday 1:00-3:00 PM

- *Textbook*

None: Lecture Notes and Readings

- *Grading*

Per Handout Syllabus

- *Test Scheduling Information*

First Midterm Test: Thursday, February 18, 2016
Second Midterm Test: Thursday, March 31, 2016
Third Midterm Test: Thursday, May 5, 2016
Midterms held in class period unless rescheduled; see below.

- *Course Schedule Information*

No Office Hours during Spring Recess and Wednesday, April 6, 2016
Reserved Time @ 6 PM on Wednesdays February 17, March 30 and May 4.
Times above for Make-up Lectures or Midterm Test Rescheduling.

- *Discussion of Prerequisites*

- *Discussion of Matlab*

- *Discussion of Homework and Grading*

1.4 Cardinal rules

Barmish

ECE 719 – Handout Cardinal
Use of References and the Cardinal Rule

Some Comments on the References: There are a number of good books on the reference list for ECE 719. I recommend the book by Chong and Zak in that it covers many of the topics in this course and is well written. The books by Rockafellar and Bertsekas tell you everything you ever wanted to know about convexity. Rockafellar covers the material in greater depth but is more time consuming to read. This course covers finite-dimensional optimization. The reader who is interested in a lucid presentation of infinite-dimensional analogues of many of these results should consult the book by Luenberger. The book by Fletcher provides a concise summary of classical numerical algorithms used in iterative optimization. For the uninitiated, I would recommend the well-written textbook by Nocedal and Wright on numerical optimization. For the reader interested in optimization with a control-theoretic slant, the book by Boyd and Baratt is nice to read.

The Cardinal Rule: In homework assignments, students should only rely on results given in class, basic mathematical facts, assigned readings and results developed in previous homework sets. That is, one should not pull results out of reference books on optimization and cite them in order to attain the result being sought. To this end, the course instructor will serve to interpret what mathematical facts qualify as "basic."

Chapter 2

Class notes

Local contents

Summary table

These are my class lecture notes written from the lectures of ECE 719 optimal systems course given by Professor B. Ross Barmish at University of Wisconsin, Madison in Spring 2016. Any errors in these notes, then all blames to me and not to the instructor.

#	date	event	Topic
1	Tuesday, 1/19/2016	First class	Introduction, handouts
2	Thursday, 1/21/2016	Multilinear	Tractable, Farming example, Multilinear, det(M) example
3	Tuesday, 1/26/2016	Real analysis	Reader on Farming, level sets, Pareto, Existence of optimal, real analysis, B-W, sub-sequences
4	Thursday, 1/28/2016	Quadratic forms	Coercivity, classical existence theorem, Quadratic forms, starting convex sets
5	Tuesday, 2/2/2016	Mixtures	Polytope, Mixtures, Extreme points, Started convex functions, maximum of collection of convex functions is convex function. epi graph.
6	Thursday, 2/4/2016	Convex	Convex functions, properties, indexed collection, Hessian theorem: $J(u)$ is convex iff, the Hessian is positive semi-definite.
7	Tuesday, 2/9/2016	Hessian	Bridging lemma. Proof of Hessian theorem for $n > 1$. Strong local minimum theorem.
8	Thursday, 2/11/2016	optimal gain	optimal gain problem.
9	Tuesday, 2/16/2016	Gradient	Gradient based optimization
10	Thursday, 2/18/2016	Exam 1	
11	Tuesday, 2/23/2016	Steepest	Handout amplifier. Finish Steepest descent. Start on Newton-Raphson
12	Thursday, 2/26/2016	Convergence	Handout Newton. Derivation of step size for Newton-Raphson. Super-linear convergence.
13	Tuesday, 3/1/2016	Gradient direction	Gradient direction, quadratic convergence, mutually conjugate vectors, quadratic convergence theorem
14	Thursday, 3/3/2016	LP	Starting linear programming
15	Tuesday, 3/8/2016	LP	patrol sector problem, mechanics of LP
16	Thursday, 3/10/2016	LP	Squeeze method, basic and feasible solutions
17	Tuesday, 3/15/2016	LP simplex	optimality theorem, extreme point theorem, unit simplex, mechanism of simplex
18	Thursday, 3/17/2016	Complete LP	Tableau method with optimality
19	Tuesday, 3/22/2016		No class. Thanks giving
20	Thursday, 3/24/2016		No class. Thanks giving
21	Tuesday, 3/29/2016	LP in control	Example using LP in control, minimum fuel. Ending LP, starting dynamic programming, review.
22	Thursday, 3/31/2016	Exam 2	
23	Tuesday, 4/5/2016	Dynamic programming	First example in dynamic programming, trip from NY to San Francisco. Toll fee optimization.
24	Thursday, 4/7/2016	No class	
25	Tuesday, 4/12/2016	D.P. and special problem	describe special problem. Dynamic programming.

26	Thursday, 4/14/2016	D.P.	LQR example. Oil and real estate example
27	Tuesday, 4/19/2016	D.P. and LQR	LQR using D.P., long example
28	Thursday, 4/21/2016	D.P. floor function	D.P. examples for variation of dynamic programming
29	Tuesday, 4/26/2016	Steady state	Finish Floor problem. Start on steady state, iterative method
30	Thursday, 4/28/2016	Steady state, Riccati	Closed form, guess method, infinite time LQR, Riccati.
31	Tuesday, 5/3/2016	Review of course	Special problem review, class review and prep for final exam
32	Thursday, 5/5/2016	Final exam	Exam

2.1 Lecture 1. Tuesday, January 19, 2016

This course is on finite dimensional optimization, which means having finite number of parameters. We now went over the syllabus. Here is a summary:

1. Convex sets and functions
2. How to certify your solution?
3. Linear programming.
4. Dynamic programming (at end of course)
5. Three exams and a final special project/problem.

Homeworks will be graded using E,S,U grades. Most will get S, few will get E. Matlab will be used.

Cardinal rule Develop an answer using given material in class only. Can use other basic things like Laplace transform, etc...

A wise person once said: "Fundamental difficulties are invariant under reformulation".

2.1.1 Objective functions, constraints and variables

In a problem, identify the objective function, constraints and variables. Optimization problems from different fields can be formulated into a common framework for solving using optimization methods.

Ingredients in this case: Set $U \subseteq \Re^n$, the constraint set. Problem has n parameters (the decision variables). Therefore $u \in U$. One dimensional problem has $n = 1$. An example is to find optimal resistor value where $U = 100 \cdots 200$ Ohms. For $n = 2$, an example is to find two resistors $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ with $100 \leq u_1 \leq 200$ and $300 \leq u_2 \leq 400$.

Reader Often we describe U graphically in $\Re^n$. Typically for only $n = 1, 2, 3$. Do this for the above example.

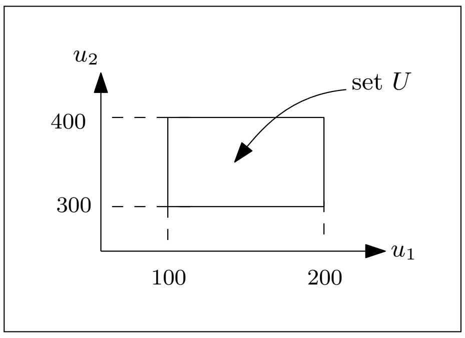


Figure 2.1: Set U with constraints

Reader design a bandpass filter with passband from ω_1 to ω_2 with $\omega_1, \omega_2 > 0$. Describe graphically the set U.

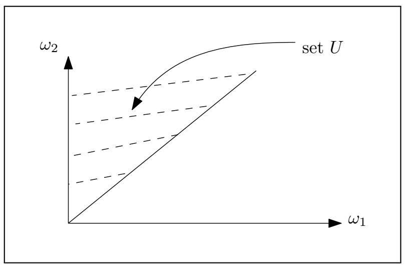


Figure 2.2: Lecture one, set U second diagram

Reader Often U is sphere in $\Re^n$ described by $\sum_{i=0}^{n} \left(u_i - u_{i,0}\right)^2 \leq R$ where $u_{i,0}$ are coordinates of center of sphere.

Reader Suppose we are designing an input voltage $u(t)$ on $t \in [0,1]$ such that $\int_0^1 u^2(t)\,dt \leq 5$. This does not fit in above framework. This is function space problem.

An important class of U: A generalization of rectangle in 2D to hypercube in $\Re^n$ with $\left|u_i - u_{i,0}\right| \leq r_i$ for $i = 1 \cdots n$.

Reader For $n = 3$ and $u_0 = \begin{bmatrix} u_{1,0} \\ u_{2,0} \\ u_{3,0} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ and $r_1 = 1, r_2 = 2, r_3 = 3$, then sketch U.

2.1.2 Constrained and Unconstrained problems

Unconstrained problem is one when we say $U = \Re^n$. There are no constraints on U. These are easier to solve than constrained problems. In practice, there will always be constraints. U is sometimes called the set of decision variables, or also called the input. The other criteria, is the objective function $J(u)$, or more formally $J : \Re^n \to \Re$. The objective function $J(u)$ is obtained from your goals.

Example we want to go from A to B in least time. Often the selection of $J(u)$ is not straight forward. Selection of $J(u)$ can be difficult in areas such as social or environmental science.

In stock markets, $J(u)$ is given and we just use it. Example is $J(u) = u^2 + 6u + e^{-u}$ in $\Re^n$. Often we test the algorithm first in $\Re^1$ or $\Re^2$ before going to higher dimensions. In $\Re^4$ an example is

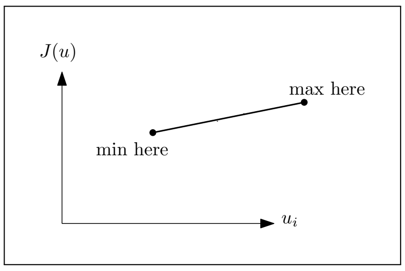
$$J(u) = u_1u_2 - 3u_1u_3u_4 + 6u_2u_3 - 6u_1 - 4u_2 - u_3u_4 + 12$$

This has 16 vertices.

Reader Can you find max $J(u)$ subject to U described by $|u_i| \leq i$ using common sense?

Answer It will be on the vertices of hypercube.

In VLSI, with hundreds of resistors, there are 2^n vertices, so the problem becomes computationally harder to solve very quickly. To proof that the optimal value is at a vertex, the idea is to freeze all other variables except for one at a time. This gives a straight line in the free variable. Hence the optimal is at ends of the line.



The main case problem: Find u^* that minimizes $J(u)$ with the constraints $u \in U$. u^* might not exist. When it exists, then

$$J(u^*) = J^* = \min_{u \in U} J(u)$$

When we begin, we do not know if u^* exist. We write

$$J(u^*) = \inf_{u \in U} J(u)$$

Example $U = \Re^n$, $J(u) = e^{-u}$. So u^* do not exist. We do not allow u^* to take values $\pm\infty$. But we allow $J(u^*)$ itself to become $\pm\infty$. For example, if $J(u) = u^2$ then $u^* = 0$.

2.2 Lecture 2. Thursday, January 21, 2016

2.2.1 Existence of optimal solution, explicit and implicit $J(u)$

Tractability. $U \subseteq \Re^n$, the decision variables. $J : \Re^n \to \Re$. Problem: find u^* such that $J(u^*) = \inf_{u\in U} J(u)$.

u^*, when it exists, is called the optimal element. We do not allow $u^* = \pm\infty$, but allow $J(u^*) = \pm\infty$. For example. $J(u) = \frac{1}{u}$ on $U = (0,\infty)$. We say $\sup_{u\in U} J(u) = J^* = 0$ but $u^* = \infty$ do not exist. Hence J^* is a limiting supremum value. Another example is $J(u) = -u$ on $\Re$. Therefore $J^* = -\infty$ but u^* do not exist.

Reader: We can consider $\max_{u\in U} J(u) \equiv \sup_{u\in U} J(u)$. Note that

$$\max_{u\in U} J(u) = -\min_{u\in U} J(u)$$

But $u^*_{\min} \neq u^*_{\max}$.

Some problems will be tractable and some are not. Some problems will not have defined algorithms and some might not have certified algorithms. Some problems have algorithms but are not tractable. NP hard problems can be either tractable or not tractable. What about stochastic problems? If u is random variable, we can not write $J(u)$. But instead we work with $\tilde{J}(u) = E(J(u))$ where E is expectation. So now $\tilde{J}(u)$ fits in the frameworks we used earlier.

We also need to make distinction between explicit and implicit $J(u)$. When we write $J(u) = u_1^2 + e^{u_2}\cos u_1 + u_2$, then $J(u)$ here is explicit. But if we have a circuit as below, where $J(u) = V_{out}$, then here $J(u)$ is implicit, since we have to solve the complicated RLC circuit to find $J(u)$, so we implicitly assume $J(u)$ exists.

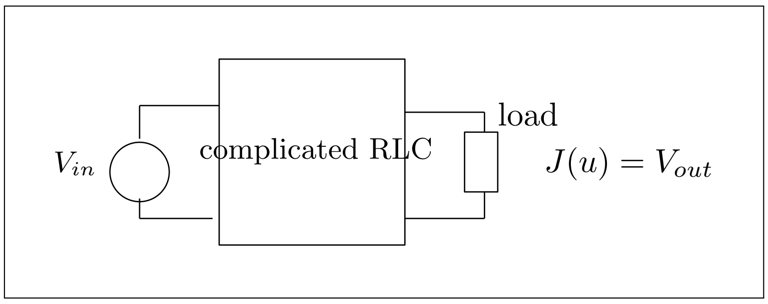


Figure 2.3: Complicated RLC

2.2.2 Farming problem

First detailed example Optimal farming example. Let $y(k)$ be the annual crop value where k is the year. This is the end of year value of the crop. At end of year, we use some of this to invest and the rest we keep as profit. Let $u(k) \in [0,1]$ be the fraction of $y(k)$ invested back. Therefore $(1-u(k))$ is the fraction of $y(k)$ which is taken out as profit. Dynamics of the problem are: $y(k+1) = y(k)$ if we invest nothing (i.e. $u = 0$). But if we invest, then

$$y(k+1) = y(k) + \omega(k) \overbrace{u(k)\,y(k)}^{\text{amount invested}}$$

Where $\omega(k)$ is an independent and identically distributed (i.i.d.) random variable which depends on the weather and other variables. Let $E(\omega(k)) = \varpi$. We have N years planning horizon. What about $J(u)$? If we model things as a convex problem, we can solve it, but if we do not, it becomes hard to solve.

$$J(u) = E\left(y(k) + \sum_{k=0}^{N-1} y(k)(1-u(k))\right) \tag{1}$$

The set U here is $\{u(0), u(1), \cdots, u(N-1)\}$. In this example $J(u)$ is implicit. We need to make $J(u)$ explicit. Let us now calculate $J(u)$ for $N = 2$

$$\begin{aligned} y_1 &= y_0 + \omega_0 u_0 y_0 \\ &= y_0(1+\omega_0 u_0) \end{aligned}$$

In class, we assumed that $y_0 = 1$. But will keep it here as Y, which is the initial conditions.

$$y_1 = Y(1 + \omega_0 u_0) \tag{2}$$

Now for the second year, we have

$$\begin{aligned} y_2 &= y_1 + \omega_1 u_1 y_1 \\ &= y_1 (1 + \omega_1 u_1) \end{aligned}$$

Substituting (2) into the above gives

$$\begin{aligned} y_2 &= Y(1 + \omega_0 u_0)(1 + \omega_1 u_1) \\ &= Y + Y\omega_0 u_0 + Y\omega_1 u_1 + Y\omega_0\omega_1 u_0 u_1 \end{aligned}$$

Therefore from (1) we have for $N = 2$

$$\begin{aligned} J(u) &= E\left(y_2 + \sum_{k=0}^{1} y_k (1 - u_k)\right) \\ &= E\left(\overbrace{Y + Y\omega_0 u_0 + Y\omega_1 u_1 + Y\omega_0\omega_1 u_0 u_1}^{y}_{2} + \left[Y(1 - u_0) + y_1 (1 - u_1)\right]\right) \\ &= E\left(Y + Y\omega_0 u_0 + Y\omega_1 u_1 + Y\omega_0\omega_1 u_0 u_1 + \left[Y(1 - u_0) + Y(1 + \omega_0 u_0)(1 - u_1)\right]\right) \\ &= E\left(Y + Y\omega_0 u_0 + Y\omega_1 u_1 + Y\omega_0\omega_1 u_0 u_1 + Y - Yu_0 + Y - Yu_1 + Y\omega_0 u_0 - Y\omega_0 u_0 u_1\right) \\ &= Y\left(3 + \varpi u_0 + \varpi u_1 + \varpi^2 u_0 u_1 - u_0 - u_1 + \varpi u_0 - \varpi u_0 u_1\right) \\ &= Y\left(3 + 2\varpi u_0 + u_1(\varpi - 1) + \varpi^2 u_0 u_1 - u_0 - \varpi u_0 u_1\right) \\ &= Y\left(3 + u_0(2\varpi - 1) + u_1(\varpi - 1) + u_0 u_1\left(\varpi^2 - \varpi\right)\right) \end{aligned}$$

Reader Use syms to find $J(u)$ for $N = 4$.

If we want maximum profit, then common sense says that ϖ should be large (good weather). Then $u^* = (1,1)$. This says $u_0 = 1$ and $u_1 = 1$. In other words, we invest everything back into the crop each year. The above was obtained by maximizing term by term since all terms are positive. If $\varpi < \frac{1}{2}$ (bad weather), then $u^* = (0,0)$ and all coefficients are negative.

In the above, $J(u)$ is multilinear function in u_0, u_1. If all u_k are held constant except for one at a time, then $J(u)$ becomes linear in the free parameter. In HW 1, we need to proof that extreme point of a multilinear function is at a vertex. So for this problem, the possible solutions are $(0,0), (0,1), (1,0), (1,1)$.

Reader Is there a value of ϖ that gives $(1,0)$ and $(0,1)$?

For arbitrary N there are 2^N vertices in hypercube for a multilinear $J(u)$ where $u \in \Re^n$.So for large n, a multilinear problem is much harder to solve than a convex optimization problem. Even simple problems demonstrate intractability. Let $M = N \times N$ matrix. Let each every m_{ij} be known within bounds $m_{ij}^- \le m_{ij} \le m_{ij}^+$. The question is: What are the bounds on the determinant of matrix M?

Reader ponder this question in the context of multilinear problem. To determine this for small n in Matlab, here is some code for $n = 4$ (uses `allcomb` which is a file exchange file)

```
a = repmat({[-1 1]},16,1);
v = allcomb(a{:});
r = arrayfun(@(i) det(reshape(v(i,:),4,4)),1:size(v,1));
max(r)
min(r)
```

.

Running the above gives

```
ans =
    16


ans =
   -16
```

2.3 Lecture 3. Tuesday, January 26, 2016

2.3.1 Multilinear functions, level sets, contours

Today will be on multilinear functions, then back to infimum of $J(u)$. Then we will go over real analysis to see when a $\min$ is reached. We need to find first if there exists a minimum before starting to search for it. For the farming problem we looked at, say $\varpi = 2$. We want to look at contours in $\Re^2$ and $\Re^3$. Try first in $\Re^2$.

Reader Obtain level sets, contour lines, defined as $\Lambda_\gamma = \left\{u : J(u) = \gamma\right\}$. In words, contour line is curve of equal values of $J(u)$. For farming problem, show contour lines of $J(u)$ looks like

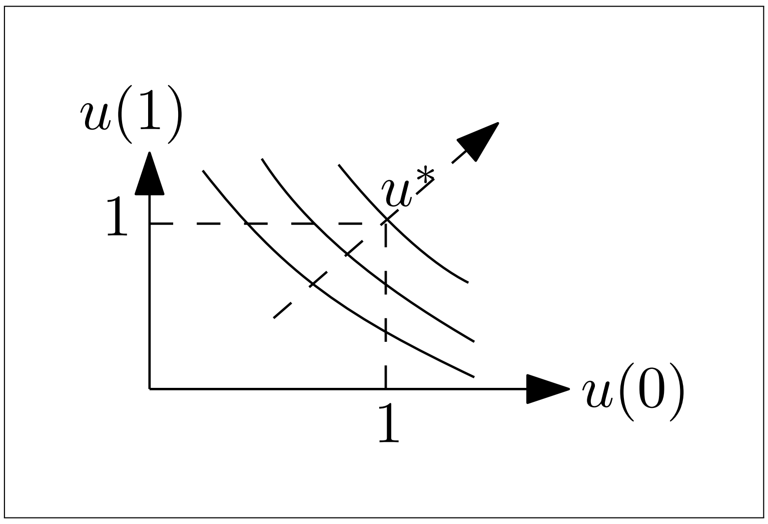


Figure 2.4: Level sets

For $\varpi = 2$, and using results from last lecture, where we had

$$J(u) = Y\left(3 + u_0(2\varpi - 1) + u_1(\varpi - 1) + u_0 u_1\left(\varpi^2 - \varpi\right)\right)$$

Where $Y = y(0) = 1$, the above becomes

$$J(u) = 3 + u_0(2\varpi - 1) + u_1(\varpi - 1) + u_0 u_1\left(\varpi^2 - \varpi\right) \tag{1}$$

Reader Use Matlab syms to obtain $J(u)$ for any N

Answer Below is a function which generates $J(u)$ for any N. Function called `nma_farm(N,y0)`, takes in N, which is how many years and $y(0)$, the initial value. $\omega 0$ below is ϖ, the mean of the ω random variable).

```
function nma_farm(N,initial)
%reader anwer for farming problem, lecture 1/21/16, ECE 719
%N is number of years
%Initial, is y(0). See class notes for ECE 719 for
%description of the problem. Second lecture.
%Nasser M. Abbasi

y = []; w=[]; u=[];
idx = [];
syms y(idx) w(idx) u(idx)
tot = 0;

for k = 0:N-1  %main loop
    tot = tot + Y(k)*(1-u(k));
end
J = Y(N)+ tot;

for i = 0:N  %use mean
    J = subs(J,w(i),'w0');
end

J=subs(J,y(0),initial);
```

```
expand(J)

%--------------------------
%   recursive function internal function
    function r = Y(k)
        if k ==0
            r = y(0);
        else
            r = Y(k-1)*(1+w(k-1)*u(k-1));
        end
    end

end
```

Here are example run outputs

```
>> nma_farm(2,1)
2*w0*u(0) - u(1) - u(0) + w0*u(1) + w0^2*u(0)*u(1) - w0*u(0)*u(1) + 3

>> nma_farm(4,1)
4*w0*u(0) - u(1) - u(2) - u(3) - u(0) + 3*w0*u(1) + 2*w0*u(2)
+ w0*u(3) + 3*w0^2*u(0)*u(1) + 2*w0^2*u(0)*u(2) + w0^2*u(0)*u(3)
+ 2*w0^2*u(1)*u(2) + w0^2*u(1)*u(3) + w0^2*u(2)*u(3) -
w0*u(0)*u(1) - w0*u(0)*u(2) - w0*u(0)*u(3) - w0*u(1)*u(2) -
w0*u(1)*u(3) - w0*u(2)*u(3) - w0^2*u(0)*u(1)*u(2) -
w0^2*u(0)*u(1)*u(3) + 2*w0^3*u(0)*u(1)*u(2) - w0^2*u(0)*u(2)*u(3)
+ w0^3*u(0)*u(1)*u(3) - w0^2*u(1)*u(2)*u(3) + w0^3*u(0)*u(2)*u(3)
+ w0^3*u(1)*u(2)*u(3) - w0^3*u(0)*u(1)*u(2)*u(3) +
w0^4*u(0)*u(1)*u(2)*u(3) + 5

>> nma_farm(5,1)

5*w0*u(0) - u(1) - u(2) - u(3) - u(4) - u(0) + 4*w0*u(1)
+ 3*w0*u(2) + 2*w0*u(3) + w0*u(4) + 4*w0^2*u(0)*u(1) +
3*w0^2*u(0)*u(2) + 2*w0^2*u(0)*u(3) + 3*w0^2*u(1)*u(2) +
w0^2*u(0)*u(4) + 2*w0^2*u(1)*u(3) + w0^2*u(1)*u(4) +
2*w0^2*u(2)*u(3) + w0^2*u(2)*u(4) + w0^2*u(3)*u(4) - w0*u(0)*u(1)
- w0*u(0)*u(2) - w0*u(0)*u(3) - w0*u(1)*u(2) - w0*u(0)*u(4)
- w0*u(1)*u(3) - w0*u(1)*u(4) - w0*u(2)*u(3) - w0*u(2)*u(4)
- w0*u(3)*u(4) - w0^2*u(0)*u(1)*u(2) - w0^2*u(0)*u(1)*u(3) +
3*w0^3*u(0)*u(1)*u(2) - w0^2*u(0)*u(1)*u(4) - w0^2*u(0)*u(2)*u(3)
+ 2*w0^3*u(0)*u(1)*u(3) - w0^2*u(0)*u(2)*u(4) - w0^2*u(1)*u(2)*u(3)
+ w0^3*u(0)*u(1)*u(4) + 2*w0^3*u(0)*u(2)*u(3) - w0^2*u(0)*u(3)*u(4)
- w0^2*u(1)*u(2)*u(4) + w0^3*u(0)*u(2)*u(4) + 2*w0^3*u(1)*u(2)*u(3)
- w0^2*u(1)*u(3)*u(4) + w0^3*u(0)*u(3)*u(4) + w0^3*u(1)*u(2)*u(4)
- w0^2*u(2)*u(3)*u(4) + w0^3*u(1)*u(3)*u(4) + w0^3*u(2)*u(3)*u(4)
- w0^3*u(0)*u(1)*u(2)*u(3) - w0^3*u(0)*u(1)*u(2)*u(4)
+ 2*w0^4*u(0)*u(1)*u(2)*u(3) - w0^3*u(0)*u(1)*u(3)*u(4)
+ w0^4*u(0)*u(1)*u(2)*u(4) - w0^3*u(0)*u(2)*u(3)*u(4)
+ w0^4*u(0)*u(1)*u(3)*u(4) - w0^3*u(1)*u(2)*u(3)*u(4) +
w0^4*u(0)*u(2)*u(3)*u(4) + w0^4*u(1)*u(2)*u(3)*u(4)
- w0^4*u(0)*u(1)*u(2)*u(3)*u(4) + w0^5*u(0)*u(1)*u(2)*u(3)*u(4) + 6
```

At $u^* = (u_0, u_1) = (1,1)$ then (1) becomes

$$\begin{aligned} J^* &= 3 + (4-1) + (2-1) + (4-2) \\ &= 9 \end{aligned}$$

Reader Look at $J(u)$ in $\Re^3$ for the farming problem. Today's topic: When does the optimal element u^* exist?

2.3.2 Pareto optimality

When we have more than one objective function, $J_i(u) : U \subseteq \Re^n \to \Re$, for $i = 1 \cdots m$. We call $u^* \in U$ a Pareto optimal, if the following holds: Given any other $u \in U$, we can not have the following two relations be true at the same time:

1. $J_i(u) \leq J_i(u^*), i = 1 \cdots m$
2. $J_k(u) < J_k(u^*)$ for some k.

This is related to efficiency in economics. So something that is not Pareto optimal, can be eliminated.

Reader Say $U = (0, \infty)$, $J_1(u) = u + 1, J_2(u) = \frac{1}{u+1}$, describe the set of Pareto optimal solutions.

Reader By creating $J(u) = \alpha_1 J_1(u) + \alpha_2 J_2(u)$, with $\alpha_i \geq 0$, show the solution (optimal $J(u)$), depends on α_i which reflects different utility functions.

2.3.3 compact and bounded sets, open and closed sets

Now we will talk about existence of optimal element u^*. We will always assume that $J(u)$ is continuous function in $u \in U$. Two different conditions on the set U can be made

1. Is the set compact?
2. Is the set Unbounded?

In both cases, we still want conditions on $J(u)$ itself as well. We begin with the definition of

$$\boxed{J^* = \inf_{u \in U} J(u)}$$

Remember that we do not allow $u = \pm\infty$, but J^* can be $\pm\infty$.

Example $J(u) = -u$ on $\Re$. Then $J^* = -\infty$, but there is no optimal element $u^* = -\infty$. But it is always true that there is a sequence $\left\{u^k\right\}_{k=1}^{\infty} = u^k$ such that $J\left(u^k\right) \to J^*$ even though there is no u^* element. We write

$$\lim_{k \to \infty} J\left(u^k\right) = J^*$$

A bad set U is often one which is not closed. Example is of a gas pedal which when pressed to the floor will cause the car to malfunction, but the goal is to obtain the shortest travel time, which will require one to press the gas pedal to the floor in order to obtain the highest car speed.

This shows that there is no optimal u^* but we can get as close to it as we want. This is an example of a set U that is not closed on one end. We always prefer to work with closed sets. An open set is one such as $U = (0, \frac{\pi}{4}]$, and a closed set is one such as $U = \left[0, \frac{\pi}{4}\right]$.

Definition of continuity If $u^k \to u_0$ then $J\left(u^k\right) \to J(u_0)$, we write $J\left(u^0\right) = \lim_{k \to \infty} J\left(u^k\right)$. This is for every sequence u^k.

Equivalent definition: There is continuity at u^0 if the following holds: Given any $\varepsilon \geq 0$ there exists δ such that $\left|J(u) - J\left(u^0\right)\right| < \varepsilon$, whenever $\left|u - u^0\right| < \delta$.

Closed sets Includes all its limit points. Examples:

1. [0,1] is closed set.
2. [0,1) not closed. Because we can approach 1 as close as we want, but never reach it.
3. $[0, \infty)$ is closed. Since it includes all its limit points. Remember we are not allowed to use $u^* = \infty$.
4. $\Re$ is closed and open at same time.

A set can be both open and closed. $\Re$ is such a set. To show a set is closed, we need to show that the limit of any sequence is also in the set.

The intersection of closed sets remains closed, but the union can be an open set. This is important in optimization. If U_i represent constraint sets, then the intersection of the constraints remain a closed set. Closed sets also contain its boundaries.

Now we will talk about boundness of set. A set is bounded if we can put it inside a sphere of some radius. We always use the Euclidean norm. If a set is bounded, using one norm, then it is bounded in all other definitions of norms. There is more than one definition of a norm. But we will use Euclidean norm.

We like to work with compact sets. A Compact set is one which is both bounded and closed. These are the best for optimization.

2.3.4 B-W (The Bolzano-Weierstrass) theorem

Each bounded sequence in $\Re^n$ has a convergent sub-sequence. This is useful, since it says if the sequence is bounded, then we can always find at least one sub-sequence in it, which is convergent. For example $u^k = \cos k$. This does not converge, but is has a subsequence in it which does converge. The same for $u^k = (-1)^k$.

2.4 Lecture 4. Thursday, January 28, 2016

2.4.1 Existence of optimal solutions

We will spend few minutes to review existence of optimal solutions then we will talk about computability and convexity. We know now what $J(u)$ being continuous means. Closed sets are very important for well posed optimization problems. Typical closed set is $u \leq k$, where k is constant. If the function $J(u)$ where $u \in U$, is a continuous function, then the set U must be closed.

Reader Give a proof.

We talked about U being closed and bounded (i.e. compact). Compact sets are best for optimization. We talked about sequence $\left\{u^k\right\}_{k=1}^{\infty} = u^k$ and a subsequence in this sequence $\left\{u^{k_i}\right\}_{i=1}^{\infty} = u^{k_i}$. If u^k converges to u^* then we say $\lim_{k\to\infty} \left\|u^k - u^*\right\| = 0$. Finally, we talked about Bolzano-Weierstrass theorem.

2.4.2 Classical existence theorem

Suppose $J : U \to \Re$ is continuous and assume U is compact (i.e. bounded and closed) and non-empty, then there exists an optimal element $u^* \in U$ such that $J(u^*) = J^* = \min_{u\in U} J(u)$. This does not say that u^* is unique. Just that it exists.

Proof Let $u^k \in U$ be a sequence such that $J\left(u^k\right) \to J^*$, then by Bolzano-Weierstrass u^{k_i} be a convergent subsequence with limit $u^* \in U$.

Reader Show that $J\left(u^{k_i}\right)$ also converges to J^*. Note: $J\left(u^{k_i}\right)$ is sequence of real numbers, which converges to a real number J^*.

Example $u^k = (-1)^k$. Let $J(u) = e^{-u^2}$ then $J(u)$ converges, but u^k does not. Hence we need to look for subsequence u^{k_i} in u^k. Now, by continuity, $J(u^*) = \lim_{i\to\infty} J\left(u^{k_i}\right) = J^*$.

There are many problems where the set is open, as in unconstrained problems. These are called open cone problems.

2.4.3 Coercive functions and Coercivity theorem

Coercive function Suppose $U \subset \Re^n$ and $J : U \to \Re$. We say that J is positive coercive if

$$\boxed{\lim_{\|u\|\to\infty} J(u) = \infty} \tag{*}$$

Initially, think of U as the whole $\Re^n$ space. So $J(u)$ blows up at ∞. Note: there is a type of uniform continuity implied by Eq (*). What Eq (*) means is that given any $\gamma \ggg 0$, arbitrarily large, there exists radius R such that $J(u) > \gamma$, whenever $\|u\| > R$. This basically says that for a Coercive function we can always find a sphere, where all values of this function are larger than some value for any $\|u\| > R$. This is useful, if we are searching for a minimum, in that we can obtain a cut off on the values in U to search for.

Example $J(u) = u^2$ is positive coercive, but $J(u) = e^u$ is not (since we can't find a sphere to limit values within it to some number), since as $u \to -\infty$, the function e^u does not blow up.

$$J(u) = au^2 + bu + c$$

Is coercive only when $a > 0$. The most famous coercive function is the positive definite quadratic form. Let A be $n \times n$ symmetric and positive definite matrix and let $b \in \Re^n$. Let c be any real number, then $J(u) = u^T A u + b^T u + c$ is coercive. This is essential to quadratic programming.

Why is $u^T A u + b^T u + c$ coercive? From matrix algebra, if A is $n \times n$ symmetric and $x \in \Re^n$, then $\lambda_{\min} \|x\|^2 \leq x^T A x \leq \lambda_{\max} \|x\|^2$. For $x, y \in \Re^n$, by Schwarz inequality, $\left|\langle x, y\rangle\right|^2 \leq \langle x, x\rangle \langle y, y\rangle$ or $x^T y \leq \|x\| \left\|y\right\|$, to establish $J(u) = u^T A u + b^T u + c$ notice then $J(u) \geq \lambda_{\min}(A) \|u\|^2 - \|b\| \|u\| + c$. Since A is positive definite, then $\lambda_{\min}(A) > 0$ (smallest eigenvalue must be positive). So this is the same as scalar problem $au^2 + by + c$. Hence $J(u)$ is coercive function in u.

Reader What if we have a physical problem, leading to $u^TAu + b^Tu + c$, but A is not symmetric, what to do? Solution: We can always work with the symmetrical part of A using $u^TAu = \frac{1}{2}u^T\left(A + A^T\right)u$, hence we work with

$$J(u) = \frac{1}{2}u^T\left(A + A^T\right)u + b^Tu + c$$

Instead.

2.4.4 Convex sets and Coercivity theorem

Coercivity theorem Suppose $J : U \to \Re$ is a continuous function and coercive. And $U \subseteq \Re^n$ is closed. Then an optimal element $u^* \in U$ exist minimizing $J(u)$. **Reader** Consider the maximization problem instead of minimization.

Now we start on a new topic: Convexity. Toward finding optimal. A set $U \subseteq \Re^n$ is said to be convex set if the following holds: For any $u^0, u^1 \in U$, and $\lambda \in [0,1]$ then it follows that $\lambda u^0 + (1-\lambda)u^1 \in U$. In words, this says that all points on the straight line between any points in the set, are also in the set. Examples:

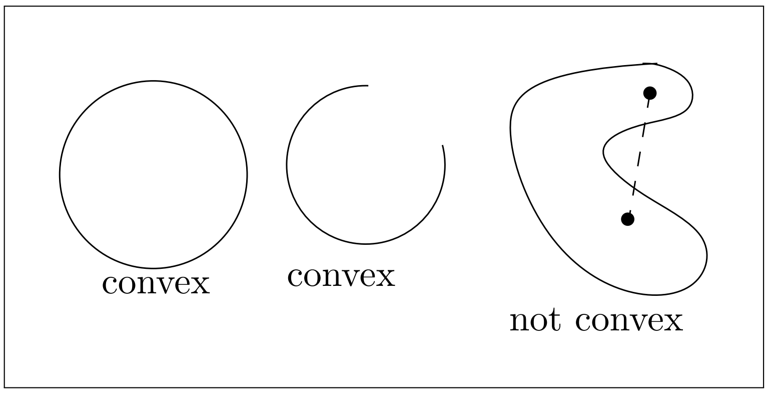


Figure 2.5: Convex sets

Reader If U_1, U_2 are both convex then show the intersection is also convex.

What about countable intersections $\cap_{i=1}^{\infty}U_i$?

Answer Yes. Example $U_i = \left\{u \in \Re^n : \|u\| \leq e^{-i}\right\}$. The union is not a convex set.

Constraints using OR are union. So harder to work with OR constraints, since union of convex sets is not convex.

Reader Describe all possible convex sets on $\Re^1$.

Reader Suppose U_i is defined by set of inequalities $a_i^Tu \leq b_i$, $i = 1 \cdots m$, these are intersections of sets. This is used in Linear programming.

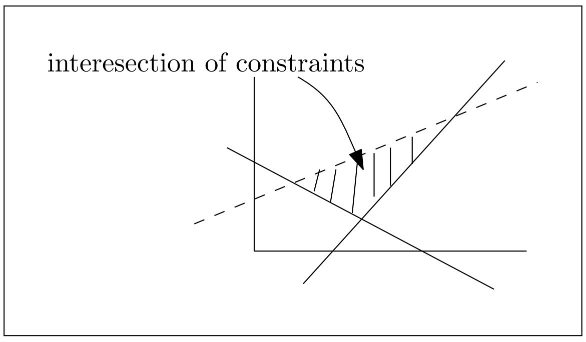


Figure 2.6: Intersection of constraints

2.5 Lecture 5. Tuesday, February 2, 2016

Back to the most important topic in optimization, which is convexity. We want to build on convex sets.

Definition A set $U \subseteq \Re^n$ is convex if the following holds: Given any $u^0, u^1 \in U$ and $\lambda \in [0,1]$, then $u^\lambda = (1-\lambda)\, u^0 + \lambda u^1$ is also in U. We will discuss convex function also soon.

Reader Show that the set $\{u \in \Re^n, \|u\| \leq r\}$ is convex.

Answer: Let $\|u_1\| \leq r, \|u_2\| \leq r$. Then $\|(1-\lambda)\, u_1 + \lambda u_2\| \leq \|(1-\lambda)\, u_1\| + \|\lambda u_2\|$ by triangle inequality. Hence

$$\|(1-\lambda)\, u_1 + \lambda u_2\| \leq (1-\lambda)\, \|u_1\| + \lambda\, \|u_2\|$$

If some center point, say $\hat{u}$ is given, then $\{u : \|u - \hat{u}\| \leq r\}$ is convex set. The translation $u + \hat{u}$ is also convex set. Other norms on $\Re^n$ are important. $\|u\|_1 = \sum_{i=1}^{n} |u_i|$ and $\|u\|_\infty = \max\{|u_1|, |u_2|, \cdots, |u_n|\}$.

Reader In $\Re^2$, sketch unit ball $\{u : \|u\| \leq r\}$ using norms $\|u\|_1$ and $\|u\|_\infty$

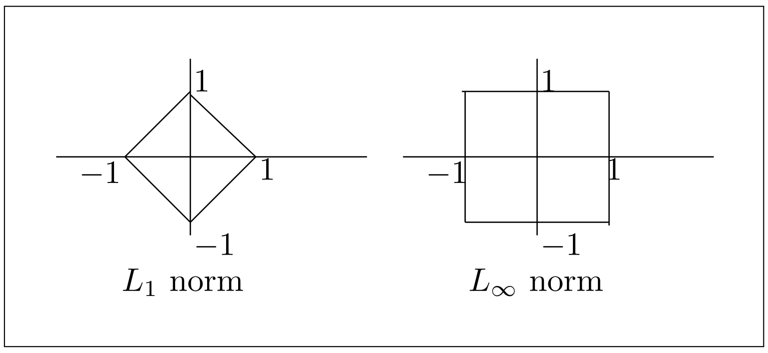
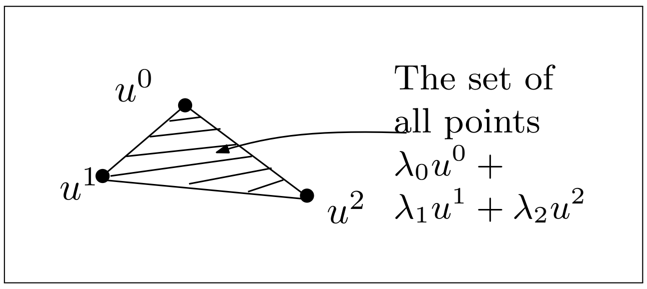


Figure 2.7: L_1 and L_∞ norms

Reader Do the above for $\Re^3$.

Reader Suppose U is convex, and $u^0, u^1, \cdots, u^m \in U$ and $\lambda_0, \lambda_1, \cdots, \lambda_m \geq 0$ with $\sum_{i=1}^{m} \lambda_i = 1$, then show that $\sum_{i=1}^{m} \lambda_i u^i \in U$. (See HW 2). For three points, u^0, u^1, u^2, then $\sum_{i=1}^{m} \lambda_i u^i$ is the mixture, which is convex set between all the three points:

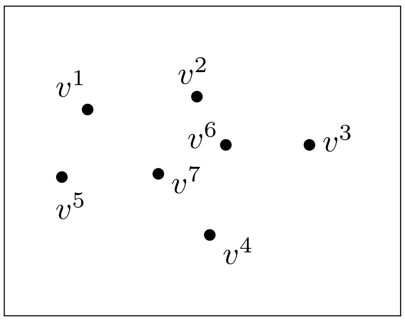

u^0
u^1
u^2
The set of all points $\lambda_0 u^0 + \lambda_1 u^1 + \lambda_2 u^2$

2.5.1 Polytope

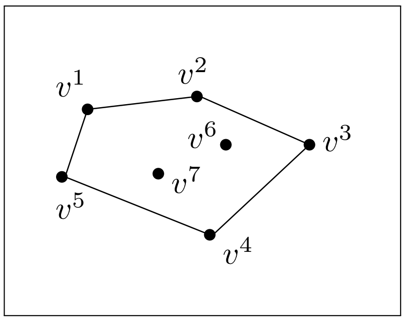
In words, these are flat sided shapes, which are convex. Let $v^1, v^2, \cdots, v^m \in \Re^n$ be given. We call set U a polytope generated by v^i if U is a set of mixtures of v^i. That is,

$$U = \left\{ \sum_{i=1}^{m} \lambda_i v^i : \lambda_i \geq 0 \text{ and } \sum_{i=1}^{m} \lambda_i = 1 \right\}$$

The following are some illustration. Given these points

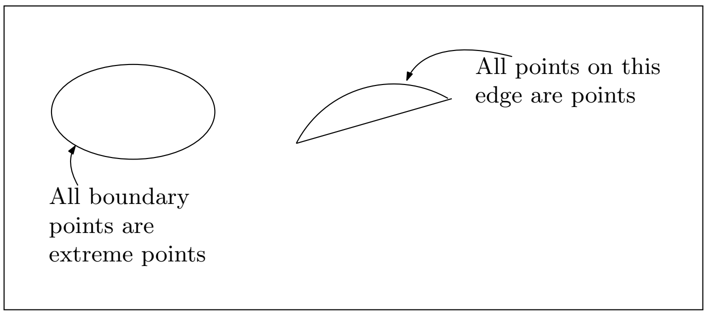


The generated polytope is



Notice that the points v^6, v^7 are redundant and have not been used. In higher dimensions, it will be harder to know which are the vertices of the extreme points.

Extreme points Let $U \subseteq \Re^n$ be convex set. A point $u \in U$ is said to be an extreme point if the following holds: u can ***not*** be written as a convex combination of other points $u^0, u^1, \cdots$. Examples:

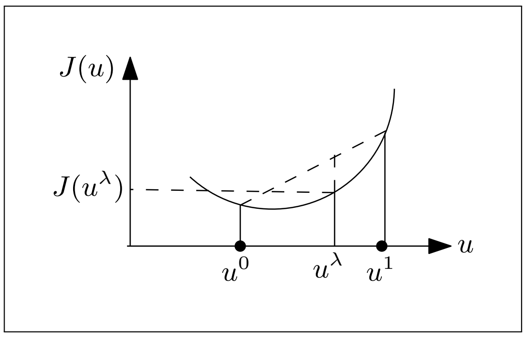


2.5.2 Convex functions

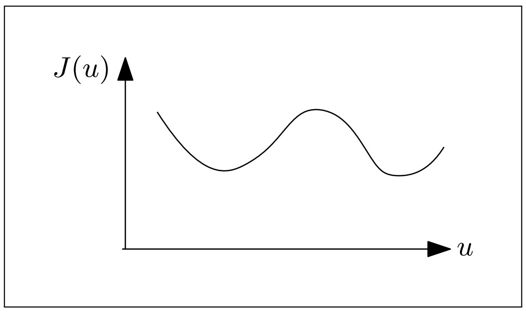
So far we talked about convex sets. Now we will talk about convex functions. $J(u) : \Re^n \rightarrow \Re$ is said to be convex function is the following is satisfied. Given any $u^0, u^1 \in U \subseteq \Re^n$ and $\lambda \in [0,1]$ then

$$\boxed{J\left((1-\lambda)\,u^0 + \lambda u^1\right) \leq (1-\lambda)\,J\left(u^0\right) + \lambda J\left(u^1\right)}$$

In words, it means the function value $J(u)$ between 2 points, is always below the straight cord points joining $J\left(u^0\right)$ and $J\left(u^1\right)$



For example, the following is not a convex function.

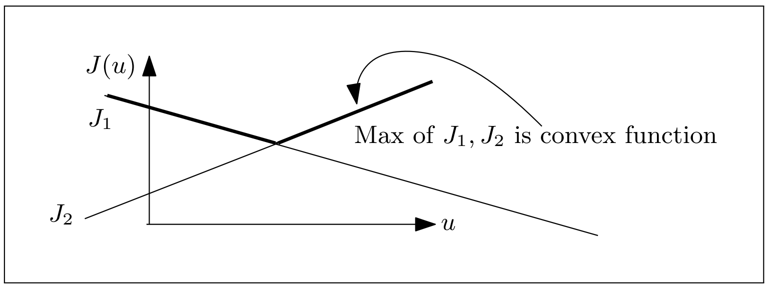


But the above is convex over some regions. But overall, it is not a convex function. We can not use this definition to check if a function is convex for higher dimensions. In that case, we have to use the Hessian to check.

Reader A function $J(u)$ is concave if $-J(u)$ is convex. Example: $e^{\alpha u}$ is convex. Any linear function is convex function. $a^T u + b$ is convex. Also $-\log(u)$ is a convex function. And $J(u) = au^2 + bu + c$ with $a > 0$ is a convex function. What about the following?

$$\max\{J_1, J_2\} = \max\left\{(a_1)^T u + b_1, (a_2)^T u + b_2\right\}$$

Is it convex function? The pointwise maximum of two or more convex functions is a convex function.



Reader Suppose $J_i : \Re^n \rightarrow \Re$ are convex functions, not necessarily linear, for $i = 1, \cdots m$, then $J(u) = \max\{J_1(u), \cdots, J_m(u)\}$ is convex function. Hint, use $(1-\lambda) J(u) + \lambda J(u) = (1-\lambda)\max(\cdots) + \cdots$.

Next is to connect convex functions to convex sets.

2.6 Lecture 6. Thursday, February 4, 2016

2.6.1 Convex functions and convex sets

An interpretation of convex function in 1D is bowl shaped. But pictures are only for low dimensions. Convex applications have taken off in the last 15 years. For example, CVX software. Why is convex so great? There are useful properties of convex functions

1. Every local minimum is also global. This is important, since once we converge to a minimum, we can stop, as there will not be any better.
2. If the objective function is strict convex, then the minimum found is unique. If it is not strict convex, then there are other minimums of the same value, hence the minimum is not unique.
3. In quadratic programming, positive definite is the same as convex.

Reader Suppose $U \subseteq \Re^n$ is convex set, and $J : U \to \Re$ is convex function. Then show the set of minimizer elements $U^* = \left\{u \in U, J(u) = \min_{u \in U} J(u)\right\}$ is a convex set.

Other nice properties of convex functions is that point wise maximum of convex functions is also a convex function. The maximum over an indexed collection is also a convex function. Let I be a set, perhaps uncountable, for each $i \in I$, suppose we have a convex function $J_i : \Re^n \to \Re$ is convex. Let $J(u) = \sup_{i \in I} J_i(u)$ and assume $J(u) < \infty$.

Reader Show that $J(u)$ is convex function. Example: $J_q(u) = 6u^2 + \left(6q - \cos q\right) + e^{-q}$. Where $\|q\| \leq 1$. Then $J(u) = \max_{|q| \leq 1} J_q(u)$.

2.6.2 convex functions and convex sets relation

Given $J : \Re^n \to \Re$, define *epiJ* as set

$$\boxed{\left\{(u, v) \in \Re^{n+1} : v \geq J(u)\right\}}$$

$J(u)$ is convex function and *epiJ* is a convex set. See HW2 *epi* problem.

2.6.3 Criterion for convexity, Gradient and Hessian

Begin with $J : \Re^n \to \Re$. How to check $J(u)$ is convex? We assume $J(u)$ is twice differentiable, called C^2. And also assume U is open and convex set.

Definition: Gradient $\nabla J(u) = \begin{bmatrix} \frac{\partial J}{\partial u_1} & \frac{\partial J}{\partial u_1} & \cdots & \frac{\partial J}{\partial u_n} \end{bmatrix}^T \in \Re^n$.

The Hessian

$$\nabla^2 J(u) = \begin{bmatrix} \frac{\partial J^2}{\partial u_1^2} & \frac{\partial^2 J}{\partial u_1 \partial u_2} & \cdots & \frac{\partial^2 J}{\partial u_1 \partial u_n} \\ \frac{\partial^2 J}{\partial u_2 \partial u_1} & \frac{\partial J^2}{\partial u_2^2} & \cdots & \frac{\partial^2 J}{\partial u_2 \partial u_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial u_n \partial u_1} & \frac{\partial^2 J}{\partial u_n \partial u_2} & \cdots & \frac{\partial J^2}{\partial u_n^2} \end{bmatrix}$$

Reader Why the Hessian is always symmetric? (Order of differentiation does not matter).

Positive definite Hessian is the same as saying second derivative is greater than zero.

2.6.4 Hessian theorem

Suppose $J : \Re^n \to \Re$ is C^2. Then J is convex function in U iff $\nabla^2 J(u)$ is positive semi-definite matrix for all $u \in U$.

Proof We first proof for $n = 1$. Sufficiency: Suppose $\nabla^2 J(u) \geq 0$ for all $u \in U$, (i.e. this is the same as saying $\frac{\partial^2 J}{\partial u^2} > 0$, and let u^0, u^1 be given in U. We need to show that for $\lambda \in [0,1]$, that $J\left(\lambda u^0+(1-\lambda)u^1\right)\leq\lambda J\left(u^0\right)+(1-\lambda)J\left(u^1\right)$. We write

$$J\left(u^\lambda\right)=J\left(u^0\right)+\int_{u^0}^{u^\lambda}J'(\xi)\,d\xi$$

Since $J'(\xi)$ non-decreasing and $J''(\xi)>0$, so the above can be upper bounded as

$$J\left(u^\lambda\right)\leq J\left(u^0\right)+J'\left(u^\lambda\right)\left(u^\lambda-u^0\right)\tag{1}$$

Similarly,

$$\begin{aligned}J\left(u^1\right)&=J\left(u^\lambda\right)+\int_{u^\lambda}^{u^1}J'(\xi)\,d\xi\\J\left(u^1\right)&\geq J\left(u^\lambda\right)+J'\left(u^\lambda\right)\left(u^1-u^\lambda\right)\\J\left(u^\lambda\right)&\leq J\left(u^1\right)+J'\left(u^\lambda\right)\left(u^\lambda-u^1\right)\end{aligned}\tag{2}$$

Therefore $\lambda\overbrace{J\left(u^\lambda\right)}^{(1)}+(1-\lambda)\overbrace{J\left(u^\lambda\right)}^{(2)}$ using (1) and (2) gives

$$\lambda J\left(u^\lambda\right)+(1-\lambda)J\left(u^\lambda\right)\leq\lambda\left[J\left(u^0\right)+J'\left(u^\lambda\right)\left(u^\lambda-u^0\right)\right]+(1-\lambda)\left[J\left(u^1\right)+J'\left(u^\lambda\right)\left(u^\lambda-u^1\right)\right]$$

Hence

$$J\left(u^\lambda\right)\leq\lambda\left[J\left(u^0\right)+u^\lambda J'\left(u^\lambda\right)-u^0J'\left(u^\lambda\right)\right]+\left[J\left(u^1\right)+J'\left(u^\lambda\right)\left(u^\lambda-u^1\right)\right]-\lambda\left[J\left(u^1\right)+J'\left(u^\lambda\right)\left(u^\lambda-u^1\right)\right]$$

Therefore

$$\begin{aligned}J\left(u^\lambda\right)&=\lambda J\left(u^0\right)+\lambda u^\lambda J'\left(u^\lambda\right)-\lambda u^0J'\left(u^\lambda\right)+J\left(u^1\right)+u^\lambda J'\left(u^\lambda\right)-u^1J'\left(u^\lambda\right)-\left[\lambda J\left(u^1\right)+\lambda J'\left(u^\lambda\right)\left(u^\lambda-u^1\right)\right]\\&=\lambda J\left(u^0\right)+\lambda u^\lambda J'\left(u^\lambda\right)-\lambda u^0J'\left(u^\lambda\right)+J\left(u^1\right)+u^\lambda J'\left(u^\lambda\right)-u^1J'\left(u^\lambda\right)-\lambda J\left(u^1\right)-u^\lambda\lambda J'\left(u^\lambda\right)+u^1\lambda J'\left(u^\lambda\right)\\&=\lambda J\left(u^0\right)+(1-\lambda)J\left(u^1\right)-\lambda u^0J'\left(u^\lambda\right)+u^\lambda J'\left(u^\lambda\right)-u^1J'\left(u^\lambda\right)+u^1\lambda J'\left(u^\lambda\right)\\&=\left[\lambda J\left(u^0\right)+(1-\lambda)J\left(u^1\right)\right]+\lambda J'\left(u^\lambda\right)\left(u^1-u^0\right)+J'\left(u^\lambda\right)\left(u^\lambda-u^1\right)\end{aligned}$$

But $u^\lambda=\lambda u^0+(1-\lambda)u^1$, hence the above becomes

$$\begin{aligned}J\left(u^\lambda\right)&\leq\left[\lambda J\left(u^0\right)+(1-\lambda)J\left(u^1\right)\right]+\lambda J'\left(u^\lambda\right)\left(u^1-u^0\right)+J'\left(u^\lambda\right)\left(\lambda u^0+(1-\lambda)u^1-u^1\right)\\&=\left[\lambda J\left(u^0\right)+(1-\lambda)J\left(u^1\right)\right]+\lambda J'\left(u^\lambda\right)\left(u^1-u^0\right)+J'\left(u^\lambda\right)\left(\lambda u^0+u^1-\lambda u^1-u^1\right)\\&=\left[\lambda J\left(u^0\right)+(1-\lambda)J\left(u^1\right)\right]+\lambda J'\left(u^\lambda\right)\left(u^1-u^0\right)+J'\left(u^\lambda\right)\left(\lambda u^0-\lambda u^1\right)\\&=\left[\lambda J\left(u^0\right)+(1-\lambda)J\left(u^1\right)\right]+\lambda J'\left(u^\lambda\right)\left(u^1-u^0\right)-\lambda J'\left(u^\lambda\right)\left(u^1-u^0\right)\\&=\lambda J\left(u^0\right)+(1-\lambda)J\left(u^1\right)\end{aligned}$$

Hence we showed that $J\left(u^\lambda\right)\leq\lambda J\left(u^0\right)+(1-\lambda)J\left(u^1\right)$. Same idea can be used to establish necessity. We now establish $\nabla^2J(u)>0$ and convexity next time. We carry this to n dimensions.

2.7 Lecture 7, Tuesday, February 9, 2016

For $n = 1$, Let $J : U \to \Re^n$, where U is open set (so that we can differentiate on it), and convex set. Let J is C^2. J is convex iff $\frac{d^2J}{du^2} \geq 0$ for all $u \in U$. We are trying to establish the Hessian theorem. Now we want to show the above for $n > 1$. We start with the bridging lemma, which will use to proof the Hessian theorem.

2.7.1 The Bridging Lemma

Given $J : U \to \Re^n$, and U is convex set, we want to know if J is a convex function. The lemma says that J is convex iff the following condition holds:

Given any $\boldsymbol{u} \in U$, $\boldsymbol{z} \in \Re^n$, then the function $\tilde{J}(\lambda) = J(\boldsymbol{u} + \lambda \boldsymbol{z})$ is convex on the set $\Lambda = \{\boldsymbol{u}, \lambda \boldsymbol{z} \in \boldsymbol{U}\}$. Notice that the function $\tilde{J}(\lambda)$ is scalar valued. It depends on scalar λ. Hence we say $\tilde{J}(\lambda) \doteqdot \Re \to \Re$. This lemma says that the function $\tilde{J}(\lambda)$ is convex in any direction we move to from $\boldsymbol{u}$ in the direction of $\boldsymbol{z}$ within the set U iff J is convex function.

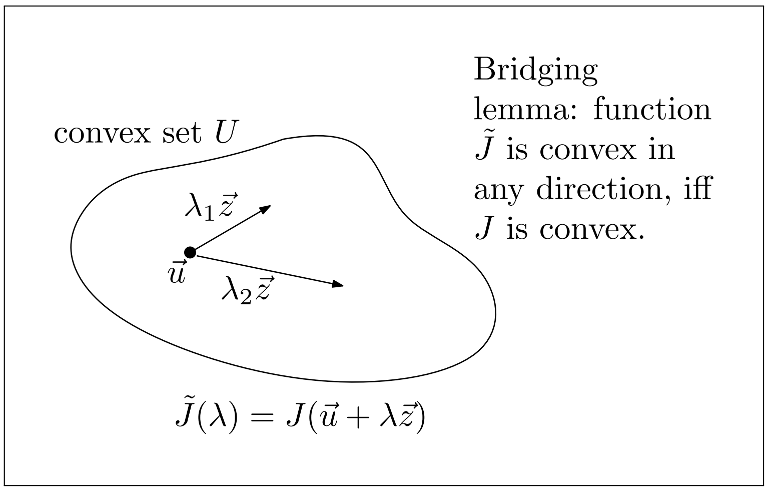

Bridging lemma: function $\tilde{J}$ is convex in any direction, iff J is convex.
convex set U
$\lambda_1\vec{z}$
$\vec{u}$
$\lambda_2\vec{z}$
$\tilde{J}(\lambda) = J(\vec{u} + \lambda\vec{z})$

Figure 2.8: Bridging lemma

Proof See also the handout. Necessity: Assume J is convex function. We must show that $\tilde{J}$ is convex function. Pick $z \in \Re$ and $\lambda \in [0,1]$ and any scalars $\alpha^0, \alpha^1 \in \Lambda$. We must show that

$$\tilde{J}\left(\lambda\alpha^0 + (1-\lambda)\alpha^1\right) \leq \lambda\tilde{J}\left(\alpha^0\right) + (1-\lambda)\tilde{J}\left(\alpha^1\right)$$

Indeed, from $\tilde{J}(\lambda) \doteqdot J(\boldsymbol{u} + \lambda \boldsymbol{z})$, then

$$\begin{aligned} \tilde{J}\left(\lambda\alpha^0 + (1-\lambda)\alpha^1\right) &\doteqdot J\left(\boldsymbol{u} + \left(\lambda\alpha^0 + (1-\lambda)\alpha^1\right)\boldsymbol{z}\right) && (1) \\ &\doteqdot J\left(\lambda\left(\boldsymbol{u} + \alpha^0\boldsymbol{z}\right) + (1-\lambda)\left(\boldsymbol{u} + \alpha^1\boldsymbol{z}\right)\right) && (2) \end{aligned}$$

Reader Going from (1) to (2) above is just a rewriting and manipulation only. Now since J is assumed convex, then

$$J\left(\lambda\left(\boldsymbol{u} + \alpha^0\boldsymbol{z}\right) + (1-\lambda)\left(\boldsymbol{u} + \alpha^1\boldsymbol{z}\right)\right) \leq \lambda J\left(\boldsymbol{u} + \alpha^0\boldsymbol{z}\right) + (1-\lambda)J\left(\boldsymbol{u} + \alpha^1\boldsymbol{z}\right)$$

Therefore (1) becomes

$$\begin{aligned} \tilde{J}\left(\lambda\alpha^0 + (1-\lambda)\alpha^1\right) &\leq \lambda J\left(\boldsymbol{u} + \alpha^0\boldsymbol{z}\right) + (1-\lambda)J\left(\boldsymbol{u} + \alpha^1\boldsymbol{z}\right) \\ &\leq \lambda\tilde{J}\left(\alpha^0\right) + (1-\lambda)\tilde{J}\left(\alpha^1\right) \end{aligned}$$

Hence $\tilde{J}$ is convex function. QED. Now to proof sufficiency. Assume that $\tilde{J}$ is convex, we need to show that this implies that J is convex on U. Since $\tilde{J}$ then

$$\begin{aligned} \tilde{J}\left((1-\lambda)\alpha^0 + \lambda\alpha^1\right) &\leq (1-\lambda)\tilde{J}\left(\alpha^0\right) + \lambda\tilde{J}\left(\alpha^1\right) \\ &= (1-\lambda)J\left(u + \alpha^0 z\right) + \lambda J\left(u + \alpha^1 z\right) && (3) \end{aligned}$$

But

$$\begin{aligned} \tilde{J}\left((1-\lambda)\alpha^0 + \lambda\alpha^1\right) &= J\left(u + \left[(1-\lambda)\alpha^0 + \lambda\alpha^1\right]z\right) && (4) \\ &= J\left((1-\lambda)\left(u + \alpha^0 z\right) + \lambda\left(u + \alpha^1 z\right)\right) && (5) \end{aligned}$$

Where the main trick was going from (4) to (5) by just rewriting, so it match what we have in (3). Now replacing (5) into LHS of (3) we find

$$J\left((1-\lambda)\left(u + \alpha^0 z\right) + \lambda\left(u + \alpha^1 z\right)\right) \leq (1-\lambda)J\left(u + \alpha^0 z\right) + \lambda J\left(u + \alpha^1 z\right)$$

Let $\left(u+\alpha^{0}z\right)\equiv u^{0},u+\alpha^{1}z\equiv u^{1}$, both in U, then the above becomes

$$J\left((1-\lambda)\,u^{0}+\lambda u^{1}\right)\leq(1-\lambda)\,J\left(u^{0}\right)+\lambda J\left(u^{1}\right)$$

Hence J is convex function. QED. Now the bridging lemma is proved. we use it to proof the Hessian theorem.

2.7.2 The Hessian Theorem, strong local minimum

Let $J=U\rightarrow\Re$, where U is open set in $\Re^{n}$. Hessian theorem says that J is convex function on U iff $\nabla^{2}J(u)$ is PSD (positive semi-definite) evaluated at each $u\in U$.

Reader Suppose $\nabla^{2}J(u)$ is PSD, does this imply strict convexity on $J(u)$? Answer: No. Need an example.

See handout Hessian for proof.

Algorithms We will now start new chapter. Looking at algorithms to find optimal of $J(u)$. Preliminaries: Begin with $J:\Re^{n}\rightarrow\Re$.

Strong local minimum u^{*} is strong local minimum if there exists $\delta>0$ such that $J(u^{*})<J(u)$ for all u such that $\|u^{*}-u\|<\delta$.

We say u^{*} is global minimum if $J(u^{*})\leq J(u)$ for all u. Henceforth, $J(u)$ is C^{2}. From undergraduate calculus, u^{*} is strong local minimum if the following is satisfied: (for $n=2$)

1. $\left.\frac{\partial J}{\partial u_{1}}\right|_{u^{*}}=0,\left.\frac{\partial J}{\partial u_{2}}\right|_{u^{*}}=0$
2. $\left.\frac{\partial^{2}J}{\partial u_{1}^{2}}\right|_{u^{*}}>0,\left(\frac{\partial^{2}J}{\partial u_{1}^{2}}\right)\left(\frac{\partial^{2}J}{\partial u_{1}^{2}}\right)-\left(\frac{\partial^{2}J}{\partial u_{1}\partial u_{2}}\right)^{2}>0$

For $J:\Re^{n}\rightarrow\Re$, in other words, in higher dimensions, define gradient

$$\boxed{(\nabla J(u))^{T}=\begin{bmatrix}\frac{\partial J}{\partial u_{1}}\\ \frac{\partial J}{\partial u_{2}}\\ \vdots\\ \frac{\partial J}{\partial u_{n}}\end{bmatrix}}$$

Normally we consider gradient as column vector. Then we say that a point $u^{*}\in\Re^{n}$ is strong local min. if $\nabla J(u)=0$, and $\nabla^{2}J(u)>0$.

Proof: Suppose u^{*} is strong local min. Then looking at neighborhood of u^{*}, let $v\in\Re^{n}$ be arbitrary. Look at $J(u^{*}+v)$ and expand in Taylor series.

$$J(u^{*}+v)=J(u^{*})+\nabla J(u^{*})\,v+v^{T}\nabla^{2}J(u)\,v+H.O.T\left(O\left(\|v\|^{3}\right)\right)$$

Since u^{*} is strong local min. then $\nabla J(u^{*})=0$. Hence $J(u^{*}+v)=J(u^{*})+v^{T}\nabla^{2}J(u)\,v+H.O.T.$ Since $J(u^{*}+v)>J(u^{*})$ (since strong local minimum), then this implies that

$$v^{T}\nabla^{2}J(u)\,v>0$$

Since $v^{T}\nabla^{2}J(u)\,v$ dominate over $H.O.T.$. This complete the proof.

2.8 Lecture 8. Thursday, February 11, 2016

Reminder, test 1 next Thursday Feb. 18, 2016. Up to and including HW 3. Today's lecture on gradient based optimization. We developed two conditions. u^* is strong local minimum when $\nabla J(u^*) = 0$ and $\nabla^2 J(u^*) < 0$.

2.8.1 gradient based optimization and line searches

We mention line searches. It is about optimization for one variable functions only. There will be reading assignment on line search. Some methods used are

1. Golden section.
2. Fibonacci.
3. Bisection

And more.

2.8.2 Optimal gain control problems, Lyapunov equation

We will now set up application areas. Optimal gain control and circuit analysis problems. In general we have

$$\dot{x} = Ax + Bu$$

Where $\dot{x}$ is $n \times 1, A$ is $n \times n$, B is $n \times m$ and u is $m \times 1$. System has n states. u is the input. This can be voltage or current sources. u is the control and x is the state. We want to select u so that $x(t)$ behaves optimally. Classical setup is to use state feedback

$$u = kx + v$$

Where k is $m \times n$ is called the feedback gain matrix and v is extra input but we will not use it. It is there for extra flexibility if needed. We use optimization to determine k. Entries of k_{ij} are our optimization variables.

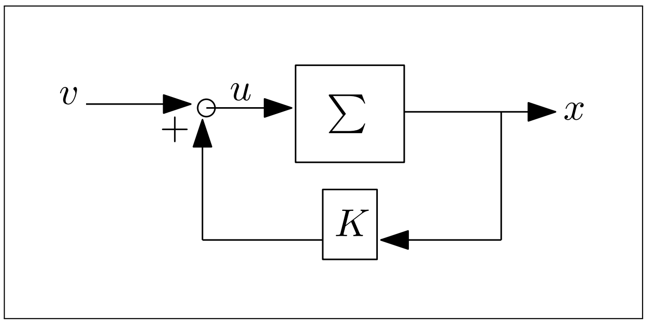


Figure 2.9: State feedback

Often, with $u = 0$, the system $\dot{x} = Ax$ may be unstable or have overshoot. We will set up a performance objective aimed at reducing or eliminating the badness of the original response (with no feedback control). Let

$$J(k) = \int_0^\infty \overbrace{x^T(t)\,x(t)}^{\|x\|^2} dt$$

In the above, $J(k)$ is implicit function of k. This cost function was found to work well in practice. We now want to make $J(k)$ explicit in k. We can solve for $x(t)$ from

$$\begin{aligned}\dot{x} &= (A + Bk)\,x \\ x &= x(0)\,e^{(A+Bk)t}\end{aligned}$$

Then $J(k) = \int_0^\infty x^T(0)\,e^{(A+Bk)^T t} dt$. But this is not practical to use. This is not closed form and hard to compute. So how can we come up with closed form for $J(k)$ which is easier to work with? Let us look at the closed loop. Let $v = 0$ and we have

$$\begin{aligned}\dot{x} &= Ax + Bkx \\ &= (A + Bk)\,x \\ &= A_c x\end{aligned}$$

Where A_c is the closed loop system matrix. Let us find a matrix P if possible such that

$$d\left(x^T(t)\,Px(t)\right) = -x^T(t)\,x(t)$$

So that now

$$\begin{aligned}J(k) &= \int_a^b x^T(t)\,x(t)\,dt \\ &= -\int_a^b d\left(x^T(t)\,Px(t)\right) \\ &= \int_b^a d\left(x^T(t)\,Px(t)\right) \\ &= x^T(a)\,Px(a) - x^T(b)\,Px(b)\end{aligned}$$

Can we find P?

$$\begin{aligned}d\left(x^T Px\right) &= x^T P\dot{x} + \dot{x}^T Px \overset{?}{=} -x^T x \\ &= x^T P\,(A_c x) + (A_c x)^T\,Px \overset{?}{=} -x^T x \\ &= x^T P\,(A_c x) + \left(x^T A_c^T\right) Px \overset{?}{=} -x^T x\end{aligned}$$

Bring all the x to LHS then

$$A_c^T x + PA_c = -I$$

Where I is the identity matrix. This is called the Lyapunov equation. This is the equation to determine P. Without loss of generality, we insist on P being symmetric matrix. Using this P, now we write

$$\begin{aligned}J(k) &= \int_0^\infty x^T(t)\,x(t)\,dt \\ &= -\int_0^\infty d\left(x^T Px\right) \\ &= x^T Px\Big|_\infty^0 \\ &= x^T(0)\,Px(0) - x^T(\infty)\,Px(\infty)\end{aligned}$$

For stable system, $x(\infty) \to 0$ (remember that we set $v = 0$, so there is no external input, hence if the system is stable, it must end up in zero state eventually). Therefore

$$J(k) = x^T(0)\,Px(0)$$

With k satisfying

$$A_c^T(k)\,x + PA_c(k) = -I$$

Example Let $y'' = u$. Hence $x_1' = x_2, x_2' = u$. Note, this is not stable with $u = 0$. Using linear state feedback,

$$\begin{aligned}u &= kx \\ u &= \begin{bmatrix} k_1 & k_2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\end{aligned}$$

Hence

$$
\begin{aligned}
x' &= Ax + Bu\\
&= Ax + Bkx\\
&= (A + Bk)\,x\\
\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} &= \left(\overbrace{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}}^{A} + \overbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}^{B} \overbrace{\begin{bmatrix} k_1 & k_2 \end{bmatrix}}^{k} \right) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\\
&= \left(\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ k_1 & k_2 \end{bmatrix} \right) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\\
&= \overbrace{\begin{bmatrix} 0 & 1 \\ k_1 & k_2 \end{bmatrix}}^{A_c} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\end{aligned}
$$

For stable closed loop, we need $k_1 < 0, k_2 < 0$ by looking at characteristic polynomial roots. Now we solve the Lyapunov equation.

$$A_c^T P + P A_c = -I$$

$$\begin{bmatrix} 0 & 1 \\ k_1 & k_2 \end{bmatrix}^T \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} + \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ k_1 & k_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & k_1 \\ 1 & k_2 \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} + \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ k_1 & k_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

Solving for P gives

$$P = \begin{bmatrix} \frac{k_2^2 - k_1 + k_1^2}{2k_1k_2} & -\frac{1}{2k_1} \\ -\frac{1}{2k_1} & \frac{1-k_1}{2k_1k_2} \end{bmatrix}$$

With $x(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, then

$$
\begin{aligned}
J(k) &= x(0)^T P x(0)\\
&= \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{k_2^2 - k_1 + k_1^2}{2k_1k_2} & -\frac{1}{2k_1} \\ -\frac{1}{2k_1} & \frac{1-k_1}{2k_1k_2} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}\\
&= \frac{k_1^2 + k_2^2 - 2k_1 - 2k_2 + 1}{2k_1k_2}
\end{aligned}
$$

Here is Matlab script to generate the above

```
syms k1 k2 p11 p12 p21 p22;
Ac   = [0 1;k1 k2];
P    = [p11 p12;p21 p22];
eq   = Ac.'*P+P*Ac==-eye(2);
sol  = solve(eq,{p11,p12,p21,p22});
P    = subs(P,sol)
x0   = [1;1];
J    = simplify(x0'*P*x0)
```

.

Let $k_1 = k_2 = k < 0$, we obtain

$$J(k) = \frac{2k^2 - 4k + 1}{2k^2}$$

As $k \to -\infty$ then $J^* \to 1$. Therefore J^* can never get to zero. This means there is no k_1^*, k_2^* such that $\nabla J(k^*) = 0$. Set k is not compact. Not coercive either. This is ill posed problem.

This can be remedied by changing the control to

$$J(k) = \int_0^\infty x^T x dt + \lambda \int_0^\infty u^T u dt$$

2.9 Lecture 9. Tuesday, February 16, 2016

2.9.1 keywords for next exam 1

1. Common sense optimization. Farming problem. Explicit vs. Implicit.
2. Minimizing $J(u)$, $\inf, \sup$
3. We want to know ahead of time if minimum can be attained. J^* but u^* might not exist.
4. Multilinear function. u^* is at a vertex. But they grow as 2^n where n is number of variables.
5. When can we be sure u^* exist? if the set is compact, we talked about W-B theory, which is used to show u^* exist always for compact sets. If the set is not compact but coercive, then we can compact it.
6. Convex sets and convex functions. For convex sets, when we find u^* then the local minimum is also a global minimum.
7. Special case of convex sets is polytope. Polyhedron is a polytope but can be unbounded.
8. Strong local minimum is when $\nabla J(u) = 0$ and $\nabla^2 J(u) > 0$. To test for convexity, find the Hessian. If the Hessian is semi positive definite, then it is convex.
9. Optimal gain control, Lyapunov equation.
10. If there are proofs, they will be simple, such as show the sum of two convex functions is also convex function.

2.9.2 Gradient based optimization

Starting new chapter. Gradient based optimization. Many algorithms involve line searches. In other words, optimization in $\Re^n$ is often solved by performing many optimization (line searches) in $\Re$.

Optimization algorithm: Starting with $\boldsymbol{u}^k$ and direction $\boldsymbol{v}$ we study $J\left(\boldsymbol{u}^k + h\boldsymbol{v}\right)$ where h is the step size. This is called line search. $h \in [0, h_{\max}]$. We want to use optimal step size h^*. Once found, then

$$\boxed{\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + h^* \boldsymbol{v}}$$

Reader Read and learn about line search. Bisection, Golden section, Fibonacci and many more. We will not cover this in this course. We also want to minimize the number of function evaluations, since these can be expensive.

One way to do line search, is to do $h = 0 \delta h_{\max}$ and then evaluate $J(h)$ and pick the minimizing h^*. For stopping criteria, we can check for the following

1. $\left\| u^{k+1} - u^k \right\| \leq \delta$
2. $\left\| J\left(u^{k+1}\right) - J\left(u^k\right) \right\| \leq \delta$
3. $\left\| \frac{J\left(u^{k+1}\right) - J\left(u^k\right)}{J\left(u^{k+1}\right)} \right\| \leq \delta$
4. $\left\| \nabla J\left(u^k\right) \right\| \leq \delta$

We also need to pick a starting point for the search. This is $\boldsymbol{u}^0$. What if we do not know where to start? We can pick multiple starting locations. And pick the best result obtained.

Reader Find $\min_{\|\boldsymbol{v}\|=1} J(\boldsymbol{u} + \boldsymbol{v})$. Show optimal $\boldsymbol{v}$ is

$$\boxed{v^* = \frac{-\nabla J(u)}{\|\nabla J(u)\|}}$$

This is called myopic local terrain. Gets us to local minimum. The steepest descent algorithm is the following:

1. Select u^0 (starting point)
2. Find step size h
3. Iterate. While $\left\|\nabla J\left(u^k\right)\right\| > \delta$ then $\boxed{u^{k+1} = u^k - h\frac{\nabla J(u)}{\|\nabla J(u)\|}}$
4. Update counter and go back to step 3 above.

See my class study notes for detailed algorithm of all search methods we did in this course.

Later we will study conjugate gradient methods.

Example: Let $u^0 = [1, 1]$. Let $h = 0.1$. Let $J(u) = u_1^2 + 2u_2^2 - 6u_1u_2 + 2u_1 + u_2 + 4$. Then

$$\nabla J(u) = \begin{bmatrix} 2u_1 - 6u_2 + 2 \\ 6u_2 - 6u_1 + 1 \end{bmatrix}$$

So $\nabla J\left(u^0\right) = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$ and $\left\|\nabla J\left(u^0\right)\right\| = \frac{1}{\sqrt{5}}$. Hence

$$\begin{aligned} u^1 &= u^0 - h\begin{bmatrix} -2 \\ 1 \end{bmatrix}\frac{1}{\sqrt{5}} \\ &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 0.1\begin{bmatrix} -2 \\ 1 \end{bmatrix}\frac{1}{\sqrt{5}} \\ &= \begin{bmatrix} 1.0894 \\ 0.95528 \end{bmatrix} \end{aligned}$$

Reader Find u^2.

2.10 Lecture 10. Thursday, February 18, 2016 (Exam 1)

Exam 1

2.11 Lecture 11. Tuesday, February 23, 2016

Watch for HW4 going out today. Implementation of steepest descent with optimal step size. See handout circuit for 2 stage amplifier.

2.11.1 Steepest descent

As the number of stages increases it becomes harder to analytically determine the optimal capacitance of each stage to produce maximum power. For two stages, by direct circuit analysis we obtain

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

Where u_i is capacitance. There are two optimal values, they are $u^* = (10, 1)$ which is a maximum and $u^* = (13, 4)$ which is a minimum. There is also a minimum at $(7, -2)$.

Geometric insight on what might go wrong with steepest descent: Gradient algorithms work best from a far but as they get close to the optimal point, there are better algorithms such as the generalized Newton Raphson method which works best when close to the optimal point. If the step size h is big, we approach the optimal fast, but because the step size is large, we can overshoot and will end up oscillating around the optimal point. If h is too small, the search will become very slow. Hence we use steepest descent but with optimal step size, where the step size is calculated at each step. Ingredients of the steepest descent algorithm are:

1. Initial guess u^0
2. maximum step size H. Here we have h_k which is the step size used at each iteration.
3. Iteration step. When at u^k define $\boxed{\tilde{J}(h) = J\left(u^k - h\frac{\nabla J\left(u^k\right)}{\left\|\nabla J\left(u^k\right)\right\|}\right)}$ and carry a line search to find optimal h which minimized $\tilde{J}(h)$, then $h_k = h_*$
4. $u^{k+1} = u^k - h_k\frac{\nabla J(u^k)}{\left\|\nabla J(u^k)\right\|}$
5. Stopping criteria. Decide how to stop the search. $\left\|\nabla J\left(u^k\right)\right\| \leq \varepsilon$

Reader: Consider oscillation issue.

Convergence result. From Polak. Let $J(u)$ be smooth and differentiable. Let u^* be strong local minimum. Assume constant $0 \leq m \leq M$ s.t.

$$mu^Tm \leq u^T\nabla^2 J(u)\,u \leq Mu^TM$$

In neighborhood of u^*. This criteria says that there is a good convexity and a bad convexity. What does this mean? We'll say more about this. In the neighborhood of u^* let $\theta = \frac{m}{M}$. Interpretation:

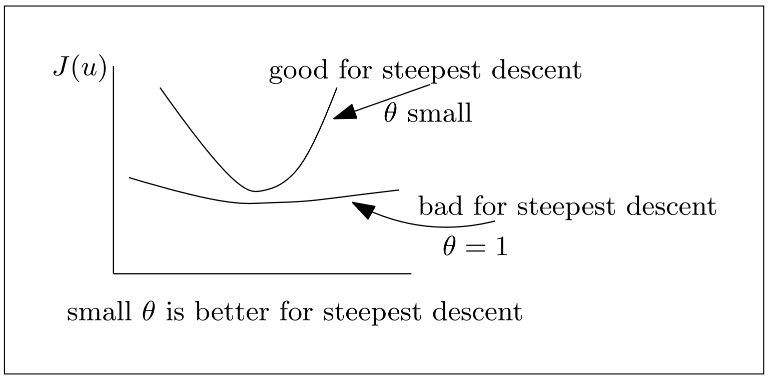

$J(u)$
good for steepest descent
θ small
bad for steepest descent
$\theta = 1$
small θ is better for steepest descent

Figure 2.10: Steepest descent diagram

Define $E = J\left(u^0\right) - J(u^*)$ then Polak says

$$0 \leq J\left(u^k\right) - J\left(u^*\right) \leq E\theta^k$$

Best case is when E is small and θ is small. This is local result.

2.11.2 Classifications of Convergence

Convergence can be

1. Linear
2. Quadratic
3. Superlinear

These are the three convergence types we will cover. The second algorithm has quadratic convergence, which is the generalized Newton-Raphson method. We will start on this now but will cover it fully next lecture.

The idea is to approximate $J(u)$ as quadratic at each step and obtain h_k. By assuming $J(k)$ is quadratic locally, we approximate $J(u)$ using Taylor and drop all terms after the Hessian. Now we find where the minimum is and use the step size to find u^{k+1}. More on this next lecture.

2.12 Lecture 12. Thursday, February 25, 2016

2.12.1 Quadratic optimization, superlinear convergence

We will start the class with a reader problem. Consider

$$J(u) = \frac{1}{2}u^T A u + b^T u + c \tag{1}$$

$$\nabla J(u) = Au^k + b \tag{2}$$

with A being symmetric positive definite (PSD) matrix. This is classic quadratic objective function. You can take a complete course on quadratic optimization. The global optimal is at the solution for $\nabla J(u) = 0$. Hence we write

$$\begin{aligned} \nabla J(u) &= 0 \\ Au^* + b &= 0 \\ u^* &= -A^{-1}b \end{aligned} \tag{3}$$

Note that since A is PSD (the Hessian is PSD), then we know that $J(u)$ is convex. Hence the local minimum is also a global minimum. Now we imagine we are doing steepest descent on this function and we are at iterate u^k with optimal step size, which we can make as large as we want. Hence we need to optimize

$$\tilde{J}(h) = J\left(u^k - h\left[\nabla J(u)\right]\right)$$

for h. Notice we did not divide by $\|\nabla J(u)\|$ here, since the step size is free to be as large as needed. Expanding the above using (2) gives

$$\begin{aligned} \tilde{J}(h) &= J\left(u^k - h\left(Au^k + b\right)\right) \\ &= J\left((I - hA)u^k - hb\right) \end{aligned}$$

Using (1) for RHS of the above gives

$$\tilde{J}(h) = \frac{1}{2}\left((I-hA)u^k - hb\right)^T A\left((I-hA)u^k - hb\right) + b^T\left((I-hA)u^k - hb\right) + c$$

The above is quadratic in h. The optimal h we are solving for. Simplifying gives

$$\tilde{J}(h) = \frac{1}{2}\left(Au^k + b\right)^T A\left(Au^k + b\right)h^2 - \left(Au^k + b\right)^T\left(Au^k + b\right)h + \text{ constant terms}$$

To find optimal h, then

$$\begin{aligned} \frac{d\tilde{J}(h)}{dh} &= 0 \\ \left(Au^k + b\right)^T A\left(Au^k + b\right)h - \left(Au^k + b\right)^T\left(Au^k + b\right) &= 0 \\ h^* &= \frac{\left(Au^k + b\right)^T\left(Au^k + b\right)}{\left(Au^k + b\right)^T A\left(Au^k + b\right)} \end{aligned}$$

In practice, we would need to check $\frac{d^2\tilde{J}(h)}{dh^2}$ also to make sure h is minimizer.

Reader Why does it take multiple iterations to get the common sense answer $u^* = -A^{-1}b$?

For quadratic objective function $J(u)$ we can obtain u^* in one step, using $u^* = -A^{-1}b$. This is the idea behind the generalized Newton-Raphson method. $J\left(u^k\right)$ is approximated as quadratic function at each step, and h^* is found from above. To elaborate, expanding by Taylor

$$J\left(u^k + \Delta u\right) = J\left(u^k\right) + \nabla J\left(u^k\right)^T \Delta u + \frac{1}{2}\Delta u^T \nabla^2 J\left(u^k\right)\Delta u + HOT$$

We approximate as quadratic by dropping higher order terms and optimize for Δu (same as h used earlier), and here $\nabla^2 J\left(u^k\right)$ is same as the A above also. Therefore we find

$$\Delta u^* = -\left[\nabla^2 J\left(u^k\right)\right]^{-1}\nabla J\left(u^k\right)$$

This converges in one step Δu^* if $J(u)$ was actually a quadratic function. Notice that Newton method is expensive if used repeatedly, as it requires finding Hessian at each step and also finding the inverse of it. The algorithm is: Initialize u^0. Then iterate, where

$$u^{k+1} = u^k - \left[\nabla^2 J\left(u^k\right)\right]^{-1} \nabla J\left(u^k\right) \tag{4}$$

Then check for convergence. If Hessian fails to be PSD, in this case $J\left(u^{k+1}\right)$ can end up increasing not decreasing. How to stop? We can try more iterations to see if $J\left(u^k\right)$ will decrease again.

Example Let

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

Pick $u^0 = (18, 3)$ then

$$\nabla J\left(u^0\right) = \begin{bmatrix} 40 \\ 100 \end{bmatrix}$$

$$\nabla^2 J\left(u^0\right) = \begin{bmatrix} 10 & 44 \\ 44 & 130 \end{bmatrix}$$

We see here that $\nabla^2 J\left(u^0\right)$ is not PSD (determinant is negative). Now we do the iterate equation (4), obtaining

$$\begin{aligned} u^1 &= u^0 - \begin{bmatrix} 10 & 44 \\ 44 & 130 \end{bmatrix} \begin{bmatrix} 40 \\ 100 \end{bmatrix} \\ &= \begin{bmatrix} 18 \\ 3 \end{bmatrix} - \begin{bmatrix} 10 & 44 \\ 44 & 130 \end{bmatrix} \begin{bmatrix} 40 \\ 100 \end{bmatrix} \\ &= \begin{bmatrix} 19.3 \\ 1.8 \end{bmatrix} \end{aligned}$$

If we look at the contour plot, we will see this point made $J(u)$ larger (away from optimal) but if we let it iterate more, it will turn and move back to the optimal point.

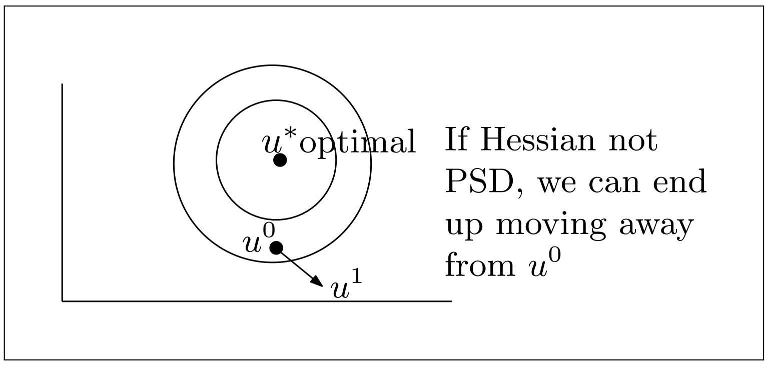


Figure 2.11: Hessian and optimal solution diagram

Now we will start on the third algorithm. Conjugate gradient algorithms (CG) are family of algorithms with property of superlinear convergence. It also has quadratic convergence.

2.12.2 Quadratic convergence

Quadratic convergence says the following: If $J(u)$ is positive definite quadratic form, the iterate $u^k \to u^*$ completes in finite number of steps N. This means if we give the algorithm a quadratic form function, it will converge in N steps to the optimal. The difference between this and Newton-Raphson, is that there is no Hessian to be calculated using this algorithm as the case was with Newton-Raphson. The conjugate direction algorithms work well from far away and also when close to the optimal point. (note: Steepest descent worked well from a far, but not when getting close to the optimal point u^*). The CG algorithms also have the property of superlinear convergence. This property do not apply to steepest descent.

2.12.3 Superlinear convergence

What is superlinear convergence? A sequence $\left\{u^k\right\}$ in $\Re^n$ is said to converge super-linearly to u^* if the following holds: Given any $\theta \in (0,1]$, then $\frac{\|u^k-u^*\|}{\theta^k} \rightarrow 0$. The important part of this definition is that the above should go to zero for any $\theta \in (0,1]$ and not some θ. Examples below illustrate this. Let $\left\{u^k\right\} = \frac{1}{k}$. Hence the sequence is $\left\{1, \frac{1}{2}, \frac{1}{3}, \cdots\right\}$. Clearly this sequence goes to $u^* = 0$. Does it converge super-linearly to u^*? Applying the definition

$$\frac{\left\|\frac{1}{k} - 0\right\|}{\theta^k} = \frac{1}{k\theta^k}$$

If we can find one θ that do not converge to zero, then we are done. Trying $\theta = \frac{3}{4}$, then the above becomes $\frac{4^k}{k3^k}$ which do not go to zero as $k \rightarrow \infty$. Hence this is not superlinear. How about $\left\{u^k\right\} = \frac{1}{k^2}$. This is still not superlinear. Similarly $\left\{u^k\right\} = \frac{1}{k^m}$. What about $\left\{u^k\right\} = \frac{1}{e^k}$. Here we get

$$\frac{\left\|\frac{1}{k} - 0\right\|}{\theta^k} = \frac{e^{-k}}{\theta^k}$$

Trying $\theta = \frac{1}{2}$ gives $\frac{2^k}{e^k}$ which is not superlinear (do not go to zero for large k). But if $\theta = \frac{2}{3}$ it will converge. But it has to converge for all θ, so $\frac{1}{e^k}$ is not superlinear. How about $\left\{u^k\right\} = \frac{1}{e^{k^2}}$ here we find it is superlinear. We obtain $\frac{e^{-k^2}}{\theta^k}$ and this goes to zero for any θ. To show this, use $\log$ on it and simplify. (Reader).

Next time we will go over conjugate direction algorithm in more details.

2.13 Lecture 13. Tuesday, March 1, 2016

2.13.1 Conjugate direction algorithms

Today lecture will be devoted to conjugate direction (C.D.) algorithms. We will start by remembering that there are many C.D. algorithms. From last lecture, be aware of: Superlinear convergence and quadratic convergence. The quadratic convergence concept is that on a P.S.D. (positive symmetric definite) form, the algorithm will converge in n steps or less (where n is the size A). Using exact arithmetic (not counting for floating point errors). We will proof this today. We will some preliminaries, then go over ingredients and go over examples, then go over properties of conjugate gradient.

Preliminaries: Let $A^{n\times n}$ be positive definite symmetric, then the pair of vectors u,v are said to be mutually conjugate w.r.t. A if

$$u^T A v = 0$$

This is generalization of orthogonality. Because we can take $A = I_n$ which is PSD.

Reader A set of distinct mutually conjugate vectors always exist for A. These are the eigenvectors of A. The proof starts with writing $Av = \lambda_1 v$ and $Au = \lambda_2 u$, then applying $u^T A v = 0$.

We will use this set of vectors as search directions. We will generate these vectors on the fly during the search and do line search along these directions. So instead of using $\nabla J(u)$ as the direction we did line search on when using steepest descent, we will now use the conjugate vectors instead.

Properties: Suppose $v^0, v^1, \cdots, v^{n-1}$ is a set of mutually conjugate vectors w.r.t A. (A is PSD). First step is to show these vectors are linearly independent.

Lemma: v^i are linearly independent. Proof: Suppose

$$\sum_{i=0}^{n-1} \alpha_i v^i = 0 \tag{1}$$

For scalars α_i. We must show that all $\alpha_i = 0$. Let use consider α_k. If we can show that $\alpha_k = 0$ for any k, then we are done. Multiply (1) by $\left(v^k\right)^T A$. Then

$$\left(v^k\right)^T A \sum_{i=0}^{n-1} \alpha_i v^i = 0$$
$$\sum_{i=0}^{n-1} \alpha_i \left(v^k\right)^T A v^i = 0$$

By mutual conjugate property, then all terms above vanish except $\alpha_k \left(v^k\right)^T A v^k$. Hence

$$\alpha_k \left(v^k\right)^T A v^k = 0$$

But A is PSD and $v^k \neq 0$, therefore $\alpha_k = 0$ is only choice. QED. We have proved that $v^0, v^1, \cdots, v^{n-1}$ are linearly independent So we can expand any vector $u \in \Re^n$ using these as basis vectors

$$u = \sum_{i=0}^{n-1} a_i v^i \tag{2}$$

Let us find the coefficients a_i. Premultiply by $\left(v^k\right)^T A$ both sides

$$\left(v^k\right)^T A u = \sum_{i=0}^{n-1} a_i \left(v^k\right)^T A v^i$$

As before, by mutual conjugate, the RHS becomes $a_k \left(v^k\right)^T A v^k$. Solving for a_k gives

$$a_k = \frac{\left(v^k\right)^T A u}{\left(v^k\right)^T A v^k}$$

Hence (2) becomes

$$u = \sum_{i=0}^{n-1} \frac{\left(v^i\right)^T Au}{\left(v^i\right)^T Av^i} v^i$$

This gives any vector u in terms of set of vectors v^i that are linearly independent

Conjugate directions algorithm ingredients are:

1. Initially given u^0. The starting guess vector
2. Iterative step u^k: Generate v^k a mutual conjugate vector to previous $n-1$ vectors v^i. For v^0 use $-\nabla J(u)$. Same as steepest descent.
3. Form line search with Max step H. To minimize $\tilde{J}(h) = J\left(u^k + hv^k\right)$. Notice there we used + sign and not − as with steepest descent. The direction takes care of the sign in this case.
4. Stopping criteria.

Example: Fletcher Reeves.

$$\begin{aligned} v^0 &= -\nabla J(u) \\ v^{k+1} &= -\nabla J\left(u^{k+1}\right) + \frac{\left\|\nabla J\left(u^{k+1}\right)\right\|^2}{\left\|\nabla J\left(u^k\right)\right\|^2} v^k \end{aligned}$$

Reader Normalize above for implementation.

Reader What is A above? Where are these v^k vectors mutually conjugate? A is the Hessian. Note: These algorithms (C.D.) converge for convex $J(u)$. If $J(u)$ is not convex or we do not know, we need to put conditions to make sure it is converging.

For Polyak-Ribieve, see homework.

2.13.2 Quadratic convergence theorem

Consider quadratic form

$$J(u) = \frac{1}{2} u^T Au + b^T u + c$$

With $A = A^T$ and positive definite $n \times n$ matrix. Let $v^0, \cdots, v^{n+1}$ be mutually conjugate w.r.t. A. Let step size be as large as we want. The conjugate direction algorithm converges to optimal $u^* = -A^{-1}b$ in n steps or less

Proof

Let u^k be the k^{th} iterate. If $u^k = u^*$ and $k \leq n$ then we are done. Without loss of generality, assume $u^k \neq u^*$. We must show that $u^n = u^*$. We first find h_k, the step size at iterate k. From

$$\begin{aligned} \tilde{J}(h) &= J\left(u^k + hv^k\right) \\ &= \frac{1}{2}\left(u^k + hv^k\right)^T A\left(u^k + hv^k\right) + b^T\left(u^k + hv^k\right) + c \end{aligned}$$

This is quadratic in h.

$$\tilde{J}(h) = \frac{1}{2}(v^k)^T Av^k h^2 + \left((v^k)^T Au^k + b^T v^k\right)h + \overbrace{\frac{1}{2}u^k Au^k + c}^{\text{constant term}}$$

Now taking derivative gives

$$\frac{d\tilde{J}(h)}{dh} = \left(v^k\right)^T Av^k h + \left(\left(v^k\right)^T Au^k + b^T v^k\right)$$

Setting this to zero and solving for h gives

$$\begin{aligned} h^* &= -\frac{\left(v^k\right)^T A u^k + b^T v^k}{\left(v^k\right)^T A v^k} \\ &= -\frac{\left(v^k\right)^T \left(A u^k + b\right)}{\left(v^k\right)^T A v^k} \end{aligned} \tag{3}$$

Hence

$$\begin{aligned} u^n &= u^0 + h_0 v^0 + \cdots + h_{n-1} v^{n-1} \\ &= u^0 + \sum_{k=0}^{n-1} h_k v^k \end{aligned}$$

Using (3) in the RHS of above, replacing each h_k with the optimal h at each iterate gives

$$\begin{aligned} u^n &= u^0 - \sum_{k=0}^{n-1} \left(\frac{\left(v^k\right)^T \left(A u^k + b\right)}{\left(v^k\right)^T A v^k} \right) v^k \\ &= u^0 - \sum_{k=0}^{n-1} \left(\frac{\left(v^k\right)^T b}{\left(v^k\right)^T A v^k} \right) v^k - \sum_{k=0}^{n-1} \left(\frac{\left(v^k\right)^T A u^k}{\left(v^k\right)^T A v^k} \right) v^k \end{aligned}$$

Replacing u^k in the second term above in the RHS with $u^0 + \sum_{i=0}^{k-1} h_i v^i$ gives

$$u^n = u^0 - \sum_{k=0}^{n-1} \left(\frac{\left(v^k\right)^T b}{\left(v^k\right)^T A v^k} \right) v^k - \sum_{k=0}^{n-1} \frac{\left(v^k\right)^T A \left(u^0 + \sum_{i=0}^{k-1} h_i v^i\right)}{\left(v^k\right)^T A v^k} v^k \tag{4}$$

But

$$\begin{aligned} \left(v^k\right)^T A \left(u^0 + \sum_{i=0}^{k-1} h_i v^i\right) &= \left(v^k\right)^T A u^0 + \sum_{i=0}^{k-1} h_i \left(v^k\right)^T v^i \\ &= \left(v^k\right)^T A u^0 \end{aligned}$$

Since all terms in $\sum_{i=0}^{k-1} h_i \left(v^k\right)^T v^i$ vanish by mutual conjugate property. Using this to simplify (4) gives

$$\begin{aligned} u^n &= u^0 - \sum_{k=0}^{n-1} \left(\frac{\left(v^k\right)^T b}{\left(v^k\right)^T A v^k} \right) v^k - \sum_{k=0}^{n-1} \left(\frac{\left(v^k\right)^T A u^0}{\left(v^k\right)^T A v^k} \right) v^k \\ &= u^0 - \sum_{k=0}^{n-1} \left(\frac{\left(v^k\right)^T \left(A u^0 + b\right)}{\left(v^k\right)^T A v^k} \right) v^k \end{aligned}$$

Reader $\left(v^k\right)^T A u^0$ is expansion of u^0. Using this in the above reduces it to

$$u^n = -\sum_{k=0}^{n-1} \left(\frac{\left(v^k\right)^T b}{\left(v^k\right)^T A v^k} \right) v^k$$

Insert AA^{-1} into the above gives

$$\begin{aligned} u^n &= -\sum_{k=0}^{n-1} \left(\frac{\left(v^k\right)^T A A^{-1} b}{\left(v^k\right)^T A v^k} \right) v^k \\ &= -\sum_{k=0}^{n-1} \left(\frac{\left(\left(v^k\right)^T A\right) \left(A^{-1} b\right)}{\left(v^k\right)^T A v^k} \right) v^k \\ &= -A^{-1} b \end{aligned}$$

But $-A^{-1}b = u^*$. QED.

Following some extra remarks added later from An introduction to optimization by Ching and Zak, 1996:

1. Conjugate direction algorithms solve quadratics of n variables in n steps
2. Algorithm does not require calculation of Hessian.
3. Algorithm requires no matrix inverse and no storage for the $A^{n\times n}$ matrix.
4. C.D. Algorithms perform better than steepest descent but not as well as Newton-Raphson (when close to optimal).
5. For quadratic $J(u) = \frac{1}{2}x^T Ax - x^T b$, the best direction at each step is the mutually conjugate direction w.r.t. A.

See Example 10.1, page 133 of the above text for illustrations how to determine each of v^i vectors for given A matrix.

2.14 Lecture 14. Thursday, March 3, 2016

2.14.1 Constraints and linear programming

We are about to enter new phase of the course with constraints and linear programming. Until now we used iterative methods to solve unconstrained problems. These are gradient based. Also looked at Newton-Raphson. We used steepest descent and conjugate directions. These methods are mainly applied to problem without constraints. i.e. u is free, where u are the variables. But in HW4 we had problem where u was the capacitance. This can not be negative. But we did not account for this. When we have constraints and want to use the above iterative methods, there are ad hoc methods to handle this, but we will not cover these ad-hoc methods in this course, but will mention some of them.

We can check that no constraint is violated during the search and start a new search. There are literature on what is called "projection methods" and other names.

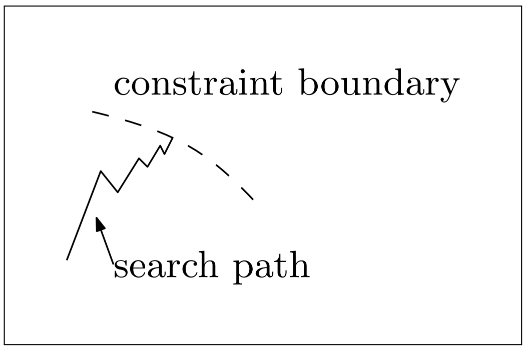


Figure 2.12: Search path near constraint

One good method is called the "penalty function method". This works as follows

$$\tilde{J}(u) = J(u) + J_{penalty}(u)$$

The original objective function is $J(u)$ and $J_{penalty}(u)$ is function we add such that it becomes very large when u constraint is violated $(u \notin U)$. (assuming we are minimizing $J(u)$).

This method works on many problems. For example, if we do not want u_1 to be negative, we can add

$$J_{penalty}(u) = -100 \min(0, u_1)$$

This way, when $u_1 \leq 0$, the result will be very large and positive. Hence $\tilde{J}(u)$ will become very large and the search will avoid this region and turn away during the line search.

But a theory with name of "Kohn Tucker" is the main way to handle such problems under heading of "non-linear programming".

Now we go back to linear programming which we will cover over the next 4–5 lectures. key points of linear programming are

1. Objective function is linear in u^i
2. Linear inequality constraints on u^i

2.14.2 History of linear programming

1. Dantzing introduced simplex algorithm. Matlab linprog implements this to find solution to L.P. problems.
2. Simplex algorithm has some problems related to what is called "klee-type pathologies". These days, L.P. have millions of variables. These days we want to solve many large scale L.P. The "Klee-type pathologies" says that there are some bad input to L.P. which causes it to become very slow. L.P. visits vertices of polytopes. We can do billion and more vertices these days on the PC with no problem. 10^{30} vertices is a small L.P. problem these days.

L.P. works fast by not visiting each vertex. But with some input L.P. can become slow and force it to visit all vertices.

3. Khachian: 1970's. Front page of NY times. Introduced ellipsoidal algorithm to solve L.P. (Faster than L.P. on the worst case problems). But it turns out that in real world problems, simplex was still faster, unless the problem had "Klee-type pathologies".

4. Small. Gave a probabilistic explanation of the "magic of simplex algorithm". Considering average probabilities. In computer science, computational complexity is defined in terms of worst case.

5. Kharmarker, from Bell Labs. Came up with new approach. Developed scaling to L.P.

2.14.3 Polytopes

Polytopes are central to L.P since polytopes are described by constraints. See handout "Polytopes" taken from textbook Barmish, Robust Control.

Polytope is convex hull of finite point set. These are the generators $v^1, \cdots, v^N$. Polytopes have extreme vertices. L.P. visits vertices. If the set is bounded, we call it polytope, else we call it polyhedron. So with linear inequalities constraints and bounded, we have polytopes.

Reader A linear function $J(u) = a^T u$ on polytope P achieves its Max. or Min. at an extreme point. We showed that the Max. of convex function is at a vertex. we can also show that Min. of concave is at a vertex. Linear functions are both concave and convex. QED.

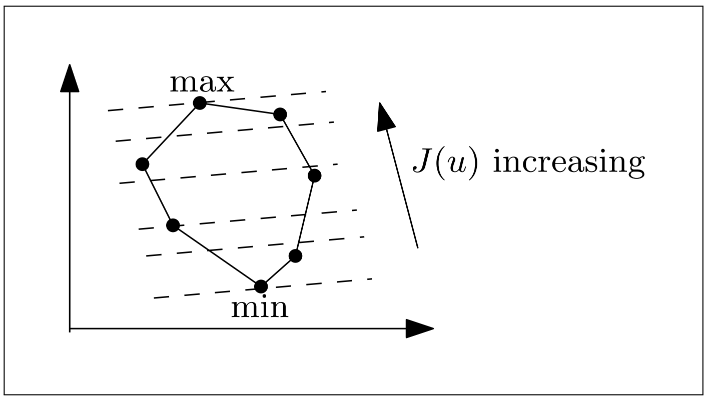


Figure 2.13: Increasing $J(u)$ diagram

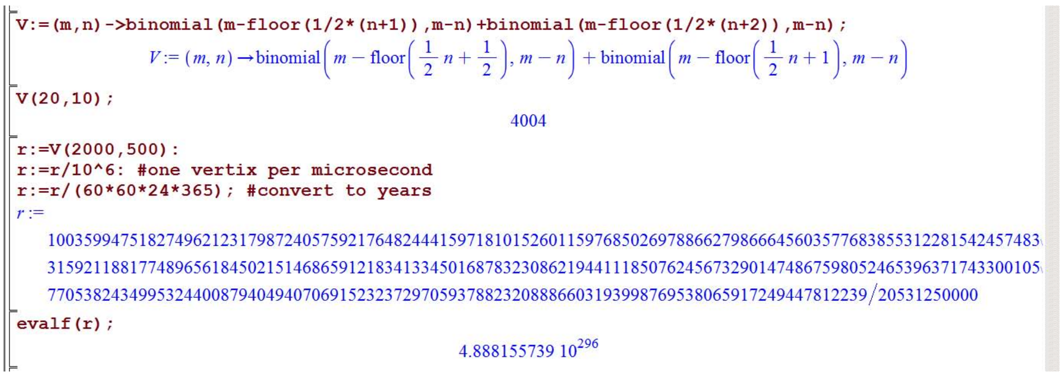
Often we do not have list of vertices. Need to first generate them. They are generated from the constraint inequalities.

How many vertices to search? McMullen's Upper Bound Theorem gives us the answer. Assuming we have m constraints and n variables, where $m \geq n$. Then

$$V(m,n) = \begin{pmatrix} m - \left\lfloor \frac{1}{2}(n+1) \right\rfloor \\ m-n \end{pmatrix} + \begin{pmatrix} m - \left\lfloor \frac{1}{2}(n+2) \right\rfloor \\ m-n \end{pmatrix}$$

Example for $m = 20, n = 10$ we get 4004 vertices. So this is a small problem.

Reader Consider $n = 500, m = 2000$ which is very modest in terms of current technology, assuming it takes 1 microsecond per vertex, then it will take order of 10^{296} years to search all the vertices.



```
V:=(m,n)->binomial(m-floor(1/2*(n+1)),m-n)+binomial(m-floor(1/2*(n+2)),m-n);
```

$$V := (m, n) \to \text{binomial}\left(m - \text{floor}\left(\frac{1}{2}\,n + \frac{1}{2}\right), m - n\right) + \text{binomial}\left(m - \text{floor}\left(\frac{1}{2}\,n + 1\right), m - n\right)$$

```
V(20,10);
                                   4004
r:=V(2000,500):
r:=r/10^6: #one vertix per microsecond
r:=r/(60*60*24*365); #convert to years
r :=
    1003599475182749621231798724057592176482444159718101526011597685026978866279866645603577683855312281542457483\
    3159211881774896561845021514686591218341334501687832308621944111850762456732901474867598052465396371743300105\
    770538243499532440087940494070691523237297059378823208886603193998769538065917249447812239/20531250000
evalf(r);
                             4.888155739 10^296
```

Figure 2.14: Verification of number of vertices using Maple

Up to last few years, L.P. was considered a completed research. But in the last 5–10 years, there is new L.P. research starting. Modern L.P. solvers use linear inequalities description as input. This is expressed and formulated as $Ax = b$. What if vertices or some vertices generating mechanism was given as input instead of the constraints themselves? How to convert the vertices to constraints? This is a difficult problem.

2.15 Lecture 15. Tuesday, March 8, 2016

2.15.1 Mechanism of linear programming

Today we will begin the mechanism of doing L.P. (linear programming). We know the extreme points is where the optimal will occur. But searching all extreme points is not practical for large N as shown before. One can take a whole course just on L.P. but here we will cover the main ideas. We basically have a linear objective function in u and linear constraints in u where u are the variables. This is called the raw L.P. formulation. This is converted to standard form L.P. and solved using the simplex method.

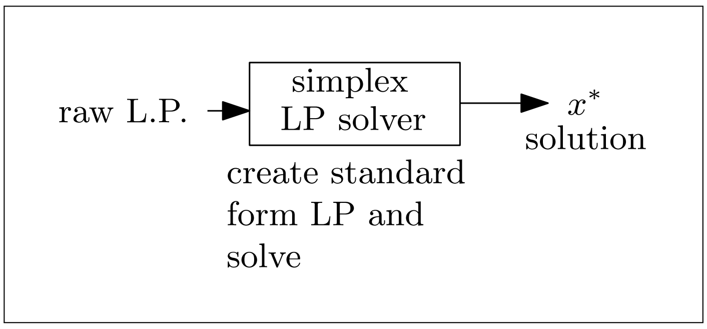


Figure 2.15: Simplex solver diagram

The solver finds the first vertex then in a clever way moves to another until it finds the optimal one. The solver solves two linear programming problems and these are:

1. First finds a feasible solution (basic)
2. moves from one vertex to another.

Standard form LP is

$$\boxed{\begin{array}{cc} \min & c^T x \\ st & Ax = b \end{array}}$$

Notice that solution might be infimum above. Example.

$$\begin{array}{cc} \min & x_1 \\ st & \begin{array}{c} x_2 \leq 5 \\ x_1 < 0 \end{array} \end{array}$$

The solution is $x_1 = -\infty$. This is closed by unbounded.

Ingredients of Linear programming are:

1. $n > m$ (number of variable is larger than number of constraints.). The matrix A is of order m by n. So A matrix is fat matrix and not thin.
2. No columns of A can all be zeros (non-degenerate).
3. Rank of A is m
4. $b \geq 0$
5. $x \geq 0$

What if we have some variables x_i which we want to be negative? We replace x_i with new variable $x_j = -x_i$. Now $x_j \geq 0$. Now in each place we have x_i which is negative and can't use, then we replace it with $-x_j$. This is now the same as before, but x_i is gone and replaced with $-x_j$ and x_j is now positive. So it is standard form.

At the end, when we obtain the solution, we replace x_j back to $-x_i$. (what about free variables?).

What if we have inequality in the raw L.P.? how to convert to equality for standard form? We use Slack variables and Surplus variables .

Example given $x_1 + 2x_2 - x_3 \leq 6$, then introduce new slack variable x_4 and rewrite the constraint as $x_1 + 2x_2 - x_3 + x_4 = 6$.

If we have constraint $x_1 + 2x_2 - x_3 \geq 6$, then we need surplus variable x_4. Rewrite as $x_1 + 2x_2 - x_3 - x_4 = 6$. Once we solve the LP problem and obtain x^*, we need to recover from this solution the actual variables of the raw LP (these are the u variables) and these do not contain any slack nor surplus variables.

2.15.2 Example, the sector patrol problem

See also handout sector patrol. The objective function is $E(T) = \frac{\frac{u_1}{3}}{10} + \frac{\frac{u_2}{3}}{5} = \frac{u_1}{30} + \frac{u_2}{15}$. Constraints are $u_1 \geq 0, u_2 \geq 0$ and

$$\begin{aligned} 2u_1 + 2u_2 &\geq 4 \\ 2u_1 + 2u_2 &\leq 10 \end{aligned}$$

And as per handout, we need to add this constraint in order to obtain a realistic solution

$$u_2 \geq 1.5u_1$$

The above is the raw LP. Convert to standard form, using x as variables. It becomes

$$\begin{aligned} 2x_1 + 2x_2 - x_4 &= 4 \\ 2x_1 + 2x_2 + x_3 &= 10 \\ -1.5x_1 + x_2 - x_5 &= 0 \end{aligned}$$

Where x_3, x_4, x_5 above were added to make it standard form. Writing it as $Ax = b$, the constraint equation is (we put the slack variables first by convention)

$$\begin{bmatrix} 2 & 2 & 1 & 0 & 0 \\ 2 & 2 & 0 & -1 & 0 \\ -1.5 & 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \\ 0 \end{bmatrix}$$

And $c^T x$ becomes (this is the objective function, notice we added the slack and surplus variables to it, but they are all zeros).

$$\begin{bmatrix} \frac{1}{30} & \frac{1}{15} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_4 \end{bmatrix}$$

Reader find common sense solution working in u_1, u_2 domain. (will do this next lecture).

2.15.3 Basic and Feasible solutions

We will define two solutions: The basic solution, and basic feasible solution.

A vector x is said to be basic solution if it solves $Ax = b$ and the non-zeros elements of x correspond to the linearly independent columns of A.

Reader Is there a basic infeasible solution?

2.16 Lecture 16. Thursday, March 10, 2016

Recall, the standard LP problem is

$$\begin{aligned} \min \quad & c^T x \\ st \quad & Ax = b \end{aligned}$$

We talked about transforming the problem from raw LP to standard LP. The patrol sector problem, solved using common sense graphical approach is given below

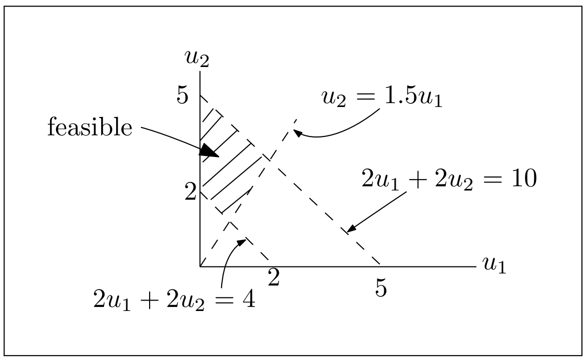


Figure 2.16: Patrol problem

The optimal u^* has to be at one of the vertices of the feasible region. It will be at the vertex shown

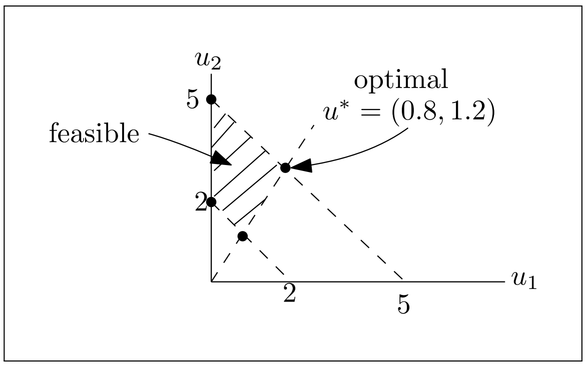


Figure 2.17: Patrol problem solution

Using Matlab, the above is solved as follows

$$\begin{bmatrix} 2 & 2 & 1 & 0 & 0 \\ 2 & 2 & 0 & -1 & 0 \\ -1.5 & 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \\ 0 \end{bmatrix}$$

And $c^T x$ becomes (this is the objective function, notice we add the slack and surplus variables to it, but they are all zeros).

$$\begin{bmatrix} \frac{1}{30} & \frac{1}{15} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_4 \end{bmatrix}$$

The code is

```
f=[1/30,1/15,0,0,0];
A=[2,2,1,0,0,;
   2,2,0,-1,0,;
   -1.5,1,0,0,-1];
```

```
b=[10,4,0];
[X,FVAL,EXITFLAG,OUTPUT]=linprog(f,[],[],A,b,zeros(size(f)),[])
```

Result of above run

```
Optimization terminated.
X =
0.799999999994766
1.20000000008299
5.99999999984448
1.55520543353025e-10
9.08450336982967e-11
FVAL =
0.106666666672025
EXITFLAG =
1
OUTPUT =
iterations: 7
algorithm: 'interior-point-legacy'
cgiterations: 0
message: 'Optimization terminated.'
constrviolation: 8.88178419700125e-16
firstorderopt: 5.07058939341966e-12
```

In the above, we only need to map $x(1)$ to u_1 and $x(2)$ to u_2 to read the result. We see that Matlab result matches the graphical solution.

definition For LP, we say $\boldsymbol{x}$ is feasible if $\boldsymbol{x}$ satisfies the constraints.

For example, for the sector patrol, let U be the feasible set in $\Re^2$ (raw LP). However, in standard LP, the feasible set is in $\Re^5$.

Reader Obtain feasible set in $\Re^5$. Obtain feasible point with either x_3, x_4, x_5 nonzero.

Basic solution A vector x is basic solution if the non-zero components of x corresponds to the linearly independent columns of A. We do not require feasibility to be basic solution).

The difference in LP and standard $A\boldsymbol{x} = \boldsymbol{b}$ solution we have seen before many times in linear algebra, is that in LP, we want to solve $A\boldsymbol{x} = \boldsymbol{b}$ but with $\boldsymbol{x} \geq 0$ and at same time have $\boldsymbol{x}$ be optimal. This what makes LP different from standard methods of solving this problem.

The algorithm takes a solution which is feasible and makes it feasible basic solution. Then after that, we move from one basic feasible solution to another basic feasible solution while at the same time making $J(u)$ smaller until it reaches the optimal value.

Reader LP has at least one basic solution.

A has m linearly independent columns, since it has rank m.

$$A\boldsymbol{x} = \boldsymbol{b}$$

$$\begin{bmatrix} A_{basic} & A_{notbasic} \end{bmatrix} \begin{bmatrix} x_{basic} \\ x_{notbasic} \end{bmatrix} = \begin{bmatrix} b_{basic} \\ b_{notbasic} \end{bmatrix}$$

2.16.1 Linear programming feasible and basic solutions

Theorem Suppose LP has feasible solution, then it has a basic feasible solution. Remember: $\boldsymbol{x}$ is feasible if it is in the feasible region (satisfies constraints), and $\boldsymbol{x}$ is basic solution if the non-zero elements of $\boldsymbol{x}$ correspond to linearly independent columns of A.

Proof say $\boldsymbol{x}$ is feasible. Let $a^1, a^2, \cdots, a^p$ denote columns of A corresponding to nonzero entries of $\boldsymbol{x}$. Without loss of generality, say the first p columns of A. (we can always rearrange A to make it so). There are two cases:

case one The a^i above are linearly independent. We are done. x is therefore basic by definition.

case two The a^i are linearly dependent. Therefore there exist scalars y_i, not all zero, such that $\sum_{i=1}^{p} y_i a^i = 0$. (this is the definition of linearly dependent columns).

Now we do the squeezing process. For $\varepsilon > 0$ define vector η^ε with components

$$\eta^\varepsilon = \begin{matrix} x_i - \varepsilon y_i & \text{for } i \leq p \\ 0 & i > p \end{matrix}$$

Reader η^ε is feasible for small ε. Hence

$$A\eta^\varepsilon = A\boldsymbol{x} - \varepsilon A \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Let $\varepsilon = \min\left\{\frac{x_i}{y_i}; y_i > 0\right\}$. Reader η^ε is basic and has at least one more zero entry than $\boldsymbol{x}$. So now $\boldsymbol{x}$ has $p-1$ columns of A corresponding to non-zero entries in $\boldsymbol{x}$. Continuing this process, we keep finding other basic feasible solutions.

example

$$\begin{bmatrix} 1 & 2 & 1 & 2 & 0 \\ 1 & 1 & 1 & 2 & 1 \\ 2 & -1 & 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 4 \end{bmatrix}$$

Let starting $\boldsymbol{x} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ This is feasible since it satisfies $A\boldsymbol{x} = \boldsymbol{b}$ with $\boldsymbol{x} \geq 0$. But not basic, since the last 3 columns are not linearly independent. (the last three columns of A. Since these are the ones that correspond to non-zero elements of $\boldsymbol{x}$. we now write

$$y_3 a^3 + y_4 a^4 + y_5 a^5 = 0$$

Where a^3, a^4, a^5 represent the last three columns of A and y^i are the scalars we want to solve for. Solving, gives

$$\begin{aligned} y_3 &= 2 \\ y_4 &= -1 \\ y_5 &= 0 \end{aligned}$$

Hence, first find $\varepsilon = \min\left\{\frac{x_i}{y_i}; y > 0\right\}$, which we find to be $\varepsilon = \frac{1}{2}$. Now we find

$$\boldsymbol{x}^{new} = \eta^\varepsilon = \begin{bmatrix} x_1 - \varepsilon y_1 \\ x_2 - \varepsilon y_2 \\ x_3 - \varepsilon y_3 \\ x_4 - \varepsilon y_4 \\ x_5 - \varepsilon y_5 \end{bmatrix} = \begin{bmatrix} 0 - \frac{1}{2}(0) \\ 0 - \frac{1}{2}(0) \\ 1 - \frac{1}{2}(2) \\ 1 - \frac{1}{2}(-1) \\ 1 - \frac{1}{2}(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

Since now a^4, a^5 are linearly independent (these are the fourth and fifth columns of A) and these correspond to the non zero of the new $\boldsymbol{x} = \eta^\varepsilon$, then the new $\boldsymbol{x}$ is basic and feasible.

So we have started with feasible solution, and from it obtained a basic feasible solution. This is not the final optimal solution x but we repeat this process now.

2.17 Lecture 17. Tuesday, March 15, 2016

If we have a feasible solution, we can obtain a basic feasible solution from it using the squeezing method. We have not talked about optimality yet. The next thing to consider is optimality. We will proof if we have an optimal feasible solution, then by obtaining a basic solution from it, the basic solution will remain optimal. This is called the optimality theorem. Then we will talk about extreme points.

2.17.1 Optimality theorem

If an optimal feasible solution exist, then an optimal feasible and basic solution exist as well.

Proof Suppose x is optimal and feasible. If x is basic, we are done. If not, now we need to do the squeeze process to make it basic, but now we have to do the squeeze making sure it remains optimal. Say $a^1, a^2, \cdots, a^p$ are the columns associated with non-zero entries in x. As before, we way, WLOG these are the first p columns in A. Hence there exist scalars y_i, not all zero, such that $\sum_{i=1}^{p} y_i a^i = 0$. Define η^ε for scalar ε such that

$$\eta^\varepsilon = \begin{matrix} x_i - \varepsilon y_i & \text{for } i \leq p \\ 0 & i > p \end{matrix}$$

Reader For small ε, say $|\varepsilon| \leq \delta$, then η^ε is still feasible. Claim: For ε suitable small, η^ε is optimal. It suffice to show that $c^T y = 0$ with $y_i = 0$ for $i > p$. This being the case, then $c^T \eta^\varepsilon = c^T x = J^*$. By contradiction: Say $c^T y \neq 0$. Let $\varepsilon = \delta sgn\left(c^T y\right)$. Let us show that η^ε is better than x. This contradicts optimality. Now

$$\begin{aligned} c^T \eta^\varepsilon &= c^T \left(x - \varepsilon y\right) \\ &= c^T x - c^T \left(\delta sgn\left(c^T y\right)\right) y \\ &= c^T x - \delta sgn\left(c^T y\right)\left(c^T y\right) \\ &= J^* - \delta \left|c^T y\right| \\ &< J^* \end{aligned}$$

This contradicts optimality. QED.

Now we will talk about extreme points. Extreme points and basic feasible solution are the same thing.

2.17.2 The extreme point theorem

Let $P = \{x \in \Re^n, x \geq 0, Ax \leq b\}$. This polyhedron is the feasible set. The set of extreme points of P are the basic feasible solution.

Proof See handout extreme send today. We need to get to the first feasible solution. This can be hard to obtain. Once we find a feasible solution, then we use it to find the first basic feasible solution, and from them we repeat the process (using the squeeze method). As we move from one basic feasible solution to another, we do this by making $J(u)$ smaller.

Example Consider LP with constraints

$$\begin{aligned} x_i &\geq 0, i = 1, 2, 3 \\ 2x_1 + 3x_2 &= 1 \\ x_1 + x_2 + x_3 &= 1 \end{aligned}$$

Note that $x_1 + x_2 + x_3 = 1$ is common in LP optimization. It is called unit simplex. Here is a plot of the above.

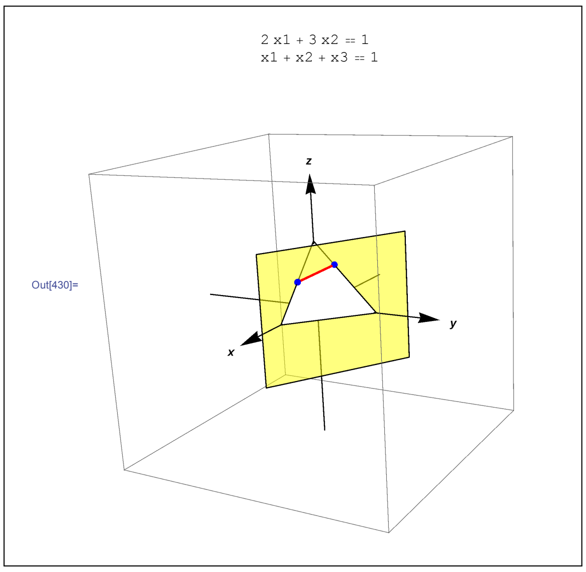


Figure 2.18: Unit simplex

Reader Plane for $2x_1 + 3x_2 = 1$ intersect the unit simplex else no feasible region exist. So there are two basic feasible solutions at $\left(0, \frac{1}{3}, \frac{2}{3}\right)$ and $\left(\frac{1}{2}, 0, \frac{1}{2}\right)$. These are the two red points shown in the above plot.

2.17.3 Mechanism the simplex method

We will use the sector patrol problem to show the mechanisms of simplex.

$$Ax = b$$

$$\begin{bmatrix} 2 & 2 & 1 & 0 & 0 \\ 2 & 2 & 0 & -1 & 0 \\ -1.5 & 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \\ 0 \end{bmatrix}$$

Step 1. Form partial tableau.

$$\begin{bmatrix} A & b \end{bmatrix} = \begin{bmatrix} 2 & 2 & 1 & 0 & 0 & 10 \\ 2 & 2 & 0 & -1 & 0 & 4 \\ -1.5 & 1 & 0 & 0 & -1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}$$

We need to identify an identify matrix. By row operations we obtain

$$\begin{bmatrix} r_1 \\ -r_2 \\ -r_3 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 1 & 0 & 0 & 10 \\ -2 & -2 & 0 & 1 & 0 & -4 \\ 1.5 & -1 & 0 & 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} r_1' \\ r_2' \\ r_3' \end{bmatrix} \tag{1}$$

Now we can read out $\boldsymbol{x}^1$ and we see that $x_3 = 10, x_4 = -4, x_5 = 0$. These are the entries in x which corresponds to the columns of the unit matrix inside A. All other entries in x are assumed zero. Hence

$$\boldsymbol{x}^1 = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 10 \\ -4 \\ 0 \end{bmatrix}$$

Note this is not feasible but basic. Now we go to next step. We have to remove an entry from x^1 and move in its place another entry. Let us pick x_3 to kick out and move in x_1. By row operations applied to (1) we obtain

$$\begin{bmatrix} \frac{r_1'}{2} \\ r_2' + 2r_1'' \\ r_3' - 1.5r_1'' \end{bmatrix} = \begin{bmatrix} 1 & 1 & \frac{1}{2} & 0 & 0 & 5 \\ 0 & 0 & 1 & 1 & 0 & 6 \\ 0 & -2.5 & -0.75 & 0 & 1 & -7.5 \end{bmatrix} \rightarrow \begin{bmatrix} r_1'' \\ r_2'' \\ r_3'' \end{bmatrix}$$

Hence the new identity matrix is a^1, a^4, a^5 and therefore $x_1 = 5, x_4 = 6, x_5 = -7.5$, These are the entries in x which corresponds to the columns of the unit matrix inside A. All other entries in x are assumed zero. Hence

$$x^2 = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 0 \\ 6 \\ -7.5 \end{bmatrix}$$

This is the second basic x we found.

Reader Find basic solution via pivoting row operations. Now we want to redo the above, with feasibility in mind which we did not consider above when moving elements out and selecting which one to move in.

Example

$$Ax = b$$

$$\begin{bmatrix} 1 & 0 & 0 & 4 & 3 & 2 \\ 0 & 1 & 0 & 2 & -1 & 2 \\ 0 & 0 & 1 & -1 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} A & b \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 4 & 3 & 2 & 1 \\ 0 & 1 & 0 & 2 & -1 & 2 & 2 \\ 0 & 0 & 1 & -1 & 2 & 1 & 3 \end{bmatrix}$$

Now we need to start with feasible solution. Last example we did not care about this, but now we need the first solution to be feasible. Here $x_1 = 1, x_2 = 2, x_3 = 3$ (read out, from the identity matrix, since the first three columns are linearly independent). We we use the squeeze process, to decide which one to kick out and which to move in. Let us for now choose arbitrarily x_4 (fourth column) to move in and we have to kick out a column. Need ε^* to decide.

$$\varepsilon^* = \min \left\{ \frac{1}{4}, \frac{1}{2} \right\} = \frac{1}{4}$$

Hence it is the first column to kick out. Remember that when doing the above to determine which x to kick out, we only divide by those entries in the column which are positive. If there is a negative entry, then do not use it. That is why in the above, we did not write $\varepsilon^* = \min \left\{ \frac{1}{4}, \frac{1}{2}, \frac{3}{-1} \right\}$. We will continue next lecture.

2.18 Lecture 18. Thursday, March 17, 2016

Class planning:

1. special problem towards end of course
2. Test 2 after spring break. Up to and including LP.
3. HW 6 will be send soon, due right after spring break.

Exercise to do

A good exercise, to be done by hand, is the following: For the sector patrol problem which we considered in class, solve the Phase One LP to obtain a basic feasible solution.
Then, use this first basic feasible solution as a starting point for the original LP and solve it via a sequence of tableau. Since you already know the answer, you will get feedback whether your calculations produce the right result.

2.18.1 Simplex method examples

We still need to know how to find first feasible solution. In the following example

x_1	x_2	x_3	x_4	x_5	x_6	b
1	0	0	2	4	6	4
0	1	0	1	2	3	3
0	0	1	-1	2	1	1

A solution which is basic and feasible is $x_1 = 4, x_2 = 3, x_3 = 1$ and all others $x_i = 0, i = 4 \cdots 6$.

$$x = \begin{pmatrix} 4 \\ 3 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

This was just read out from the identity matrix in the first 3 columns above. Let us now assume we want fourth column to be in the basis. We have to remove one of the current columns in the basis in order to let another column in. $\min\left\{\frac{4}{2}, \frac{3}{1}\right\} = 2$. This means the first row is the pivot row. Notice we do not consider $\frac{1}{-1}$ when doing the minimum operation. Any negative value in a column is bypassed. Now we know that first row is pivot row and that we want fourth column in. This is all what we need to go to the next step. We know need to normalize entry $(1,4)$ to one. (before it was 2). After normalizing the pivot row (by dividing the whole row by 2) we obtain

x_1	x_2	x_3	x_4	x_5	x_6	b
$\frac{1}{2}$	0	0	1	2	3	2
0	1	0	1	2	3	3
0	0	1	-1	2	1	1

Only now we start applying row operations, with row one as pivot row, we make all other entries in fourth column below $(1,4)$ zero, This gives the following

	x_1	x_2	x_3	x_4	x_5	x_6	b
r_1	$\frac{1}{2}$	0	0	1	2	3	2
$r_2 - r_1$	$-\frac{1}{2}$	1	0	0	0	0	1
$r_3 + r_1$	$\frac{1}{2}$	0	1	0	4	4	3

Now that we have a new identity matrix, we read out the new solution which is $x_2 = 1, x_4 = 2, x_3 = 3$ and all other x entries zero.

$$x = \begin{pmatrix} 0 \\ 1 \\ 3 \\ 2 \\ 0 \\ 0 \end{pmatrix}$$

We will now revisit this, with optimality in mind. This means we need to know which column to bring into the basis. The question is, which x_i to bring in. Set up this tableau[1]

$$\overbrace{\begin{pmatrix} 1 & \cdots & 0 & a(1,m+1) & \cdots & a(1,n) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \cdots \\ 0 & \cdots & 1 & a(m,m+1) & \cdots & a(m,n) \end{pmatrix}}^{A} \overbrace{\begin{pmatrix} y(1,0) \\ y(2,0) \\ \vdots \\ y(m,0) \end{pmatrix}}^{b}$$

Therefore, the current feasible and basic solution is $x_1 = y(1,0), x_2 = y(2,0), \cdots, x_m = y(m,0)$. All other $x_i = 0$. We need now to do a feasibility preserving perturbation.

Allow $x_{m+1}, x_{m+2}, \cdots, x_n$ to be part of the solution.

$$\begin{aligned} x_1 + \overbrace{\sum_{i=m+1}^{n} a(1,i)\, x_i}^{\text{we allow this in}} &= y(1,0) \\ x_2 + \sum_{i=m+1}^{n} a(2,i)\, x_i &= y(2,0) \\ &\vdots \\ x_m + \sum_{i=m+1}^{n} a(m,i)\, x_i &= y(2,0) \end{aligned}$$

Now $x = \begin{pmatrix} x_1 & x_2 & \cdots & x_m & 0 & \cdots & 0 \end{pmatrix}^T$ is still feasible, but no longer basic. We know that $J = c^T x$, hence

$$J = c_1 \left(y(1,0) - \sum_{i=m+1}^{n} a(1,i)x_i \right) + c_2 \left(y(2,0) - \sum_{i=m+1}^{n} a(2,i)x_i \right) + \cdots + c_m \left(y(m,0) - \sum_{i=m+1}^{n} a(m,i)x_i \right) + c_{m+1}x_{m+1} + \cdots + c_n x$$

Hence

$$J = \overbrace{\sum_{i=1}^{m} c_i y(i,0)}^{\text{current } J_0 \text{value}} - c_1 \sum_{i=1}^{m} a(1,i)x_i - c_2 \sum_{i=1}^{m} a(2,i)x_i - \cdots - c_m \sum_{i=1}^{m} a(m,i)x_i + c_{m+1}x_{m+1} + \cdots + c_n x_n$$

Therefore

$$J = J_0 - \left(-c_{m+1} + c_1 a(1,m+1) + c_2 a(2,m+1) + \cdots + c_m a(m,m+1)\right) x_{m+1} - \left(-c_{m+1} + c_1 a(2,m+1) + \cdots + c_m a(m, m-\right.$$

Define cost coefficients, for $j = m+1, \cdots, n$

$$r_j = c_j - \sum_{i=1}^{m} c_i a(i,j)$$

[1] in class, we used $y(1,m+1)$ etc.. for entries in A matrix. I changed it in these notes to $a(1,m+1)$ etc... not to confuse myself with the RHS $y(1,0)$, etc.. entries in the b vector.

Hence

$$J = J_0 + \sum_{j=m+1}^{n} r_j x_j$$

To minimize J we need to pick the most negative r_j. This tells us which x_j we need to bring into the basis in order to reduce J. Now we add extra row (last row) which is $J(u)$ to the table, to keep cost in it. Here is the table with the cost coefficient row added

$$\overbrace{\begin{pmatrix}1 & \cdots & 0 & a(1,m+1) & \cdots & a(1,n)\\ \vdots & \ddots & \vdots & \vdots & \ddots & \cdots \\ 0 & \cdots & 1 & a(m,m+1) & \cdots & a(m,n) \\ 0 & 0 & 0 & r_{m+1} & \cdots & r_n\end{pmatrix}}^{A}\overbrace{\begin{pmatrix} y(1,0)\\ y(2,0)\\ \vdots \\ -J_0\end{pmatrix}}^{b}$$

state street vendor example Selling rings and bracelets. Ring has 3 oz. gold., 1 oz. silver. Bracelet has 1 oz. gold, 2 oz. silver. Profit on ring is \$4 and profit on bracelet is \$5. Initially we have 8 oz. gold and 9 0z silver. How many rings and bracelets to produce to maximize profit? This is the heart of many production problems. Note: we need integer LP to solve this. But if the quantities are very large, it will make little difference. So for now we can ignore the issue of integer programming.

$$\begin{aligned} u_1 &= \text{number of rings} \\ u_2 &= \text{number of bracelets} \\ J(u) &= 4u_1 + 5u_2 \end{aligned}$$

Since we want to maximize this, then

$$J(u) = -4u_1 - 5u_2$$

With $u_i \geq 0$. Constraints are $3u_1 + u_2 \leq 8$ and $u_1 + 2u_2 \leq 9$. We convert to standard LP with slack variables and make the first table

	x_1	x_2	x_3	x_4	b
row 1	3	2	1	0	8
row 2	1	2	0	1	9
$J(u)$	-4	-5	0	0	0

The first basic feasible solution is $x_3 = 8, x_4 = 9$. To decide which column to bring in, we see the most negative is column 2 (last row is -5). To find the pivot row, we see it is first row since $\min\left\{\frac{8}{2}, \frac{9}{2}\right\} = \{4, 4.5\} = 4$. Now we do the first stage.

Reader obtain the following

	x_1	x_2	x_3	x_4	b
row 1	2.5	0	1	-0.5	3.5
row 2	0.5	1	0	0.5	4.5
$J(u)$	-1.5	0	0	2.5	22.5

Hence $x_2 = 4.5, x_3 = 3.5$. Since there is still a negative entry in the last row we need to repeat the process again. We Keep doing this until there are no negative entries in the last (third) row.

Now we will now talk about how to find the first basic feasible solution. There are two cases. If the number of slack variables is m then first basic feasible solution can be read out. This means there is no phase one LP. Case two. The number of slack variables is $z < m$. Now we need to solve the first phase LP. Then use its result to solve second phase LP. In phase one, we introduce new artificial variables y_i as many as $m - z$ and new artificial cost function $J(y)$ which we want to minimize to zero.

See HW6, first problem for an example of how to solve first phase LP.

2.19 Lecture 19. Tuesday, March 22, 2016 (No class)

No class.

2.20 Lecture 20. Thursday, March 24, 2016 (No class)

No class.

2.21 Lecture 21. Tuesday, March 29, 2016

Coming up soon is the special problem. It is like one HW but can count up to two HW's weight. Note: Possible rescheduling for April 6, 2016 remain in place.

2.21.1 Second exam keywords

1. Test two will be more application oriented.
2. local minimum. Strong and weak. We had sufficient conditions. Gradient, Hessian.
3. If we have convexity, we can do much better. If not, we need iterative algorithms. Line search is central to iterative search. Step size, optimal step size.
4. we looked at steepest descent with and without optimal step size. For simple problems we can easily find optimal step size.
5. We talked about convergence for iterative algorithms. For steepest descent we talked about $E\theta^k$.
6. We talked about generalized Newton-Raphson. We talked about quadratic convergence. Conjugate direction method has quadratic convergence (we proved this). It will converge in N steps or less, where N is A matrix size. Newton-Raphson will converge in one step for quadratic function. We also talked about superlinear convergence.
7. This brought us to the end of iterative algorithms. Then we went to Linear programming. The number of vertices is large. So trying to check them all is not possible. Polytope is central to LP. The basic theory of LP
   (a) Feasible solutions
   (b) Basic solutions
   (c) basic and feasible solutions
   (d) The method of squashing using η^ϵ
   (e) Basic feasible ⇔ extreme points.
   (f) Relative cost coefficients.
8. Then we talked about simplex algorithm.

Now we will start on today lecture. Often a problem is given to you, but it is not an LP problem. Sometimes it is not obvious how to convert it to an LP problem. Sometime we need algebra or cleaner reformulation of the problem to make it an LP problem. For example, in HW6, we had a min-max problem but it is was possible to convert it to an LP problem. See key solution for HW6.

2.21.2 Application of Linear programming to control problems

Another application area for LP is control. Example is the minimum fuel problem. In this, we want to go from some state to final state with minimum control effort. The control effort is generic name which can mean many things depending on the problem itself. We also want to do this in minimal time. We begin with the discrete state equation

$$x(k+1) = Ax(k) + Bu(k)$$

With $x(0)$ given as the initial state. In the above, x is an $n \times 1$ vector, and A is $n \times n$ and B is $n \times m$ where m is number of inputs, and u is $m \times 1$ input vector.

We want to select $u(k)$ sending $x(0)$ to some target x^* at some future time $k = N$. With N being minimal, and control effort minimum. Assume for now that u is scalar, which means one input, then an energy measure is

$$\sum_{k=0}^{N-1} |u(k)|^2$$

On the other hand, a peak measure is

$$\max\{u(k), k = 0, \dots, N-1\}$$

But we will consider the fuel measure given by

$$\sum_{k=0}^{N-1} |u(k)|$$

We will use fuel measure in the LP problem. The constraint is

$$|u(k)| \leq U \tag{*}$$

Which says that control is bounded. Note that is (A, B) is controllable, we can get from initial state to final state in one step if we want, but the effort will be very large. We also want $x(N) = x^*$. The above two are the constraints in this problem. The objective function is

$$J(u) = \sum_{k=0}^{N-1} |u(k)|$$

Therefore

$$\begin{aligned}
x(1) &= Ax(0) + Bu(0)\\
x(2) &= A^2 x(0) + ABu(0) + Bu(1)\\
x(3) &= A^3 x(0) + A^2 Bu(0) + ABu(1) + Bu(2)\\
&\vdots\\
x(N) &= \underbrace{A^N x(0) + \sum_{k=0}^{N-1} A^{N-1-k} Bu(k)}_{x^* \text{This is the linear constraint}}
\end{aligned}$$

Now we rewrite the constraint $|u(k)| \leq U$ as

$$u(k) = u_p(k) - u_n(k)$$

with $u_p(k), u_n(k)$ being positive. The objective function becomes (where we now put N as parameter, to say this is for a specific value of N

$$\begin{aligned}
J_N(u) &= \sum_{k=0}^{N-1} |u(k)|\\
&= \sum_{k=0}^{N-1} u_p(k) + u_n(k)
\end{aligned}$$

Equation $*$ above becomes

$$\begin{aligned}
u_p(k) &\leq U\\
u_n(k) &\leq U
\end{aligned}$$

So minimizing $J_N(u)$ is now an LP problem in $2N$ raw variables (we still need to add the needed slack variables). So by doubling the number of variables, we were able to convert this control problem to an LP problem. Let

$$N^* = \inf\{N : LP_N \text{ feasible}\}$$

Reader Argue that l^∞ measure also lead to an LP problem. l^∞ measure is $\max\{u(0), u(1), \dots\}$.

Example Let $A = \begin{pmatrix} 1 & 1-e^{-T} \\ 0 & e^{-T} \end{pmatrix}$ where $T = 1$ is the sample time. And let $B = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. The bound on $u(k) = 1$. This means $U = 1$. Let $x(0) = \begin{pmatrix} -40.91 \\ 43.50 \end{pmatrix}$ and let the target $x^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$. Find u^* and N^*. Running LP for different values of N from $1, \dots, 4$ we find the first feasible solution at

$N = 4$. These is the resulting optimal effort $u(k)$

$$
\begin{aligned}
u(0) &= -0.3009\\
u(1) &= -1\\
u(2) &= -0.2999\\
u(3) &= -1
\end{aligned}
$$

And the corresponding optimal objective function $J^* = 2.6008$. In the above, we have priorities on minimal time first.

2.21.3 Starting dynamic programming

Now we will start on the next topic, which is dynamic programming. This involves discrete decisions. In this course we will cover only discrete dynamic programming and not continuous dynamic programming. We will be making sequential decisions in time. For example, if $u(k) = \{-1, 0, 1\}$ then the decision tree will look like

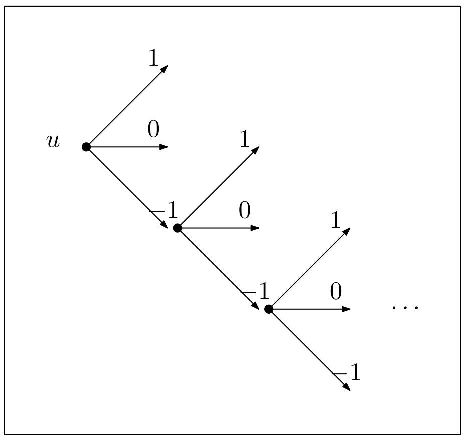


Figure 2.19: decision tree

We will get tree with potentially large number of branches. Combinatories arise. We get the curse of dimensionality problem again. For dynamic programming, Bellman is considered the person who originated the subject.

2.22 Lecture 22. Thursday, March 31, 2016. Second midterm exam

Exam.

2.23 Lecture 23. Tuesday, April 5, 2016

For steepest descent problem, with optimal step size, max is 1, need to use $\|u^{k+1} - u^k\| \leq 1$.

Next we will have special problem. Expect it next week.

Back to dynamic programming. Bellman secret is simple but theory is powerful. Main things to take from this course are

1. Linear programming.
2. Iterative solutions to optimization problems
3. dynamic programming

2.23.1 First dynamic programming problem, trip from NY to SFO

Suppose we want to take trip from NY to San Francisco, such that the total toll is minimum. We also must go west at each step we take once we start from NY. Not allowed to go east direction, even if the cost might be lower.

This diagram shows the possible routes and the cost (toll) for each segment.

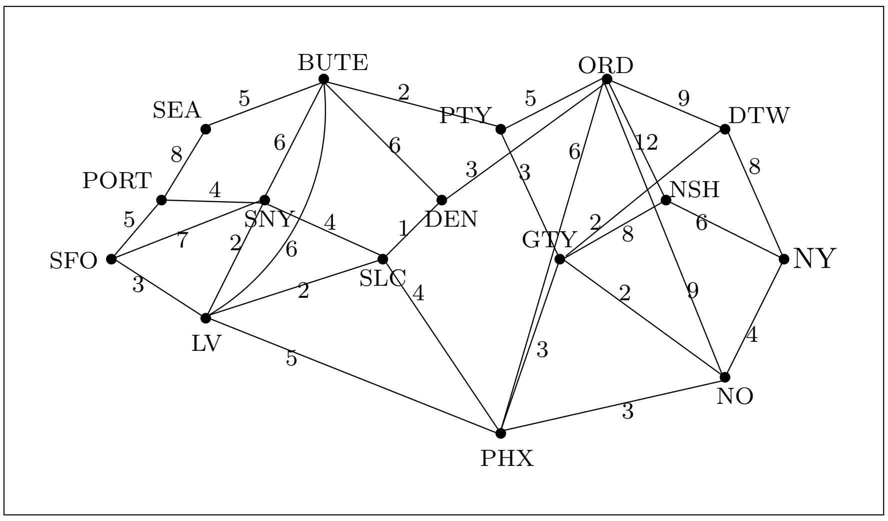


Figure 2.20: NY to SF tree one

Dynamic programming is now used to find optimal route in the above problem. (i.e. the route with least toll (cost) from NY to SFO). Dynamic programming is based on what if decisions. Instead of starting from NY and trying every possible route, we instead start backwards, and ask, what if we were in PORT, which route would we take?

Clearly the only route PORT to SFO with cost of 5 exist. Then we ask, what if we were in LV, which route would we take? we see it is LV to SFO with cost of 3. Then we ask what if we were in SNY? Then since LV had cost 3, then the route SNY → LV → SFO would be the one to take, with cost of $2 + 3 = 5$. Each time we find the cost from one city to SFO, we label the city with this cost. We keep moving back to the east, doing the same. When we arrive all the way to JFK, then we see that the lowest cost is

$$J^* = \text{JFK} \rightarrow \text{NO} \rightarrow \text{PHX} \rightarrow \text{LV} \rightarrow \text{SFO}$$

The following diagram shows the route above, with the cost of moving from each city to SFO given next to each city name on the diagram.

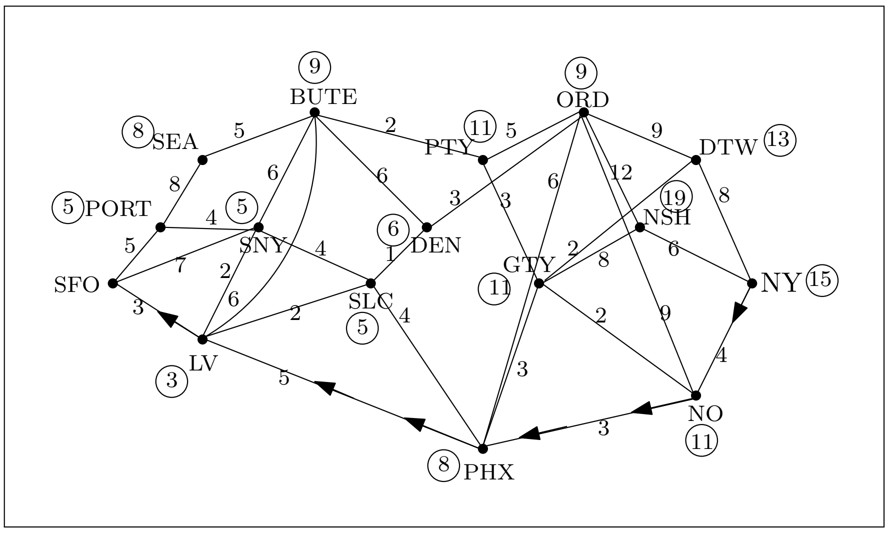


Figure 2.21: NY to SF tree two

If we were to solve the above problem using direct evaluation the number of computations is of order (assuming even n is $\frac{(n-1)!n!}{\frac{n}{2}!\frac{n}{2}!}$ while with dynamic programming method as explained above, it is $\frac{n^2}{2}+n$. For 20 cities, this given 220 for dynamic programming compare to over one million computation for the direct approach (trying all the possible routes).

We will only consider discrete dynamic programming. This is an optimization problem. The variables are not continuous. The variable take in discrete fixed values of choices each time. These applications are useful for integrate programming problems. An integer programming problem is much harder than continuous ones with much larger complexity.

When we are given an integer programming problem which is hard, we can try to approximate it to continuous programming problem and solve it more easily that way. For example

$$\min_x c^T x \quad \text{subject to} \quad Ax = b$$

where x is allowed to be integers, is a hard problem. But if we relax it and allow x to take any value so that the problem becomes continuous, then it will become much easier to solve using Linear Programming.

There are papers written on when we can approximate integer programming problems as continuous.

We will be making sequential decisions. $u_0, u_1, \dots, u_{N-1}$ i.e. N decisions where $u \in \mathbb{R}^n$. If we are given a problem which is not sequential, we can treat it as one for this purpose. For example, the car toll problem above, we formulate it to sequential but we did not have to do this. But the final answer J^* should come out the same no matter how it was formulated of course.

There will be states $x(k)$, $k = 0, 1, \dots, N$ where $x(N)$ is the terminal state. The constraints are state dependent. Because it depends on where we are when making the decision. For the car toll problem above, the decision depended on which city we were in. We denote the decision as $u(k) \in \Omega$ and the state equation is

$$x(k+1) = f(x(k), u(k), k)$$

and the objective function is

$$J(u) = \underbrace{\sum_{k=0}^{N-1} J(x(k), u(k), k)}_{\text{stage cost}} + \underbrace{\psi(x(N))}_{\text{terminal cost}}$$

The terminal cost, is a cost applied once we reach the terminal state.

2.23.2 Subproblem in dynamic programming

Now we need to define a subproblem. Notion of subproblem: Will begin at $k = L$ and will be in state $x(L)$. The cost when in state L is therefore

$$\boxed{J_L(u) = \sum_{k=L}^{N-1} J(x(k), u(k), k) + \psi(x(N))}$$

Suppose u^* is the optimal decision when we are at state $x(k)$. Let x^*k be the optimal trajectory from $x(k)$. i.e. $x^*(k)$ is corresponding state path beginning at given $x(0)$. Hence

$$x^*(k+1) = f(x^*(k), u^*(k), k), \ \ k = 0, 1, \dots, N-1$$

2.23.3 Bellman principle of optimality

If the subproblem begins at $x(L) = x^*(L)$ i.e. we being subproblem along optimal trajectory of the original problem, then $u^*(L), u^*(L+1), \dots, u^*(N-1)$ is optimal for the subproblem.

What all this means, is that if the subproblem is optimal, then its trajectory has to be part of the overall problem optimal trajectory. An optimal subproblem, can not become sub-optimal when viewed as part of the main problem.

2.24 Lecture 24. Thursday, April 7, 2016 (No class)

No class.

2.25 Lecture 25. Tuesday, April 12, 2016

The first part of the lecture was on describing the special problem we have to do. This is described in the special problem HW itself included in HW chapters, under special problem.

2.25.1 Dynamic programming state equation

Now we go back to dynamic programming. The state equation is

$$x(k+1) = f(x(k), u(k), k) \qquad k = 0, 1, \cdots N \quad \text{and} \quad u(k) \in \Omega_k$$

And the objective function is

$$J = \Psi(x(N)) + \sum_{k=0}^{N-1} J_k(x(k), u(k))$$

Where $\Psi(x(N))$ is the cost of the terminal stage.

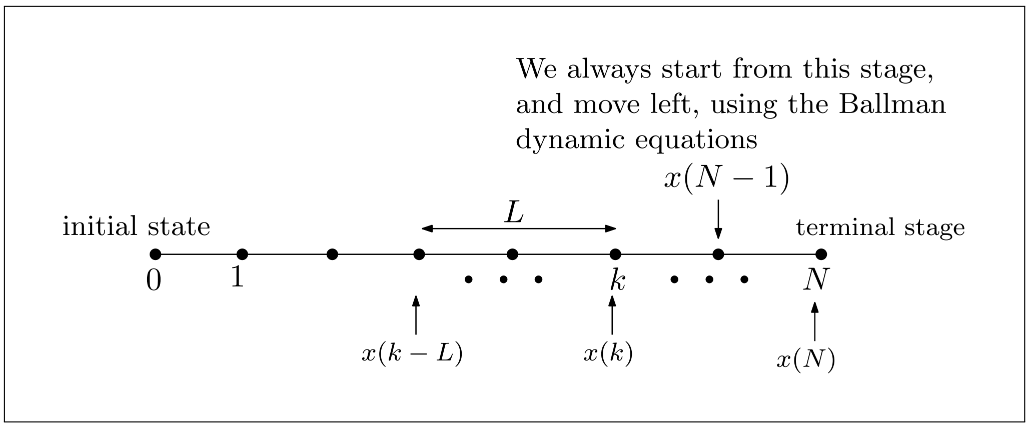

We always start from this stage, and move left, using the Ballman dynamic equations
$x(N-1)$
initial state
L
terminal stage
0
1
k
N
$x(k-L)$
$x(k)$
$x(N)$

Figure 2.22: Showing dynamic programming block diagram

2.25.2 Subproblems and principle of optimality (POO)

A subproblem is defined at intermediate point.

P.O.O. (principle Of optimality): if initial state of a subproblem is on the optimal trajectory of original problem, then the subproblem is also optimal.

Proof by contradiction Let

$$u(k) = \begin{cases} u^*(k) & k = 0, \cdots, L-1 \\ u^*_{new}(k) & k = L, \cdots N-1 \end{cases}$$

Plug the above in the original problem. We will get a suboptimal solution.

Translating POO. to dynamic programming.

$$\boxed{I(x(N-1), 1) = \min_{u(N-1)\in\Omega_{N-1}} \left\{ \overbrace{J(x(N-1), u(N-1))}^{\text{cost of one step}} + \Psi(x(N)) \right\}}$$

The above is the optimal cost with one step to terminal stage. This is similar to what we did for the routing problem from NY to San Francisco before. The above is when we are standing in Portland and looking for the last step to take to San Francisco.

To minimize the above, we have to express everything in the same state $x(N-1)$. So we write the above, using the state equation, as

$$I(x(N-1), 1) = \min_{u(N-1)\in\Omega_{N-1}} \left\{ J(x(N-1), u(N-1)) + \Psi\left(f(x(N-1), u(N-1))\right) \right\} \qquad (1)$$

Now we find $u^{*}(N-1)$ of the above, using

$$\frac{dI(x(N-1),1)}{du(N-1)}=\frac{d}{du(N-1)}J(x(N-1),u(N-1))+\Psi\left(f(x(N-1),u(N-1))\right)=0$$

And solve for $u(N-1)$ using standard calculus. Once we find $u^{*}(N-1)$, we plug it back into (1) and obtain

$$I^{*}(x(N-1),1)=J(x(N-1),u^{*}(N-1))+\Psi\left(f(x(N-1),u^{*}(N-1))\right)$$

Notice now there is no $\min_{u(N-1)\in\Omega_{N-1}}$ since we already done this. We now apply POO. again to find cost from stage $N-2$

$$I(x(N-2),2)=\min_{u(N-2)\in\Omega_{N-2}}\{J(x(N-2),u(N-2))+I^{*}(x(N-1),1)\}$$

Notice the difference now. For all stages back, beyond $N-1$, we use the cost found from the ahead stage, which is $I^{*}(x(N-1),1)$ in the above case. We now repeat the process, and find optimal $u^{*}(x(N-2),2)$. See HW 7, problem 2 for detailed example how to do this. More generally,

$$I(x(L),N-L)=\min_{u(L)\in\Omega_{L}}\{J(x(L),u(L))+I^{*}(x(L+1),N-L-1)\}$$

The above is called the dynamic programming equation.

2.26 Lecture 26. Thursday, April 14, 2016

2.26.1 Stages in dynamic programming

At stage L the optimal cost from stage L with $N-L$ steps to go is

$$\boxed{I(x_L, N-L) = \min_{u(N-L)\in\Omega_L} \{J(x_L, u(L)) + I(x_{L+1}, N-(L+1))\}}$$

With appropriate initialization.

Some comments: The trickiest part is how to use this equation. Must be careful. Think of $u(L)$ as feedback. We call the optimal $u^*(L)$.

Warning. There is a constraint on u. Do not use derivative to find optimal without being careful about the limits and constraints. For example, if $|u(L)| \leq 1$ and we have quadratic form. We will now use an example to show how to use these equations. Let

$$x_{k+1} = x_k - u_k \tag{1}$$

Where u_k is free to take any value. Let the objective function be

$$J(x_k, u_k) = \sum_{k=0}^{N-1} (x_{k+1} - u_k)^2 + u_k^2 \tag{2}$$

Reflecting a simple tracking mechanism. We always start at $x(N-1)$ with one stage to go. Hence the optimal cost from x_{N-1} with one step to go is

$$I(x_{N-1}, 1) = \min_{u(N-1)\in\Omega_1} \{J(x_{N-1}, u_{N-1}) + \Psi(x_N)\}$$

$\Psi(x_N)$ is the terminal cost. Let us now remove it from the rest of the computation to simplify things. We also replace J in the above from (2) and obtain

$$\begin{aligned} I(x_{N-1}, 1) &= \min_{u(N-1)} \left\{ \left(x_{((N-1)+1)} - u_{N-1}\right)^2 + u_{N-1}^2 \right\} \\ &= \min_{u(N-1)} \left\{ (x_N - u_{N-1})^2 + u_{N-1}^2 \right\} \\ &= \min_{u(N-1)} \left\{ (x_N - u_{N-1})^2 + u_{N-1}^2 \right\} \end{aligned}$$

We want everything in terms of x_{N-1}. So we use (1) to write $x_N = x_{N-1} - u_{N-1}$ and plug it back in the last equation above to obtain

$$\begin{aligned} I(x_{N-1}, 1) &= \min_{u_{N-1}} \left\{ (x_{N-1} - u_{N-1} - u_{N-1})^2 + u_{N-1}^2 \right\} \\ &= \min_{u_{N-1}} \left\{ (x_{N-1} - 2u_{N-1})^2 + u_{N-1}^2 \right\} \end{aligned} \tag{3}$$

Only now to take derivative, in order to find u_{N-1}^*. therefore

$$\begin{aligned} \frac{dI(x_{N-1}, 1)}{u_{N-1}} &= 0 \\ 2(x_{N-1} - 2u_{N-1})(-2) + 2u_{N-1} &= 0 \\ u_{N-1}^* &= \frac{2}{5} x_{N-1} \end{aligned}$$

Now that we found the optimal u_{N-1}^* we go back to (3) and replace u_{N-1} in (3) by u_{N-1}^*. Hence

$$\begin{aligned} I(x_{N-1}, 1) &= \left(x_{N-1} - 2\left(\frac{2}{5} x_{N-1}\right)\right)^2 + \left(\frac{2}{5} x_{N-1}\right)^2 \\ &= \frac{1}{5} x_{N-1}^2 \end{aligned}$$

Now we backup one step. We need to find

$$I(x_{N-2}, 2) = \min_{u_{N-2}} \{J(x_{N-2}, u_{N-1}) + I(x_{N-1}, 1)\} \tag{4}$$

Notice that we used $I(x_{N-1}, 1)$ in place of what we had before, which was the terminal cost $\Psi(x(N))$. Since now we are two steps behind. All the work before was for finding optimal $I(x_{N-1}, 1)$. So now (4) becomes

$$I(x_{N-2}, 2) = \min_{u_{N-2}} \left\{ J(x_{N-1}, u_{N-1}) + \frac{1}{5} x_{N-1}^2 \right\} \tag{5}$$

But from (2)

$$J(x_{N-1},u_{N-2}) = (x_{N-1}-u_{N-2})^2 + u_{N-2}^2$$

Hence (5) becomes

$$I(x_{N-2},2) = \min_{u_{N-2}}\left\{(x_{N-1}-u_{N-2})^2 + u_{N-2}^2 + \frac{1}{5}x_{N-1}^2\right\}$$

We need to use the state equation $x_{k+1} = x_k - u_k$ to rewrite x_{N-1} in the above, since we want everything in $N-2$ terms. Therefore the above becomes

$$\begin{aligned} I(x_{N-2},2) &= \min_{u_{N-2}}\left\{((x_{N-2}-u_{N-2})-u_{N-2})^2 + u_{N-2}^2 + \frac{1}{5}(x_{N-2}-u_{N-2})^2\right\} \\ &= \min_{u_{N-2}}\left\{\frac{26}{5}u_{N-2}^2 - \frac{22}{5}u_{N-2}x_{N-2} + \frac{6}{5}x_{N-2}^2\right\} \qquad (6) \end{aligned}$$

Now we take derivative, to find u_{N-2}^*

$$\begin{aligned} \frac{dI(x_{N-2},2)}{u_{N-2}} &= 0 \\ \frac{52}{5}u_{N-2} - \frac{22}{5}x_{N-2} &= 0 \\ u_{N-2}^* &= \frac{11}{26}x_{N-2} \end{aligned}$$

Now that we found the optimal u_{N-2}^*, we go back to (6) and replace u_{N-2} there with u_{N-2}^*

$$\begin{aligned} I(x_{N-2},2) &= \frac{26}{5}\left(\frac{11}{26}x_{N-2}\right)^2 - \frac{22}{5}\left(\frac{11}{26}x_{N-2}\right)x_{N-2} + \frac{6}{5}x_{N-2}^2 \\ &= \frac{7}{26}x_{N-2}^2 \end{aligned}$$

Reader Carry out one more stage and obtain $J^* = (x(0),3)$

Answer

$$\begin{aligned} I(x_{N-3},3) &= \min_{u_{N-3}}\{J(x_{N-3},u_{N-3}) + I(x_{N-2},2)\} \\ &= \min_{u_{N-3}}\left\{J(x_{N-3},u_{N-3}) + \frac{7}{26}x_{N-2}^2\right\} \end{aligned}$$

But from (2) $J(x_{N-3},u_{N-3}) = (x_{N-2}-u_{N-3})^2 + u_{N-3}^2$, hence the above becomes

$$I(x_{N-3},3) = \min_{u_{N-3}}\left\{(x_{N-2}-u_{N-3})^2 + u_{N-3}^2 + \frac{7}{26}x_{N-2}^2\right\}$$

We need to use the state equation $x_{k+1} = x_k - u_k$ to rewrite x_{N-2} in the above, since we want everything in $N-3$ terms. Therefore the above becomes

$$\begin{aligned} I(x_{N-3},3) &= \min_{u_{N-3}}\left\{((x_{N-3}-u_{N-3})-u_{N-3})^2 + u_{N-3}^2 + \frac{7}{26}(x_{N-3}-u_{N-3})^2\right\} \\ &= \min_{u_{N-3}}\left\{\frac{137}{26}u_{N-3}^2 - \frac{59}{13}u_{N-3}x_{N-3} + \frac{33}{26}x_{N-3}^2\right\} \qquad (7) \end{aligned}$$

Now we take derivative, to find u_{N-3}^*

$$\begin{aligned} \frac{dI(x_{N-3},3)}{u_{N-3}} &= 0 \\ \frac{274}{26}u_{N-3} - \frac{59}{13}x_{N-3} &= 0 \\ u_{N-3}^* &= \frac{59}{137}x_{N-3} \end{aligned}$$

Replace this back in (7)

$$\begin{aligned} I(x_{N-3},3) &= \frac{137}{26}\left(\frac{59}{137}x_{N-3}\right)^2 - \frac{59}{13}\left(\frac{59}{137}x_{N-3}\right)x_{N-3} + \frac{33}{26}x_{N-3}^2 \\ &= \frac{40}{137}x_{N-3}^2 \end{aligned}$$

Let us do one more one, $N = 4$.

$$\begin{aligned} I(x_{N-4}, 4) &= \min_{u_{N-4}} \{J(x_{N-4}, u_{N-4}) + I(x_{N-3}, 3)\} \\ &= \min_{u_{N-4}} \left\{ J(x_{N-4}, u_{N-4}) + \frac{40}{137} x_{N-3}^2 \right\} \end{aligned}$$

But from (2) $J(x_{N-4}, u_{N-4}) = (x_{N-3} - u_{N-4})^2 + u_{N-4}^2$, hence the above becomes

$$I(x_{N-4}, 4) = \min_{u_{N-4}} \left\{ (x_{N-3} - u_{N-4})^2 + u_{N-4}^2 + \frac{40}{137} x_{N-3}^2 \right\}$$

We need to use the state equation $x_{k+1} = x_k - u_k$ to rewrite x_{N-3} in the above, since we want everything in $N-4$ terms. Therefore the above becomes

$$\begin{aligned} I(x_{N-4}, 4) &= \min_{u_{N-4}} \left\{ ((x_{N-4} - u_{N-4}) - u_{N-4})^2 + u_{N-4}^2 + \frac{40}{137} (x_{N-4} - u_{N-4})^2 \right\} \\ &= \min_{u_{N-4}} \left\{ \frac{725}{137} u_{N-4}^2 - \frac{628}{137} u_{N-4} x_{N-4} + \frac{177}{137} x_{N-4}^2 \right\} \end{aligned} \tag{8}$$

Now we take derivative, to find u_{N-4}^*

$$\begin{aligned} \frac{dI(x_{N-4}, 4)}{u_{N-4}} &= 0 \\ \frac{1450}{137} u_{N-4} - \frac{628}{137} x_{N-4} &= 0 \\ u_{N-4}^* &= \frac{314}{725} x_{N-4} \end{aligned}$$

Therefore (8) becomes

$$\begin{aligned} I(x_{N-4}, 4) &= \frac{725}{137} \left(\frac{314}{725} x_{N-4} \right)^2 - \frac{628}{137} \left(\frac{314}{725} x_{N-4} \right) x_{N-4} + \frac{177}{137} x_{N-4}^2 \\ &= \frac{217}{725} x_{N-4}^2 \end{aligned}$$

A table of the summary

L	$I(x_{N-L}, L)$
1	$0.2\ x_{N-1}^2$
2	$0.2692\ x_{N-2}^2$
3	$0.29197\ x_{N-3}^2$
4	$0.29931\ x_{N-4}^2$

So for $N = 4$

L	$I(x_{4-L}, L)$
1	$0.2\ x_3^2$
2	$0.2692\ x_2^2$
3	$0.29197\ x_1^2$
4	$0.29931\ x_0^2$

Using $x_{k+1} = x_k - u_k$ to write everything in terms of x_0

L	$I(x_{4-L}, L)$
1	$0.2\ (((x_0 - u_0) - u_1) - u_2)^2$
2	$0.2692\ ((x_0 - u_0) - u_1)^2$
3	$0.29197\ x_1^2\ (x_0 - u_0)^2$
4	$0.29931\ x_0^2$

So total cost is

$$I = 0.2\ (((x_0 - u_0) - u_1) - u_2)^2 + 0.2692\ ((x_0 - u_0) - u_1)^2 + 0.29197\ x_1^2 (x_0 - u_0)^2 + 0.29931\ x_0^2$$

This example is special case of LQR. What happens in this example above, or more generally when $N\to\infty$? As $N\to\infty$ we will see later that the feedback gains become time invariant. This is called steady state LQR and will we arrive at the Riccati equations.

2.26.2 Allocation problem, applying DP to investment problem

Next example is allocation problem We will do it in two steps. We can solve this without using D.P. but will use D.P. to illustrate the method. Consider two investments.

1. Invest \$1 in real estate, with return of \$2.
2. Invest \$1 in oil, with return of \$4.

Let say with start with fixed amount of money k dollars. We have constraint: b_r is maximum allowed amount of investment in real estate, b_o is the maximum amount allowed for oil investment. To avoid trivial solution, assume also that $b_r+b_o>k$. Let u_r the amount invested in real estate, and let u_o amount invested in oil.

Common sense solution is $u_0^*=k-b_o$ and $u_1^*=b_o$ since investment in oil has higher return. u_1 is investment in oil, and u_0 is investment in real estate. Let do this using D.P. The objective function is

$$J(u)=2u_0+4u_1$$

And the state equation is (we only have two states x_1,x_0)

$$x_1=x_0-u_0$$

Initial state is $x_0=k$ which is the money we have at the start. There are two stages. Hence $N=2$. We start with

$$\begin{aligned}I(x_{N-1},1)&=I(x_1,1)\\&=\max_{u_1\in\Omega_1}\{J(u)\}\end{aligned}$$

i.e. we assume we have one stage to go, and that we have made initial investment in real estate and now we are making investment in oil. Hence

$$\Omega_1=[0\cdots\min\{b_o,x_1\}]$$

Hence $u_1^*=\min\{b_o,x_1\}$ and

$$I(x_1,1)=4\min\{b_o,x_1\}$$

Now backup one step.

$$\begin{aligned}I(x_0,2)&=\max_{u_0\in\Omega_0}\{J(u)+I(x_1,1)\}\\&=\max_{u_0\in\Omega_0}\{2u_0+4\min\{b_o,x_1\}\}\end{aligned}$$

Where $\Omega_0=[0\cdots b_o]$. Therefore since $x_1=x_0-u_0$ the above becomes

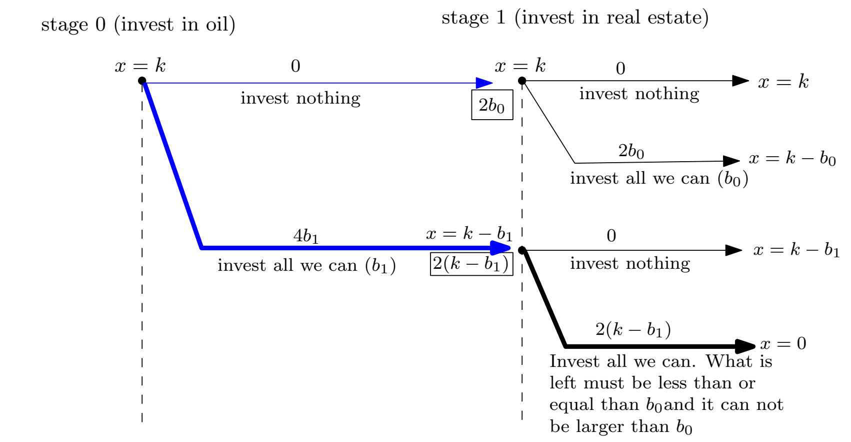
$$I(x_0,2)=\max_{u_0\in\Omega_0}\{2u_0+4\min\{b_o,x_0-u_0\}\}$$

Let

$$\begin{aligned}F(u_0)&=2u_0+4\min\{b_o,x_0-u_0\}\\&=2u_0+4\min\{b_o,k-u_0\}\end{aligned}$$

Find the maximum using graphics method. This gives $u_o^*=k-b_o$ which is the same using the common sense approach.

The following diagram shows the solution of the above using the dynamic graph method.


stage 0 (invest in oil)
stage 1 (invest in real estate)
$x = k$
0
invest nothing
$x = k$
$2b_0$
0
invest nothing
$x = k$
$2b_0$
$x = k - b_0$
invest all we can (b_0)
$4b_1$
invest all we can (b_1)
$x = k - b_1$
$2(k - b_1)$
0
invest nothing
$x = k - b_1$
$2(k - b_1)$
$x = 0$
Invest all we can. What is left must be less than or equal than $b_0$and it can not be larger than b_0

The constriants are

$$
\begin{aligned}
b_0 + b_1 &> k \\
b_0 &\leq k \\
b_1 &\leq k
\end{aligned}
$$

Note also, if we invest b_1 in stage 0, what is left is $k - b_1$ and this amount can not be larger than b_0. It can only be equal or less than b_0.

The maximum path is taken by investing b_1 in stage zero, and then investing what is left in stage one. This is becuase we had to decide if $2b_0$ is larger than $4b_1 + 2(k - b_1)$ to decide which path to take.

It is clear than $2b_1 + 2k \geq 2b_0$ since $k \geq b_0$, hence the path selected is the one shown.

In the above, b_1 is amount to invest in oil and b_0 is amount to invest in real estate, and k is maximum total amount we allowed to invest.

Figure 2.23: Solution to the oil and real estate problem using Branch and Bound graph method

2.27 Lecture 27. Tuesday, April 19, 2016

2.27.1 LQR and dynamic programming

In the following, some of the terms were re-written with the index being as subscript, as it is easier to see on the screen, Hence instead of $x(k)$ as was done in the lecture, it becomes x_k and $u(k)$ becomes u_k and so on.

Now we start with the state equation

$$\boxed{x_{k+1} = Ax_k + Bu_k}$$

In the above, A is $n \times n$ and B is $n \times m$ where m is the number of inputs, and u_k is column vector of size m, and similarly for x_{k+1} and x_k. In continuous time, the above is

$$\boxed{\dot{x}(t) = Ax(t) + Bu(t)}$$

We can also allow time varying A, B in the above, but in this discussion we assumed these are constant matrices. The goal is to make $x(k)$ track, or approach some desired $x^d(k)$ with as little effort $u(k)$ as possible (what about also as fast as possible at same time?). $u(k)$ is called the control effort. This diagram illustrates this

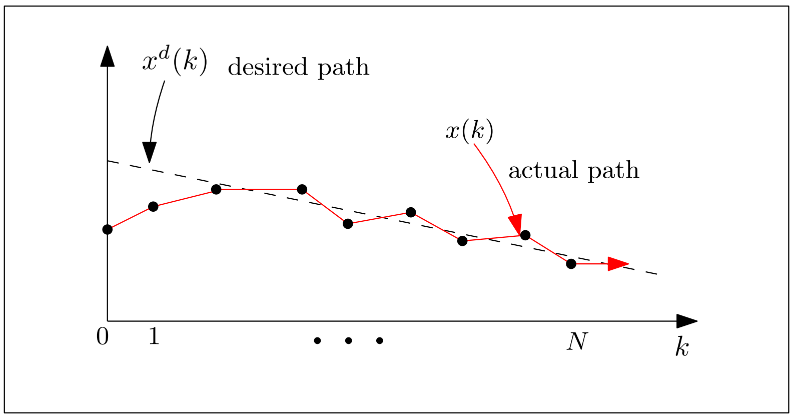


Figure 2.24: Goal is to track desired path

When $x^d(\infty) = 0$, this means we want to bring the system to stable state. We want now to quantify the goal $J(u)$. So the problem is to bring the state $x(k)$ to zero using as small effort u as possible. We write the cost $J(u)$ as

$$\boxed{J(u) = \sum_{k=0}^{N-1} x_{k+1}^T Q x_{k+1} + u_k^T R u_k}$$

This is just something we have to accept as given. We did not derive this, but it makes sense. Notice for example, when $x(k)$ is small, then $J(u)$ is small. But from now on, we just use the above as given. In the above, Q and R are called <u>weighting matrices</u>. For example, if $u(1)$ is more important than $u(2)$, we adjust the values in Q to reflect different weights we want to assign. In the above, Q is $n \times m$ and R is $m \times m$. Both Q and R are <u>positive definite and symmetric</u>. Now we start using the Bellman dynamic equations.

$$\begin{aligned} I(x_{N-1}, 1) &= \min_{u_{N-1} \in \Omega_{N-1}} \{J(x_{N-1}, u_{N-1}), \Psi(x_N)\} \\ &= \min_{u_{N-1} \in \Omega_{N-1}} \left\{x_N^T Q x_N + u_{N-1}^T R u_{N-1}\right\} \qquad (1) \end{aligned}$$

We ignore the terminal cost $\Psi(x_N)$ for now. Be careful with the indices. Notice in the above, after replacing $J(u)$, that x has index N and the u has the $N-1$ index on it. This is due to how $J(u)$ is given to us to use, which has x_{k+1} in it already. EQ (1) is our starting point. Now we start applying the dynamic equations recursively. First , we replace the state equation in the above and obtain

$$I(x_{N-1}, 1) = \min_{u_{N-1} \in \Omega_{N-1}} \left\{(Ax_{N-1} + Bu_{N-1})^T Q (Ax_{N-1} + Bu_{N-1}) + u_{N-1}^T R u_{N-1}\right\} \qquad (2)$$

Notice that now all indices on x are $N-1$, this is because the state equation being $x_{k+1} = Ax_k + Bu_k$. Notice that (2) is quadratic in u_{N-1}. This is important. Doing one more simplification on (2) gives (where the leading $\min_{u_{N-1}\in\Omega_{N-1}}$ is now removed, just to make the equations fit), but it is assumed to be on each equation on what follows

$$
\begin{aligned}
I\left(x_{N-1},1\right) &= \left(\left(Ax_{N-1}\right)^T + \left(Bu_{N-1}\right)^T\right) Q\left(Ax_{N-1}+Bu_{N-1}\right) + u_{N-1}^T R u_{N-1}\\
&= \left(x_{N-1}^T A^T + u_{N-1}^T B^T\right) Q\left(Ax_{N-1}+Bu_{N-1}\right) + u_{N-1}^T R u_{N-1}\\
&= \left(x_{N-1}^T A^T Q + u_{N-1}^T B^T Q\right)\left(Ax_{N-1}+Bu_{N-1}\right) + u_{N-1}^T R u_{N-1}\\
&= \left(x_{N-1}^T A^T Q + u_{N-1}^T B^T Q\right)\left(Ax_{N-1}\right) + \left(x_{N-1}^T A^T Q + u_{N-1}^T B^T Q\right)\left(Bu_{N-1}\right) + u_{N-1}^T R u_{N-1}\\
&= x_{N-1}^T A^T Q A x_{N-1} + u_{N-1}^T B^T Q A x_{N-1} + x_{N-1}^T A^T Q B u_{N-1} + u_{N-1}^T B^T Q B u_{N-1} + u_{N-1}^T R u_{N-1}\\
&= \left(u_{N-1}^T B^T Q B u_{N-1} + u_{N-1}^T R u_{N-1}\right) + u_{N-1}^T B^T Q A x_{N-1} + x_{N-1}^T A^T Q B u_{N-1} + x_{N-1}^T A^T Q A x_{N-1}\\
&= u_{N-1}^T\left(B^T Q B + R\right) u_{N-1} + \left(u_{N-1}^T B^T Q A x_{N-1} + x_{N-1}^T A^T Q B u_{N-1}\right) + \left(x_{N-1}^T A^T Q A x_{N-1}\right)\\
&= u_{N-1}^T\left(B^T Q B + R\right) u_{N-1} + \left(u_{N-1}^T B^T Q A x_{N-1} + \left(\left(QBu_{N-1}\right)^T\left(x_{N-1}^T A^T\right)^T\right)^T\right) + x_{N-1}^T A^T Q A x_{N-1}\\
&= u_{N-1}^T\left(B^T Q B + R\right) u_{N-1} + \left(u_{N-1}^T B^T Q A x_{N-1} + \left(\left(QBu_{N-1}\right)^T\left(Ax_{N-1}\right)\right)^T\right) + x_{N-1}^T A^T Q A x_{N-1}\\
&= u_{N-1}^T\left(B^T Q B + R\right) u_{N-1} + \left(u_{N-1}^T B^T Q A x_{N-1} + \left(\left(u_{N-1}^T\left(QB\right)^T\right)\left(Ax_{N-1}\right)\right)^T\right) + x_{N-1}^T A^T Q A x_{N-1}\\
&= u_{N-1}^T\left(B^T Q B + R\right) u_{N-1} + \left(u_{N-1}^T B^T Q A x_{N-1} + \left(u_{N-1}^T\left(B^T Q^T\right)\left(Ax_{N-1}\right)\right)^T\right) + x_{N-1}^T A^T Q A x_{N-1}\\
&= u_{N-1}^T\left(B^T Q B + R\right) u_{N-1} + \left(u_{N-1}^T B^T Q A x_{N-1} + \left(u_{N-1}^T B^T Q^T A x_{N-1}\right)^T\right) + x_{N-1}^T A^T Q A x_{N-1}
\end{aligned}
$$

But $Q^T = Q$ and the above becomes

$$
\begin{aligned}
I\left(x_{N-1},1\right) = u_{N-1}^T\left(B^T Q B + R\right) u_{N-1} &+ \left(u_{N-1}^T B^T Q A x_{N-1} + \left(u_{N-1}^T B^T Q A x_{N-1}\right)^T\right)\\
&+ x_{N-1}^T A^T Q A x_{N-1} \qquad (2)
\end{aligned}
$$

Using

$$\boxed{H = \frac{1}{2}\left(H + H^T\right)}$$

On the middle term $\left(u_{N-1}^T B^T Q A x_{N-1} + \left(u_{N-1}^T B^T Q A x_{N-1}\right)^T\right)$, where $H = u_{N-1}^T B^T Q A x_{N-1}$ reduces (2) to

$$I\left(x_{N-1},1\right) = \min_{u_{N-1}\in\Omega_{N-1}}\left\{u_{N-1}^T\left(B^T Q B + R\right) u_{N-1} + 2\left(u_{N-1}^T B^T Q A x_{N-1}\right) + x_{N-1}^T A^T Q A x_{N-1}\right\} \qquad (3)$$

The above is in the form $I = a_1 u^2 + 2a_2 u + a_3$, therefore it is quadratic in u_{N-1}. Taking derivative w.r.t. u_{N-1} since this is what we are minimizing I with respect to, we obtain from (3)

$$
\begin{aligned}
\frac{\partial I\left(x_{N-1},1\right)}{\partial u_{N-1}} &= 0\\
0 &= 2\left(B^T Q B + R\right) u_{N-1} + 2\left(B^T Q A x_{N-1}\right)
\end{aligned}
$$

R is positive definite, and Q is positive definite. Solving for u_{N-1} gives

$$
\begin{aligned}
u_{N-1}^* &= -\left(B^T Q B + R\right)^{-1}\left(B^T Q A x_{N-1}\right)\\
&= -\left(B^T Q B + R\right)^{-1} B^T Q A x_{N-1} \qquad (4)
\end{aligned}
$$

Reader $\left(B^T Q B + R\right)$ is the Hessian. Show it is positive definite matrix. Since the Hessian is P.D., then u_{N-1}^* is global min. Eq (4) is linear feedback on state $(N-1)$. i.e. we write

$$u^*\left(N-1\right) = K\left(N-1\right) x\left(N-1\right)$$

Where $K(N-1)$ is called the gain matrix which is

$$K\left(N-1\right) = -\left(B^T Q B + R\right)^{-1} B^T Q A$$

In all the expressions below, we see the term $\left(B^T Q B + R\right)$ repeating. So to simplify things and make the equations smaller, let

$$\boxed{\Phi = B^T Q B + R}$$

Hence

$$K\left(N-1\right) = -\Phi^{-1} B^T Q A$$

And therefore

$$\begin{aligned} u^*_{N-1} &= \left(-\Phi^{-1}B^TQA\right)x_{N-1} \\ &= -\left(B^TQB+R\right)^{-1}B^TQAx_{N-1} \end{aligned}$$

Now we go back to (3) and replace the u_{N-1} in that expression with u^*_{N-1} we found in (4). (we remove the $\min_{u_{N-1}\in\Omega_{N-1}}$ we had in (3), since it is now the minimum)

$$\begin{aligned} I^*\left(x_{N-1},1\right) &= u^{*T}_{N-1}\Phi u^*_{N-1} + 2\left(u^{*T}_{N-1}B^TQAx_{N-1}\right) + x^T_{N-1}A^TQAx_{N-1} \\ &= \left(\left(-\Phi^{-1}B^TQA\right)x_{N-1}\right)^T\Phi\left(-\Phi^{-1}B^TQA\right)x_{N-1} \\ &\quad + 2\left(\left(-\Phi^{-1}B^TQA\right)x_{N-1}\right)^T\left(B^TQAx_{N-1}\right) + x^T_{N-1}A^TQAx_{N-1} \\ &= -x^T_{N-1}\left(\Phi^{-1}B^TQA\right)^T\Phi\left(-\Phi^{-1}B^TQA\right)x_{N-1} - 2x^T_{N-1}\left(\Phi^{-1}B^TQA\right)^T\left(B^TQAx_{N-1}\right) \\ &\quad + x^T_{N-1}A^TQAx_{N-1} \\ &= x^T_{N-1}\left[\left(\Phi^{-1}B^TQA\right)^T\left(B^TQA\right) - 2\left(\Phi^{-1}B^TQA\right)^T\left(B^TQA\right) + A^TQA\right]x_{N-1} \\ &= x^T_{N-1}\left[\left(B^TQA\right)^T\left(\Phi^{-1}\right)^T\left(B^TQA\right) - 2\left(B^TQA\right)^T\left(\Phi^{-1}\right)^T\left(B^TQA\right) + A^TQA\right]x_{N-1} \\ &= x^T_{N-1}\left[(QA)^TB\left(\Phi^{-1}\right)^T\left(B^TQA\right) - 2(QA)^TB\left(\Phi^{-1}\right)^T\left(B^TQA\right) + A^TQA\right]x_{N-1} \\ &= x^T_{N-1}\left[A^TQ^TB\left(\Phi^{-1}\right)^T\left(B^TQA\right) - 2A^TQ^TB\left(\Phi^{-1}\right)^T\left(B^TQA\right) + A^TQA\right]x_{N-1} \end{aligned}$$

But $Q = Q^T$ and $\left(\Phi^{-1}\right)^T = \Phi^{-1}$. Note that Φ is the Hessian matrix, and it is positive definite, and assumed symmetric. That is why $\left(\Phi^{-1}\right)^T = \Phi^{-1}$. But we did not proof this. It was a reader to show this is positive definite. The above therefore becomes

$$\begin{aligned} I^*\left(x_{N-1},1\right) &= x^T_{N-1}\left[A^TQB\Phi^{-1}B^TQA - 2A^TQ^TB\Phi^{-1}B^TQA + A^TQA\right]x_{N-1} \\ &= x^T_{N-1}\left[A^TQ - A^TQB\Phi^{-1}B^TQA\right]x_{N-1} \\ &= x^T_{N-1}A^TQ\left[I - B\Phi^{-1}B^TQA\right]x_{N-1} \\ &= x^T_{N-1}A^TQ\left[Q^{-1} - B\Phi^{-1}B^T\right]QAx_{N-1} \end{aligned}$$

Let

$$M_{N-1} = A^TQ\left[Q^{-1} - B\Phi^{-1}B^T\right]QA$$

Then

$$I^*\left(x_{N-1},1\right) = x^T_{N-1}M_{N-1}x_{N-1}$$

Now that we found $I^*\left(x_{N-1},1\right)$, we go back one more step

$$\begin{aligned} I\left(x_{N-2},2\right) &= \min_{u_{N-2}\in\Omega_{N-2}}\left\{J\left(x_{N-2},u_{N-2}\right) + I^*\left(x_{N-1},1\right)\right\} \\ &= \min_{u_{N-2}\in\Omega_{N-2}}\left\{x^T_{N-1}Qx_{N-1} + u^T_{N-2}Ru_{N-2} + I^*\left(x_{N-1},1\right)\right\} \\ &= \min_{u_{N-2}\in\Omega_{N-2}}\left\{x^T_{N-1}Qx_{N-1} + u^T_{N-2}Ru_{N-2} + x^T_{N-1}M_{N-1}x_{N-1}\right\} \\ &= \min_{u_{N-2}\in\Omega_{N-2}}\left\{x^T_{N-1}\left(Q+M_{N-1}\right)x_{N-1} + u^T_{N-2}Ru_{N-2}\right\} \end{aligned} \tag{6}$$

Think of $\left(Q+M_{N-1}\right)$ as the new Q matrix at stage $N-2$. We need to replace everything to be at $N-2$ stage, using the state equation $x_{k+1} = Ax_k + Bu_k$ then

$$x_{N-1} = Ax_{N-2} + Bu_{N-2}$$

Hence (6) becomes

$$I\left(x_{N-2},2\right) = \min_{u_{N-2}\in\Omega_{N-2}}\left\{\left(Ax_{N-2}+Bu_{N-2}\right)^T\left(Q+M_{N-1}\right)\left(Ax_{N-2}+Bu_{N-2}\right) + u^T_{N-2}Ru_{N-2}\right\}$$

As above, remove $\min_{u_{N-2}\in\Omega_{N-2}}$ in what follows so that equations fit on the page and let

$$\boxed{Q' = Q + M_N - 1}$$

Then

$$
\begin{align}
I\left(x_{N-2},2\right) &= \left(\left(Ax_{N-2}\right)^T + \left(Bu_{N-2}\right)^T\right) Q'\left(Ax_{N-2}+Bu_{N-2}\right) + u_{N-2}^T R u_{N-2} \notag\\
&= \left(\left(Ax_{N-2}\right)^T Q' + \left(Bu_{N-2}\right)^T Q'\right)\left(Ax_{N-2}+Bu_{N-2}\right) + u_{N-2}^T R u_{N-2} \notag\\
&= \left(Ax_{N-2}\right)^T Q'\left(Ax_{N-2}+Bu_{N-2}\right) + \left(Bu_{N-2}\right)^T Q'\left(Ax_{N-2}+Bu_{N-2}\right) + u_{N-2}^T R u_{N-2} \notag\\
&= \left(Ax_{N-2}\right)^T Q' Ax_{N-2} + \left(Ax_{N-2}\right)^T Q' Bu_{N-2} + \left(Bu_{N-2}\right)^T Q' Ax_{N-2} + \left(Bu_{N-2}\right)^T Q' Bu_{N-2} + \tag{2.1}\\
&\quad u_{N-2}^T R u_{N-2} \notag\\
&= x_{N-2}^T A^T Q' A x_{N-2} + x_{N-2}^T A^T Q' B u_{N-2} + u_{N-2}^T B^T Q' A x_{N-2} + u_{N-2}^T B^T Q' B u_{N-2} + u_{N-2}^T R u_{N-2} \notag\\
&= u_{N-2}^T\left(B^T Q' B + R\right) u_{N-2} + x_{N-2}^T\left(A^T Q' A\right) x_{N-2} + x_{N-2}^T A^T Q' B u_{N-2} + u_{N-2}^T B^T Q' A x_{N-2} \tag{7}
\end{align}
$$

But

$$
\begin{aligned}
x_{N-2}^T A^T Q' B u_{N-2} + u_{N-2}^T B^T Q' A x_{N-2} &= \left(u_{N-2}^T\left(x_{N-2}^T A^T Q' B\right)^T\right)^T + u_{N-2}^T B^T Q' A x_{N-2}\\
&= \left(u_{N-2}^T\left(\left(A^T Q' B\right)^T x_{N-2}\right)\right)^T + u_{N-2}^T B^T Q' A x_{N-2}\\
&= \left(u_{N-2}^T\left(\left(Q'B\right)^T A x_{N-2}\right)\right)^T + u_{N-2}^T B^T Q' A x_{N-2}\\
&= \left(u_{N-2}^T\left(B^T\left(Q'\right)^T A x_{N-2}\right)\right)^T + u_{N-2}^T B^T Q' A x_{N-2}
\end{aligned}
$$

But $Q' = \left(Q'\right)^T$ since symmetric[2] then the above becomes

$$
\begin{align}
x_{N-2}^T A^T Q' B u_{N-2} + u_{N-2}^T B^T Q' A x_{N-2} &= \left(u_{N-2}^T B^T Q' A x_{N-2}\right)^T + u_{N-2}^T B^T Q' A x_{N-2} \notag\\
&= 2\left(u_{N-2}^T B^T Q' A x_{N-2}\right) \tag{8}
\end{align}
$$

Using $H = \frac{1}{2}\left(H + H^T\right)$. Replacing (8) into (7) gives

$$
I\left(x_{N-2},2\right) = u_{N-2}^T\left(B^T Q' B + R\right) u_{N-2} + x_{N-2}^T\left(A^T Q' A\right) x_{N-2} + 2\left(u_{N-2}^T B^T Q' A x_{N-2}\right) \tag{9}
$$

We now find u_{N-2}^*

$$
\begin{aligned}
\frac{\partial I\left(x_{N-2},2\right)}{\partial u_{N-2}} &= 0\\
0 &= 2u_{N-2}\left(B^T Q' B + R\right) + 2B^T Q' A x_{N-2}
\end{aligned}
$$

Hence

$$
u_{N-2}^* = \left(-\left(B^T Q' B + R\right)^{-1} B^T Q' A\right) x_{N-2}
$$

Where $Q' = Q + M_{N-1}$. Hence

$$
\begin{aligned}
K(N-2) &= -\left(B^T Q' B + R\right)^{-1} B^T Q' A\\
&= -\left(B^T\left(Q + M_{N-1}\right) B + R\right)^{-1} B^T\left(Q + M_{N-1}\right) A
\end{aligned}
$$

Now we go back to $I\left(x_{N-2},2\right)$ and replace u_{N-2} with u_{N-2}^* we just found. From (9)

$$
\begin{aligned}
I^*\left(x_{N-2},2\right) &= \left(-\left(B^T Q' B + R\right)^{-1} B^T Q' A x_{N-2}\right)^T \left(B^T Q' B + R\right)\left(-\left(B^T Q' B + R\right)^{-1} B^T Q' A x_{N-2}\right) +\\
&\qquad x_{N-2}^T\left(A^T Q' A\right) x_{N-2} + 2\left(\left(-\left(B^T Q' B + R\right)^{-1} B^T Q' A x_{N-2}\right)^T B^T Q' A x_{N-2}\right)\\
&= \left(B^T Q' A x_{N-2}\right)^T \left(B^T Q' B + R\right)^{-1}\left(B^T Q' B + R\right)\left(B^T Q' B + R\right)^{-1} B^T Q' A x_{N-2} +\\
&\qquad x_{N-2}^T\left(A^T Q' A\right) x_{N-2} - 2\left(\left(B^T Q' B + R\right)^{-1} B^T Q' A x_{N-2}\right)^T B^T Q' A x_{N-2}\\
&= \left(B^T Q' A x_{N-2}\right)^T \left(B^T Q' B + R\right)^{-1} B^T Q' A x_{N-2} + x_{N-2}^T\left(A^T Q' A\right) x_{N-2}\\
&\quad - 2\left(B^T Q' A x_{N-2}\right)^T \left(B^T Q' B + R\right)^{-1} B^T Q' A x_{N-2}\\
&= x_{N-2}^T\left(B^T Q' A\right)^T \left(B^T Q' B + R\right)^{-1} B^T Q' A x_{N-2} + x_{N-2}^T\left(A^T Q' A\right) x_{N-2}\\
&= x_{N-2}^T\left[\left(A^T Q'^T B\right)\left(B^T Q' B + R\right)^{-1} B^T Q' A + \left(A^T Q' A\right)\right] x_{N-2}
\end{aligned}
$$

Reader Argue that $I^*\left(x_{N-2},2\right)$ looks like

$$
I^*\left(x_{N-2},2\right) = x_{N-2}^T M_{N-2} x_{N-2}
$$

Reader Find M_{N-2}

[2] Need proof

$$
\begin{aligned}
M_{N-2} &= \left(A^TQ'^TB\right)\left(B^TQ'B+R\right)^{-1}B^TQ'A+\left(A^TQ'A\right)\\
&= \left(A^T\left(Q+M_N-1\right)^TB\right)\left(B^T\left(Q+M_N-1\right)B+R\right)^{-1}B^T\left(Q+M_N-1\right)A\\
&\quad+\left(A^T\left(Q+M_N-1\right)A\right)
\end{aligned}
$$

But $M_N-1=A^TQ\left[Q^{-1}-B\Phi^{-1}B^T\right]QA$, hence

$$
\begin{aligned}
M_{N-2} = &\left(A^T\left(Q+\left(A^TQ\left[Q^{-1}-B\Phi^{-1}B^T\right]QA\right)\right)^TB\right)\left(B^T\left(Q+M_N-1\right)B+R\right)^{-1}B^T\\
&\left(Q+\left(A^TQ\left[Q^{-1}-B\Phi^{-1}B^T\right]QA\right)\right)A+\left(A^T\left(Q+\left(A^TQ\left[Q^{-1}-B\Phi^{-1}B^T\right]QA\right)\right)A\right)
\end{aligned}
$$

But $\Phi=B^TQB+R$, hence the above becomes

$$
M_{N-2}=\left(A^T\left(Q+\left(A^TQ\left[Q^{-1}-B\left(B^TQB+R\right)^{-1}B^T\right]QA\right)\right)^TB\right)\left(B^T\left(Q+M_N-1\right)B+R\right)^{-1}B^T\left(Q+\left(A^TQ\left[Q^{-1}-B
$$

Summary

$$
\boxed{\begin{aligned}
u^*_{N-i}&=K_{N-i}x_{N-i}\\
I^*\left(x_{N-i}\right)&=x^T_{N-i}M_{N-i}x_{N-i}
\end{aligned}}
$$

Reader Find a formula for $K(N-3)$.

The expression for K_{N-i} is

$$
\begin{aligned}
K_{N-1}&=-\left(B^TQB+R\right)^{-1}B^TQA\\
K_{N-2}&=-\left(B^T\left(Q+M_{N-1}\right)B+R\right)^{-1}B^T\left(Q+M_{N-1}\right)A\\
&=-\left(B^TQB+R+B^TM_{N-1}B\right)^{-1}\left(B^TQA+B^TM_{N-1}A\right)
\end{aligned}
$$

Where $M_{N-1}=A^TQ\left[Q^{-1}-B\Phi^{-1}B^T\right]QA$ and $\Phi=B^TQB+R$, hence

$$
\begin{aligned}
K_{N-2}&=-\left(B^TQB+R+B^T\left(A^TQ\left[Q^{-1}-B\Phi^{-1}B^T\right]QA\right)B\right)^{-1}\\
&\quad\left(B^TQA+B^T\left(A^TQ\left[Q^{-1}-B\Phi^{-1}B^T\right]QA\right)A\right)\\
&=-\left(B^TQB+R+B^T\left(A^TQ\left[Q^{-1}-B\left(B^TQB+R\right)^{-1}B^T\right]QA\right)B\right)^{-1}\\
&\qquad\left(B^TQA+B^T\left(A^TQ\left[Q^{-1}-B\left(B^TQB+R\right)^{-1}B^T\right]QA\right)A\right)
\end{aligned}
$$

The final optimal cost will be

$$
J^*=x_0^TM_0x_0
$$

2.27.2 Example LQR using dynamic programming

Reader Let $A=\begin{pmatrix}1&-2\\-1&3\end{pmatrix}, B=\begin{pmatrix}0\\1\end{pmatrix}, Q=\begin{pmatrix}2&1\\1&2\end{pmatrix}$ and $R=1$, the weight on input. For $N=3$, find $K(2), K(1), K(0)$ solving for LQR problem. We will get optimal gain $K(i)$ and these will not be the same. We do not like time varying gains, as in this case. We like the gain matrix to be constant, as it is easier to manger and more safe to use. If we make N very large, then gain will become constant. We start from very large N and go back to zero.

Solution

$N=3$,

$$\begin{aligned}
K(N-1) &= -\left(B^TQB+R\right)^{-1}B^TQA\\
K(2) &= -\left(\begin{pmatrix}0\\1\end{pmatrix}^T\begin{pmatrix}2&1\\1&2\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}+1\right)^{-1}\begin{pmatrix}0\\1\end{pmatrix}^T\begin{pmatrix}2&1\\1&2\end{pmatrix}\begin{pmatrix}1&-2\\-1&3\end{pmatrix}\\
&= -(2+1)^{-1}\begin{pmatrix}-1&4\end{pmatrix}\\
&= \frac{-1}{3}\begin{pmatrix}-1&4\end{pmatrix}\\
&= \begin{pmatrix}\frac{1}{3}&\frac{-4}{3}\end{pmatrix}
\end{aligned}$$

Hence

$$\boxed{K(2)=\begin{pmatrix}\frac{1}{3}&\frac{-4}{3}\end{pmatrix}=\begin{pmatrix}0.33333&-1.3333\end{pmatrix}}$$

Will do things from now on using the state equations directly, as it seems easier. Starting again

$$\begin{aligned}
I(x(3-1),1) &= \min_{u(2)}\{J(x_2,u_2)\}\\
I(x_2,1) &= \min_{u_2}\left\{x_3^TQx_3+u_2^TRu_2\right\}
\end{aligned}$$

But $x_3=Ax_2+Bu_2$ from state equation, hence the above becomes

$$\begin{aligned}
I(x_2,1) &= \min_{u_2}\left\{(Ax_2+Bu_2)^TQ(Ax_2+Bu_2)+u_2^TRu_2\right\}\\
&= \min_{u_2}\left\{\left(\begin{pmatrix}1&-2\\-1&3\end{pmatrix}x_2+\begin{pmatrix}0\\1\end{pmatrix}u_2\right)^T\begin{pmatrix}2&1\\1&2\end{pmatrix}\left(\begin{pmatrix}1&-2\\-1&3\end{pmatrix}x_2+\begin{pmatrix}0\\1\end{pmatrix}u_2\right)+u_2^T(1)u_2\right\}\\
&= \min_{u_2}\left\{\left(\begin{pmatrix}1&-2\\-1&3\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}+\begin{pmatrix}0\\u_2\end{pmatrix}\right)^T\begin{pmatrix}2&1\\1&2\end{pmatrix}\left(\begin{pmatrix}1&-2\\-1&3\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}+\begin{pmatrix}0\\u_2\end{pmatrix}\right)+u_2^2\right\}\\
&= \min_{u_2}\left\{\begin{pmatrix}x_{11}-2x_{21}\\u_2-x_{11}+3x_{21}\end{pmatrix}^T\begin{pmatrix}2&1\\1&2\end{pmatrix}\begin{pmatrix}x_{11}-2x_{21}\\u_2-x_{11}+3x_{21}\end{pmatrix}+u_2^2\right\}\\
&= \min_{u_2}\left\{(x_{11}-2x_{21})(u_2+x_{11}-x_{21})+(2u_2-x_{11}+4x_{21})(u_2-x_{11}+3x_{21})+u_2^2\right\}\\
&= \min_{u_2}\left\{2u_2^2-2u_2x_{11}+8u_2x_{21}+2x_{11}^2-10x_{11}x_{21}+14x_{21}^2+u_2^2\right\}\\
&= \min_{u_2}\left\{3u_2^2-2u_2x_{11}+8u_2x_{21}+2x_{11}^2-10x_{11}x_{21}+14x_{21}^2\right\}
\end{aligned}$$

Hence

$$\begin{aligned}
\frac{\partial I(x_2,1)}{\partial u_2} &= 0\\
0 &= 6u_2-2x_{11}+8x_{21}\\
0 &= 6u_2+\begin{pmatrix}-2&8\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}\\
0 &= 6u_2+\begin{pmatrix}-2&8\end{pmatrix}x_1\\
u_2^* &= -\frac{1}{6}\begin{pmatrix}-2&8\end{pmatrix}x_1\\
&= \begin{pmatrix}\frac{2}{6}&\frac{-8}{6}\end{pmatrix}x_1
\end{aligned}$$

Therefore

$$\boxed{u_2^*=\begin{pmatrix}\frac{1}{3}&\frac{-4}{3}\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}}$$

And

$$\boxed{K(2)=\begin{pmatrix}\frac{1}{3}&\frac{-4}{3}\end{pmatrix}=\begin{pmatrix}0.33333&-1.3333\end{pmatrix}}$$

Which is the same as above using $-\left(B^TQB+R\right)^{-1}B^TQA$.

Now we find $I^*(x_2,1)$ by using u_2^* found above back in $I(x_2,1)$

$$I^*(x_2,1)=\left(\begin{pmatrix}1&-2\\-1&3\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}+\begin{pmatrix}0\\1\end{pmatrix}u_2\right)^T\begin{pmatrix}2&1\\1&2\end{pmatrix}$$
$$\left(\begin{pmatrix}1&-2\\-1&3\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}+\begin{pmatrix}0\\1\end{pmatrix}u_2\right)+u_2^T(1)u_2$$

$$=\left(\begin{pmatrix}1&-2\\-1&3\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}+\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}\frac{1}{3}&\frac{-4}{3}\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}\right)^T$$
$$\begin{pmatrix}2&1\\1&2\end{pmatrix}\left(\begin{pmatrix}1&-2\\-1&3\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}+\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}\frac{1}{3}&\frac{-4}{3}\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}\right)$$

$$+\left(\begin{pmatrix}\frac{1}{3}&\frac{-4}{3}\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}\right)^T\begin{pmatrix}\frac{1}{3}&\frac{-4}{3}\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}$$
$$=\begin{pmatrix}x_{11}-2x_{21}\\\frac{5}{3}x_{21}-\frac{2}{3}x_{11}\end{pmatrix}^T\begin{pmatrix}2&1\\1&2\end{pmatrix}\begin{pmatrix}x_{11}-2x_{21}\\\frac{5}{3}x_{21}-\frac{2}{3}x_{11}\end{pmatrix}$$
$$+x_{11}\left(\frac{1}{9}x_{11}-\frac{4}{9}x_{21}\right)-x_{21}\left(\frac{4}{9}x_{11}-\frac{16}{9}x_{21}\right)$$

$$=(x_{11}-2x_{21})\left(\frac{4}{3}x_{11}-\frac{7}{3}x_{21}\right)+\left(\frac{1}{3}x_{11}-\frac{4}{3}x_{21}\right)\left(\frac{2}{3}x_{11}-\frac{5}{3}x_{21}\right)$$
$$+x_{11}\left(\frac{1}{9}x_{11}-\frac{4}{9}x_{21}\right)-x_{21}\left(\frac{4}{9}x_{11}-\frac{16}{9}x_{21}\right)$$

$$=\frac{5}{3}x_{11}^2-\frac{22}{3}x_{11}x_{21}+\frac{26}{3}x_{21}^2$$

We have to convert this to $x_1^Tx_1$ to be able to use it in the next stage since we need to apply the state equation to it. We see that

$$\frac{5}{3}x_{11}^2-\frac{22}{3}x_{11}x_{21}+\frac{26}{3}x_{21}^2=\begin{pmatrix}x_{11}&x_{21}\end{pmatrix}\begin{pmatrix}c_{11}&c_{12}\\c_{12}&c_{22}\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}$$

Solving gives $\begin{pmatrix}c_{11}&c_{12}\\c_{12}&c_{22}\end{pmatrix}=\begin{pmatrix}\frac{5}{3}&-\frac{22}{6}\\-\frac{22}{6}&\frac{26}{3}\end{pmatrix}$, hence

$$\boxed{I^*(x_2,1)=x_2^T\begin{pmatrix}\frac{5}{3}&-\frac{22}{6}\\-\frac{22}{6}&\frac{26}{3}\end{pmatrix}x_2}$$

Now we find $I(x(N-2),2)=I(x(3-2),2)=I(x_1,2)$

$$\begin{aligned}I(x_1,2)&=\min_{u_1}\{J(x_1,u_1)+I^*(x_2,1)\}\\&=\min_{u_3}\left\{x_2^TQx_2+u_1^TRu_1+I^*(x_2,1)\right\}\end{aligned}$$

But $x_2 = Ax_1 + Bu_1$ from state equation, hence the above becomes

$$\begin{aligned}
I(x_1,2) &= \min_{u_3}\left\{(Ax_1+Bu_1)^T Q(Ax_1+Bu_1) + u_1^T R u_1 + I^*(x_2,1)\right\}\\
&= \left(\begin{pmatrix}1 & -2\\ -1 & 3\end{pmatrix}\begin{pmatrix}x_{11}\\ x_{21}\end{pmatrix} + \begin{pmatrix}0\\ 1\end{pmatrix}u_1\right)^T \begin{pmatrix}2 & 1\\ 1 & 2\end{pmatrix}\left(\begin{pmatrix}1 & -2\\ -1 & 3\end{pmatrix}\begin{pmatrix}x_{11}\\ x_{21}\end{pmatrix} + \begin{pmatrix}0\\ 1\end{pmatrix}u_1\right) + u_1^2\\
&+ x_2^T\begin{pmatrix}\frac{5}{3} & -\frac{22}{6}\\ -\frac{22}{6} & \frac{26}{3}\end{pmatrix}x_2
\end{aligned}$$

Now we have to use the state equations in $I^*(x_2,1)$ to update the last term above, this is important, since everything should be at the same state

$$\begin{aligned}
I(x_1,2) &= 3u_1^2 - 2u_1x_{11} + 8u_1x_{21} + 2x_{11}^2 - 10x_{11}x_{21} + 14x_{21}^2\\
&+ \left(\begin{pmatrix}1 & -2\\ -1 & 3\end{pmatrix}\begin{pmatrix}x_{11}\\ x_{21}\end{pmatrix} + \begin{pmatrix}0\\ 1\end{pmatrix}u_1\right)^T \begin{pmatrix}\frac{5}{3} & -\frac{22}{6}\\ -\frac{22}{6} & \frac{26}{3}\end{pmatrix}\left(\begin{pmatrix}1 & -2\\ -1 & 3\end{pmatrix}\begin{pmatrix}x_{11}\\ x_{21}\end{pmatrix} + \begin{pmatrix}0\\ 1\end{pmatrix}u_1\right)\\
&= 3u_1^2 - 2u_1x_{11} + 8u_1x_{21} + 2x_{11}^2 - 10x_{11}x_{21} + 14x_{21}^2\\
&\left(\begin{pmatrix}1 & -2\\ -1 & 3\end{pmatrix}\begin{pmatrix}x_{11}\\ x_{21}\end{pmatrix} + \begin{pmatrix}0\\ u_1\end{pmatrix}\right)^T \begin{pmatrix}\frac{5}{3} & -\frac{22}{6}\\ -\frac{22}{6} & \frac{26}{3}\end{pmatrix}\left(\begin{pmatrix}1 & -2\\ -1 & 3\end{pmatrix}\begin{pmatrix}x_{11}\\ x_{21}\end{pmatrix} + \begin{pmatrix}0\\ u_1\end{pmatrix}\right)\\
&= 3u_1^2 - 2u_1x_{11} + 8u_1x_{21} + 2x_{11}^2 - 10x_{11}x_{21} + 14x_{21}^2\\
&\frac{26}{3}u_1^2 - \frac{74}{3}u_1x_{11} + \frac{200}{3}u_1x_{21} + \frac{53}{3}x_{11}^2 - \frac{286}{3}x_{11}x_{21} + \frac{386}{3}x_{21}^2\\
&= \frac{35}{3}u_1^2 - \frac{80}{3}u_1x_{11} + \frac{224}{3}u_1x_{21} + \frac{59}{3}x_{11}^2 - \frac{316}{3}x_{11}x_{21} + \frac{428}{3}x_{21}^2
\end{aligned}$$

Now we take derivative to find optimal u_1^*

$$\begin{aligned}
\frac{\partial I(x_1,2)}{\partial u_1} &= 0\\
0 &= \frac{70}{3}u_1 - \frac{80}{3}x_{11} + \frac{224}{3}x_{21}\\
u_1^* &= \frac{3}{70}\left(\frac{80}{3}x_{11} - \frac{224}{3}x_{21}\right)\\
&= \frac{8}{7}x_{11} - \frac{16}{5}x_{21}\\
&= \begin{pmatrix}\frac{8}{7} & -\frac{16}{5}\end{pmatrix}\begin{pmatrix}x_{11}\\ x_{21}\end{pmatrix}
\end{aligned}$$

Hence

$$\boxed{u_1^* = \begin{pmatrix}\frac{8}{7} & -\frac{16}{5}\end{pmatrix}\begin{pmatrix}x_{11}\\ x_{21}\end{pmatrix}}$$

And

$$K(1) = \begin{pmatrix}\frac{8}{7} & -\frac{16}{5}\end{pmatrix} = \begin{pmatrix}1.1429 & -3.2\end{pmatrix}$$

Therefore, we find $I^*(x_1,2)$ using u_1^*

$$I^*(x_1,2) = \left(\begin{pmatrix}1 & -2\\ -1 & 3\end{pmatrix}\begin{pmatrix}x_{11}\\ x_{21}\end{pmatrix} + \begin{pmatrix}0\\ 1\end{pmatrix}u_1^*\right)^T \begin{pmatrix}2 & 1\\ 1 & 2\end{pmatrix}\left(\begin{pmatrix}1 & -2\\ -1 & 3\end{pmatrix}\begin{pmatrix}x_{11}\\ x_{21}\end{pmatrix} + \begin{pmatrix}0\\ 1\end{pmatrix}u_1^*\right) + u_1^{*T}(1)\,u_1^*$$

But $u_1^* = \begin{pmatrix} \frac{8}{7} & -\frac{16}{5} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix}$, hence the above becomes

$$\begin{aligned} I^*(x_1, 2) &= \begin{pmatrix} x_{11} - 2x_{21} \\ \frac{1}{7}x_{11} - \frac{1}{5}x_{21} \end{pmatrix}^T \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x_{11} - 2x_{21} \\ \frac{1}{7}x_{11} - \frac{1}{5}x_{21} \end{pmatrix} + \left(\begin{pmatrix} \frac{8}{7} & -\frac{16}{5} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix} \right)^T \begin{pmatrix} \frac{8}{7} & -\frac{16}{5} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix} \\ &= (x_{11} - 2x_{21}) \left(\frac{15}{7}x_{11} - \frac{21}{5}x_{21} \right) - \left(\frac{1}{5}x_{21} - \frac{1}{7}x_{11} \right) \left(\frac{9}{7}x_{11} - \frac{12}{5}x_{21} \right) \\ &+ x_{11} \left(\frac{64}{49}x_{11} - \frac{128}{35}x_{21} \right) - x_{21} \left(\frac{128}{35}x_{11} - \frac{256}{25}x_{21} \right) \\ &= \frac{178}{49}x_{11}^2 - \frac{82}{5}x_{11}x_{21} + \frac{478}{25}x_{21}^2 \end{aligned}$$

We have to write the above as x^TCx in order to update the state the next stage. As before, we solve for C from

$$\begin{aligned} \frac{178}{49}x_{11}^2 - \frac{82}{5}x_{11}x_{21} + \frac{478}{25}x_{21}^2 &= \begin{pmatrix} x_{11} & x_{21} \end{pmatrix} \begin{pmatrix} c_{11} & c_{12} \\ c_{12} & c_{22} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix} \\ &= c_{11}x_{11}^2 + 2c_{12}x_{11}x_{21}c_{22}x_{21}^2 \end{aligned}$$

Hence $c_{11} = \frac{178}{49}, c_{22} = \frac{478}{25}, c_{12} = -\frac{82}{10}$, therefore

$$I^*(x_1, 2) = \begin{pmatrix} x_{11} & x_{21} \end{pmatrix} \begin{pmatrix} \frac{178}{49} & \frac{82}{10} \\ \frac{82}{10} & \frac{478}{25} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix}$$

Now we back one more final step to find $K(0)$. We find $I(x(N-3), 3) = I(x(0), 3)$

$$\begin{aligned} I(x_0, 3) &= \min_{u0} \{J(x_0, u_0) + I^*(x_1, 2)\} \\ &= \min_{u_0} \left\{ x_1^TQx_1 + u_0^TRu_0 + I^*(x_1, 2) \right\} \end{aligned}$$

But $x_1 = Ax_0 + Bu_0$ from state equation, hence the above becomes

$$\begin{aligned} I(x_0, 3) &= \min_{u_0} \left\{ (Ax_0 + Bu_0)^T Q (Ax_0 + Bu_0) + u_0^TRu_0 + I^*(x_1, 2) \right\} \\ &= \left(\begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u_0 \right)^T \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \left(\begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u_0 \right) + u_0^2 \\ &+ x_1^T \begin{pmatrix} \frac{178}{49} & \frac{82}{10} \\ \frac{82}{10} & \frac{478}{25} \end{pmatrix} x_1 \end{aligned}$$

Now we have to use the state equations in $I^*(x_1, 2)$ to update the last term above, this is important, since everything should be at the same state

$$\begin{aligned} I(x_0, 3) &= \left(\begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix} + \begin{pmatrix} 0 \\ u_0 \end{pmatrix} \right)^T \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \left(\begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix} + \begin{pmatrix} 0 \\ u_0 \end{pmatrix} \right) + u_0^2 \\ &+ \left(\begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix} + \begin{pmatrix} 0 \\ u_0 \end{pmatrix} \right)^T \begin{pmatrix} \frac{178}{49} & \frac{82}{10} \\ \frac{82}{10} & \frac{478}{25} \end{pmatrix} \left(\begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix} + \begin{pmatrix} 0 \\ u_0 \end{pmatrix} \right) \\ &= (x_{11} - 2x_{21})(u_0 + x_{11} - x_{21}) + u_0^2 + (2u_0 - x_{11} + 4x_{21})(u_0 - x_{11} + 3x_{21}) \\ &+ (x_{11} - 2x_{21}) \left(\frac{41}{5}u_0 - \frac{1119}{245}x_{11} + \frac{4247}{245}x_{21} \right) + \\ &\left(\frac{478}{25}u_0 - \frac{273}{25}x_{11} + \frac{1024}{25}x_{21} \right)(u_0 - x_{11} + 3x_{21}) \\ &= \frac{553}{25}u_0^2 - \frac{596}{25}u_0x_{11} + \frac{2248}{25}u_0x_{21} + \frac{10\,232}{1225}x_{11}^2 - \frac{70\,132}{1225}x_{11}x_{21} + \frac{125\,208}{1225}x_{21}^2 \end{aligned}$$

Now we take derivative to find optimal u_0^*

$$\frac{\partial I(x_0,3)}{\partial u_0}=0$$

$$0=2\frac{553}{25}u_0-\frac{596}{25}x_{11}+\frac{2248}{25}x_{21}$$

Hence

$$u_0^*=\frac{596}{603}x_{11}-\frac{2248}{603}x_{21}$$

Therefore

$$\boxed{u_0^*=\begin{pmatrix}\frac{596}{603} & -\frac{2248}{603}\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}=\begin{pmatrix}0.98839 & -3.728\end{pmatrix}\begin{pmatrix}x_{11}\\x_{21}\end{pmatrix}}$$

And

$$K(0)=\begin{pmatrix}0.988\,39 & -3.728\end{pmatrix}$$

Matlab `dlqr` gives a slightly different result and the signs are switched. This needs to be looked at it more. Let us verify $K(1)$ using the Bellman dynamic equations we derived earlier which is

$$\begin{aligned}K(1)=&-\left(B^TQB+R+B^T\left(A^TQ\left[Q^{-1}-B\left(B^TQB+R\right)^{-1}B^T\right]QA\right)B\right)^{-1}\\&\left(B^TQA+B^T\left(A^TQ\left[Q^{-1}-B\left(B^TQB+R\right)^{-1}B^T\right]QA\right)A\right)\end{aligned}\tag{10}$$

Let $\Delta=B^T\left(A^TQ\left[Q^{-1}-B\left(B^TQB+R\right)^{-1}B^T\right]QA\right)$, hence

$$K(1)=-\left(B^TQB+R+\Delta B\right)^{-1}\left(B^TQA+\Delta A\right)$$

We already found $K(1)=\begin{pmatrix}1.1429 & -3.2\end{pmatrix}$ using the direct method. Just need to verify using the dynamic equations found.

$$\begin{aligned}\Delta&=\begin{pmatrix}0\\1\end{pmatrix}^T\left(\begin{pmatrix}1&-2\\-1&3\end{pmatrix}^T\begin{pmatrix}2&1\\1&2\end{pmatrix}\left[\begin{pmatrix}2&1\\1&2\end{pmatrix}^{-1}-\begin{pmatrix}0\\1\end{pmatrix}\left(\begin{pmatrix}0\\1\end{pmatrix}^T\begin{pmatrix}2&1\\1&2\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}+1\right)^{-1}\begin{pmatrix}0\\1\end{pmatrix}^T\right]\begin{pmatrix}2&1\\1&2\end{pmatrix}\begin{pmatrix}1&-2\\-1&3\end{pmatrix}\right)\\&=\begin{pmatrix}0\\1\end{pmatrix}^T\begin{pmatrix}1&-1\\-1&4\end{pmatrix}\begin{pmatrix}\frac{2}{3}&-\frac{1}{3}\\-\frac{1}{3}&\frac{1}{3}\end{pmatrix}\begin{pmatrix}1&-1\\-1&4\end{pmatrix}\\&=\begin{pmatrix}-\frac{11}{3}&\frac{26}{3}\end{pmatrix}\end{aligned}$$

Hence (10) becomes

$$\begin{aligned}K(1)&=-\left(\begin{pmatrix}0\\1\end{pmatrix}^T\begin{pmatrix}2&1\\1&2\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}+1+\begin{pmatrix}-\frac{11}{3}&\frac{26}{3}\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\right)^{-1}\left(\begin{pmatrix}0\\1\end{pmatrix}^T\begin{pmatrix}2&1\\1&2\end{pmatrix}\begin{pmatrix}1&-2\\-1&3\end{pmatrix}+\begin{pmatrix}-\frac{11}{3}&\frac{26}{3}\end{pmatrix}\begin{pmatrix}1&-2\\-1&3\end{pmatrix}\right)\\&=-\begin{pmatrix}-\frac{8}{7}&\frac{16}{5}\end{pmatrix}\\&=\begin{pmatrix}1.142\,9&-3.2\end{pmatrix}\end{aligned}$$

Which matches the direct method used above. Similar results are obtained using Matlab. Matlab has the signs reversed, this needs to be investigated to find out why.

```
>> A=[1,-2;-1,3];
>> B=[0;1];
>> Q=[2,1;1,2];
>> R=1;
>> [k,s,e]=dlqr(A,B,Q,R)
k =
-1.3195    3.6050
s =
7.0346  -10.3887
-10.3887   28.3824
e =
0.2679
0.1270
```

In Mathematica:

```
ssm = StateSpaceModel[{{{1, -2}, {-1, 3}}, {{0}, {1}}}];
q = {{1, -2}, {-1, 3}};
r = {{1}};
DiscreteLQRegulatorGains[ssm, {q, r}, 1.0]
{{-1.44328, 3.8633}}
```

.

Next we will talk about variations of dynamic programming. Floor and ceiling. We might want to optimize for maximum of some variable. For economy, this could be ceiling of unemployment. We can modify the dynamic equations to handle these problems. Floor and ceiling violate the additive process we used in D.P. but we can still manage to use the dynamic equations with some modifications.

2.28 Lecture 28. Thursday, April 21, 2016

2.28.1 Variations of dynamic programming, floor and ceiling

Today's lecture is on variations of dynamic programming. Many integer programming problem can be cast as D.P. The emphasis will be heuristics more than proofs.

One such variation is when the objective function is in terms of the floor or ceiling of variable. Another variation is steady state (this is when the number of stages becomes very large and goes to infinity).

Floor and ceiling To begin, say that x_k is scalar. We are interested in the floor of x_k. This is $\min_k x_k$. We can also talk about ceiling. This is $\max_k x(k)$. More formally

$$x_{k+1} = f(x_k, u_k)$$

Where $u(k) \in \Omega$. We introduce a monitoring function $g(x(k))$. Therefore, the floor problem can be stated as

$$\max_{u_k \in \Omega_k} \min_{k=1\cdots N} g(x_k)$$

For example, the ceiling problem could be to minimize the maximum of unemployment, stated as

$$\min_{u_k \in \Omega_k} \max_{k=1\cdots N} g(x_k)$$

Modeling our D.P. analysis Assume we are doing the floor problem now. Then we write

$$I(x_{N-1}, 1) = \max_{u_{N-1} \in \Omega_{N-1}} g(x_N)$$

And for the ceiling

$$I(x_{N-1}, 1) = \min_{u_{N-1} \in \Omega_{N-1}} g(x_N)$$

From now on, we continue with the ceiling problem. For the next stage we obtain

$$I(x_{N-2}, 2) = \min_{u_{N-2} \in \Omega_{N-2}} \left\{ \max \left\{ g(x_N), I^*(x_{N-1}, 1) \right\} \right\}$$

And the recursion formula becomes

$$I(x_L, N-L) = \min_{u_L \in \Omega_L} \left\{ \max \left\{ g(x_{L+1}), I^*(x_{L+1}, N-L-1) \right\} \right\}$$

When applying the above, we see one difference from previous examples of D.P., now we have Ω_L which might depends on $u_{L-1}, u_{L-2}, \cdots$. Now we will go over one example. A simplified economy model example. Let N be the years of horizon planning. Let y_k be the national income for the k year. Let c_k be the consumer expenditure. Let I_k be the business investment. And let u_k be the government expenditure. The constraint is

$$\sum_{k=0}^{N-1} u_k \leq U$$

Where U is the budget given and $u_k \geq 0$. Hence in a given year we have

$$y_k = c_k + I_k + u_{k-1} \tag{1}$$

$$c_k = \alpha y_{k-1} \tag{2}$$

Where α is propensity factor to consume.

$$I_k = \beta \left(c_k - c_{k-1} \right) \tag{3}$$

Reader Eliminate c_k, I_k from (1,2,3) to obtain

$$y_k = \left(1 + \beta\right) \alpha y_{k-1} - \alpha\beta y_{k-2} + u_{k-1} \tag{4}$$

Our goal is to control the output y_k using the input u_k.

Reader Let $z_k = y_k - y_{k-1}$

Therefore, now

$$y_{k-2} = y_{k-1} - z_{k-1}$$

Substituting the above in (4) gives

$$\begin{aligned} y_k &= \left(1 + \beta\right) \alpha y_{k-1} - \alpha\beta \left(y_{k-1} - z_{k-1}\right) + u_{k-1} \\ &= \alpha y_{k-1} + \alpha\beta z_{k-1} + u_{k-1} \end{aligned}$$

We have state equation

$$y_{k+1} = \alpha y_k + \alpha\beta z_k + u_k$$

Next we want state equation for z_k

Reader

$$z_{k+1} = \left(\alpha - 1\right) y_k + \alpha\beta z_k + u_k$$

Now define state $x_1\left(k\right) = y_k$ and $x_2\left(k\right) = z_k$, hence the state equation in matrix form becomes

$$x_{k+1} = \begin{pmatrix} \alpha & \alpha\beta \\ \alpha - 1 & \alpha\beta \end{pmatrix} x_k + \begin{pmatrix} 1 \\ 1 \end{pmatrix} u_k$$

We want to maximize the floor of the national income y_k using D.P. To illustrate two stages, i.e. $N = 2$, let $U = 1$ dollar, and let $\alpha = \beta = \frac{1}{2}$ Hence

$$\begin{pmatrix} x_1\left(k+1\right) \\ x_2\left(k+1\right) \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} \\ \frac{-1}{2} & \frac{1}{4} \end{pmatrix} \begin{pmatrix} x_1\left(k\right) \\ x_2\left(k\right) \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} u_k$$

Monitor function is $g\left(x\left(k\right)\right) = x_1\left(k\right)$

Begin with one stage to go

$$I\left(x\left(1\right), 1\right) = \max_{u(1)\in\Omega_1} x_1\left(2\right) \tag{5}$$

We know $u\left(1\right) \leq 1 - u\left(0\right)$

Hence (5) becomes

$$I\left(x\left(1\right), 1\right) = \max_{u(1)\in[0\cdots 1-u(0)]} \left(\frac{1}{2} x_1\left(1\right) + \frac{1}{4} x_2\left(1\right) + u\left(1\right)\right)$$

The minimizer is $u^*\left(1\right) = 1 - u\left(0\right)$, therefore

$$I^*\left(x\left(1\right), 1\right) = \frac{1}{2} x_1\left(1\right) + \frac{1}{4} x_2\left(1\right) + 1 - u\left(0\right)$$

Now we go back one more step

$$I\left(x\left(0\right), 2\right) = \max_{u(0)\in\Omega_0} \min \left\{g\left(x\left(1\right), I\left(x\left(1\right), 1\right)\right)\right\}$$

Where $\Omega_0 = [0 \dots 1]$ since we have one dollar to start with. Hence

$$I\left(x\left(0\right),2\right) = \max_{u(0)\in[0\dots1]} \min\left\{x_1\left(1\right), \frac{1}{2}x_1\left(1\right) + \frac{1}{4}x_2\left(1\right) + 1 - u\left(0\right)\right\}$$

Back everything to get an equation in $u\left(0\right)$

$$I\left(x\left(0\right),2\right) = \max_{u(0)\in[0\dots1]} \min\left\{\frac{1}{2}x_1\left(0\right) + \frac{1}{4}x_2\left(0\right) + u\left(0\right), \frac{1}{2}\left(\frac{1}{2}x_1\left(0\right) + \frac{1}{4}x_2\left(0\right) + u\left(0\right)\right) + \frac{1}{4}\left(-\frac{1}{2}x_1\left(0\right) + \frac{1}{4}x_2\left(0\right) + u\left(0\right)\right)1 -$$

This reduces to

$$I\left(x\left(0\right),2\right) = \max_{u(0)\in[0\dots1]} \min\left\{A + u\left(0\right), B\right\}$$

Where

$$\begin{aligned} A &= \frac{1}{2}x_1\left(0\right) + \frac{1}{4}x_2\left(0\right) + u\left(0\right) \\ B &= \frac{1}{8}x\left(0\right) + \frac{3}{16}x_2\left(0\right) + 1 - \frac{1}{4}u\left(0\right) \end{aligned}$$

Hence the function to minimize is

$$F\left(u\left(0\right)\right) = \min\left\{A + u\left(0\right), B\right\}$$

Consider case when $A \geq B$ and case $A < B$.

Reader work out the different cases.

2.29 Lecture 29. Tuesday, April 26, 2016

2.29.1 Detailed example for a floor problem

We will start today with one more dynamic problem which will be useful for HW problem. Then we will start on steady state. Example is a floor problem.

$$\begin{aligned}x_1(k+1) &= \min\{x_1(k), x_2(k)\} + u(k)\\ x_2(k+1) &= x_1(k)\,u(k)\end{aligned}$$

And initial state is

$$\begin{aligned}x_1(0) &= 1\\ x_2(0) &= -1\end{aligned}$$

And

$$J = \min_{k=1,2} x_2(k)$$

With $|u(k)| \leq M$. One step to go is

$$\begin{aligned}I(x(1),1) &= \max_{u(1)\in\Omega_1}\{\min J\}\\ &= \max_{u(1)\in\Omega_1} x_2(2)\\ &= \max_{|u(1)|\leq M} x_1(1)\,u(1)\end{aligned}$$

Hence $u^* = M$ sign$(x_1(x))$, therefore

$$I^*(x(1),1) = M\,abs(x_1(1))$$

With two steps

$$\begin{aligned}I(x(0),2) &= \max_{u(0)\in\Omega_0}\{\min\{J(x(1), I(x(1),1))\}\}\\ &= \max_{u(0)\in\Omega_0}\min\{x_2(1), M|x_1(1)|\}\\ &= \max_{u(0)\in\Omega_0}\min\{x_1(0)\,u(0), M|\min\{x_1(0), x_2(0)\} + u(0)|\}\\ &= \max_{u(0)\in\Omega_0}\min\{u(0), M|u(0)-1|\}\end{aligned}$$

Where $F(u) = \min\{u(0), M|u(0)-1|\}$. Consider the case $0 < M \leq \sqrt{2}$ and case $M > \sqrt{2}$. See key solution for HW 7 for the solution.

We now start talking about steady state. Notion of functional equations., then iterative solution to steady state problem, then Riccati equation. We begin with

$$x_{k+1} = f(x_k, u_k)$$

With a constraint on u_k given by $u_k \in \Omega_k$. The branch cost is

$$J = \Psi(x_N) + \sum_{k=0}^{N-1} J_k(x_k, u_k)$$

Then let $N \to \infty$. We are hoping to get J_N and obtain $u_N^* = u_0^*, u_1^*, \cdots, u_{N-1}^*$ optimal controls at each stage. Notice that when N changes, then the whole sequence u_N^* changes also, and not just one term. We now ask, does J_N converge? does u_N^* converge? To make sense of the above, we remove the terminal cost $\Psi(x_N)$ from this analysis. We also want Ω_k to be fixed for any N. This means we have the same constraints all the time.

If steady state solution exist, then it satisfies

$$I(x) = \min_{u\in\Omega}\left\{J(x,u) + I\left(f(x,u)\right)\right\} \tag{1}$$

This is a functional equation. Solving it means finding u^*. The solution u^* in (1) is a function of x. i.e.. u at state x is a feedback. Say $u^* = \sigma(x)$. Substitute this in (1) gives

$$I(x) = \min_{u\in\Omega}\left\{J(x,\sigma(x)) + I\left(f(x,\sigma(x))\right)\right\}$$

We do not know $I(x)$ here. Before, we knew I, but now we do not know I. So we have a function space issue to find $I(x)$.

Example

$$\begin{aligned} J(x,u) &= x^2 + xu + u^2 \\ f(x,u) &= xu + x + u \\ x(k+1) &= x(k)\,u(k) + x(k) + u(k) \end{aligned}$$

Let u be free variable with no constraints. Hence (1) becomes

$$I(x) = \min_u \left\{ x^2 + xu + u^2 + I(xu + x + u) \right\}$$

2.29.2 Functional equations in dynamic programming

This is a functional equation since $I(x)$, appears on both sides of the equation. There are two famous methods to solve functional equations. The first is the iterative method. Begin with initial $I_o(x)$, then $I_{k+1}(x) = \min_{u(k)} \left\{ J(x,u) + I_k\left(f(x,u)\right) \right\}$. We get sequence of solutions of $u(k)$ and $I(x(k))$ and check for convergence.

Example

$$\begin{aligned} J(x,u) &= u^2 + (x-u)^2 \\ x(k+1) &= x(k) - u(k) \end{aligned}$$

$I_0(x) = 0$, hence

$$I_1(x) = \min_u \left\{ u^2 + (x-u)^2 \right\}$$

$\frac{d}{du} = 0$ gives $u^* = \frac{x}{2}$, hence $I_1^*(x) = \frac{x^2}{4} + \frac{x^2}{4} = \frac{x^2}{2}$. Next stage becomes

$$\begin{aligned} I_2(x) &= \min_u \left\{ u^2 + (x-u)^2 + I_1^* \right\} \\ &= \min_u \left\{ u^2 + (x-u)^2 + \frac{(x-u)^2}{2} \right\} \end{aligned}$$

$\frac{d}{du} = 0$ gives $u^* = \frac{3}{5}x$, hence $I_1^*(x) = \frac{3}{5}x^2$.

Reader Continue this process. Does it converge? It will converge eventually leading to $u^* \to kx$

2.30 Lecture 30. Thursday, April 28, 2016

The special problem will be returned next Tuesday.

2.30.1 Steady state and functional equations

The plan for today: we have been talking about steady state. This lead to functional equations in D.P. so far, we talked about iterative solution. Today we will talk about closed form solution. Analogy between differential equations and functional equations. In differential equations, the iterative method is called Picard iterations method.

For linear state equations, we can get closed form solution for the functional equation. We start with a guess of the solution with a parameter to find. Now we will use our main example to illustrate this.

$$\begin{aligned} x_{k+1} &= x_k - u_k \\ J &= \sum u_k^2 + (x_{k+1} - u_k)^2 \end{aligned}$$

Notice the cross term with x and u in it. Now we guess a form for I. Let $I = px^2$ and then we try to find p. First write J above so that all indices are the same with the help of the state equation. This will reduce the chance of error later on

$$\begin{aligned} J &= \sum u_k^2 + ((x_k - u_k) - u_k)^2 \\ &= \sum u_k^2 + x_k^2 + 4u_k^2 - 4x_k u_k \\ &= \sum x_k^2 + 5u_k^2 - 4x_k u_k \end{aligned}$$

Consider

$$\begin{aligned} I(x) &= \min_{u\in\Omega} \left\{ J(x,u) + I\left(f(x,u)\right) \right\} \\ px^2 &= \min_{u\in\Omega} \left\{ \left(5u^2 + x^2 - 4xu\right) + p(x-u)^2 \right\} \qquad (1) \end{aligned}$$

$$\frac{d\left(5u^2 + x^2 - 4xu\right) + p(x-u)^2}{du} = 0$$

$$0 = 2u - 4(x - 2u) - 2p(x-u)$$

Solving gives $u^* = \frac{2+p}{5+p}x$. Substitute back in (1)

$$px^2 = \left(5u^2 + x^2 - 4x\left(\frac{2+p}{5+p}x\right)\right) + p\left(x - \left(\frac{2+p}{5+p}x\right)\right)^2$$

And obtain an equation in p and solve for p. We find roots are $p = 0.302$ and $p = -3.3$. If everything was done correct, there should be a positive root. Always pick the positive one. This is special case of LQR. In LQR there is no cross term between x, u. While in the above there was. **Reader** For $x(0) = 1$ find J^*.

Example Consider

$$x(k+1) = x(k) + 2u(k)$$

With constraint $u(k) \in [-1,1]$

$$J = \sum_{k=0}^{\infty} e^{x(k+1)}$$

Guess $I = ae^x$ then

$$I(x) = \min_{u\in[-1,1]} \left(e^{x+2u} + ae^{x+2u}\right)$$

Reader $u^* = -1$

Therefore

$$ae^x = \left(e^{x-2} + ae^{x-2}\right)$$

Solving gives

$$a = \frac{1}{e^2 - 1} > 0$$

For LQR, the steady state is given by

$$x(k+1) = Ax(k) + Bu(k)$$

$$J = \sum_{k=0}^{\infty} x^T(k+1)\,Qx(k+1) + u^T(k)\,Ru(k)$$

Where Q, R are weight matrices and are positive definite symmetric. I should be quadratic in the state $x(k)$.

$$I(x) = \min_u x^TQx + u^TRu$$

Guess $I = x^TPx$ and now solve for P, this leads to Riccati matrix equation.

$$I(x) = \min_u \left\{x^TQx + u^TRu + (Ax+Bu)^T P(Ax+Bu)\right\} \tag{2}$$

Taking gradient w.r.t. u and setting to zero, gives

$$2Ru + 2B^TPBu + 2B^TPAx = 0$$

$$u^* = -\left(R + B^TPB\right)^{-1} B^TPAx$$

Back to (2) we find

$$\begin{aligned} x^TPx = x^TQx &+ \left(-\left(R+B^TPB\right)^{-1} B^TPAx\right)^T R\left(-\left(R+B^TPB\right)^{-1} B^TPAx\right) \\ &+ \left(Ax + B\left(-\left(R+B^TPB\right)^{-1} B^TPAx\right)\right)^T P\left(Ax + B\left(-\left(R+B^TPB\right)^{-1} B^TPAx\right)\right) \end{aligned}$$

Solving to P, we obtain the Riccati matrix equation

$$\boxed{P = A^TPA - A^TPB\left(R + B^TPB\right)^{-1} B^TPA + Q}$$

In Matlab, use `dare()` to solve this for P.

Remarks Are we sure the solution exist? i.e. does there exist positive definite P that satisfies the Riccati equation above? Yes, a solution exist if (A, B) is controllable system. Notice that the Riccati equation is not linear in P. This is solved numerically. Consider the special case of LQR with one state and one input. Hence everything is scalar. We obtain

$$p = a^2p - apb\frac{1}{r + b^2p}bpa + q$$

Solving for p, show that there is solution $p > 0$. Assume $b > 0$ for controllability.

2.31 Lecture 31. Tuesday, May 3, 2016

This is the last lecture. Final exam is next lecture. Review of special problem and results obtained by different reports. General approaches to solving the special problem included: Cluster analysis, noisy gradient and random search.

2.31.1 Final review for final exam

We talked about steady state. Quadratic regulator has no cross coupling terms between x and u

$$J=\sum_{k=0}^{\infty}x^{T}\left(x+1\right)Qx\left(k+1\right)+u^{T}Ru$$

For general regulator, one can get a cross term as in $J=ax^{2}+bxu+cu$ but we did not discuss this.

Test 3, will have 4 questions on dynamic programming. With dynamic programming, one can solve the problem using the Bellman equations or using the graphical method. If there are finite stages, and the state x is discrete $x\left(k+1\right)=f\left(x\left(k\right),u\left(k\right)\right)$ and if u is discrete, then a graphical method can be used. If there are constraints, this will reduce the size of the tree more.

Course review

We can take an integer linear programming problem, which is hard to solve and treat it as continuous problem under special conditions and solve it much easier. We did not discuss non-linear programming and kuhn-Tucker conditions. But for many non-linear programming problem, it is possible to use the penalty method. There is also large scale linear programming, where sparsity becomes important. Also parallel programming become important for these problems. For dynamic programming, most of the books are on continuous time, and very few discusses discrete dynamic programming problems.

2.32 Lecture 32. Thursday, May 5, 2016

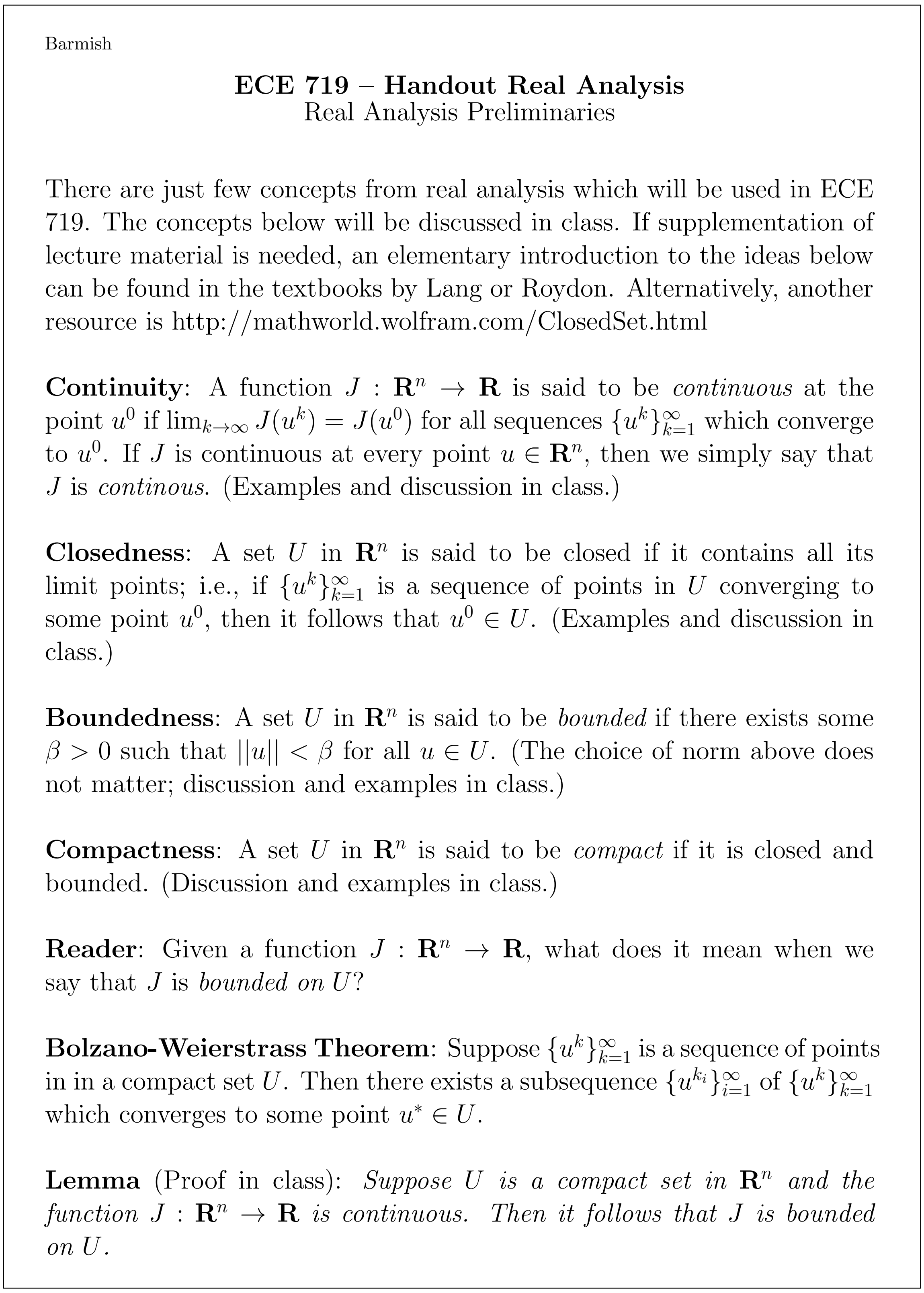
Final exam

Chapter 3

Handouts

Local contents

3.1 Real analysis. January 21, 2016



Barmish

ECE 719 – Handout Real Analysis
Real Analysis Preliminaries

There are just few concepts from real analysis which will be used in ECE 719. The concepts below will be discussed in class. If supplementation of lecture material is needed, an elementary introduction to the ideas below can be found in the textbooks by Lang or Roydon. Alternatively, another resource is http://mathworld.wolfram.com/ClosedSet.html

Continuity: A function $J : \mathbf{R}^n \rightarrow \mathbf{R}$ is said to be *continuous* at the point u^0 if $\lim_{k\rightarrow\infty} J(u^k) = J(u^0)$ for all sequences $\{u^k\}_{k=1}^{\infty}$ which converge to u^0. If J is continuous at every point $u \in \mathbf{R}^n$, then we simply say that J is *continous.* (Examples and discussion in class.)

Closedness: A set U in $\mathbf{R}^n$ is said to be closed if it contains all its limit points; i.e., if $\{u^k\}_{k=1}^{\infty}$ is a sequence of points in U converging to some point u^0, then it follows that $u^0 \in U$. (Examples and discussion in class.)

Boundedness: A set U in $\mathbf{R}^n$ is said to be *bounded* if there exists some $\beta > 0$ such that $||u|| < \beta$ for all $u \in U$. (The choice of norm above does not matter; discussion and examples in class.)

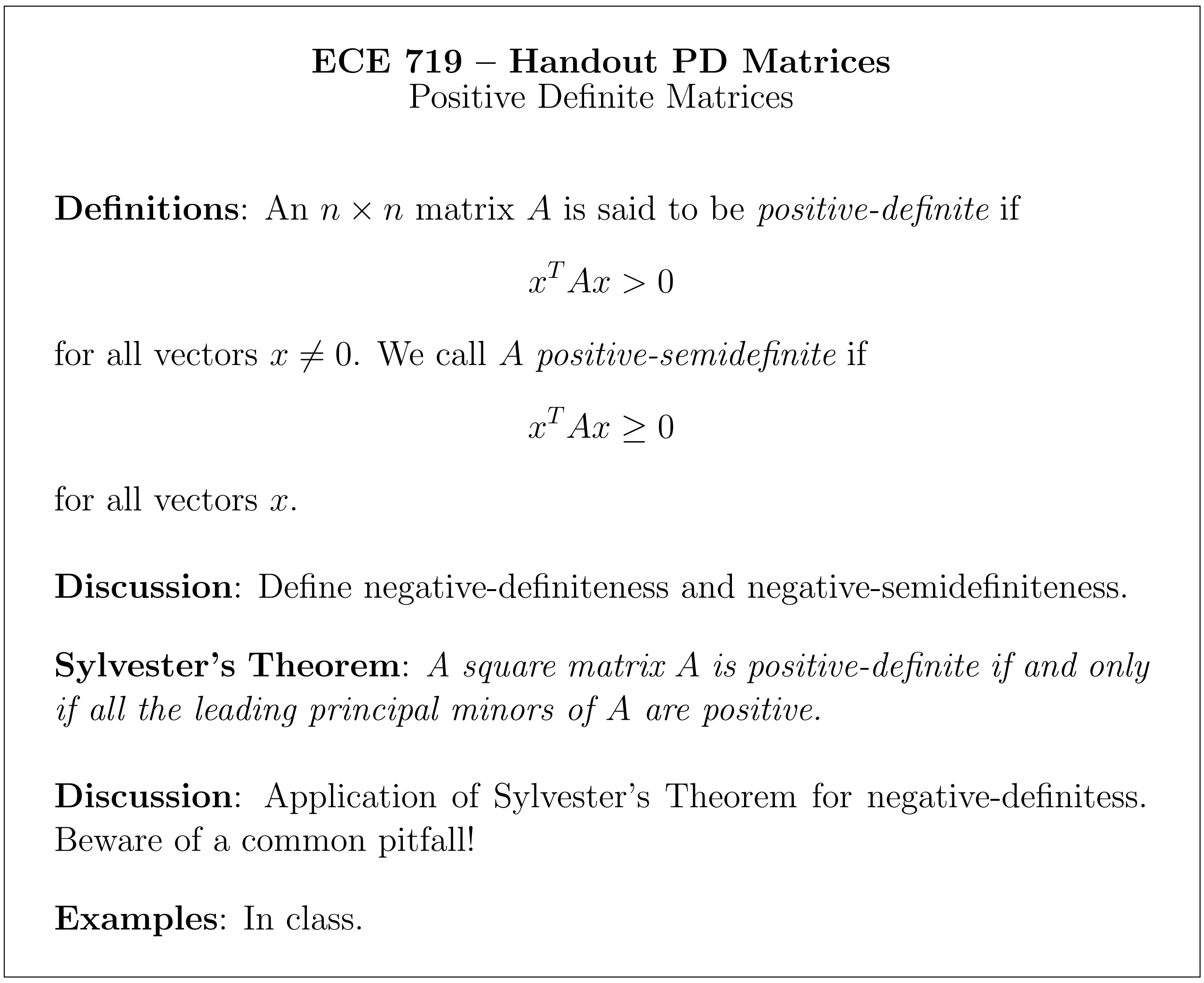
Compactness: A set U in $\mathbf{R}^n$ is said to be *compact* if it is closed and bounded. (Discussion and examples in class.)

Reader: Given a function $J : \mathbf{R}^n \rightarrow \mathbf{R}$, what does it mean when we say that J is *bounded on* U?

Bolzano-Weierstrass Theorem: Suppose $\{u^k\}_{k=1}^{\infty}$ is a sequence of points in in a compact set U. Then there exists a subsequence $\{u^{k_i}\}_{i=1}^{\infty}$ of $\{u^k\}_{k=1}^{\infty}$ which converges to some point $u^* \in U$.

Lemma (Proof in class): *Suppose U is a compact set in $\mathbf{R}^n$ and the function $J : \mathbf{R}^n \rightarrow \mathbf{R}$ is continuous. Then it follows that J is bounded on U.*

3.2 Positive-Definite Matrices. January 26, 2016



ECE 719 – Handout PD Matrices
Positive Definite Matrices

Definitions: An $n \times n$ matrix A is said to be *positive-definite* if

$$x^T Ax > 0$$

for all vectors $x \neq 0$. We call A *positive-semidefinite* if

$$x^T Ax \geq 0$$

for all vectors x.

Discussion: Define negative-definiteness and negative-semidefiniteness.

Sylvester's Theorem: *A square matrix A is positive-definite if and only if all the leading principal minors of A are positive.*

Discussion: Application of Sylvester's Theorem for negative-definitess. Beware of a common pitfall!

Examples: In class.

3.3 Coercivity Theorem. January 27, 2016

Barmish

ECE 719 – Handout Coercivity Theorem

We now provide a result on existence of an optimal element in the absence of a compactness assumption on U.

Theorem: *Suppose $J : U \to R$ is continuous and (positively) coercive and $U \subseteq R^n$ is non-empty and closed. Then an optimal element $u^* \in U$ exists which minimizes J.*

Proof: Select any point $u^0 \in U$. Now, by coercivity, there exists some radius $R > 0$ such that

$$J(u) > J(u^0)$$

for all $u \in U$ with $||u|| > R$. Hence,

$$\inf_{u \in U} J(u) = \inf_{u \in U, ||u|| \leq R} J(u).$$

Notice that the new constraint set on the right hand side above, described by $u \in U$ and $||u|| \leq R$, is the intersection of the closed set U and the compact set given by $||u|| \leq R$. Hence, this set is compact. Now using the previous existence theorem, an optimal element $u^* \in U$ exists minimizing the objective function $J(u)$.

Reader: For maximization problems, define negative coercivity and note that a similar result holds.

3.4 Hessian Theorem. February 8, 2016

Barmish

ECE 719 – Handout Hessian Theorem
A Criterion for Convexity

The result below will be proven in class. It is often quite helpful in deciding if a function is convex.

Hessian Theorem: *Assume $U \subseteq \mathbf{R}^n$ is open and convex and that the function $J : U \to \mathbf{R}$ is twice continuously differentiable. Define the $n \times n$ Hessian matrix $\nabla^2 J(u)$ with (i, j)-th entry*

$$[\nabla^2 J(u)]_{i,j} = \frac{\partial^2 J}{\partial u_i \partial u_j}.$$

Then J is convex on U if and only if $\nabla^2 J(u)$ is positive semi-definite for all $u \in U$.

Reader: Provide a counterexample to show that the theorem cannot be strengthened to assure equivalence between strict convexity and positive-definiteness of the Hessian.

3.5 Proof of Hessian theorem. February 8, 2016

Barmish

ECE 719 – Handout Proof of Hessian Theorem

Preamble: Before proving the theorem, we review two results. The first, already covered in class, is for $n = 1$. That is, we already know that convexity of J on $U = (\alpha, \beta)$ is equivalent satisfaction of the condition

$$\frac{d^2J}{dt^2} \geq 0.$$

The second result, to be proven in class, is for the n-dimensional case: It will be shown that that $J(u)$ is convex on the open convex set U if and only if the following condition is satisfied: Given any point $u \in U$ and any direction $z \in \mathbf{R}^n$, the function

$$\tilde{J}(\lambda) = J(u + \lambda z)$$

is convex for λ in the set

$$\Lambda_z = \{\lambda : u + \lambda z \in U\}.$$

Proof of Hessian Theorem: Using the result for $n = 1$, we know that for $u \in \mathbf{R}^n$, the function $J(u)$ is convex if and only if

$$\frac{d^2J(u + \lambda z)}{d\lambda^2} \geq 0$$

for all $z \in \mathbf{R}^n$ and all $\lambda \in \Lambda_z$. Taking the first derivative above and using the chain rule, we have

$$\frac{d}{d\lambda}J(u + \lambda z) = \frac{\partial J}{\partial u_1}|_{u+\lambda z} z_1 + \frac{\partial J}{\partial u_2}|_{u+\lambda z} z_2 + \cdots + \frac{\partial J}{\partial u_n}|_{u+\lambda z} z_n$$

Similarly, by differentiating once again, we obtain

$$\begin{aligned} \frac{d^2}{d\lambda^2}J(u + \lambda z) &= \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 J}{\partial u_i \partial u_j}|_{u+\lambda z} z_i z_j \\ &= z^T \nabla^2 J(u + \lambda z) z. \end{aligned}$$

Now, to complete the proof, we observe the following: Satisfaction of the necessary and sufficient convexity condition

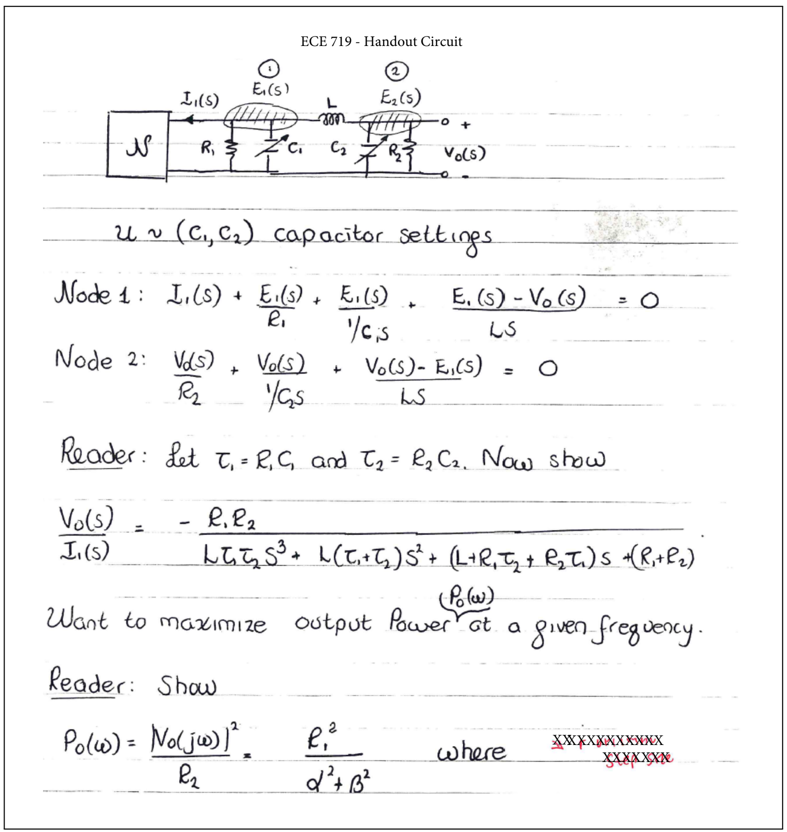
$$z^T \nabla^2 J(u + \lambda z) z \geq 0$$

for all admissible (u, z, λ) triples is equivalent to

$$z^T \nabla^2 J(u) z \geq 0$$

for all $u \in U$. Now, this condition is equivalent to the requirement that $\nabla^2 J(u)$ is positive semi-definite for all $u \in U$.

3.6 Handout circuit. February 23, 2016


①
②
$I_1(s)$
$E_1(s)$
L
$E_2(s)$
$\mathcal{N}$
R_1
C_1
C_2
R_2
+
$V_0(s)$
-

ECE 719 - Handout Circuit

$u \sim (C_1, C_2)$ capacitor settings

Node 1: $I_1(s) + \frac{E_1(s)}{R_1} + \frac{E_1(s)}{1/C_1 s} + \frac{E_1(s) - V_0(s)}{Ls} = 0$

Node 2: $\frac{V_0(s)}{R_2} + \frac{V_0(s)}{1/C_2 s} + \frac{V_0(s) - E_1(s)}{Ls} = 0$

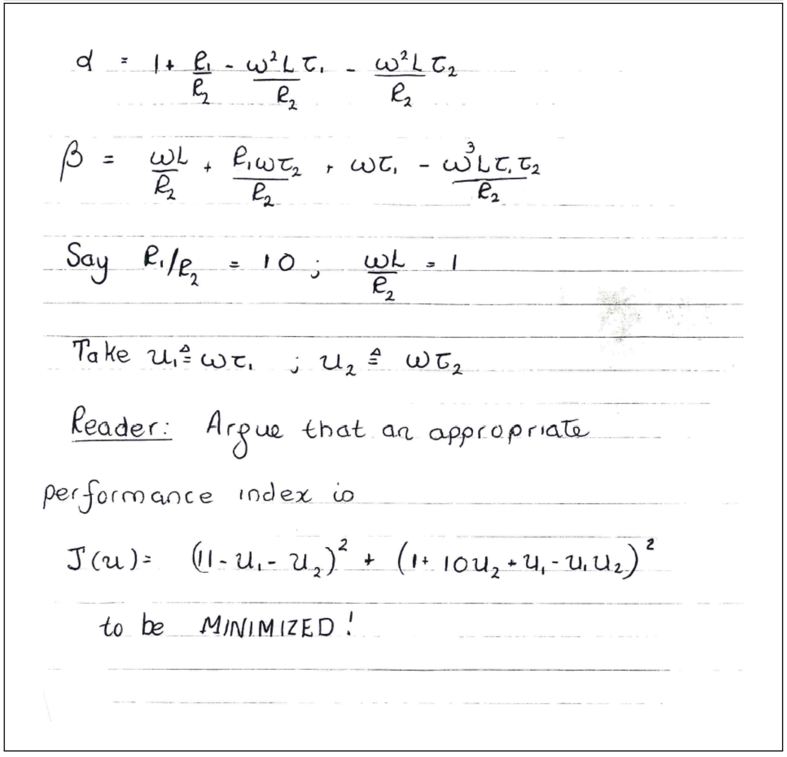
Reader: Let $\tau_1 = R_1 C_1$ and $\tau_2 = R_2 C_2$. Now show

$$\frac{V_0(s)}{I_1(s)} = \frac{-R_1 R_2}{L\tau_1\tau_2 s^3 + L(\tau_1+\tau_2)s^2 + (L + R_1\tau_2 + R_2\tau_1)s + (R_1+R_2)}$$

Want to maximize output $\overbrace{\text{Power}}^{P_0(\omega)}$ at a given frequency.

Reader: Show

$P_0(\omega) = \frac{|V_0(j\omega)|^2}{R_2} = \frac{R_1^2}{\alpha^2 + \beta^2}$ where XXXXXXXXXXXX XXXXXXX



$$\alpha = 1 + \frac{R_1}{R_2} - \frac{\omega^2 L \tau_1}{R_2} - \frac{\omega^2 L}{R_2}\tau_2$$

$$\beta = \frac{\omega L}{R_2} + \frac{R_1 \omega \tau_2}{R_2} + \omega\tau_1 - \frac{\omega^3 L \tau_1 \tau_2}{R_2}$$

Say $R_1/R_2 = 10$; $\frac{\omega L}{R_2} = 1$

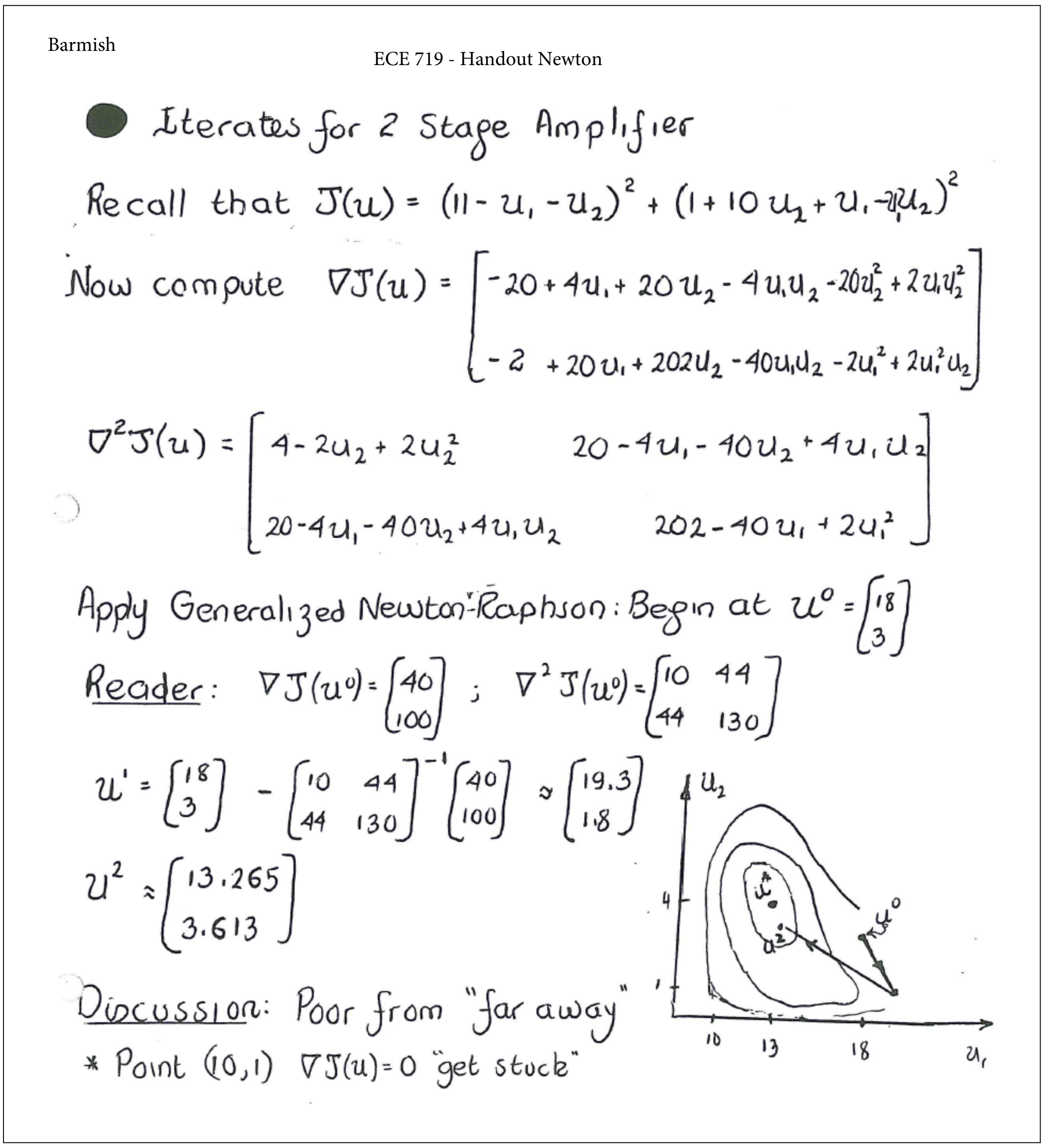
Take $u_1 \triangleq \omega\tau_1$; $u_2 \triangleq \omega\tau_2$

Reader: Argue that an appropriate performance index is

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

to be MINIMIZED!

3.7 Handout Newton. February 26, 2016



Barmish

ECE 719 - Handout Newton

● Iterates for 2 Stage Amplifier

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

Now compute $\nabla J(u) = \begin{bmatrix} -20 + 4u_1 + 20u_2 - 4u_1u_2 - 20u_2^2 + 2u_1u_2^2 \\ -2 + 20u_1 + 202u_2 - 40u_1u_2 - 2u_1^2 + 2u_1^2u_2 \end{bmatrix}$

$$\nabla^2 J(u) = \begin{bmatrix} 4 - 2u_2 + 2u_2^2 & 20 - 4u_1 - 40u_2 + 4u_1u_2 \\ 20 - 4u_1 - 40u_2 + 4u_1u_2 & 202 - 40u_1 + 2u_1^2 \end{bmatrix}$$

Apply Generalized Newton-Raphson: Begin at $u^0 = \begin{bmatrix} 18 \\ 3 \end{bmatrix}$

<u>Reader</u>: $\nabla J(u^0) = \begin{bmatrix} 40 \\ 100 \end{bmatrix}$; $\nabla^2 J(u^0) = \begin{bmatrix} 10 & 44 \\ 44 & 130 \end{bmatrix}$

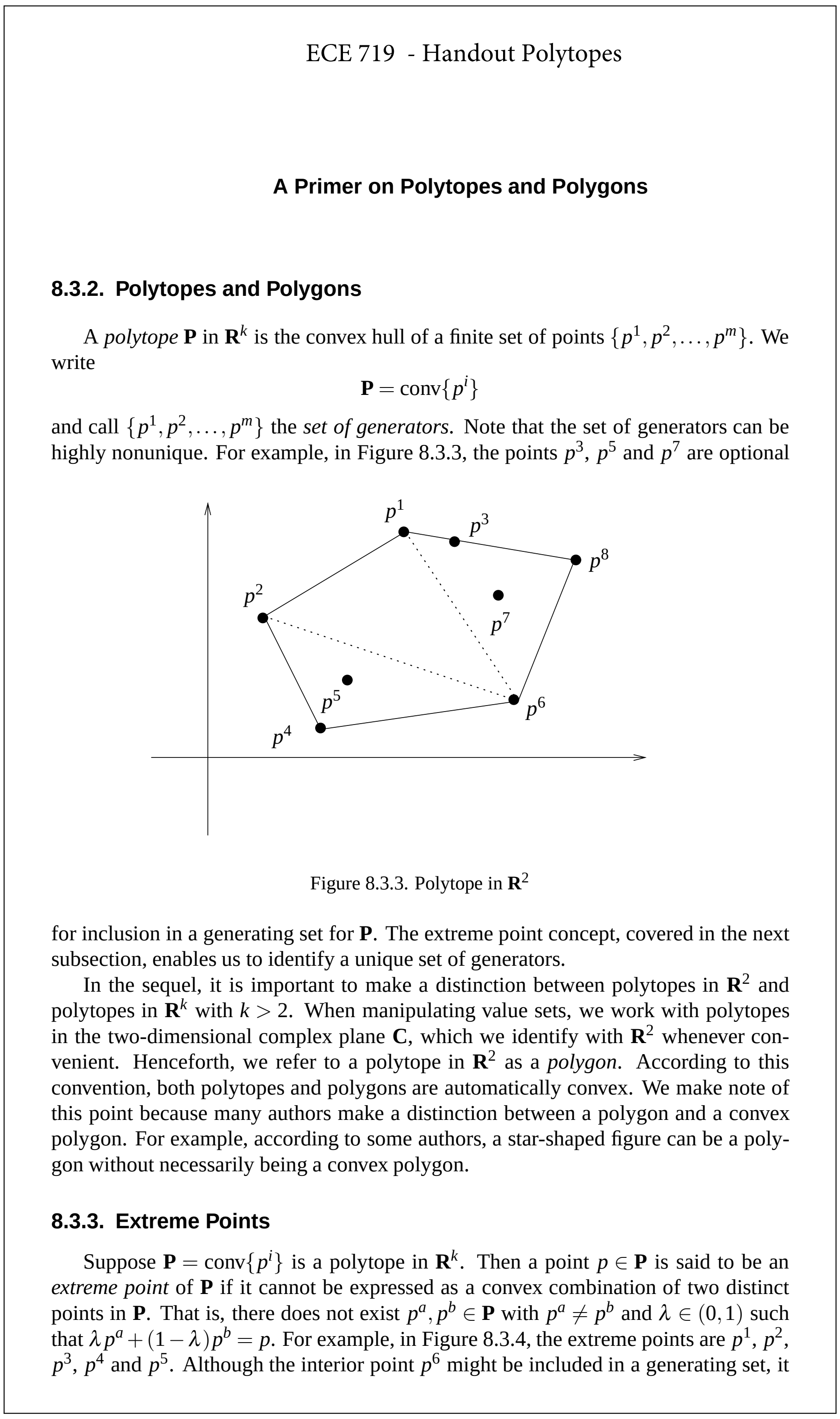
$$u^1 = \begin{bmatrix} 18 \\ 3 \end{bmatrix} - \begin{bmatrix} 10 & 44 \\ 44 & 130 \end{bmatrix}^{-1} \begin{bmatrix} 40 \\ 100 \end{bmatrix} \approx \begin{bmatrix} 19.3 \\ 1.8 \end{bmatrix}$$

$$u^2 \approx \begin{bmatrix} 13.265 \\ 3.613 \end{bmatrix}$$

<u>Discussion</u>: Poor from "far away"

* Point (10,1) $\nabla J(u) = 0$ "get stuck"

3.8 Handout polytopes, march 3 2016



ECE 719 - Handout Polytopes

A Primer on Polytopes and Polygons

8.3.2. Polytopes and Polygons

A *polytope* $\mathbf{P}$ in $\mathbf{R}^k$ is the convex hull of a finite set of points $\{p^1, p^2, \ldots, p^m\}$. We write

$$\mathbf{P} = \text{conv}\{p^i\}$$

and call $\{p^1, p^2, \ldots, p^m\}$ the *set of generators*. Note that the set of generators can be highly nonunique. For example, in Figure 8.3.3, the points p^3, p^5 and p^7 are optional for inclusion in a generating set for $\mathbf{P}$. The extreme point concept, covered in the next subsection, enables us to identify a unique set of generators.

Figure 8.3.3. Polytope in $\mathbf{R}^2$

In the sequel, it is important to make a distinction between polytopes in $\mathbf{R}^2$ and polytopes in $\mathbf{R}^k$ with $k > 2$. When manipulating value sets, we work with polytopes in the two-dimensional complex plane $\mathbf{C}$, which we identify with $\mathbf{R}^2$ whenever convenient. Henceforth, we refer to a polytope in $\mathbf{R}^2$ as a *polygon*. According to this convention, both polytopes and polygons are automatically convex. We make note of this point because many authors make a distinction between a polygon and a convex polygon. For example, according to some authors, a star-shaped figure can be a polygon without necessarily being a convex polygon.

8.3.3. Extreme Points

Suppose $\mathbf{P} = \text{conv}\{p^i\}$ is a polytope in $\mathbf{R}^k$. Then a point $p \in \mathbf{P}$ is said to be an *extreme point* of $\mathbf{P}$ if it cannot be expressed as a convex combination of two distinct points in $\mathbf{P}$. That is, there does not exist $p^a, p^b \in \mathbf{P}$ with $p^a \neq p^b$ and $\lambda \in (0,1)$ such that $\lambda p^a + (1-\lambda)p^b = p$. For example, in Figure 8.3.4, the extreme points are p^1, p^2, p^3, p^4 and p^5. Although the interior point p^6 might be included in a generating set, it

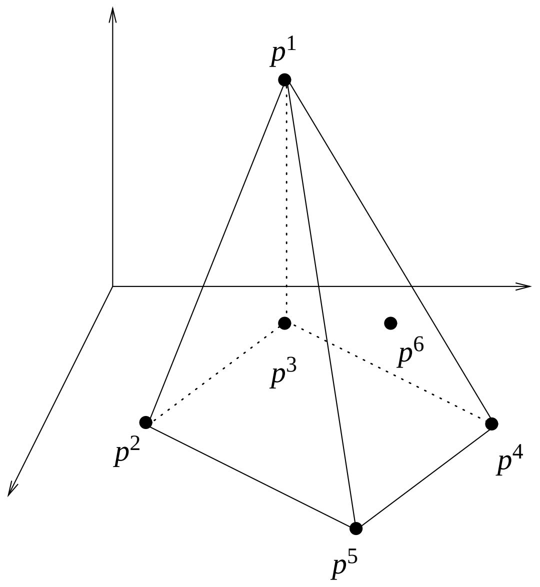


Figure 8.3.4. Polytope in $\mathbf{R}^3$

is not an extreme point. Given a finite set of generators $\{p^i\}$ for a polytope $\mathbf{P}$, the set of extreme points is a subset of the set of generators. Furthermore, the set of extreme points can be called a *minimal generating set* in the sense that any other generating set contains the set of extremes.

In many applications, generators or extreme points of a polytope are specified implicitly rather than explicitly. A prime example occurs in the theory of linear programming where polytopes are described by a set of linear inequalities in the matrix form $Ax \leq b$.

8.3.4. Convex Combination Property

Given a polytope $\mathbf{P} = \text{conv}\{p^1, p^2, \ldots, p^m\}$, every point $p \in \mathbf{P}$ can be expressed as a *convex combination* of the p^i; that is, there exist real scalars $\lambda_1,\lambda_2,\ldots,\lambda_m \geq 0$ such that

$$p = \sum_{i=1}^{m} \lambda_i p^i$$

and

$$\sum_{i=1}^{m} \lambda_i = 1.$$

In the sequel, it is sometimes convenient to describe the constraint set for λ using the notation

$$\Lambda = \{\lambda \in \mathbf{R}^m : \lambda_i \geq 0 \text{ for } i = 1, 2, \ldots, m \text{ and } \sum_{i=1}^{m} \lambda_i = 1\}.$$

For such cases, Λ is called a *unit simplex*.

Primer on Polytopes and Polygons

To illustrate the notion of convex combinations, consider the polygon $\mathbf{P}$ in Figure 8.3.3. Observe that $\mathbf{P}$ is the union of three triangles given by $\mathbf{P}_1 = \mathrm{conv}\{p^1, p^6, p^8\}$, $\mathbf{P}_2 = \mathrm{conv}\{p^1, p^2, p^6\}$ and $\mathbf{P}_3 = \mathrm{conv}\{p^2, p^4, p^6\}$. Now, any point $p \in \mathbf{P}_1$ can be expressed as a convex combination $\lambda_1 p^1 + \lambda_6 p^6 + \lambda_8 p^8$. For example, a point such as p^7 might be obtained with $\lambda_1 = \lambda_6 = \lambda_8 = 1/3$, a point such as p^3 is generated with $\lambda_1 \neq 0$, $\lambda_8 \neq 0$ and $\lambda_6 = 0$ and finally, an extreme point such as p^6 is obtained with $\lambda_6 = 1$ and $\lambda_1 = \lambda_8 = 0$. To conclude, we observe that the description of a point $p \in \mathbf{P}$ as a convex combination of extreme points is nonunique. For example, a point such as p^7 can be expressed as a convex combination of p^1, p^6 and p^8 or p^2, p^6 and p^8.

The fact that we can describe every point $p \in \mathbf{P}$ in Figure 8.3.3 as a convex combination of three or less extreme points is not particular to the example at hand. In fact, Cartheodory's Theorem tells us: Every point in a polytope $\mathbf{P} \subset \mathbf{R}^k$ is expressible as a convex combination of $k+1$ extreme points at most; e.g., see Rockafellar (1970).

8.3.5. Edges of a Polytope

Given any two points x^a and x^b in $\mathbf{R}^k$, we denote the straight line segment joining these points by $[x^a, x^b]$. Notice that every point $x \in [x^a, x^b]$ can be expressed uniquely as a convex combination of x^a and x^b; that is,

$$x = \lambda x^a + (1-\lambda) x^b$$

for some unique $\lambda \in [0,1]$. Furthermore, if $x^a = x^b$, $[x^a, x^b]$ degenerates to a *point* which is viewed as a special case of a line.

We now consider lines of the form $[p^{i_1}, p^{i_2}]$, where p^{i_1} and p^{i_2} are extremes of a given polytope $\mathbf{P}$ and $p^{i_1} \neq p^{i_2}$. We say that $[p^{i_1}, p^{i_2}]$ is an *edge* of $\mathbf{P}$ if the following condition holds: Given any $p^a, p^b \in \mathbf{P}$ with $p^a, p^b \notin [p^{i_1}, p^{i_2}]$, it follows that $[p^a, p^b] \cap [p^{i_1}, p^{i_2}] = \phi$. In two or three dimensions, the edges of a polytope are apparent by inspection. For example, in Figure 8.3.3, edges of the polytope $\mathbf{P}$ are $[p^1, p^2]$, $[p^2, p^4]$, $[p^4, p^6]$, $[p^6, p^8]$ and $[p^8, p^1]$ and in Figure 8.3.4, the edges of the polygon $\mathbf{P}$ are $[p^1, p^2]$, $[p^1, p^3]$, $[p^1, p^4]$, $[p^1, p^5]$, $[p^2, p^3]$, $[p^2, p^5]$, $[p^3, p^4]$ and $[p^4, p^5]$.

8.3.6. Operations on Polytopes

In this subsection, we provide a number of basic facts about operations on polytopes.

LEMMA 8.3.7 (Direct Sum for Two Polytopes): *Given two polytopes* $\mathbf{P}_1 = conv\{p^{1,i_1}\}$ *and* $\mathbf{P}_2 = conv\{p^{2,i_2}\}$ *in* $\mathbf{R}^k$*, the direct sum*

$$\mathbf{P}_1 + \mathbf{P}_2 = \{p^1 + p^2 : p^1 \in \mathbf{P}_1; p^2 \in \mathbf{P}_2\}$$

is a polytope. Moreover,

$$\mathbf{P}_1 + \mathbf{P}_2 = conv\{p^{1,i_1} + p^{2,i_2}\}.$$

REMARKS 8.3.8 (Direct Sum for Polytope and Point): For the special case when $\mathbf{P}_2$ consists of a single point, $\mathbf{P}_1+\mathbf{P}_2$ corresponds to a translation of $\mathbf{P}$. In the lemma below, we provide another useful characterization of $\mathbf{P}_1+\mathbf{P}_2$.

LEMMA 8.3.9 (Another Direct Sum Description): *Given two polytopes* $\mathbf{P}_1 = conv\{p^{1,i}\}$ *and* $\mathbf{P}_2$ *in* $\mathbf{R}^k$, *it follows that*

$$\mathbf{P}_1+\mathbf{P}_2 = conv\bigcup_i (p^{1,i}+\mathbf{P}_2).$$

EXAMPLE 8.3.10 (Illustration of Direct Sum): To illustrate formation of the direct sum via application of the lemma above, suppose that $\mathbf{P}_1 = \text{conv}\{2+2j, 4-2j, 6+6j\}$ and $\mathbf{P}_2$ is the unit square in the complex plane. Then, the lemma leads to the direct sum, which is shown in Figure 8.3.5.

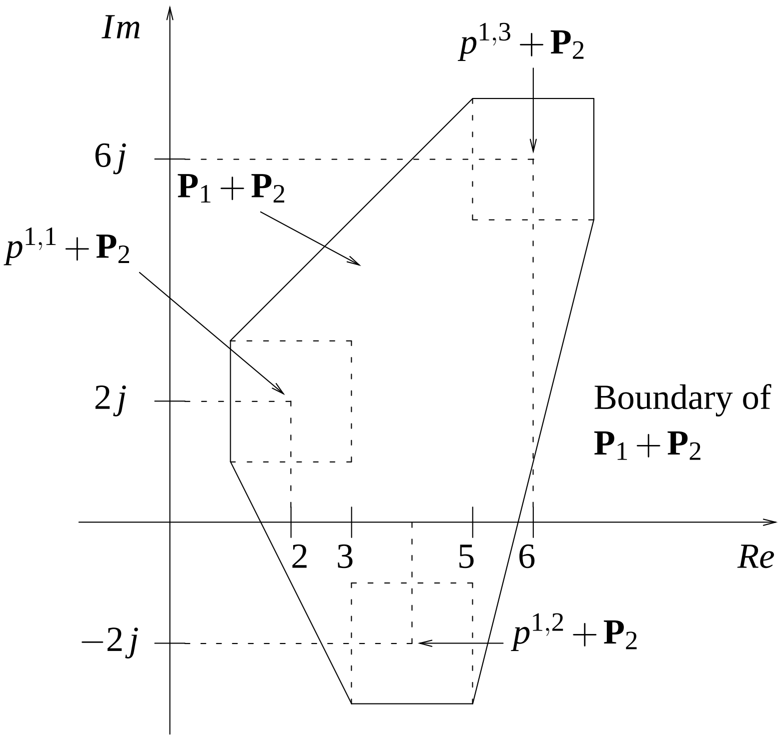
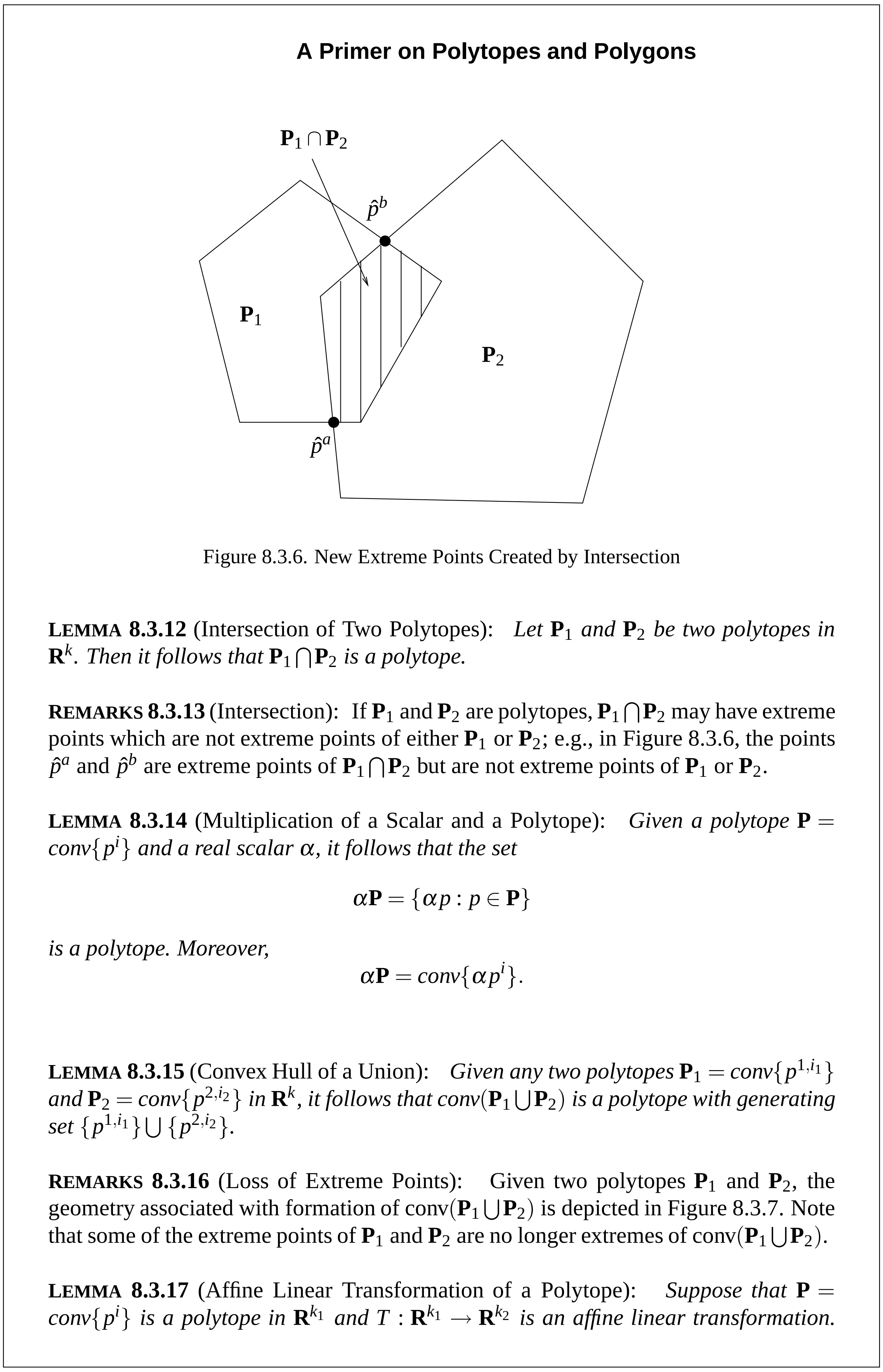


Figure 8.3.5. Formation of Direct Sum in Example 8.3.10

EXERCISE 8.3.11 (Less Restriction on $\mathbf{P}_2$): Argue that Lemma 8.3.9 remains valid when $\mathbf{P}_2$ is an arbitrary convex set which is not necessarily polytopic. Illustrate by considering Example 8.3.10 with $\mathbf{P}_2$ being the unit disc rather than the unit square. Sketch the resulting direct sum $\mathbf{P}_1+\mathbf{P}_2$.



A Primer on Polytopes and Polygons

$\mathbf{P}_1 \cap \mathbf{P}_2$
$\hat{p}^b$
$\mathbf{P}_1$
$\mathbf{P}_2$
$\hat{p}^a$

Figure 8.3.6. New Extreme Points Created by Intersection

LEMMA 8.3.12 (Intersection of Two Polytopes): *Let* $\mathbf{P}_1$ *and* $\mathbf{P}_2$ *be two polytopes in* $\mathbf{R}^k$. *Then it follows that* $\mathbf{P}_1 \bigcap \mathbf{P}_2$ *is a polytope.*

REMARKS 8.3.13 (Intersection): If $\mathbf{P}_1$ and $\mathbf{P}_2$ are polytopes, $\mathbf{P}_1 \bigcap \mathbf{P}_2$ may have extreme points which are not extreme points of either $\mathbf{P}_1$ or $\mathbf{P}_2$; e.g., in Figure 8.3.6, the points $\hat{p}^a$ and $\hat{p}^b$ are extreme points of $\mathbf{P}_1 \bigcap \mathbf{P}_2$ but are not extreme points of $\mathbf{P}_1$ or $\mathbf{P}_2$.

LEMMA 8.3.14 (Multiplication of a Scalar and a Polytope): *Given a polytope* $\mathbf{P} = conv\{p^i\}$ *and a real scalar* α, *it follows that the set*

$$\alpha\mathbf{P} = \{\alpha p : p \in \mathbf{P}\}$$

is a polytope. Moreover,

$$\alpha\mathbf{P} = conv\{\alpha p^i\}.$$

LEMMA 8.3.15 (Convex Hull of a Union): *Given any two polytopes* $\mathbf{P}_1 = conv\{p^{1,i_1}\}$ *and* $\mathbf{P}_2 = conv\{p^{2,i_2}\}$ *in* $\mathbf{R}^k$, *it follows that* $conv(\mathbf{P}_1 \bigcup \mathbf{P}_2)$ *is a polytope with generating set* $\{p^{1,i_1}\} \bigcup \{p^{2,i_2}\}$.

REMARKS 8.3.16 (Loss of Extreme Points): Given two polytopes $\mathbf{P}_1$ and $\mathbf{P}_2$, the geometry associated with formation of conv$(\mathbf{P}_1 \bigcup \mathbf{P}_2)$ is depicted in Figure 8.3.7. Note that some of the extreme points of $\mathbf{P}_1$ and $\mathbf{P}_2$ are no longer extremes of conv$(\mathbf{P}_1 \bigcup \mathbf{P}_2)$.

LEMMA 8.3.17 (Affine Linear Transformation of a Polytope): *Suppose that* $\mathbf{P} = conv\{p^i\}$ *is a polytope in* $\mathbf{R}^{k_1}$ *and* $T : \mathbf{R}^{k_1} \to \mathbf{R}^{k_2}$ *is an affine linear transformation.*

3.9 Sector patrol. March 8, 2016

ECE 719 - Handout Sector Patrol

Thus far, our sector patrol optimization problem is described by the non-negativity constraints

$$u_1 \geq 0, \;\; u_2 \geq 0,$$

the perimeter constraints

$$2u_1 + 2u_2 \leq 10,$$

$$2u_1 + 2u_2 \geq 4$$

and the objective function

$$J(u) = \frac{u_1}{30} + \frac{u_2}{15}.$$

We found the optimum (minimizer) to be

$$(u_1^*, u_2^*) = (2, 0).$$

Reader: Suppose the model, based on traffic analysis, is enhanced to include constraint

$$u_2 \geq 1.5u_1.$$

Carry out a simple graphical analysis in the (u_1, u_2) plane to obtain optimum

$$(u_1^*, u_2^*) = (0.8, 1.2).$$

3.10 Handout Extreme. March 15, 2016

Barmish

ECE 719 – Handout Extreme

We begin with a standard form LP and want to establish the following: *A vector x is an extreme point of the constraint set if and only if x is basic and feasible.*

To establish sufficiency, we assume x is basic and feasible. We must show x is extreme. Without loss of generality, say $x_1, x_2, \ldots, x_m$ are the potentially non-zero components of x. Hence, with a^i being the i-th column of A, we have

$$x_1 a^1 + x_2 a^2 + \cdots + x_m a^m = b.$$

Now proceeding by contradiction, say x is not extreme. Then, there exist two points $v^0, v^1 \in \mathcal{P}$, different from x, and $\lambda \in (0,1)$ such that

$$x = (1-\lambda)v^0 + \lambda v^1. \qquad (*)$$

Since, v^1 and v^2 are feasible, their components are non-negative. Using this non-negativity, the fact that $(*)$ holds and the fact that $x_i = 0$ for $i > m$, it follows that for $i > m$,

$$v_i^1 = 0 = v_i^2$$

for $i > m$. Furthermore, by feasibility of v^1 and v^2, we also have

$$v_1^1 a^1 + v_2^1 a^2 + \cdots + v_m^1 a^m = b; \qquad v_1^2 a^1 + v_2^2 a^2 + \cdots + v_m^2 a^m = b.$$

Therefore, from the equations above,

$$(v_1^1 - x_1)a^1 + (v_2^1 - x_2)a^2 + \cdots + (v_m^1 - x_m)a^m = 0.$$

Now, by linear independence of columns $a^1, a^2, \ldots, a^m$, the equality above forces $x = v^1$. By a similar argument, we also get $x = v^2$. We have now contradicted the fact that the v^i differ from x.

To establish necessity, we now assume x is extreme and must show x is basic and feasible. Again proceeding by contradiction, without loss of generality, say $x_1, x_2, \ldots, x_k$ are the non-vanishing components of x and assume that the corresponding columns of A, $a^1, a^2, \ldots, a^k$, are linearly dependent. Now, pick vector y with first k components, not all zero, and satisfying

$$y_1 a^1 + y_2 a^2 + \ldots y_k a^k = 0$$

and last $n-k$ components y_i being zero. Now, since $x_i > 0$ for $i \leq k$, using the definition of y, the two points $v^1 = x + \epsilon y$ and $v^2 x - \epsilon y$ are feasible for suitably small $\epsilon > 0$. Noting that x can be expressed as the convex combination

$$x = \frac{1}{2}v^1 + \frac{1}{2}v^2,$$

we have contradicted the extremality of x.

Chapter 4

HWs

Local contents

4.1 HW 1

4.1.1 Problem 1

PROBLEM DESCRIPTION

Barmish

ECE 719 – Homework Multilinear

Suppose U is a hypercube in $\mathbf{R}^n$ and $J : U \to \mathbf{R}$ is multilinear function. Argue that the maximum of $J(u)$ over U is attained at a vertex of U. **Remarks**: If your argument involves working with one coordinate at a time, I suggest you review your solution to this problem to see if it is consistent with your solution to the next problem called "Homework Multilinear Revisited." Also note that the minumum of J is also attained at a vertex.

SOLUTION

A multilinear function $f(x_1, \cdots, x_n)$ is one which is linear with respect to each of its independent variables taken one at a time. In other words, when fixing all the independent variables except for one, then it reduces to a linear function in the one variable which is free to change. For an example, for $n = 2$,

$$\boxed{f\left(x, y\right) = ax + by + cxy}$$

is a multilinear function in x, y. When fixing x to some specific value x_0 in $\Re$, the above becomes

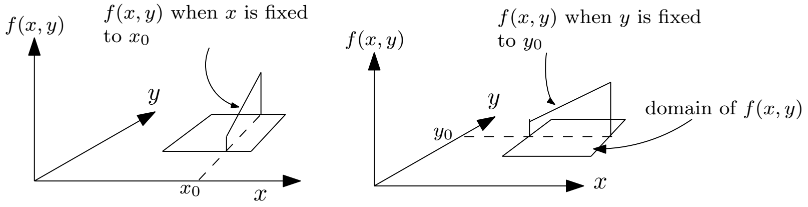
$$\begin{aligned} f\left(x, y\right)\Big|_{x=x_0} &= ax_0 + by + cx_0y \\ &= y\left(b + cx_0\right) + ax_0 \\ &= Ay + B \end{aligned}$$

Where all the constants a, b, c, x_0 have been combined into A and B. Similarly when fixing $y = y_0$, then

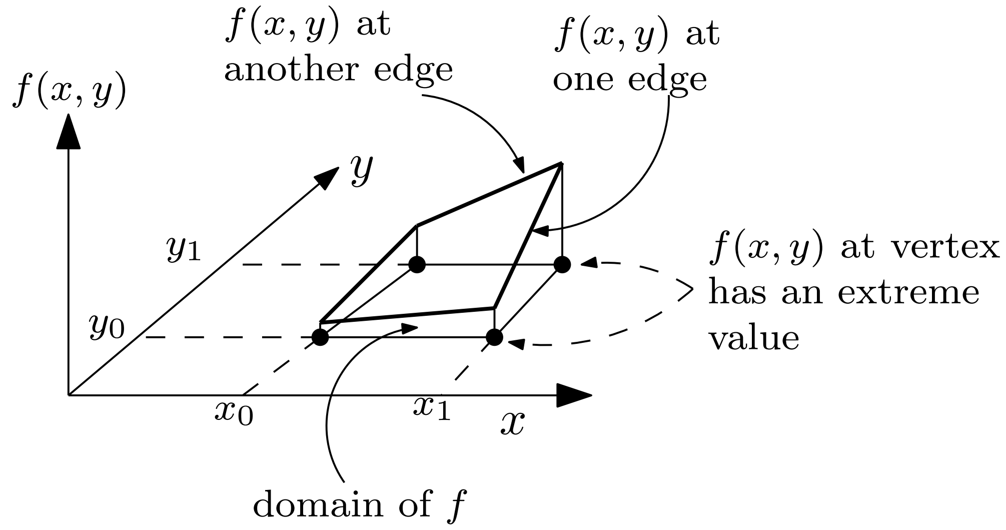
$$f\left(x, y\right)\Big|_{y=y_0} = Cx + D$$

A linear function has it extreme values at the start or at the end of its allowed range of values (The function can be either increasing or decreasing or a constant), this shows that $f(x_1, \cdots, x_n)$ will have its extreme values at one end of the boundaries of each of its variables $x_1, \cdots, x_n$.

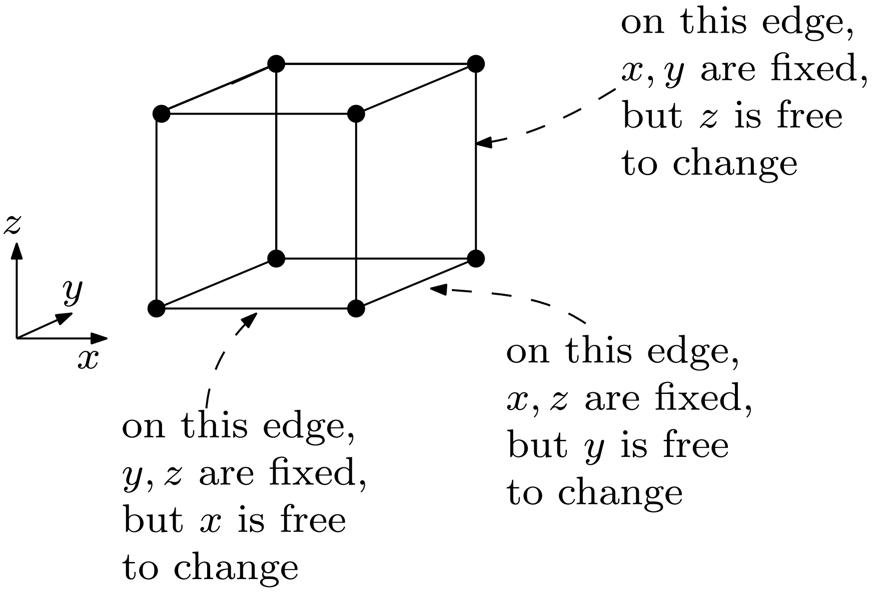
To illustrate what was said so far, taking $n = 2$ and fixing $x = x_0$, then the function will be $f\left(x, y\right)\Big|_{x=x_0}$ and when fixing $y = y_0$ the function will be $f\left(x, y\right)\Big|_{y=y_0}$



To show that the extreme points must be at a "corner" or a vertex, is now straight forward. From the above, the extreme value of the multilinear function must be on an edge. But on any edge, only one of the coordinates is free to change while all the others are fixed. Therefore on any edge of the hypercube (when in higher dimensions) the multilinear function is linear in only one of the parameters at an edge. Hence the function must be either increasing or decreasing on that edge (or be constant). Therefore the function extreme values on the edge is where the edge starts or ends, which is a vertex node. This is illustrated by this diagram for the $\Re^2$case.



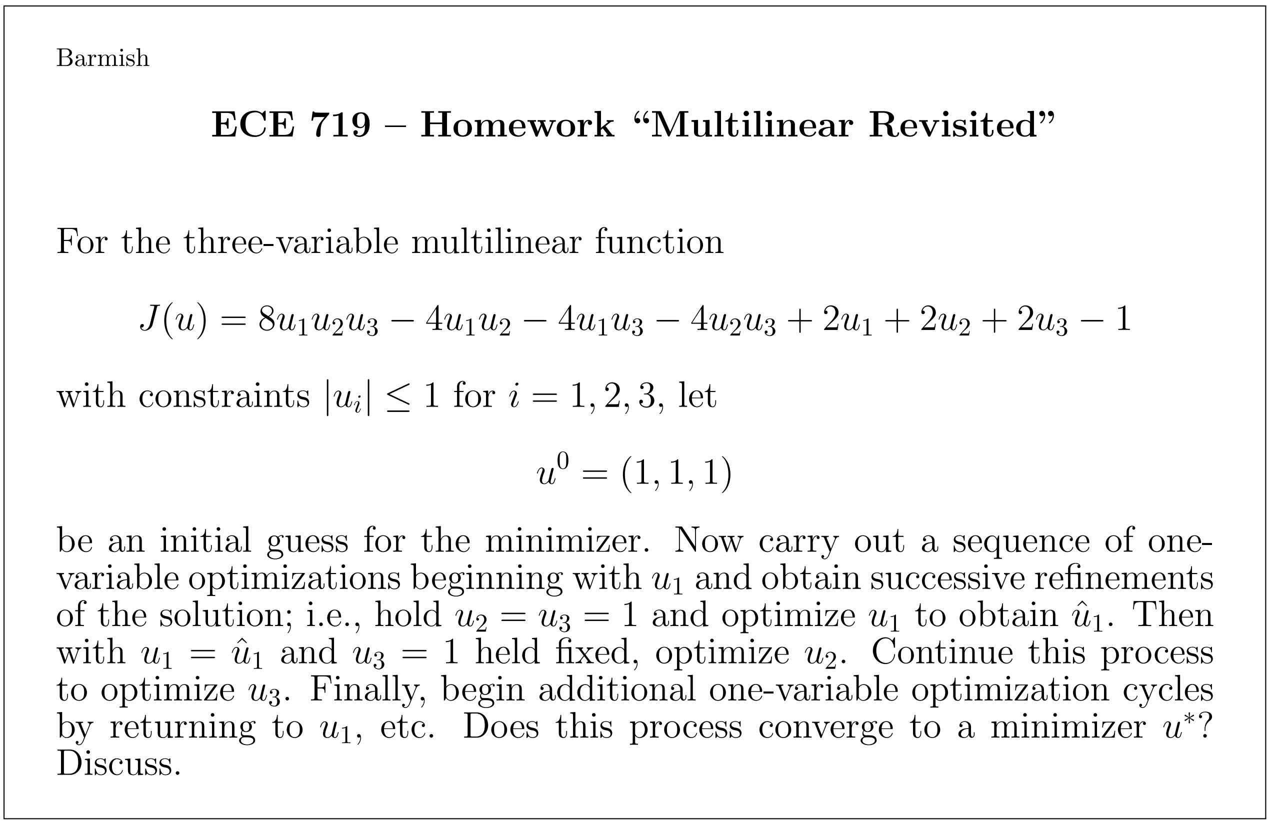
The following illustrates the case for $\Re^3$ by showing few edges and (the function value $f\left(x,y,z\right)$ is hard to show here, since we would need fourth dimension).


on this edge, x,y are fixed, but z is free to change
z
y
x
on this edge, x,z are fixed, but y is free to change
on this edge, y,z are fixed, but x is free to change

This process carry over to higher dimensions hypercube.

4.1.2 Problem 2

PROBLEM DESCRIPTION



Barmish

ECE 719 – Homework "Multilinear Revisited"

For the three-variable multilinear function

$$J(u) = 8u_1u_2u_3 - 4u_1u_2 - 4u_1u_3 - 4u_2u_3 + 2u_1 + 2u_2 + 2u_3 - 1$$

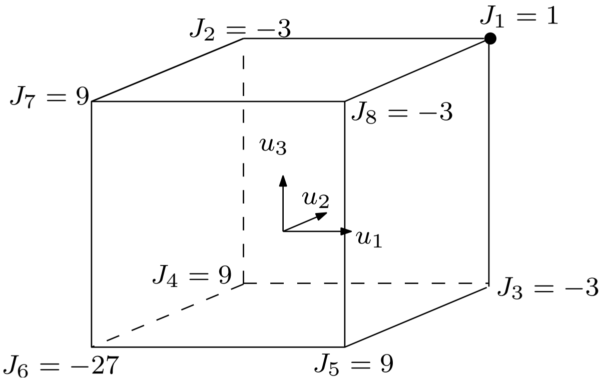
with constraints $|u_i| \leq 1$ for $i = 1, 2, 3$, let

$$u^0 = (1, 1, 1)$$

be an initial guess for the minimizer. Now carry out a sequence of one-variable optimizations beginning with u_1 and obtain successive refinements of the solution; i.e., hold $u_2 = u_3 = 1$ and optimize u_1 to obtain $\hat{u}_1$. Then with $u_1 = \hat{u}_1$ and $u_3 = 1$ held fixed, optimize u_2. Continue this process to optimize u_3. Finally, begin additional one-variable optimization cycles by returning to u_1, etc. Does this process converge to a minimizer u^*? Discuss.

SOLUTION

The function $J(u)$ is defined on a cube. Since it is multilinear in u_1, u_2, u_3, the minimizer point must be at one of the 8 vertices of the cube as shown in problem one. The optimization method the problem asks to perform ***does not converge*** to a minimizer u^* in general. And it does not in this problem. The value of J at each corner are[1] as shown below



In the first stage, we select between J_1 and J_2, (this is the result of fixing $u_2 = u_3 = 1$ and optimizing $J(u)$ to find $\hat{u}_1$). We find that J_2 wins, since it is smaller. Then we select between J_2 and J_7 and find that J_2 still wins. Then we select between J_2 and J_4, and find that J_2 also wins. So at the end of the first phase we mark J_2 as the winner (the minimum so far).

Now we go back to J_1 but select the second edge leaving this node, and select between J_1 and J_8, and find that J_8 wins. Now we select between J_8 and J_7, and find that J_8 wins. Then select between J_8 and J_5, and J_8 still wins. This ends the second phase.

Between $J_8 = -3$ and $J_2 = -3$ (winner of phase one), there is a tie so far.

We go back to J_1 (this is the final stage) and now take the third edge leaving this node, and select between J_1 and J_3. J_3 wins. Then select between J_3 and J_4 and J_3 wins. Then select between J_3 and J_5 and $J_3 = -3$ still is the winner.

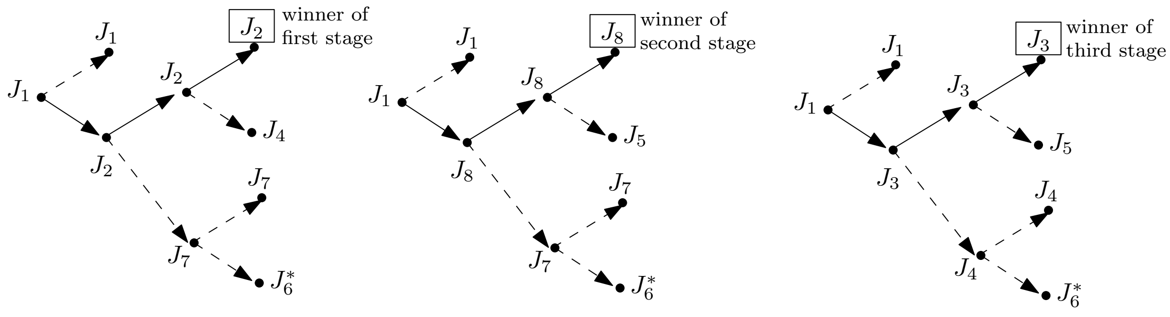
From vertex J_1 we have followed this algorithm over all the three edges leaving it, and found that overall minimum is $J(u) = -3$. If we continue this process, it will just repeat all

[1] small matlab script in the appendix

over.

But $J(u) = -3$ is not the correct value for the global minimum, since the global minimum is at vertex $J_6^* = -27$, which we never got the chance to compare with due to the nature of how this algorithm works. If, for example, J_7 had been smaller than J_2, say -4, then we would had the chance to select J_7 and then compare J_6 with J_7 to find that it is the overall minimum. So this algorithm is ***not guaranteed to converge to the global minimum*** in general.

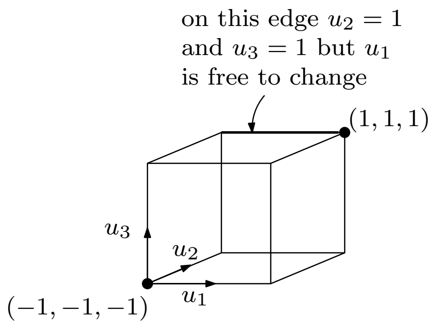
Here is the decision tree used for the above process.



We see that the global minimum at vertex $J_6 = -27$ was not visited. Wrong turn was taken in each stage.

4.1.2.1 Appendix

This appendix can be skipped. It shows the basic calculations for the first phase for illustration, and Matlab code used. Starting at (1,1,1), and fixing $u_2 = u_3 = 1$.


on this edge $u_2 = 1$ and $u_3 = 1$ but u_1 is free to change
$(1,1,1)$
u_3
u_2
u_1
$(-1,-1,-1)$

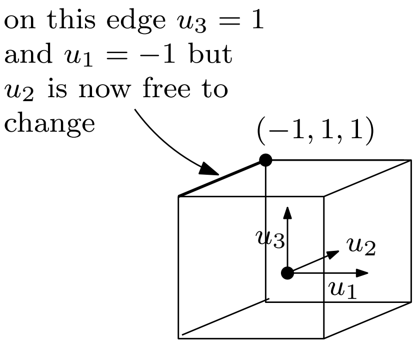
Hence on the first edge above, we have

$$J(u_1,u_2,u_3) = 8u_1u_2u_3 - 4u_1u_2 - 4u_1u_3 - 4u_2u_3 + 2u_1 + 2u_2 + 2u_3 - 1$$

Fixing $u_2 = u_3 = 1$ gives

$$\begin{aligned} J(u_1,1,1) &= 8u_1 - 4u_1 - 4u_1 - 4 + 2u_1 + 2 + 2 - 1 \\ &= 2u_1 - 1 \end{aligned}$$

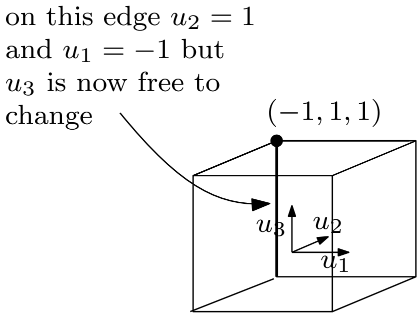
This is minimum when $u_1 = -1$. Hence $\hat{u}_1 = -1$. Now we see that on the above edge, J is smaller on vertex (−1,1,1) than on (1,1,1). Now we look at the next edge, where $u_1 = -1$ and $u_3 = 1$


on this edge $u_3 = 1$ and $u_1 = -1$ but u_2 is now free to change
$(-1,1,1)$
u_3
u_2
u_1

Fixing u_3 at 1 and $u_1 = \hat{u}_1 = -1$ then

$$\begin{aligned} J(\hat{u}_1, u_2, u_3) &= J(-1, u_2, 1) \\ &= 8\hat{u}_1 u_2 u_3 - 4\hat{u}_1 u_2 - 4\hat{u}_1 u_3 - 4u_2 u_3 + 2\hat{u}_1 + 2u_2 + 2u_3 - 1 \\ &= -8u_2 + 4u_2 + 4 - 4u_2 - 2 + 2u_2 + 2 - 1 \\ &= 3 - 6u_2 \end{aligned}$$

Hence $J(\hat{u}_1, u_2, 1)$ is minimum when $u_2 = 1$. Therefore $\hat{u}_2 = 1$. This tells us that J at vertex (−1,1,1) is smaller than on (−1,1,−1). Traveling down the edge between (−1,1,1) and (−1,1,−1), which is done by fixing u_2, u_1 and changing u_3 gives



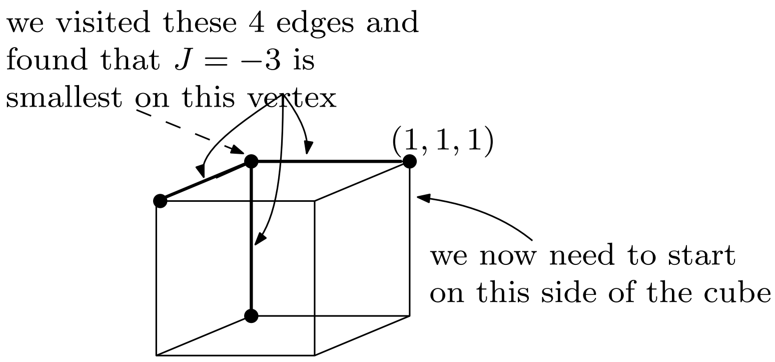
Now we need to find $\hat{u}_3$

$$\begin{aligned} J(\hat{u}_1, \hat{u}_2, u_3) &= J(-1, 1, u_3) \\ &= 8\hat{u}_1 \hat{u}_2 u_3 - 4\hat{u}_1 \hat{u}_2 - 4\hat{u}_1 u_3 - 4\hat{u}_2 u_3 + 2\hat{u}_1 + 2\hat{u}_2 + 2u_3 - 1 \\ &= -8u_3 + 4 + 4u_3 - 4u_3 - 2 + 2 + 2u_3 - 1 \\ &= 3 - 6u_3 \end{aligned}$$

This is minimum at $u_3 = 1$, Therefore $\hat{u}_3 = 1$. This means that J is still smallest at vertex (−1,1,1). We have so far visited 3 edges, and looked at 4 vertices and found that J is smallest at (−1,1,1).

$$\begin{aligned} J(-1,1,1) &= 8u_1 u_2 u_3 - 4u_1 u_2 - 4u_1 u_3 - 4u_2 u_3 + 2u_1 + 2u_2 + 2u_3 - 1 \\ &= -8 + 4 + 4 - 4 - 2 + 2 + 2 - 1 \\ &= -3 \end{aligned}$$

Here is a diagram to illustrate what we did so far



We now repeat the process starting from (1,1,1). Fixing $u_1 = 1$, $u_3 = 1$, but vary u_2. The calculation is similar to the above, and will not be shown. A small Matlab script is given below that was used to verify the results.

```
function nma_HW1_problem2_ECE719
%function to evaluate J at corner of the cube and do
%some syms caluclations.
%1/22/16

syms u1 u2 u3;
J = 8*u1*u2*u3-4*u1*u2-4*u1*u3-4*u2*u3+2*u1+2*u2+2*u3-1;

%first find J value at all the corners, the coordinates  are
a      = {[-1 1] [-1 1] [-1 1]};
```

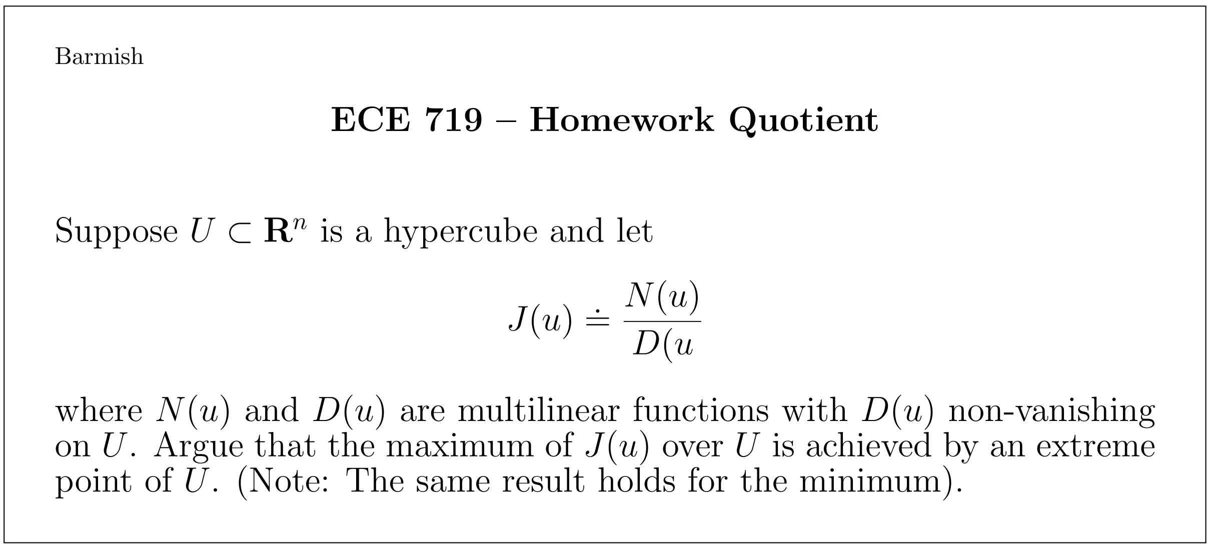
```
coords = allcomb(a{:});

%function to evalute J at each coordinate
f = @(c)subs(J,{u1,u2,u3},c);

%print values at each corner of the cube
vpa(arrayfun(@(i) f(coords(i,:)),1:length(coords),'Uni',false))

J0    = subs(J,{u2,u3},{1,1});
u1Hat = getuHat(J0,u1)
J0    = subs(J,{u1,u3},{u1Hat,1});
u2Hat = getuHat(J0,u2)
J0    = subs(J,{u1,u2},{u1Hat,u2Hat});
u3Hat = getuHat(J0,u3)

end

function uHat = getuHat(J0,u)
uHat = 1;
u0  = subs(J0,u,-1);
if u0<subs(J0,u,1)
    uHat = -1;
end
end
```

Output of the above is

```
>> nma_HW1_problem2_ECE719
[ -27.0, 9.0, 9.0, -3.0, 9.0, -3.0, -3.0, 1.0]
u1Hat =
-1
u2Hat =
1
u3Hat =
1
```

4.1.3 Problem 3

PROBLEM DESCRIPTION



Barmish

ECE 719 – Homework Quotient

Suppose $U \subset \mathbf{R}^n$ is a hypercube and let

$$J(u) \doteq \frac{N(u)}{D(u}$$

where $N(u)$ and $D(u)$ are multilinear functions with $D(u)$ non-vanishing on U. Argue that the maximum of $J(u)$ over U is achieved by an extreme point of U. (Note: The same result holds for the minimum).

SOLUTION

Since $N(u)$ is multilinear, its maximum and minimum values will be on a vertex. Similarly for $D(u)$. Therefore, we only need to compare the ratios $\frac{N(u)}{D(u)}$ on the vertices to find the largest ratio. For example, for $n = 2$, we look at the four ratios, $\frac{N_1}{D_1}, \frac{N_2}{D_2}, \frac{N_3}{D_3}, \frac{N_4}{D_4}$. Where N_i means the value of $N(u)$ at vertex i, and similarly for D_i.

Since one of these N_i will be the largest value that $N(u)$ can take, and one of these D_i

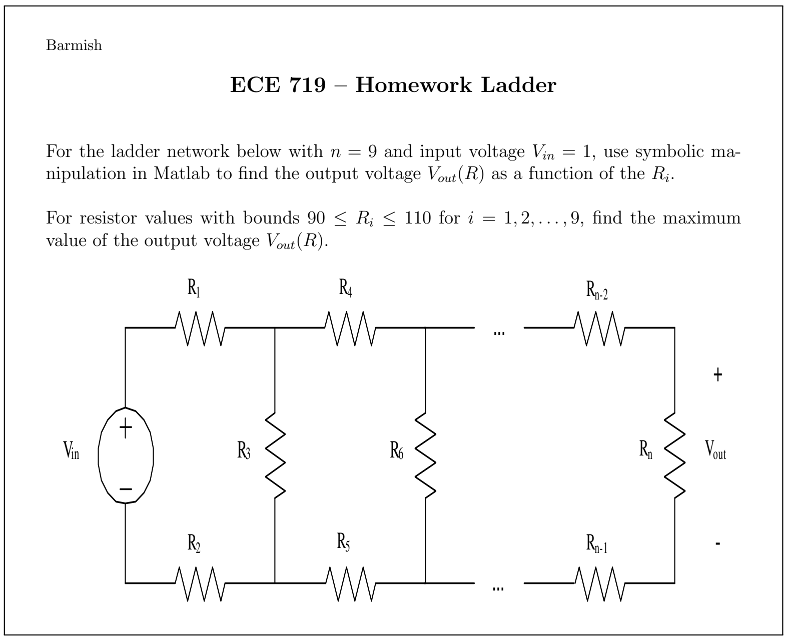
values will be the smallest $D(u)$ can take, then the maximum of $J(u) = \frac{N(u)}{D(u)}$ will be one of these four values.

It can not be at any other u_i location. Proof by contradiction: Let us assume there is a point somewhere else in the domain of $J(u)$, say an internal point u_i where $\frac{N_i}{D_i}$ was the largest. This would imply that N_i is so large as to make $\frac{N_i}{D_i}$ larger than any value at the vertices regardless of what D_i happened to be at u_i, which means that N_i is the maximum of $N(u)$, but this is not possible since the maximum of $N(u)$ must be at a vertex.

Or it could mean that D_i is so small such that $\frac{N_i}{D_i}$ is larger than any value at the vertices regardless of what N_i happened to be, which means that D_i is the minimum of $D(u)$, but this is also not possible, since the minimum of $N(u)$ is at a vertex. Therefore the maximum of $J(u)$ must be at a vertex and can not be at any other point.

4.1.4 Problem 4

PROBLEM DESCRIPTION



Barmish

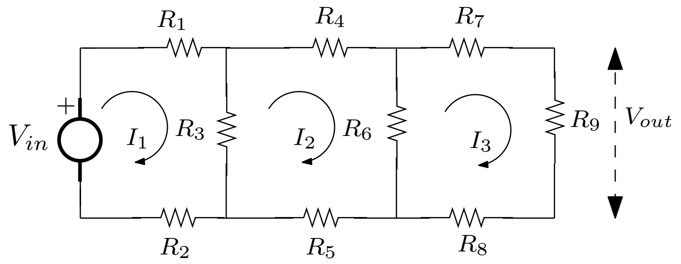
ECE 719 – Homework Ladder

For the ladder network below with $n = 9$ and input voltage $V_{in} = 1$, use symbolic manipulation in Matlab to find the output voltage $V_{out}(R)$ as a function of the R_i.

For resistor values with bounds $90 \leq R_i \leq 110$ for $i = 1, 2, \ldots, 9$, find the maximum value of the output voltage $V_{out}(R)$.

R1
R4
...
Rn-2
+
Vin
+
−
R3
R6
Rn
Vout
R2
R5
...
Rn-1
-

SOLUTION



The total network resistance (Input impedance) is

$$Z_{in} = R_1 + R_2 + R_3 || (R_4 + R_5 + R_6 || (R_7 + R_9 + R_8))$$

Using $X||Y = \frac{XY}{X+Y}$ since in parallel the above become

$$\begin{aligned} Z_{in} &= R_1 + R_2 + \frac{R_3 (R_4 + R_5 + R_6 || (R_7 + R_9 + R_8))}{R_3 + (R_4 + R_5 + R_6 || (R_7 + R_9 + R_8))} \\ &= R_1 + R_2 + \frac{R_3 \left(R_4 + R_5 + \frac{R_6(R_7+R_9+R_8)}{R_6+(R_7+R_9+R_8)}\right)}{R_3 + \left(R_4 + R_5 + \frac{R_6(R_7+R_9+R_8)}{R_6+(R_7+R_9+R_8)}\right)} \end{aligned}$$

The above is the overall ladder network resistance. Let $X = R_4 + R_5 + \frac{R_6(R_7+R_9+R_8)}{R_6+(R_7+R_9+R_8)}$ to simplify the equation

$$Z_{in} = R_1 + R_2 + \frac{XR_3}{R_3 + X}$$

The output impedance is

$$\begin{aligned} Z_{out} &= R_3 \| \left(R_4 + R_5 + R_6 \| \left(R_7 + R_9 + R_8\right)\right) \\ &= \frac{XR_3}{R_3 + X} \end{aligned}$$

Hence V_{out} is now found, using $V_{in} = 1$, from

$$\begin{aligned} \frac{V_{out}}{V_{in}} &= \frac{Z_{out}}{Z_{in}} \\ V_{out} &= \frac{\frac{XR_3}{R_3+X}}{R_1 + R_2 + \frac{XR_3}{R_3+X}} = \frac{XR_3}{R_1\left(R_3 + X\right) + R_2\left(R_3 + X\right) + XR_3} \\ &= \frac{XR_3}{X\left(R_1 + R_2 + R_3\right) + R_1R_3 + R_2R_3} \end{aligned}$$

Since V_{out} multilinear function in $R_i, i = 1 \cdots 9$, the maximum and minimum will occur at the end range values of each resistance, which is 90 and 110. so there are 2^9 different cases to check. A small script is below which calculate these vertex values and shows the maximum V_{out} found. The maximum is

$$V_{out_{\max}} = 0.3147 \text{ volt}$$

Using R_1 and R_2 at 90 ohm, and the rest of the resistors using 110 ohm.

```
function nma_HW1_problem4_ECE719
%function to evaluate Vout at corners of the R^9
%Nasser M. Abbasi
%1/23/16

a = repmat({[90 110]},9,1);
v = allcomb(a{:});
r = arrayfun(@(i) vOut(v(i,:)),1:size(v,1));

%done. Now print min and max, and the vertix at each
[maxValue,indx] = max(r);
fprintf('max is %f at U=\n',maxValue);
v(indx,:)

[minValue,indx] = min(r);
fprintf('min is %f at U=\n',minValue);
v(indx,:)

end
function r = vOut(R)
%evaluate objective function at vertex. See HW
X = R(4)+R(5)+R(6)*(R(7)+R(9)+R(8))/(R(6)+(R(7)+R(9)+R(8)));
r = X*R(3)/(X*(R(1)+R(2)+R(3))+R(1)*R(2)+R(2)*R(3));
end
```

Which generates when run:

```
>> nma_HW1_problem4_ECE719
max is 0.314732 at U=
90    90    110    110    110    110    110    110    110
min is 0.225627 at U=
110    110    90    90    90    90    90    90    90
```

4.1.5 Problem 5

PROBLEM DESCRIPTION

Barmish

ECE 719 – Homework Common Sense

For constraint set $U \subseteq \mathbf{R}^n$ and $J : U \to \mathbf{R}$ suppose that for $k = 1, 2, \ldots, n$, the following condition is satisfied: With all u_i frozen except for $i = k$, the resulting one variable J function is minimized over U by $u_k = u_k^*$, independently of the frozen values of the remaining u_i.

(a) Argue that $u^* = (u_1^*, u_2^*, \ldots, u_n^*)$ minimizes $J(u)$ over U with all u_i being allowed to vary simultaneously.

(b) Give an example with $n = 2$ for which the result in Part (a) applies.

(c) Give an example with $n = 2$ for which the result in Part (a) does not apply.

SOLUTION

4.1.5.1 Part(a)

Since u_k^* minimizes $f(u)$ when all u_i are fixed except for u_k, then this is the same as saying that solving for $\frac{\partial f}{\partial u_k} = 0$ gives u_k^*. Since this is the necessary condition for an extreme point from unconstrained calculus (we still need to check the Hessian for u_k^* being the minimum or the maximum, but we are told here it is a minimizer).

But $\frac{\partial f}{\partial u_k}$ is partial derivative of $f(u)$ w.r.t. to u_k when all other u_i are fixed (by definition). For $\Re^n$ this carries over and becomes the gradient. Therefore

$$\nabla f = 0$$
$$\begin{pmatrix} \frac{\partial f}{\partial u_1} \\ \frac{\partial f}{\partial u_2} \\ \vdots \\ \frac{\partial f}{\partial u_n} \end{pmatrix} = 0$$

Leads to minimum being at $\begin{pmatrix} u_1^* \\ u_2^* \\ \vdots \\ u_n^* \end{pmatrix}$ since we are told that each $\frac{\partial f}{\partial u_k} = 0$ results in u_k^* as the solution.

4.1.5.2 Part(b)

An example where the above applies is $J(x, y) = xy$ on $U = [0,1]$ Since

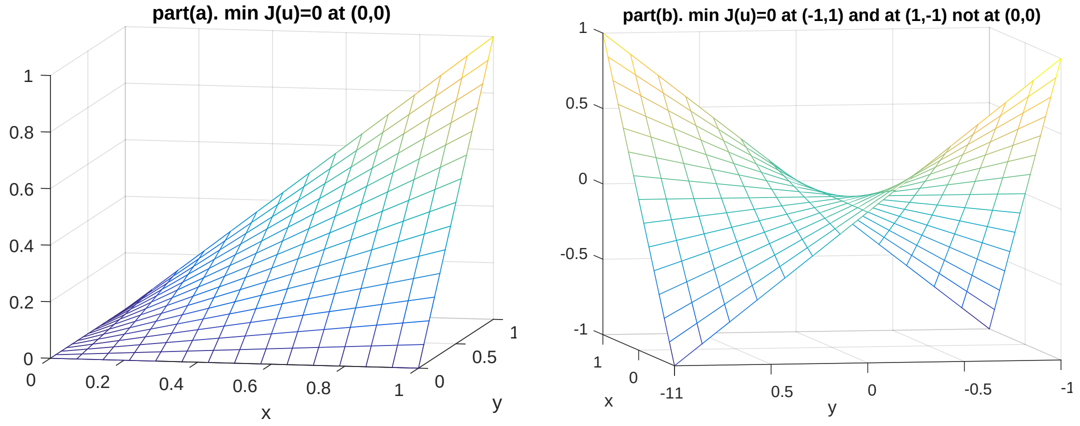
$$\nabla f = 0$$
$$\begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} y \\ x \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Hence a minimizer is $u^* = \begin{pmatrix} x = 0 \\ y = 0 \end{pmatrix}$ and $J(u^*) = 0$ is the global minimum.

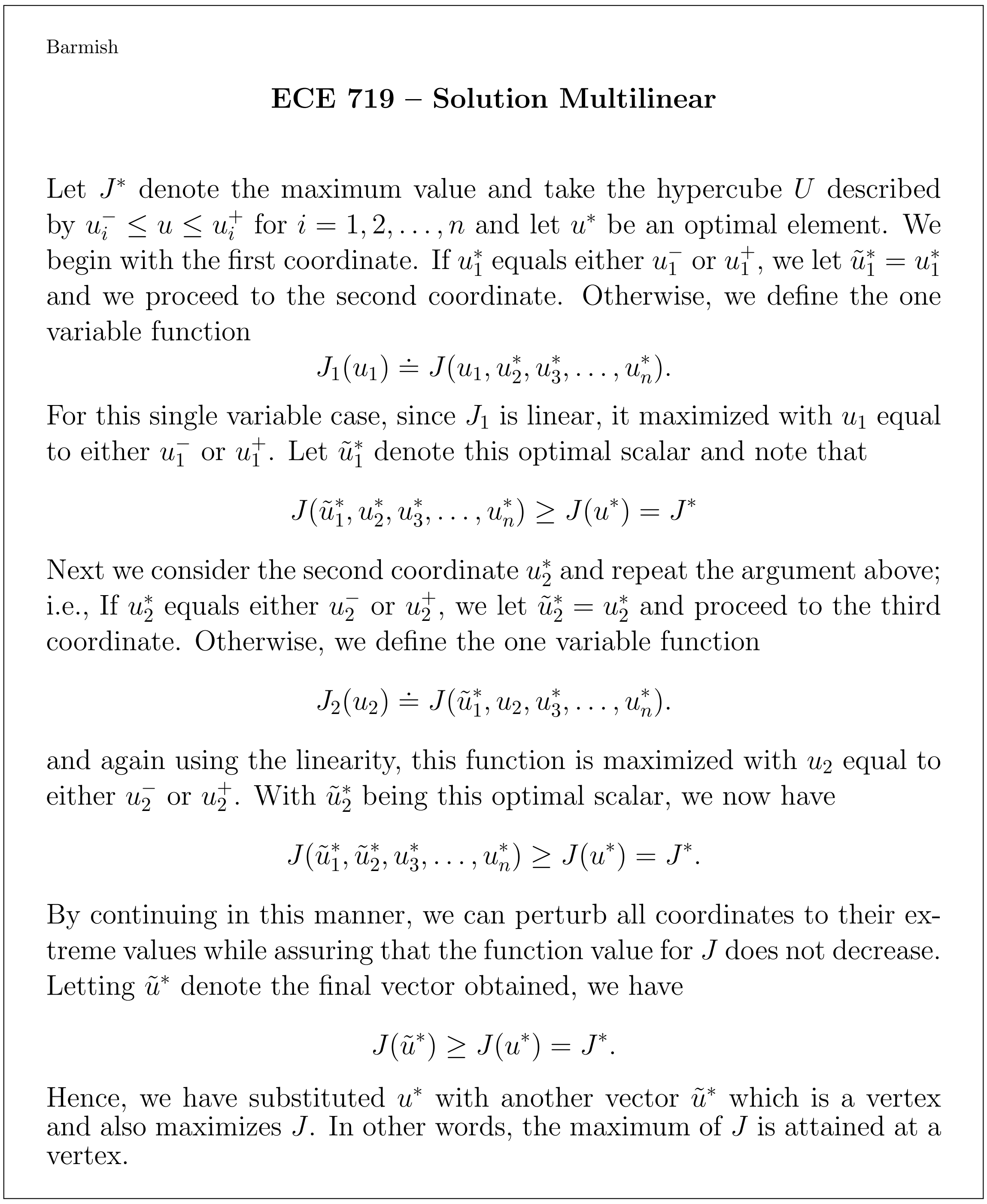
4.1.5.3 Part(c)

An example where part (a) does not apply is $J(x, y) = xy$ on $U = [-1,1]$. Now $u^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ is not the minimizer, since if $x = -1$ and $y = 1$ or if $x = 1$ and $y = -1$, we find $J(u^*) = -1 < 0$.

The following is a plot of $J(x,y) = xy$ of different sets of constraints to illustrate part(b) and (c).



4.1.6 HW 1 key solution



Barmish

ECE 719 – Solution Multilinear

Let J^* denote the maximum value and take the hypercube U described by $u_i^- \leq u \leq u_i^+$ for $i = 1, 2, \ldots, n$ and let u^* be an optimal element. We begin with the first coordinate. If u_1^* equals either u_1^- or u_1^+, we let $\tilde{u}_1^* = u_1^*$ and we proceed to the second coordinate. Otherwise, we define the one variable function

$$J_1(u_1) \doteq J(u_1, u_2^*, u_3^*, \ldots, u_n^*).$$

For this single variable case, since J_1 is linear, it maximized with u_1 equal to either u_1^- or u_1^+. Let $\tilde{u}_1^*$ denote this optimal scalar and note that

$$J(\tilde{u}_1^*, u_2^*, u_3^*, \ldots, u_n^*) \geq J(u^*) = J^*$$

Next we consider the second coordinate u_2^* and repeat the argument above; i.e., If u_2^* equals either u_2^- or u_2^+, we let $\tilde{u}_2^* = u_2^*$ and proceed to the third coordinate. Otherwise, we define the one variable function

$$J_2(u_2) \doteq J(\tilde{u}_1^*, u_2, u_3^*, \ldots, u_n^*).$$

and again using the linearity, this function is maximized with u_2 equal to either u_2^- or u_2^+. With $\tilde{u}_2^*$ being this optimal scalar, we now have

$$J(\tilde{u}_1^*, \tilde{u}_2^*, u_3^*, \ldots, u_n^*) \geq J(u^*) = J^*.$$

By continuing in this manner, we can perturb all coordinates to their extreme values while assuring that the function value for J does not decrease. Letting $\tilde{u}^*$ denote the final vector obtained, we have

$$J(\tilde{u}^*) \geq J(u^*) = J^*.$$

Hence, we have substituted u^* with another vector $\tilde{u}^*$ which is a vertex and also maximizes J. In other words, the maximum of J is attained at a vertex.

Barmish

ECE 719 – Solution Multilinear Revisited

Fixing $u_2 = u_3 = 1$, we obtain the single variable function

$$J(u_1, 1, 1) = 2u_1 - 1$$

which is minimized with $u_1 = -1$. Hence the next solution is

$$u^1 = (-1, 1, 1)$$

with associated cost

$$J(u^1) = -3.$$

For the next step, we minimize the single variable function

$$J(-1, u_2, 1) = -6u_2 + 3$$

and obtain $u_2 = 1$ as the minimizer. Hence $u^2 = u^1$ and we obtain cost given by

$$J(u^2) = J(u^1) = -3.$$

By continuing the iteration we see that

$$u^k = u^1; \;\; J(u^k) = -3$$

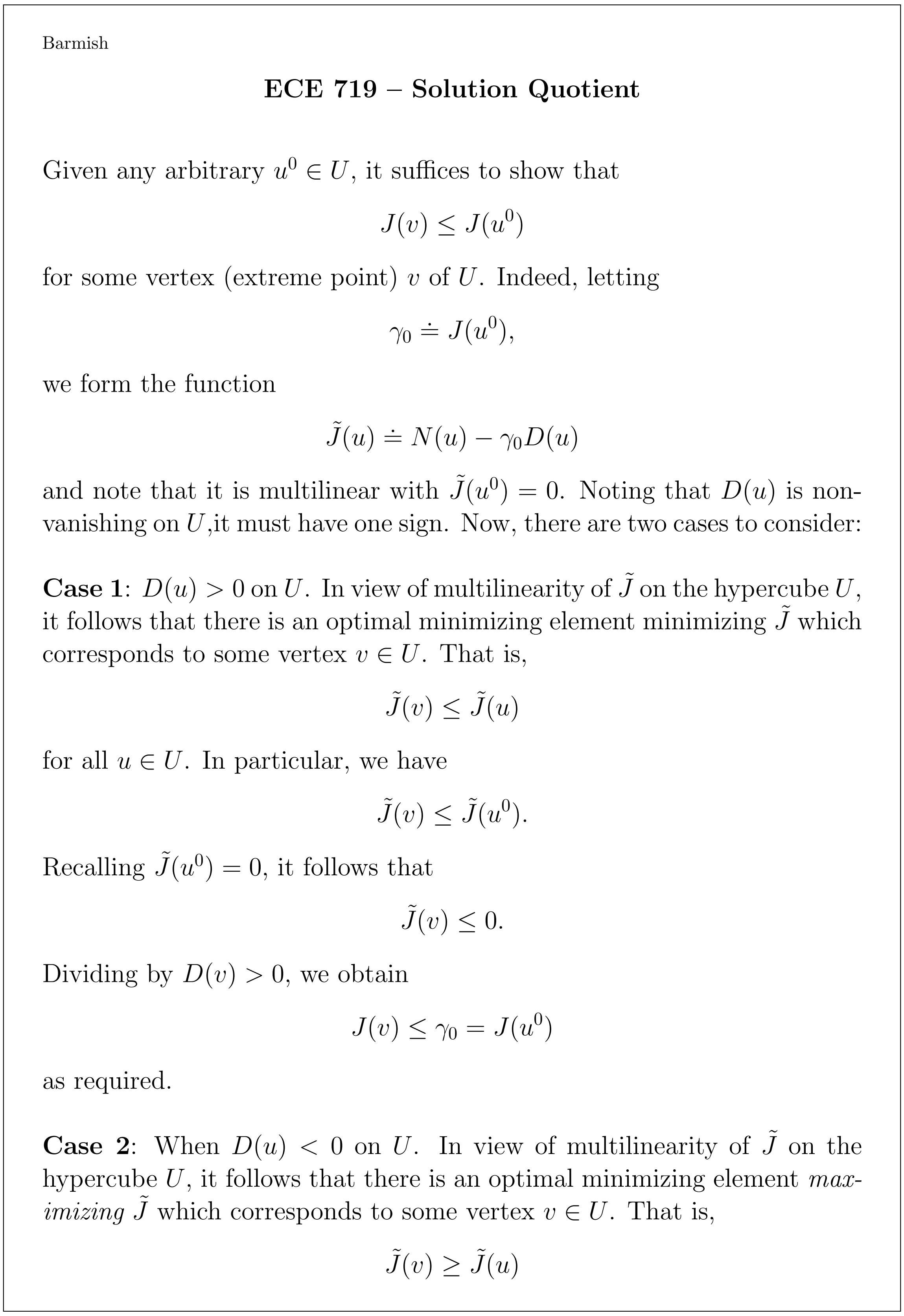
for all $k \geq 3$. That is, the algorithm gets stuck at $(-1, 1, 1)$ whereas the optimal element, obtainable via substitution of vertices, is

$$u^* = (-1, -1, -1)$$

with associated optimal cost

$$J^* = J(u^*) = -27.$$

This example shows one potential pitfall associated with the solution of an n-variable problem via a sequence on one-dimensional optimizations. Notice for the multilinear case above, each new iterate is a "nearest vertex neighbor" of its predecessor.



Barmish

ECE 719 – Solution Quotient

Given any arbitrary $u^0 \in U$, it suffices to show that

$$J(v) \leq J(u^0)$$

for some vertex (extreme point) v of U. Indeed, letting

$$\gamma_0 \doteq J(u^0),$$

we form the function

$$\tilde{J}(u) \doteq N(u) - \gamma_0 D(u)$$

and note that it is multilinear with $\tilde{J}(u^0) = 0$. Noting that $D(u)$ is nonvanishing on U,it must have one sign. Now, there are two cases to consider:

Case 1: $D(u) > 0$ on U. In view of multilinearity of $\tilde{J}$ on the hypercube U, it follows that there is an optimal minimizing element minimizing $\tilde{J}$ which corresponds to some vertex $v \in U$. That is,

$$\tilde{J}(v) \leq \tilde{J}(u)$$

for all $u \in U$. In particular, we have

$$\tilde{J}(v) \leq \tilde{J}(u^0).$$

Recalling $\tilde{J}(u^0) = 0$, it follows that

$$\tilde{J}(v) \leq 0.$$

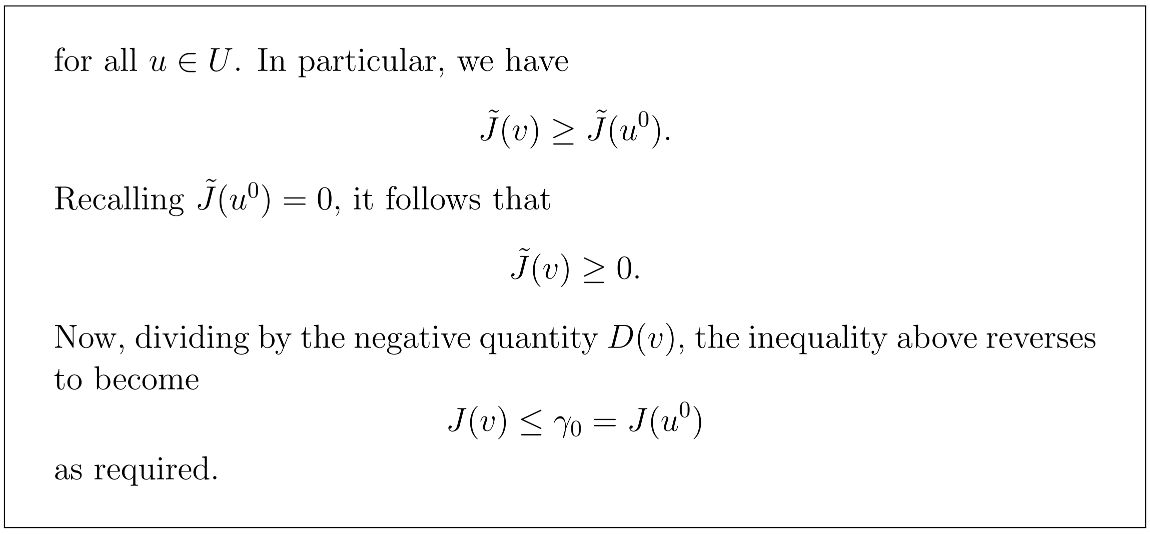
Dividing by $D(v) > 0$, we obtain

$$J(v) \leq \gamma_0 = J(u^0)$$

as required.

Case 2: When $D(u) < 0$ on U. In view of multilinearity of $\tilde{J}$ on the hypercube U, it follows that there is an optimal minimizing element *maximizing* $\tilde{J}$ which corresponds to some vertex $v \in U$. That is,

$$\tilde{J}(v) \geq \tilde{J}(u)$$



for all $u \in U$. In particular, we have

$$\tilde{J}(v) \geq \tilde{J}(u^0).$$

Recalling $\tilde{J}(u^0) = 0$, it follows that

$$\tilde{J}(v) \geq 0.$$

Now, dividing by the negative quantity $D(v)$, the inequality above reverses to become

$$J(v) \leq \gamma_0 = J(u^0)$$

as required.

Barmish

ECE 719 – Solution Ladder

For the ladder network with $n = 9$, the solution of the circuit equation is

$$V_{out}(R) = \frac{N(R)}{D(R)}$$

where

$$N(R) = R_3R_6R_9$$

and

$$\begin{aligned} D(R) \quad = \quad & R_2R_4R_6 + R_6R_3R_8 + R_3R_9R_1 + R_6R_8R_1 + R_3R_7R_2 + R_7R_4R_2 + R_9R_6R_1 + R_4R_8R_2 \\ & +R_4R_9R_1 + R_3R_8R_2 + R_3R_6R_4 + R_6R_7R_2 + R_8R_4R_1 + R_3R_6R_9 + R_7R_4R_1 + R_9R_4R_2 \\ & +R_6R_4R_1 + R_6R_7R_1 + R_6R_8R_2 + R_9R_6R_2 + R_3R_7R_1 + R_3R_8R_1 + R_3R_9R_2 + R_3R_6R_1 \\ & +R_6R_3R_2 + R_3R_7R_4 + R_3R_8R_4 + R_3R_9R_4 + R_3R_7R_6 + R_5R_7R_1 + R_5R_7R_2 + R_5R_8R_1 \\ & +R_5R_8R_2 + R_9R_5R_1 + R_9R_5R_2 + R_6R_5R_1 + R_6R_5R_2 + R_3R_7R_5 + R_3R_8R_5 + R_3R_9R_5 \\ & +R_3R_6R_5. \end{aligned}$$

Now, from Homework Quotient, we know the maximum is attained in the vertices of the hypercube. Hence R_i can only take values $\{90, 110\}$. Now by exhaustive search of the 2^9 vertices, we obtain $V^*_{out} \approx 0.0350$, with the choice of $R^* = (90, 90, 110, 90, 90, 110, 90, 90, 110)$.

Barmish

ECE 719 – Solution Common Sense

(a) Given any $u^0 = (u^0_1, u^0_2, ..., u^0_n) \in U$, it suffices to show that

$$J(u^*) \leq J(u^0).$$

Indeed, with

$$\hat{u}^{*,1} \doteq (u^*_1, u^0_2, ..., u^0_n),$$

it follows, from the definition of u^*_1, that

$$J(\hat{u}^{*,1}) \leq J(u^0).$$

Similarly, for

$$\hat{u}^{*,2} \doteq (u^*_1, u^*_2, u^0_3, ..., u^0_n),$$

we have

$$J(\hat{u}^{*,2}) \leq J(\hat{u}^{*,1}) \leq J(u^0).$$

Continuing the same reasoning, one arrives at

$$J(u^*) \leq J(\hat{u}^{*,n-1}) \leq ... \leq J(\hat{u}^{*,1}) \leq J(u^0),$$

i.e., u^* minimizes $J(u)$ over U.

(b) Let $U = [-1, 1] \times [-1, 1]$, and $J(u) = u^2_1 + u^2_2$. For u_1 fixed, $u^*_2 = 0$; For u_2 fixed, $u^*_1 = 0$. The minimum of $J(u)$ is also attained at $u^* = (0, 0)$.

(c) Let $U = [-1, 1] \times [-1, 1]$, and $J(u) = u_1u_2$. For u_1 fixed, u^*_2 is either $+1$ or -1, dependent on the value of u_1. For u_2 fixed, u^*_1 is either $+1$ or -1, dependent on the value of u_2. This violates the assumption on u^*_i in part (a).

4.2 HW 2

4.2.1 Problem 1

PROBLEM DESCRIPTION

Barmish

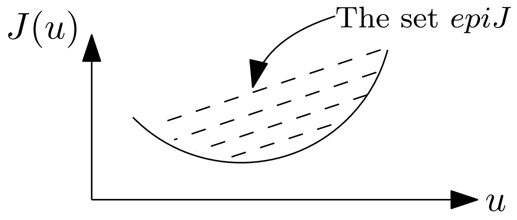
ECE 719 – Homework Epigraph

Give a function $J : \mathbf{R}^n \to \mathbf{R}$, we recall that its *epigraph* is the a set in $\mathbf{R}^{n+1}$ given by

$$epi\ J = \{(u, \alpha) \in \mathbf{R}^{n+1} : \alpha \geq J(u)\}.$$

Now prove that J is a convex function if and only if *epi* J is a convex set.

SOLUTION The following diagram illustrates epi J for $n = 1$. In words, it is the set of all points above the curve of the function $J(u)$

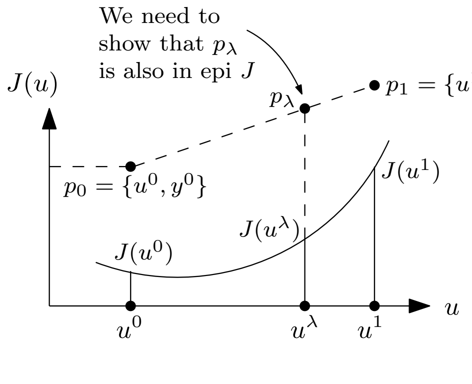


This is an iff proof, hence we need to show the following

1. Given J is convex function, then show that epi J is a convex set.

2. Given that epi J is a convex set, then show that J is a convex function.

Proof of first direction We pick any two arbitrary points in epi J, such as $p_0 = (u^0, y^0)$ and $p_1 = (u^1, y^1)$. To show epi J is a convex set, we need now to show that any point on the line between p_0, p_1 is also in epi J. The point between them is given by $p_\lambda = \left(u^\lambda, y^\lambda\right)$ where $\lambda \in [0, 1]$. The following diagram helps illustrates this for $n = 1$.



The point p_λ is given by

$$\begin{aligned}
\left(u^\lambda, y^\lambda\right) &= (1-\lambda) p_0 + \lambda p_1 \\
&= (1-\lambda)\left(u^0, y^0\right) + \lambda\left(u^1, y^1\right) \\
&= \left((1-\lambda) u^0 + \lambda u^1, (1-\lambda) y^0 + \lambda y^1\right)
\end{aligned}$$

Therefore $y^\lambda = (1-\lambda) y^0 + \lambda y^1$. Since p_0, p_1 are in epi J, then by the definition of epi J, we know that $y^0 \geq J\left(u^0\right)$ and $y^1 \geq J\left(u^1\right)$. Therefore we conclude that

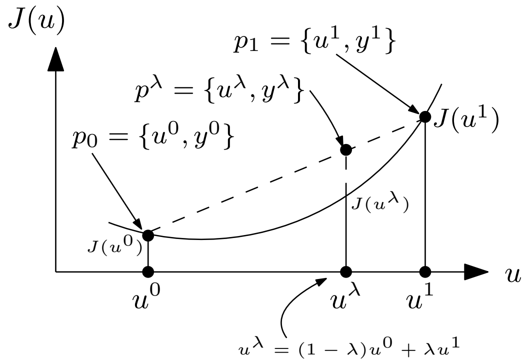
$$y^\lambda \geq (1-\lambda) J\left(u^0\right) + \lambda J\left(u^1\right) \tag{1}$$

But since we assumed J is a convex function, then we also know that $(1-\lambda) J\left(u^0\right) + \lambda J\left(u^1\right) \geq J\left(u^\lambda\right)$ where $u^\lambda = (1-\lambda) u^0 + \lambda u^1$. Therefore (1) becomes

$$y^\lambda \geq J\left(u^\lambda\right)$$

This implies the arbitrary point p_λ is in epi J.

We now need to proof the other direction. Given that epi J is a convex set, then show that J is a convex function. Since epi J is a convex set, we pick two arbitrary points in epi J, such as p_0, p_1. We can choose $p_0 = \left(u^0, J\left(u^0\right)\right)$ and $p_1 = \left(u^1, J\left(u^1\right)\right)$. These are still in epi J, but on the lower bound, on the edge with $J(u)$ curve.



Since p_0, p_1 are two points in a convex set, then any point p^λ on a line between them is also in epi J (by definition of a convex set). And since $p^\lambda = (1-\lambda)p_0 + \lambda p_1$ this implies

$$\begin{aligned}
p^\lambda &= \left(u^\lambda, y^\lambda\right)\\
&= \left((1-\lambda)p_0 + \lambda p_1\right)\\
&= \left((1-\lambda)\left(u^0, J\left(u^0\right)\right) + \lambda\left(u^1, J\left(u^1\right)\right)\right)\\
&= \left((1-\lambda)u^0 + \lambda u^1, \overbrace{(1-\lambda)J\left(u^0\right) + J\left(u^1\right)}^{y^\lambda}\right) \qquad (1)
\end{aligned}$$

Since p^λ is in epi J then by definition of epi J

$$y^\lambda \geq J\left(u^\lambda\right) \qquad (2)$$

But from (1) we see that $y^\lambda = (1-\lambda)J\left(u^0\right) + J\left(u^1\right)$, therefore (2) is the same as writing

$$(1-\lambda)J\left(u^0\right) + J\left(u^1\right) \geq J\left(u^\lambda\right) \qquad (3)$$

But $u^\lambda = (1-\lambda)u^0 + \lambda u^1$, hence the above becomes

$$(1-\lambda)J\left(u^0\right) + J\left(u^1\right) \geq J\left((1-\lambda)u^0 + \lambda u^1\right)$$

But the above is the definition of a convex function. Therefore $J(u)$ is a convex function. QED.

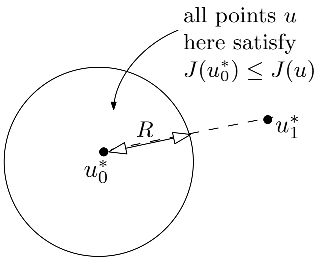
4.2.2 Problem 2

PROBLEM DESCRIPTION

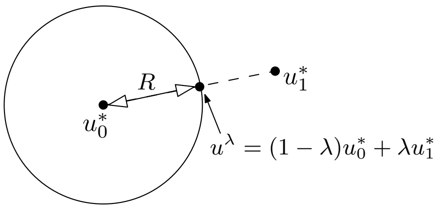
> Barmish
>
> **ECE 719 – Homework Unique Minimum**
>
> Suppose $J : \mathbf{R}^n \to \mathbf{R}$ is strictly convex. Then prove the following: If a minimizing element $u^* \in \mathbf{R}^n$ exists, it must be unique.

SOLUTION Let u_0^* and u_1^* be any two different minimizing elements in $\Re^n$ such that $J\left(u_0^*\right) < J\left(u_1^*\right)$. We will show that this leads to contradiction. Since u_0^* is a minimizer, then there exists some $R > 0$, such that all points u that satisfy $\|u^* - u\| \leq R$ also satisfy

$$J\left(u_0^*\right) \leq J(u)$$



We will consider all points along the line joining u_0^*, u_1^*, and pick one point u^λ that satisfies $\left\| u^* - u^\lambda \right\| \leq R$, where $\lambda \in [0,1]$ is selected to make the convex mixture $u^\lambda = (1-\lambda)\, u_0^* + \lambda u_1^*$ satisfied. Therefore any $\lambda \leq \frac{R}{\|u_0^* - u_1^*\|}$ will put u^λ inside the sphere of radius R.



Hence now we can say that

$$J\left(u_0^*\right) \leq J\left(u^\lambda\right) \tag{1}$$

But given that $J(u)$ is a strict convex function, then

$$J(u^\lambda) < (1-\lambda)\, J\left(u_0^*\right) + \lambda J\left(u_1^*\right) \tag{2}$$

Since we assumed that $J\left(u_0^*\right) < J\left(u_1^*\right)$, then if we replace $J\left(u_1^*\right)$ by $J\left(u_0^*\right)$ in the RHS of (2), it will change from $<$ to $\leq$ resulting in

$$J(u^\lambda) \leq (1-\lambda)\, J\left(u_0^*\right) + \lambda J\left(u_0^*\right)$$
$$J(u^\lambda) \leq J\left(u_0^*\right) \tag{3}$$

We see that equations (3) and (1) are a contradiction. Therefore our assumption is wrong and there can not be more than one minimizing element and u_0^* must be the same as u_1^*.

4.2.3 Problem 3

PROBLEM DESCRIPTION

Barmish

ECE 719 – Homework Global Minimum

Preamble: Suppose $J : \mathbf{R}^n \to \mathbf{R}$. A point $u^* \in \mathbf{R}^n$ is said to be a *local minimum of* J if there exists some suitably small $\delta > 0$ leading to satisfaction of the following condition:

$$J(u^*) \leq J(u)$$

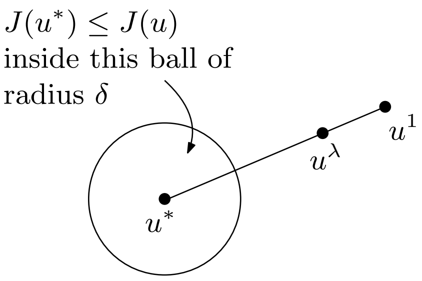
for all u such that $||u - u^*|| < \delta$. Said another way, u^* is a minimizing element over a suitably small open neighborhood. For the case when $J(u^*) \leq J(u)$ for all u, we call u^* a *global minimum* of J.

The Homework Problem: Suppose $J : \mathbf{R}^n \to \mathbf{R}$ is convex. Prove that every local minimum of J is a global minimum.

SOLUTION We are given that $J(u^*) \leq J(u)$ for all u such that $\|u^* - u\| < \delta$. Let us pick any arbitrary point u^1, outside ball of radius δ. Then any point on the line between u^* and u^1 is given by

$$u^\lambda = (1-\lambda)\, u^* + \lambda u^1$$

In picture, so far we have this setup

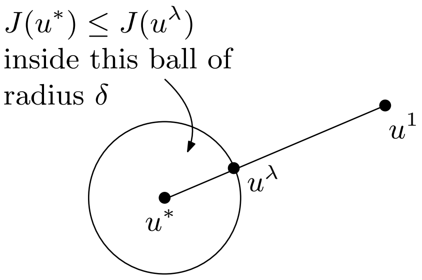


We now need to show that $J(u^*) \leq J\left(u^1\right)$ even though u^1 is outside the ball. Since J is a convex function, then

$$J\left(u^\lambda\right) \leq (1-\lambda) J(u^*) + \lambda J\left(u^1\right) \tag{1}$$

We can now select λ to push u^λ to be inside the ball. We are free to change λ as we want while keeping u^1 fixed, outside the ball. If we do this we then we have

$$J(u^*) \leq J\left(u^\lambda\right)$$



Hence (1) becomes

$$J(u^*) \leq (1-\lambda) J(u^*) + \lambda J\left(u^1\right) \tag{2}$$

Where we replaced $J\left(u^\lambda\right)$ by $J(u^*)$ in (1) and since $J(u^*) \leq J\left(u^\lambda\right)$ the $\leq$ relation remained valid. Simplifying (2) gives

$$\begin{aligned} J(u^*) &\leq J(u^*) - \lambda J(u^*) + \lambda J\left(u^1\right) \\ \lambda J(u^*) &\leq \lambda J\left(u^1\right) \end{aligned}$$

For non-zero λ this means $J(u^*) \leq J\left(u^1\right)$. This completes the proof, since u^1 was arbitrary point anywhere. Hence u^* is global minimum. QED

4.2.4 Problem 4

PROBLEM DESCRIPTION

Barmish

ECE 719 – Homework Multiple Combinations

For a convex function $J : \mathbf{R}^n \rightarrow \mathbf{R}$, prove the following condition is satisfied: Given any points $u^1, u^2, ..., u^N \in \mathbf{R}^n$ and any scalars $\lambda_1, \lambda_2, ..., \lambda_N \geq 0$ such that

$$\sum_{i=1}^{N} \lambda_i = 1,$$

it follows that

$$J\left(\sum_{i=1}^{N} \lambda_i u^i\right) \leq \sum_{i=1}^{N} \lambda_i J(u^i).$$

SOLUTION

We need to show that $J\left(\sum_{i=1}^{N} \lambda_i u^i\right) \leq \sum_{i=1}^{N} \lambda_i J\left(u^i\right)$ where $\sum_{i=1}^{N} \lambda_i = 1$. Proof by induction. For $N = 1$

and since $\lambda_1 = 1$, then we have

$$J\left(u^1\right) = J\left(u^1\right)$$

The case for $N = 2$ comes for free, from the definition of J being a convex function

$$J\left((1-\lambda)\,u^1 + \lambda u^2\right) \leq (1-\lambda)\,J\left(u^1\right) + \lambda J\left(u^2\right) \tag{A}$$

By making $(1-\lambda) \equiv \lambda_1, \lambda \equiv \lambda_2$, the above can be written as

$$J\left(\lambda_1 u^1 + \lambda_2 u^2\right) \leq \lambda_1 J\left(u^1\right) + \lambda_2 J\left(u^2\right)$$

We now assume it is true for $N = k-1$. In other words, the inductive hypothesis below is given as true

$$J\left(\sum_{i=1}^{k-1} \lambda_i u^i\right) \leq \sum_{i=1}^{k-1} \lambda_i J\left(u^i\right) \tag{*}$$

Now we have to show it will also be true for $N = k$, which is

$$\begin{aligned}
\sum_{i=1}^{k} \lambda_i J\left(u^i\right) &= \lambda_1 J\left(u^1\right) + \lambda_1 J\left(u^1\right) + \cdots + \lambda_k J\left(u^k\right) \\
&= (1-\lambda_k)\left(\frac{\lambda_1}{(1-\lambda_k)} J\left(u^1\right) + \frac{\lambda_1}{(1-\lambda_k)} J\left(u^1\right) + \cdots + \frac{\lambda_{k-1}}{(1-\lambda_k)} J\left(u^{k-1}\right) + \frac{\lambda_k}{(1-\lambda_k)} J\left(u^k\right)\right) \\
&= (1-\lambda_k)\left(\frac{\lambda_1}{(1-\lambda_k)} J\left(u^1\right) + \frac{\lambda_1}{(1-\lambda_k)} J\left(u^1\right) + \cdots + \frac{\lambda_{k-1}}{(1-\lambda_k)} J\left(u^{k-1}\right)\right) + \lambda_k J\left(u^k\right) \\
&= (1-\lambda_k)\left(\sum_{i=1}^{k-1} \frac{\lambda_i}{(1-\lambda_k)} J\left(u^i\right)\right) + \lambda_k J\left(u^k\right) \qquad (1)
\end{aligned}$$

Now we take advantage of the inductive hypothesis Eq. (*) on $k-1$, which says that $J\left(\sum_{i=1}^{k-1} \frac{\lambda_i}{(1-\lambda_k)} u^i\right) \leq \sum_{i=1}^{k-1} \frac{\lambda_i}{(1-\lambda_k)} J\left(u^i\right)$. Using this in (1) changes it to $\geq$ relation

$$\sum_{i=1}^{k} \lambda_i J\left(u^i\right) \geq (1-\lambda_k)\,J\left(\sum_{i=1}^{k-1} \frac{\lambda_i}{(1-\lambda_k)} u^i\right) + \lambda_k J\left(u^k\right) \tag{2}$$

We now take advantage of the case of $N = 2$ in (A) by viewing RHS of (2) as $(1-\lambda_k)\,J\left(u^1\right) + \lambda_k J\left(u^2\right)$, where we let $u^1 \equiv \sum_{i=1}^{k-1} \frac{\lambda_i}{(1-\lambda_k)} u^i, u^2 \equiv u^k$. Hence we conclude that

$$(1-\lambda_k)\,J\left(\sum_{i=1}^{k-1} \frac{\lambda_i}{(1-\lambda_k)} u^i\right) + \lambda_k J\left(u^k\right) \geq J\left((1-\lambda_k) \sum_{i=1}^{k-1} \frac{\lambda_i}{(1-\lambda_k)} u^i + \lambda_k u^k\right) \tag{3}$$

Using (3) in (2) gives (the $\geq$ relation remains valid, even more now, since we replaced something in RHS of (2), by something smaller)

$$\begin{aligned}
\sum_{i=1}^{k} \lambda_i J\left(u^i\right) &\geq J\left((1-\lambda_k) \sum_{i=1}^{k-1} \frac{\lambda_i}{(1-\lambda_k)} u^i + \lambda_k u^k\right) \\
&= J\left(\left(\sum_{i=1}^{k-1} \lambda_i u^i\right) + \lambda_k u^k\right)
\end{aligned}$$

Hence

$$\sum_{i=1}^{k} \lambda_i J\left(u^i\right) \geq J\left(\sum_{i=1}^{k} \lambda_i u^i\right)$$

QED.

4.2.5 Problem 5

PROBLEM DESCRIPTION

Barmish

ECE 719 – Homework Hessian

For $u \in \mathbf{R}^n$, define
$$J(u) = -(u_1 u_2 u_3 \cdots u_n)^{1/n}.$$
Prove that $J(u)$ is convex on the positive orthant; i.e., the set defined by $u_i > 0$ for $i = 1, 2, ..., n$.

SOLUTION

Assuming $J(u)$ is twice continuously differentiable (C^2) in $u_1, u_2, \cdots, u_n$, then if we can show that the Hessian $\nabla^2 J(u)$ is positive semi-definite on $u_i > 0$, then this implies $J(u)$ is convex. The first step is to determined $\nabla^2 J(u)$.

$$\frac{\partial J}{\partial u_i} = -\frac{1}{n}(u_1 u_2 \cdots u_n)^{\frac{1}{n}-1} \prod_{k=1,k\neq i}^{n} u_k = \frac{1}{n}\frac{J(u)}{(u_1 u_2 \cdots u_n)} \prod_{k=1,k\neq i}^{n} u_k = \frac{1}{n}\frac{J(u)}{\prod_{k=1}^{n} u_k} \prod_{k=1,k\neq i}^{n} u_k$$
$$= \frac{1}{n}\frac{J(u)}{u_i}$$

And

$$\begin{aligned}
\frac{\partial^2 J}{\partial u_i^2} &= \frac{1}{n}\frac{\left(\frac{1}{n}\frac{J(u)}{u_i}\right)}{u_i} - \frac{1}{n}\frac{J(u)}{u_i^2} \\
&= \frac{1}{n^2}\frac{J(u)}{u_i^2} - \frac{1}{n}\frac{J(u)}{u_i^2} \\
&= \frac{1}{n}\frac{J(u)}{u_i^2}\left(\frac{1}{n} - 1\right)
\end{aligned}$$

And the cross derivatives are

$$\begin{aligned}
\frac{\partial^2 J}{\partial u_i \partial u_j} &= \frac{\partial}{\partial u_j}\left(\frac{1}{n}\frac{J(u)}{u_i}\right) \\
&= \frac{1}{n}\frac{\frac{1}{n}\frac{J(u)}{u_j}}{u_i} \\
&= \frac{1}{n^2}\frac{J(u)}{u_i u_j}
\end{aligned}$$

Therefore

$$\boxed{\nabla^2 J(u) = \begin{pmatrix}
\frac{1}{n^2}\frac{J(u)}{u_1^2}(1-n) & \frac{1}{n^2}\frac{J(u)}{u_1 u_2} & \cdots & \frac{1}{n^2}\frac{J(u)}{u_1 u_n} \\
\frac{1}{n^2}\frac{J(u)}{u_2 u_1} & \frac{1}{n^2}\frac{J(u)}{u_2^2}(1-n) & \cdots & \frac{1}{n^2}\frac{J(u)}{u_2 u_n} \\
\vdots & \cdots & \ddots & \vdots \\
\frac{1}{n^2}\frac{J(u)}{u_n u_1} & \frac{1}{n^2}\frac{J(u)}{u_n u_2} & \cdots & \frac{1}{n^2}\frac{J(u)}{u_n^2}(1-n)
\end{pmatrix}}$$

Now we need to show that $\nabla^2 J(u)$ is positive semi-definite. For $n = 1$, the above reduces to

$$\nabla^2 J(u) = \frac{J(u)}{u_1^2}(1-1) = 0$$

Hence non-negative. This is the same as saying the second derivative is zero. For $n = 2$

$$\nabla^2 J(u) = \begin{pmatrix}
\frac{1}{4}J(u)\frac{1-2}{u_1^2} & \frac{1}{u_1 u_2}\frac{1}{4}J(u) \\
\frac{1}{u_2 u_1}\frac{1}{4}J(u) & \frac{1}{4}J(u)\frac{1-2}{u_2^2}
\end{pmatrix} = \begin{pmatrix}
\frac{-1}{u_1^2}\frac{1}{4}J(u) & \frac{1}{u_1 u_2}\frac{1}{4}J(u) \\
\frac{1}{u_2 u_1}\frac{1}{4}J(u) & \frac{-1}{u_2^2}\frac{1}{4}J(u)
\end{pmatrix}$$

The first leading minor is $\frac{-1}{4u_1^2}J(u)$, which is positive, since $J(u) < 0$ and $u_i > 0$ (given). The second leading minor is

$$\Delta_2 = \begin{vmatrix}
\frac{-1}{u_1^2}\frac{1}{4}J(u) & \frac{1}{u_1 u_2}\frac{1}{4}J(u) \\
\frac{1}{u_2 u_1}\frac{1}{4}J(u) & \frac{-1}{u_2^2}\frac{1}{4}J(u)
\end{vmatrix} = 0$$

Hence all the leasing minors are non-negative. Which means $\nabla^2 J(u)$ is semi-definite. We will look at $n = 3$

$$\nabla^2 J(u) = \begin{pmatrix} \frac{-2}{u_1^2}\frac{1}{9}J(u) & \frac{1}{u_1u_2}\frac{1}{9}J(u) & \frac{1}{u_1u_3}\frac{1}{9}J(u) \\ \frac{1}{u_2u_1}\frac{1}{9}J(u) & \frac{-2}{u_2^2}\frac{1}{9}J(u) & \frac{1}{u_2u_3}\frac{1}{9}J(u) \\ \frac{1}{u_3u_1}\frac{1}{9}J(u) & \frac{1}{u_3u_2}\frac{1}{9}J(u) & \frac{-2}{u_3^2}\frac{1}{9}J(u) \end{pmatrix}$$

The first leading minor is $\frac{-2}{9u_1^2}J(u)$, which is positive again, since $J(u) < 0$ for $u_i > 0$ (given). And the second leading minor is $\frac{1}{27}J^2\frac{u^2}{u_1^2u_2^2}$

which is positive, since all terms are positive. The third leading minor is

$$\Delta_3 = \begin{vmatrix} \frac{-2}{u_1^2}\frac{1}{9}J(u) & \frac{1}{u_1u_2}\frac{1}{9}J(u) & \frac{1}{u_1u_3}\frac{1}{9}J(u) \\ \frac{1}{u_2u_1}\frac{1}{9}J(u) & \frac{-2}{u_2^2}\frac{1}{9}J(u) & \frac{1}{u_2u_3}\frac{1}{9}J(u) \\ \frac{1}{u_3u_1}\frac{1}{9}J(u) & \frac{1}{u_3u_2}\frac{1}{9}J(u) & \frac{-2}{u_3^2}\frac{1}{9}J(u) \end{vmatrix} = 0$$

Hence non-of the leading minors are negative. Therefore $\nabla^2 J(u)$ is semi-definite. The same pattern repeats for higher values of n. All leading minors are positive, except the last leading minor will be zero.

4.2.5.1 Appendix

Another way to show that $\nabla^2 J(u)$ is positive semi-definite is to show that $x^T\left(\nabla^2 J(u)\right)x \geq 0$ for any vector x. (since $\nabla^2 J(u)$ is symmetric).

$$x^T\left(\nabla^2 J(u)\right)x = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix} \begin{pmatrix} \frac{1}{n}\frac{J(u)}{u_1^2}\left(\frac{1}{n}-1\right) & \frac{1}{n^2}\frac{J(u)}{u_1u_2} & \cdots & \frac{1}{n^2}\frac{J(u)}{u_1u_n} \\ \frac{1}{n^2}\frac{J(u)}{u_2u_1} & \frac{1}{n}\frac{J(u)}{u_2^2}\left(\frac{1}{n}-1\right) & \cdots & \frac{1}{n^2}\frac{J(u)}{u_2u_n} \\ \vdots & \cdots & \ddots & \vdots \\ \frac{1}{n^2}\frac{J(u)}{u_nu_1} & \frac{1}{n^2}\frac{J(u)}{u_nu_2} & \cdots & \frac{1}{n}\frac{J(u)}{u_n^2}\left(\frac{1}{n}-1\right) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Now the idea is to set $n = 1, 2, 3, \cdots$ and show that the resulting values ≥ 0 always. For $n = 1$, we obtain 0 as before. For $n = 2$, let

$\Delta = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} \frac{1}{n}\frac{J(u)}{u_1^2}\left(\frac{1}{n}-1\right) & \frac{1}{n^2}\frac{J(u)}{u_1u_2} \\ \frac{1}{n^2}\frac{J(u)}{u_2u_1} & \frac{1}{n}\frac{J(u)}{u_2^2}\left(\frac{1}{n}-1\right) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$. Expanding gives

$$\begin{aligned} \Delta &= \begin{pmatrix} x_1\frac{1}{n}\frac{J(u)}{u_1^2}\left(\frac{1}{n}-1\right) + x_2\frac{1}{n^2}\frac{J(u)}{u_2u_1} & x_1\frac{1}{n^2}\frac{J(u)}{u_1u_2} + x_2\frac{1}{n}\frac{J(u)}{u_2^2}\left(\frac{1}{n}-1\right) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= x_1^2\frac{1}{n}\frac{J(u)}{u_1^2}\left(\frac{1}{n}-1\right) + x_1x_2\frac{1}{n^2}\frac{J(u)}{u_2u_1} + x_2x_1\frac{1}{n^2}\frac{J(u)}{u_1u_2} + x_2^2\frac{1}{n}\frac{J(u)}{u_2^2}\left(\frac{1}{n}-1\right) \\ &= x_1^2\frac{1}{2}\frac{J(u)}{u_1^2}\left(\frac{1}{2}-1\right) + x_1x_2\frac{1}{4}\frac{J(u)}{u_2u_1} + x_2x_1\frac{1}{4}\frac{J(u)}{u_1u_2} + x_2^2\frac{1}{2}\frac{J(u)}{u_2^2}\left(\frac{1}{2}-1\right) \end{aligned}$$

The RHS above becomes, and by replacing $J(u) = -\sqrt{u_1u_2}$ for $n = 2$

$$\begin{aligned} -\frac{1}{4}x_1^2\frac{J(u)}{u_1^2} + x_1x_2\frac{1}{2}\frac{J(u)}{u_2u_1} - \frac{1}{4}x_2^2\frac{J(u)}{u_2^2} &= \frac{1}{4}x_1^2\frac{\sqrt{u_1u_2}}{u_1^2} - x_1x_2\frac{1}{2}\frac{\sqrt{u_1u_2}}{u_2u_1} + \frac{1}{4}x_2^2\frac{\sqrt{u_1u_2}}{u_2^2} \\ &= \left(\frac{1}{\sqrt{4}}\frac{(u_1u_2)^{\frac{1}{4}}}{u_1}x_1 - \frac{1}{\sqrt{4}}\frac{(u_1u_2)^{\frac{1}{4}}}{u_2}x_2\right)^2 \end{aligned}$$

Where we completed the square in the last step above. Hence $x^T\left(\nabla^2 J(u)\right)x \geq 0$. The same process can be continued for n higher. Hence $\nabla^2 J(u)$ is positive semi-definite.

4.2.6 HW 2 key solution

Barmish

ECE 719 – Solution Epigraph

We want to show that J is a convex function if and only if epi J is a convex set. To establish necessity, we assume convexity of J and must show epi J is a convex set. Indeed, let (u^0, α_0) and (u^1, α_1) be two points in epi J and, for $\lambda \in [0,1]$, consider the convex combination

$$(1-\lambda)(u^0, \alpha_0) + \lambda(u^1, \alpha_1) = ((1-\lambda)u^0 + \lambda u^1, (1-\lambda)\alpha_0 + \lambda\alpha_1)$$

To complete the proof of necessity, we must show this combination is in epi J. Indeed, using the convexity of J we have

$$J((1-\lambda)u^0 + \lambda u^1) \leq (1-\lambda)J(u^0) + \lambda J(u^1).$$

Now using the fact that

$$\alpha_0 \geq J(u^0); \quad \alpha_1 \geq J(u^1),$$

we have

$$J((1-\lambda)u^0 + \lambda u^1) \leq (1-\lambda)\alpha_0 + \lambda\alpha_1.$$

Hence,

$$((1-\lambda)u^0 + \lambda u^1, (1-\lambda)\alpha_0 + \lambda\alpha_1) \in \text{epi } J.$$

To establish sufficiency, we assume epi J is a convex set and must show J is a convex function. Indeed, for any two points u^0, u^1 and $\lambda \in [0,1]$, with $\alpha_0 = J(u^0)$ and $\alpha_1 = J(u^1)$, we have two points (u^0, α_0) and (u^1, α_1) in the epigraph. By convexity of the epigraph we have

$$((1-\lambda)u^0 + \lambda u^1, (1-\lambda)\alpha_0 + \lambda\alpha_1) \in \text{epi } J.$$

Therefore

$$(1-\lambda)\alpha_0 + \lambda\alpha_1 \geq (1-\lambda)J(u^0) + \lambda J(u^1).$$

Substituting for α_0, α_1 above then yields the desired result.

Barmish

ECE 719 – Solution Unique Minimum

With J being strictly convex, either no minimizer exists (for example, consider the function $J(u) = e^{-u}$ with $u \in \mathbf{R}$) or the minimum is unique. To consider the case of uniqueness, suppose at least one minimizer u^* exists. We claim there can be no other minimizer. Proceeding by contradiction, suppose $\tilde{u}$ is a second minimizer. So we have $J(u^*) = J(\tilde{u}) = J^*$. Now define

$$\hat{u} \doteq \frac{1}{2}u^* + \frac{1}{2}\tilde{u}.$$

Now, by strict convexity of J, with $\lambda = 1/2$, we have

$$J(\hat{u}) = J(\frac{1}{2}u^* + \frac{1}{2}\tilde{u}) < \frac{1}{2}J(u^*) + \frac{1}{2}J(\tilde{u}) = J^*.$$

This contradicts the optimality of u^*. That is, we have found another element with a lower J value.

Barmish

ECE 719 – Solution Global Minimum

Assume J is convex and u^* is a local minimum. We must show that u^* is also a global minimum. Proceeding by contradiction, suppose that $J(\hat{u}) < J(u^*)$. That is, say u^* is a local minimum but not a global minimum. Now for positive integers k, define the sequence of points

$$u^k = \frac{1}{k}\hat{u} + (1 - \frac{1}{k})u^*$$

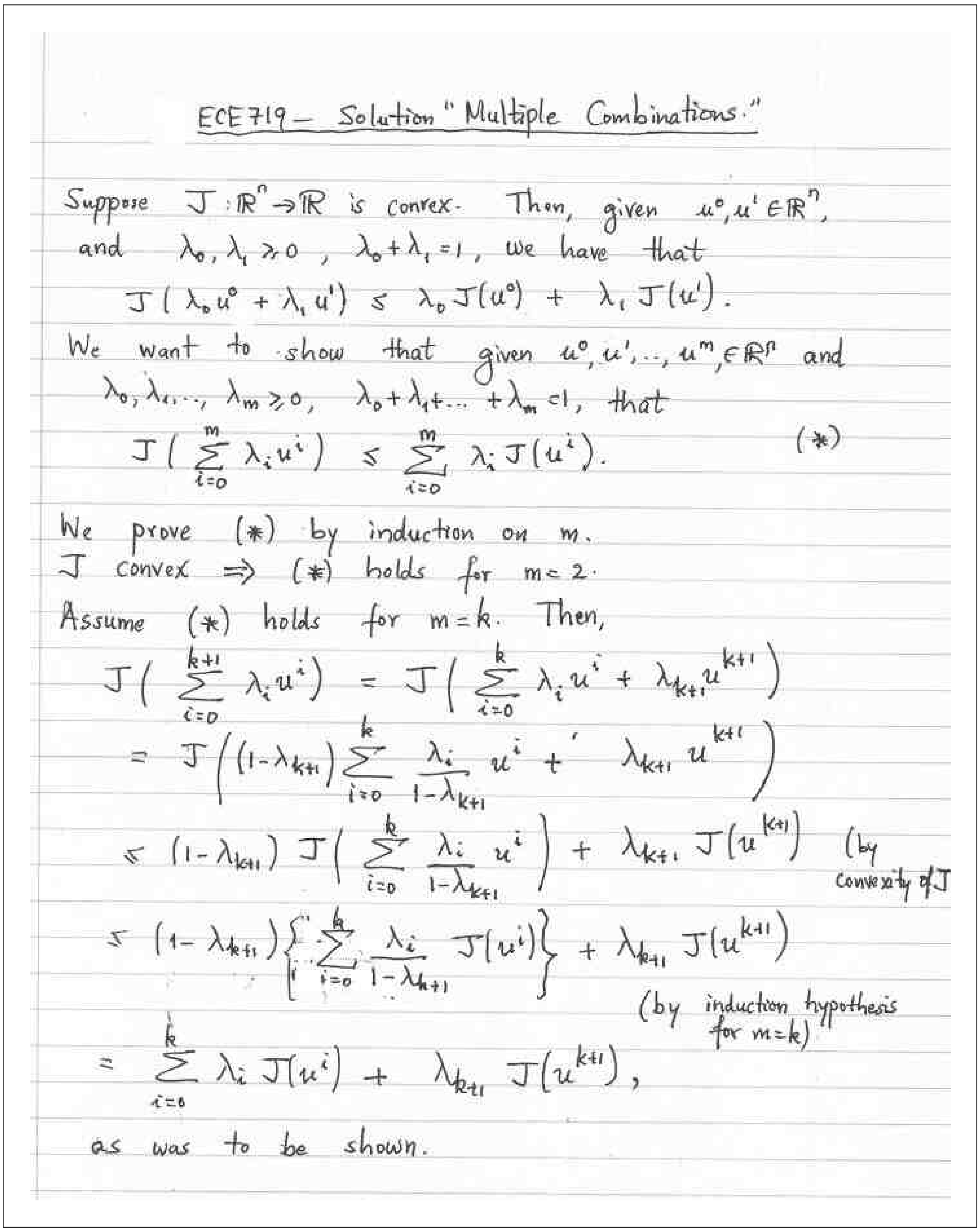
and note that convexity assures that

$$J(u^k) \leq \frac{1}{k}J(\hat{u}) + (1 - \frac{1}{k})J(u^*),$$

and using the fact that $J(\hat{u}) < J(u^*)$, the inequality above becomes

$$J(u^k) < J(u^*)$$

for all k. Also notice that this sequence converges to u^*. Hence, no matter how small a neighborhood we take around u^*, there is a point in it with a smaller J value. This contradicts the assumption that u^* is a local minimum.



ECE719 — Solution "Multiple Combinations."

Suppose $J:\mathbb{R}^n \to \mathbb{R}$ is convex. Then, given $u^0, u^1 \in \mathbb{R}^n$, and $\lambda_0, \lambda_1 \geq 0$, $\lambda_0 + \lambda_1 = 1$, we have that

$$J(\lambda_0 u^0 + \lambda_1 u^1) \leq \lambda_0 J(u^0) + \lambda_1 J(u^1).$$

We want to show that given $u^0, u^1, \ldots, u^m, \in \mathbb{R}^n$ and $\lambda_0, \lambda_1, \ldots, \lambda_m \geq 0$, $\lambda_0 + \lambda_1 + \ldots + \lambda_m = 1$, that

$$J\left(\sum_{i=0}^{m} \lambda_i u^i\right) \leq \sum_{i=0}^{m} \lambda_i J(u^i). \qquad (*)$$

We prove (*) by induction on m.
J convex ⇒ (*) holds for m = 2.
Assume (*) holds for m = k. Then,

$$J\left(\sum_{i=0}^{k+1} \lambda_i u^i\right) = J\left(\sum_{i=0}^{k} \lambda_i u^i + \lambda_{k+1} u^{k+1}\right)$$

$$= J\left((1-\lambda_{k+1}) \sum_{i=0}^{k} \frac{\lambda_i}{1-\lambda_{k+1}} u^i + \lambda_{k+1} u^{k+1}\right)$$

$$\leq (1-\lambda_{k+1}) J\left(\sum_{i=0}^{k} \frac{\lambda_i}{1-\lambda_{k+1}} u^i\right) + \lambda_{k+1} J(u^{k+1}) \quad \text{(by convexity of } J\text{)}$$

$$\leq (1-\lambda_{k+1}) \left\{\sum_{i=0}^{k} \frac{\lambda_i}{1-\lambda_{k+1}} J(u^i)\right\} + \lambda_{k+1} J(u^{k+1})$$

(by induction hypothesis for m = k)

$$= \sum_{i=0}^{k} \lambda_i J(u^i) + \lambda_{k+1} J(u^{k+1}),$$

as was to be shown.

Barmish

ECE 719 – Solution Hessian

To show that

$$J(u) = -(u_1 u_2 \cdots u_n)^{\frac{1}{n}}$$

is convex on the positive orthant, we compute its Hessian. Indeed, taking two derivatives, we obtain

$$H_{ii} = \frac{\partial^2 J}{\partial u_i^2} = -\frac{(n-1)J(u)}{n^2 u_i^2}$$

for diagonal terms of the Hessian and

$$H_{ij} = \frac{\partial^2 J}{\partial u_i \partial u_j} = \frac{J(u)}{n^2 u_i u_j}$$

for off-diagonal terms. Now, for arbitrary $x \in \mathbf{R}^n$, we consider the quadratic form

$$\begin{aligned} x^T \nabla^2 J(u) x &= \sum_{i,j} H_{ij} x_i x_j \\ &= \sum_{i \neq j} \frac{x_i x_j J(u)}{n^2 u_i u_j} - \sum_i \frac{(n-1)J(u)x_i^2}{n^2 u_i^2} \\ &= \sum_{i,j} \frac{x_i x_j J(u)}{n^2 u_i u_j} - \sum_i \frac{J(u)x_i^2}{n u_i^2}. \end{aligned}$$

Now defining the real numbers

$$\alpha_i = \frac{x_i}{u_i}$$

for $i = 1, 2, \ldots, n$ and noting that $J(u) < 0$ over the positive orthant, to establish positive semi-definiteness of the Hessian, it suffices to show that

$$n \sum_i \alpha_i^2 \geq \sum_{i,j} \alpha_i \alpha_j$$

Now recognizing that the right hand side is

$$\sum_{i,j} \alpha_i \alpha_j = (\sum_i \alpha_i)^2,$$

positive semi-definiteness is equivalent to

$$n \sum_i \alpha_i^2 \geq (\sum_i \alpha_i)^2 \qquad (*)$$

To establish $(*)$, let

$$\overline{\alpha} = \frac{1}{n} \sum_i \alpha_i$$

and observe that

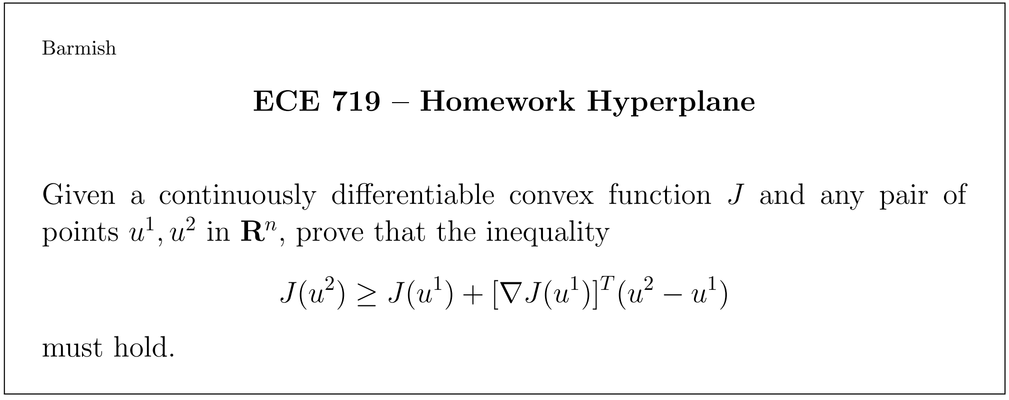
$$\begin{aligned} 0 &\leq \sum_i (\alpha_i - \overline{\alpha})^2 \\ &= \sum_i \alpha_i^2 - 2\overline{\alpha} \sum_i \alpha_i + n\overline{\alpha}^2 \\ &= \sum_i \alpha_i^2 - n\overline{\alpha}^2 \\ &= \sum_i \alpha_i^2 - \frac{1}{n}(\sum_i \alpha_i)^2. \end{aligned}$$

Now, inequality $(*)$ immediately follows.

4.3 HW 3

4.3.1 Problem 1

PROBLEM DESCRIPTION



Barmish

ECE 719 – Homework Hyperplane

Given a continuously differentiable convex function J and any pair of points u^1, u^2 in $\mathbf{R}^n$, prove that the inequality

$$J(u^2) \geq J(u^1) + [\nabla J(u^1)]^T(u^2 - u^1)$$

must hold.

SOLUTION

Since $J(\boldsymbol{u})$ is a convex function $J : \Re^n \to \Re$, then by definition of convex functions we write

$$J\left((1-\lambda)\boldsymbol{u}^1 + \lambda\boldsymbol{u}^2\right) \leq (1-\lambda) J\left(\boldsymbol{u}^1\right) + \lambda J\left(\boldsymbol{u}^2\right)$$

Where $\lambda \in (0,1)$. Rewriting the above as follows

$$J\left(\boldsymbol{u}^1 - \lambda\boldsymbol{u}^1 + \lambda\boldsymbol{u}^2\right) \leq J\left(\boldsymbol{u}^1\right) - \lambda J\left(\boldsymbol{u}^1\right) + \lambda J\left(\boldsymbol{u}^2\right)$$
$$J\left(\boldsymbol{u}^1 + \lambda\left(\boldsymbol{u}^2 - \boldsymbol{u}^1\right)\right) - J\left(\boldsymbol{u}^1\right) \leq \lambda\left(J\left(\boldsymbol{u}^2\right) - J\left(\boldsymbol{u}^1\right)\right)$$

Dividing both sides by $\lambda \neq 0$ gives

$$\frac{J\left(\boldsymbol{u}^1 + \lambda\left(\boldsymbol{u}^2 - \boldsymbol{u}^1\right)\right) - J\left(\boldsymbol{u}^1\right)}{\lambda} \leq J\left(\boldsymbol{u}^2\right) - J\left(\boldsymbol{u}^1\right)$$

Taking the limit $\lambda \to 0$ results in

$$\lim_{\lambda\to 0}\frac{J\left(\boldsymbol{u}^1 + \lambda\left(\boldsymbol{u}^2 - \boldsymbol{u}^1\right)\right) - J\left(\boldsymbol{u}^1\right)}{\lambda} \leq \lim_{\lambda\to 0} J\left(\boldsymbol{u}^2\right) - J\left(\boldsymbol{u}^1\right)$$

But $\lim_{\lambda\to 0}\frac{J\left(\boldsymbol{u}^1 + \lambda\left(\boldsymbol{u}^2 - \boldsymbol{u}^1\right)\right) - J\left(\boldsymbol{u}^1\right)}{\lambda} = \left.\frac{\partial J(\boldsymbol{u})}{\partial\left(\boldsymbol{u}^2 - \boldsymbol{u}^1\right)}\right|_{\boldsymbol{u}^1} = \left[\nabla J\left(\boldsymbol{u}^1\right)\right]^T\left(\boldsymbol{u}^2 - \boldsymbol{u}^1\right)$ (appendix below shows how this came about). Therefore the above becomes

$$\left[\nabla J\left(\boldsymbol{u}^1\right)\right]^T\left(\boldsymbol{u}^2 - \boldsymbol{u}^1\right) \leq J\left(\boldsymbol{u}^2\right) - J\left(\boldsymbol{u}^1\right)$$
$$J\left(\boldsymbol{u}^2\right) \geq J\left(\boldsymbol{u}^1\right) + \left[\nabla J\left(\boldsymbol{u}^1\right)\right]^T\left(\boldsymbol{u}^2 - \boldsymbol{u}^1\right)$$

QED.

4.3.1.1 Appendix

More details are given here on why

$$\lim_{\lambda\to 0}\frac{J\left(\boldsymbol{u}^1 + \lambda\left(\boldsymbol{u}^2 - \boldsymbol{u}^1\right)\right) - J\left(\boldsymbol{u}^1\right)}{\lambda} = \left[\nabla J\left(\boldsymbol{u}^1\right)\right]^T\left(\boldsymbol{u}^2 - \boldsymbol{u}^1\right)$$

Let $\boldsymbol{u}^2 - \boldsymbol{u}^1 = \boldsymbol{d}$. This is a directional vector, its tail starts at $\boldsymbol{u}^1$ going to tip of $\boldsymbol{u}^2$ point. Evaluating $\lim_{\lambda\to 0}\frac{J\left(\boldsymbol{u}^1 + \lambda\boldsymbol{d}\right) - J\left(\boldsymbol{u}^1\right)}{\lambda}$ is the same as saying

$$\left.\frac{\partial J(\boldsymbol{u})}{\partial \boldsymbol{d}}\right|_{\boldsymbol{u}^1} = \lim_{\lambda\to 0}\frac{J\left(\boldsymbol{u}^1 + \lambda\boldsymbol{d}\right) - J\left(\boldsymbol{u}^1\right)}{\lambda}$$
$$= \left.\frac{d}{d\lambda}J\left(\boldsymbol{u}^1 + \lambda\boldsymbol{d}\right)\right|_{\lambda=0}$$

Using the chain rule gives

$$\begin{aligned}\frac{d}{d\lambda}J\left(\boldsymbol{u}^1+\lambda\boldsymbol{d}\right)\Big|_{\lambda=0} &= \left[\nabla J\left(\boldsymbol{u}^1+\lambda\boldsymbol{d}\right)\right]^T\frac{d}{d\lambda}\left(\boldsymbol{u}^1+\lambda\boldsymbol{d}\right)\Big|_{\lambda=0}\\ &= \left[\nabla J\left(\boldsymbol{u}^1+\lambda\boldsymbol{d}\right)\right]^T\boldsymbol{d}\Big|_{\lambda=0}\\ &= \left[\nabla J\left(\boldsymbol{u}^1\right)\right]^T\boldsymbol{d}\end{aligned}$$

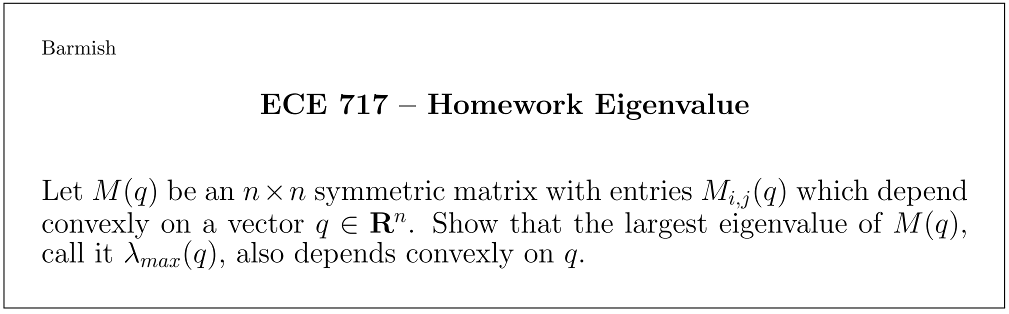
Replacing $\boldsymbol{u}^2-\boldsymbol{u}^1=\boldsymbol{d}$, the above becomes

$$\begin{aligned}\lim_{\lambda\to0}\frac{J\left(\boldsymbol{u}^1+\lambda\left(\boldsymbol{u}^2-\boldsymbol{u}^1\right)\right)-J\left(\boldsymbol{u}^1\right)}{\lambda} &= \frac{\partial J(\boldsymbol{u})}{\partial\left(\boldsymbol{u}^2-\boldsymbol{u}^1\right)}\Bigg|_{\boldsymbol{u}^1}\\ &= \left[\nabla J\left(\boldsymbol{u}^1\right)\right]^T\left(\boldsymbol{u}^2-\boldsymbol{u}^1\right)\end{aligned}$$

Where $\nabla J\left(\boldsymbol{u}^1\right)$ is the gradient vector of $J(\boldsymbol{u})$ evaluated at $\boldsymbol{u}=\boldsymbol{u}^1$.

4.3.2 Problem 2

PROBLEM DESCRIPTION



Barmish

ECE 717 – Homework Eigenvalue

Let $M(q)$ be an $n\times n$ symmetric matrix with entries $M_{i,j}(q)$ which depend convexly on a vector $q\in\mathbf{R}^n$. Show that the largest eigenvalue of $M(q)$, call it $\lambda_{max}(q)$, also depends convexly on q.

SOLUTION

Since each $m_{ij}\left(q\right)$ is convex function in q, then

$$m_{ij}\left((1-\alpha)q^1+\alpha q^2\right)\leq(1-\alpha)m_{ij}\left(q^1\right)+\alpha m_{ij}\left(q^2\right) \tag{1}$$

For $\alpha\in[0,1]$. We also know by Rayleigh quotient theorem which applies for symmetric matrices that largest eigenvalue of a symmetric matrix is given by

$$\lambda_{\max}=\max_{x\in\Re^n,\|x\|=1}x^TMx$$

Therefore, evaluated at point $q^\alpha=(1-\alpha)q^1+\alpha q^2$, the above become

$$\lambda_{\max}\left((1-\alpha)q^1+\alpha q^2\right)=\max_{\|x\|=1}\sum_{i,j}^n m_{ij}\left((1-\alpha)q^1+\alpha q^2\right)x_ix_j \tag{2}$$

Applying (1) in RHS (2) changes = to ≤ giving

$$\begin{aligned}\lambda_{\max}\left((1-\alpha)q^1+\alpha q^2\right) &\leq \max_{\|x\|=1}\sum_{i,j}^n\left((1-\alpha)m_{ij}\left(q^1\right)+\alpha m_{ij}\left(q^2\right)\right)x_ix_j\\ &= \max_{\|x\|=1}\left(\sum_{i,j}^n(1-\alpha)m_{ij}\left(q^1\right)x_ix_j+\sum_{i,j}^n\alpha m_{ij}\left(q^2\right)x_ix_j\right)\\ &= (1-\alpha)\left(\max_{\|x\|=1}\sum_{i,j}^n m_{ij}\left(q^1\right)x_ix_j\right)+\alpha\left(\max_{\|x\|=1}\sum_{i,j}^n m_{ij}\left(q^2\right)x_ix_j\right)\end{aligned} \tag{3}$$

Since

$$\max_{\|x\|=1}\sum_{i,j}^n m_{ij}\left(q^1\right)x_ix_j=\lambda_{\max}\left(q^1\right)$$

And

$$\max_{\|x\|=1}\sum_{i,j}^n m_{ij}\left(q^2\right)x_ix_j=\lambda_{\max}\left(q^2\right)$$

Then (3) becomes

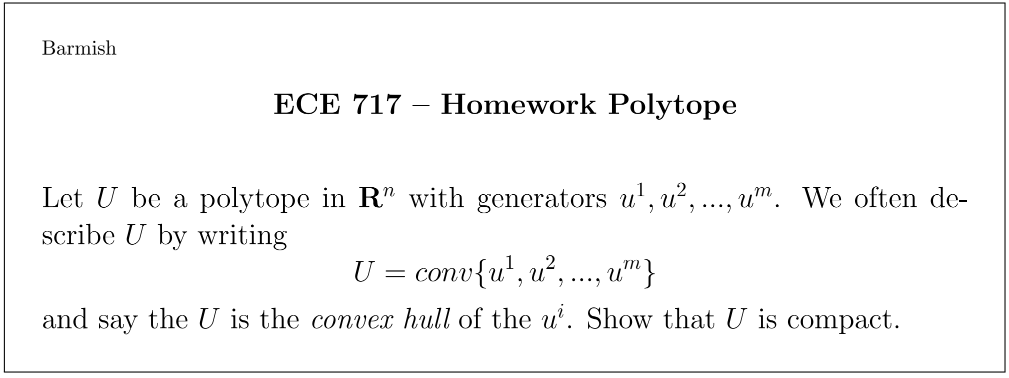
$$\lambda_{\max}\left((1-\alpha)q^1+\alpha q^2\right)\leq(1-\alpha)\lambda_{\max}\left(q^1\right)+\alpha\lambda_{\max}\left(q^2\right)$$

This is the definition of convex function, therefore $\lambda_{\max}$ is a convex function in q.

Note: I tried also to reduce this to a problem where I could argue that the pointwise maximum of convex functions is also a convex function to solve it. I could not get a clear way to do this, so I solved it as above. I hope I did not violate the cardinal rule by using $\lambda_{\max} = \max_{x\in\Re^n,\|x\|=1} x^T M x$.

4.3.3 Problem 3

PROBLEM DESCRIPTION



Barmish

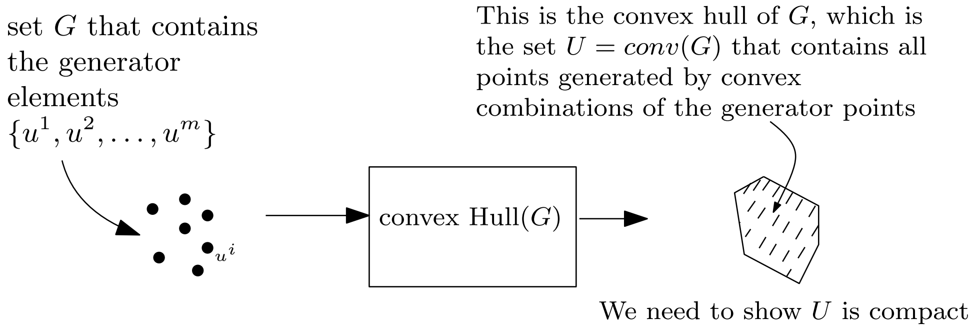
ECE 717 – Homework Polytope

Let U be a polytope in $\mathbf{R}^n$ with generators $u^1, u^2, ..., u^m$. We often describe U by writing

$$U = conv\{u^1, u^2, ..., u^m\}$$

and say the U is the *convex hull* of the u^i. Show that U is compact.

SOLUTION


set G that contains the generator elements $\{u^1, u^2, \ldots, u^m\}$
u^i
convex Hull(G)
This is the convex hull of G, which is the set $U = conv(G)$ that contains all points generated by convex combinations of the generator points
We need to show U is compact

To show U is bounded, a proof by induction is used. From the definition of constructing U

$$U = \left\{ x \in \Re^n : x = \sum_{i=1}^m \lambda_i u^i \right\}$$

Where $\sum_{i=1}^m \lambda_i = 1$ and $\lambda_i \geq 0$.

For $m = 1$, $x = \lambda u^1$. So U contains just one element u^1. Since $\lambda = 1$ and u^1 is given and bounded, then this is closed and bounded set with one element. Hence compact. Now we assume U is compact for $m = k-1$ and we need to show it is compact for $m = k$. In other words, we assume that each $x^* \in U$ generated using

$$x^* = \sum_{i=1}^{k-1} \lambda_i u^i$$

Is such that $\|x^*\| < \infty$ and $x^* \in U$. Now we need to show that U is bounded when generator contains k elements. Now

$$\begin{aligned} x &= \sum_{i=1}^{k} \lambda_i u^i \\ &= \lambda_1 u^1 + \lambda_2 u^2 + \cdots + \lambda_{k-1} u^{k-1} + \lambda_k u^k \end{aligned}$$

Multiply and divide by $(1-\lambda_k)$

$$\begin{aligned} x &= (1-\lambda_k)\left(\frac{\lambda_1 u^1}{(1-\lambda_k)} + \frac{\lambda_2}{(1-\lambda_k)} u^2 + \cdots + \frac{\lambda_{k-1} u^{k-1}}{(1-\lambda_k)} + \frac{\lambda_k}{(1-\lambda_k)} u^k \right) \\ &= (1-\lambda_k)\left(\sum_{i=1}^{k-1} \frac{\lambda_i}{(1-\lambda_k)} u^i + \frac{\lambda_k}{(1-\lambda_k)} u^k \right) \\ &= (1-\lambda_k)\left(\sum_{i=1}^{k-1} \frac{\lambda_i}{(1-\lambda_k)} u^i \right) + \lambda_k u^k \end{aligned}$$

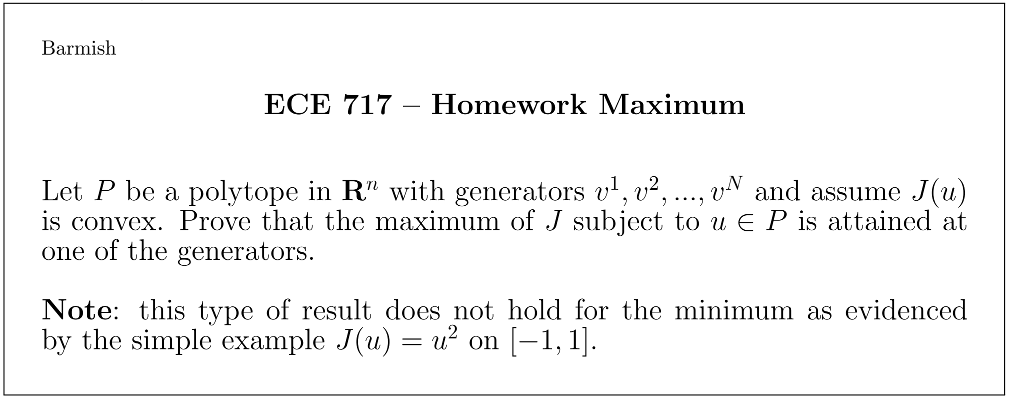
But $\sum_{i=1}^{k-1} \frac{\lambda_i}{(1-\lambda_k)} u^i = x^*$ which we assumed in U. Hence the above becomes

$$x = (1-\lambda_k)\, x^* + \lambda_k u^k$$

Since u^k is element in the generator set G and it is in U by definition, then the above is convex combination of two elements in U. Hence x in also in U (it is on a line between x^* and u^k, both in U). Therefore U is closed and bounded for any m in the generator set. Hence U is compact.

4.3.4 Problem 4

PROBLEM DESCRIPTION



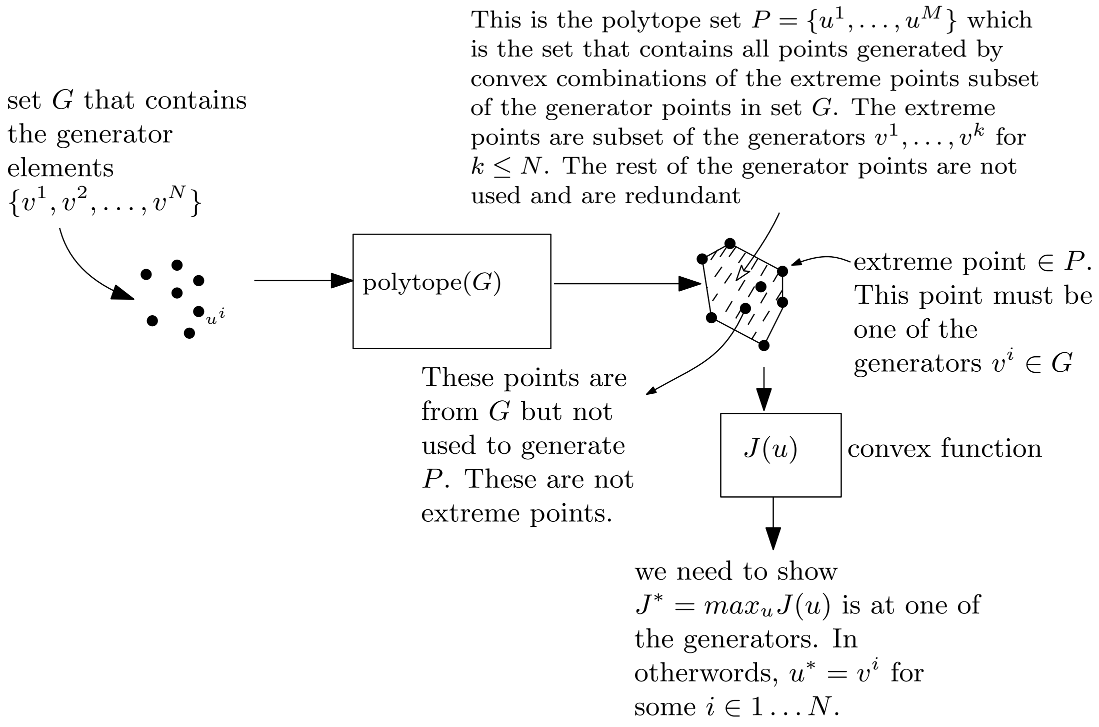
Barmish

ECE 717 – Homework Maximum

Let P be a polytope in $\mathbf{R}^n$ with generators $v^1, v^2, ..., v^N$ and assume $J(u)$ is convex. Prove that the maximum of J subject to $u \in P$ is attained at one of the generators.

Note: this type of result does not hold for the minimum as evidenced by the simple example $J(u) = u^2$ on $[-1, 1]$.

SOLUTION



This is the polytope set $P = \{u^1, \ldots, u^M\}$ which is the set that contains all points generated by convex combinations of the extreme points subset of the generator points in set G. The extreme points are subset of the generators $v^1, \ldots, v^k$ for $k \leq N$. The rest of the generator points are not used and are redundant

set G that contains the generator elements $\{v^1, v^2, \ldots, v^N\}$

extreme point $\in P$. This point must be one of the generators $v^i \in G$

These points are from G but not used to generate P. These are not extreme points.

we need to show $J^* = max_u J(u)$ is at one of the generators. In otherwords, $u^* = v^i$ for some $i \in 1 \ldots N$.

The extreme points of P are subset of G. They are the points used to generate P. The set P is compact (by problem 3) and convex set (by construction, since it is convex combinations of its extreme points). If we can show that J^* is at an extreme point of P, then we are done, since an extreme point of P is in G.

Let $u^* \in P$ be the point where $J(u)$ is maximum. u^* is a convex combinations of all extreme points of P, (these are also subset from G but they can be the whole set G also if there were no redundant generators), Therefore

$$u^* = \sum_{i=1}^{k} \lambda_i v^i$$

where $k \leq N$ and $v^i \in G$. If it happens that all points in G are extreme points of P, then $k = N$. Therefore

$$J^* = J(u^*) = J\left(\sum_{i=1}^{k} \lambda_i v^i\right)$$

Where $\sum_{i=1}^{k} \lambda_i = 1$ and $\lambda_i \geq 0$. But J is convex function (given). Hence by definition of convex function

$$J^* = J\left(\sum_{i=1}^{k}\lambda_i v^i\right) \leq \sum_{i=1}^{k}\lambda_i J\left(v^i\right) \tag{1}$$

The above is generalization of $J\left((1-\lambda)u^1 + \lambda u^2\right) \leq (1-\lambda)J\left(u^1\right) + \lambda J\left(u^2\right)$ applied to convex mixtures. Now we look at $J\left(v^i\right)$ term in the above. We pick the maximum of J over all v^i. There must be a point in G where $J(v)$ is largest. We call this value J_G^*. This is the value of J where it attains its maximum over generator elements $v^i : i = 1 \cdots k$. Eq (1) becomes

$$J^* \leq \sum_{i=1}^{k}\lambda_i J_G^*$$

Where we replaced $J\left(v^i\right)$ by one value, the maximum J_G^*. But J_G^* does not depend on i now, and can take it outside the sum

$$J^* \leq J_G^*\left(\sum_{i=1}^{k}\lambda_i\right)$$

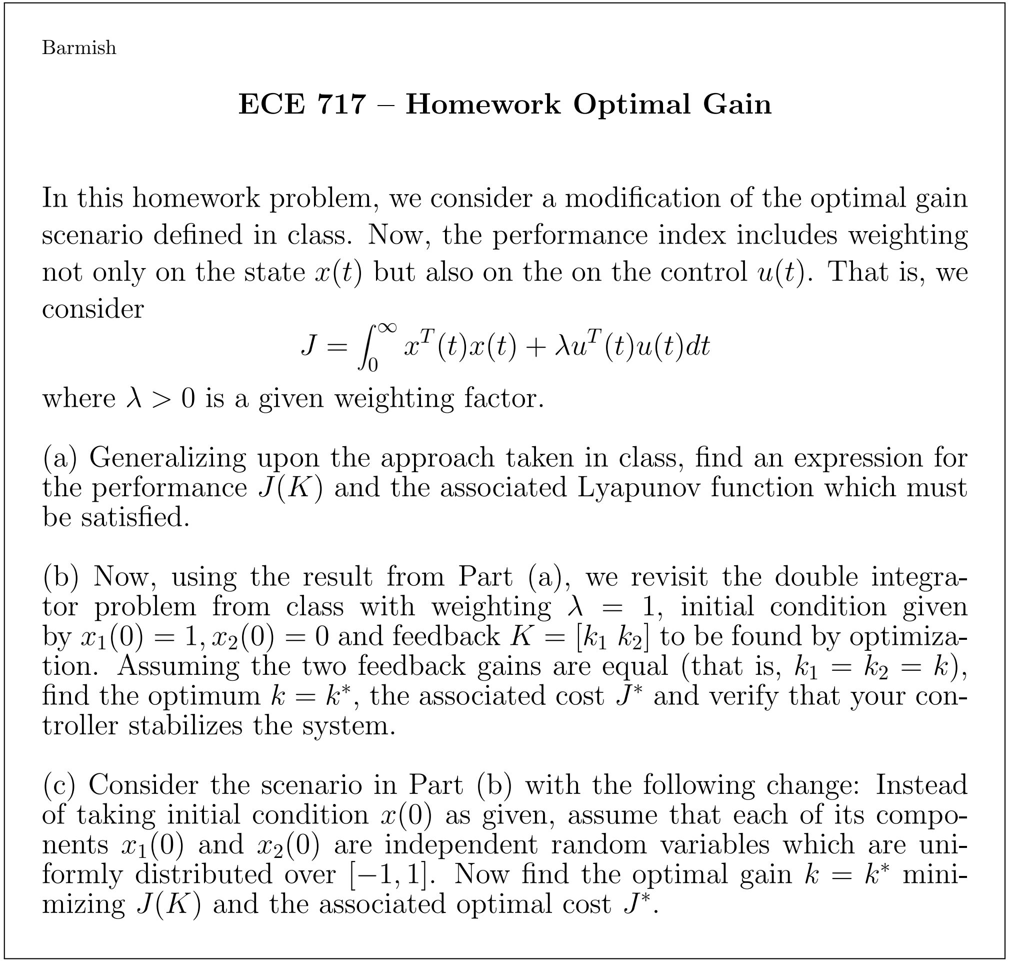
But $\sum_{i=1}^{k}\lambda_i = 1$ by definition. Therefore the above becomes

$$J^* \leq J_G^*$$

We now see that the maximum of $J(u)$ over P is smaller (or equal) than the maximum of $J(u)$ over the generator set G. Hence a maximum occurs at one of the extreme points v^i, since these are by definition taken from G. which is what we are asked to show.

4.3.5 Problem 5

PROBLEM DESCRIPTION



Barmish

ECE 717 – Homework Optimal Gain

In this homework problem, we consider a modification of the optimal gain scenario defined in class. Now, the performance index includes weighting not only on the state $x(t)$ but also on the on the control $u(t)$. That is, we consider

$$J = \int_0^\infty x^T(t)x(t) + \lambda u^T(t)u(t)dt$$

where $\lambda > 0$ is a given weighting factor.

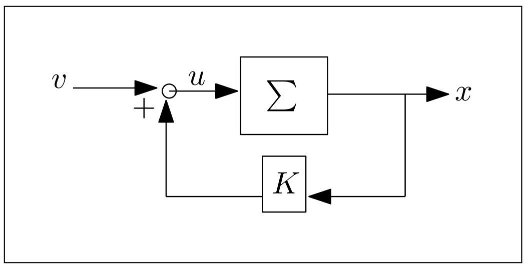
(a) Generalizing upon the approach taken in class, find an expression for the performance $J(K)$ and the associated Lyapunov function which must be satisfied.

(b) Now, using the result from Part (a), we revisit the double integrator problem from class with weighting $\lambda = 1$, initial condition given by $x_1(0) = 1, x_2(0) = 0$ and feedback $K = [k_1\ k_2]$ to be found by optimization. Assuming the two feedback gains are equal (that is, $k_1 = k_2 = k$), find the optimum $k = k^*$, the associated cost J^* and verify that your controller stabilizes the system.

(c) Consider the scenario in Part (b) with the following change: Instead of taking initial condition $x(0)$ as given, assume that each of its components $x_1(0)$ and $x_2(0)$ are independent random variables which are uniformly distributed over $[-1, 1]$. Now find the optimal gain $k = k^*$ minimizing $J(K)$ and the associated optimal cost J^*.

SOLUTION

4.3.5.1 Part (a)



Let us look at the closed loop. Let $v=0$ and we have, since $u(t)=kx(t)$

$$\begin{aligned}\dot{x}&=Ax+Bkx\\&=(A+Bk)x\\&=A_cx\end{aligned}$$

Where A_c is the closed loop system matrix. Since $J(k)=\int_0^\infty x^T(t)x(t)+\lambda u^T(t)u(t)dt$, where $u(t)=kx(t)$, then

$$\begin{aligned}J(k)&=\int_0^\infty x^Tx+\lambda(kx)^T(kx)dt\\&=\int_0^\infty x^Tx+\lambda x^T\left(k^Tk\right)xdt\end{aligned}$$

Let us find a matrix P, if possible such that

$$d\left(x^TPx\right)=-\left(x^Tx+\lambda x^T\left(k^Tk\right)x\right)$$

Can we find P? Since

$$d\left(x^TPx\right)=x^TP\dot{x}+\dot{x}^TPx$$

Then we need to solve

$$\begin{aligned}x^TP\dot{x}+\dot{x}^TPx&=-\left(x^Tx+\lambda x^T\left(k^Tk\right)x\right)\\x^TP(A_cx)+(A_cx)^TPx&=-\left(x^Tx+\lambda x^T\left(k^Tk\right)x\right)\\x^TP(A_cx)+\left(x^TA_c^T\right)Px&=-\left(x^Tx+\lambda x^T\left(k^Tk\right)x\right)\end{aligned}$$

Bring all the x to LHS then

$$\begin{aligned}x^Tx+\lambda x^T\left(k^Tk\right)x+x^TP(A_cx)+\left(x^TA_c^T\right)Px&=0\\\lambda\left(k^Tk\right)+PA_c+A_c^TP&=-I\end{aligned}$$

Hence the Lyapunov equation to solve for P is

$$\boxed{\lambda\left(k^Tk\right)+PA_c+A_c^TP=-I}$$

Where I is the identity matrix. This is the equation to determine matrix P. Without loss of generality, we insist on P being symmetric matrix. Using this P, now we write

$$\begin{aligned}J(k)&=\int_0^\infty x^Tx+\lambda(kx)^T(kx)dt\\&=-\int_0^\infty d\left(x^TPx\right)\\&=x^TPx\Big|_\infty^0\\&=x^T(0)Px(0)-x^T(\infty)Px(\infty)\end{aligned}$$

For stable system, $x(\infty)\rightarrow 0$ (since we set $v=0$, there is no external input, hence if the system is stable, it must end up in zero state eventually). In part (b) we check for k range so that the roots are in the left hand side. Therefore

$$J(k)=x^T(0)P(k)x(0)$$

With $P(k)$ satisfying solution of Lyapunov equation found above.

4.3.5.2 Part(b)

For $k = \begin{bmatrix} k_1 & k_2 \end{bmatrix}, x(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and system $y'' = u$. Hence $x_1' = x_2, x_2' = u$. Since

$$u = \begin{bmatrix} k_1 & k_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

The system $\dot{x} = Ax + Bu$ becomes

$$\begin{aligned}
x' &= Ax + Bu \\
&= Ax + Bkx \\
&= (A + Bk)\, x \\
\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} &= \left(\overbrace{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}}^{A} + \overbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}^{B} \overbrace{\begin{bmatrix} k_1 & k_2 \end{bmatrix}}^{k} \right) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\
&= \left(\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ k_1 & k_2 \end{bmatrix} \right) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\
&= \overbrace{\begin{bmatrix} 0 & 1 \\ k_1 & k_2 \end{bmatrix}}^{A_c} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\end{aligned}$$

For stable system, we need $k_1, k_2 < 0$ from looking at the roots of the characteristic equation. Now we solve the Lyapunov equation.

$$\lambda \left(k^T k \right) + PA_c + A_c^T P = -I$$

$$\begin{aligned}
\lambda \begin{bmatrix} k_1 & k_2 \end{bmatrix}^T \begin{bmatrix} k_1 & k_2 \end{bmatrix} + \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ k_1 & k_2 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ k_1 & k_2 \end{bmatrix}^T \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} &= \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\
\lambda \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} \begin{bmatrix} k_1 & k_2 \end{bmatrix} + \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ k_1 & k_2 \end{bmatrix} + \begin{bmatrix} 0 & k_1 \\ 1 & k_2 \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} &= \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\
\lambda \begin{bmatrix} k_1^2 & k_1 k_2 \\ k_1 k_2 & k_2^2 \end{bmatrix} + \begin{bmatrix} k_1 p_{12} & p_{11} + k_2 p_{12} \\ k_1 p_{22} & p_{21} + k_2 p_{22} \end{bmatrix} + \begin{bmatrix} k_1 p_{21} & k_1 p_{22} \\ p_{11} + k_2 p_{21} & p_{12} + k_2 p_{22} \end{bmatrix} &= \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\
\begin{bmatrix} k_1 \left(p_{12} + p_{21} + \lambda k_1 \right) & p_{11} + k_1 p_{22} + k_2 p_{12} + \lambda k_1 k_2 \\ p_{11} + k_1 p_{22} + k_2 p_{21} + \lambda k_1 k_2 & \lambda k_2^2 + 2 p_{22} k_2 + p_{12} + p_{21} \end{bmatrix} &= \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}
\end{aligned}$$

Hence we have 4 equations to solve for $p_{11}, p_{12}, p_{21}, p_{22}$. (but we know also that $p_{12} = p_{21}$). Now let $\lambda = 1$ per the problem, and we obtain the four equations from above as

$$\begin{aligned}
k_1^2 + k_1 p_{12} + k_1 p_{21} &= -1 \\
p_{11} + k_1 k_2 + k_1 p_{22} + k_2 p_{12} &= 0 \\
p_{11} + k_1 k_2 + k_1 p_{22} + k_2 p_{21} &= 0 \\
k_2^2 + 2 p_{22} k_2 + p_{12} + p_{21} &= -1
\end{aligned}$$

Solution is (Using Matlab syms).

```
clear;
syms k1 k2 p11 p12 p21 p22;
k   = [k1,k2];
A   = [0,1;0,0];
B   = [0;1];
Ac  = A+B*k;
P   = [p11 p12;p21 p22];
lam = 1;
eq  =  lam*(k.')*k + (Ac.')*P + P*Ac == -eye(2);
sol = solve(eq,{p11,p12,p21,p22});
P   = simplify(subs(P,sol))
x0  = [1;0];
```

```
J1  = simplify(x0'*P*x0)
```

.

Gives

```
P =
[ -(k1^3 - k1^2 + k1 - k2^2)/(2*k1*k2),  -(k1^2 + 1)/(2*k1)]
[ -(k1^2 + 1)/(2*k1), -(- k1^2 + k1*k2^2 + k1 - 1)/(2*k1*k2)]
J1 =
-(k1^3 - k1^2 + k1 - k2^2)/(2*k1*k2)
```

$$P=\begin{bmatrix}-\frac{k_1-k_1^2+k_1^3-k_2^2}{2k_1k_2} & -\frac{k_1^2+1}{2k_1}\\ -\frac{k_1^2+1}{2k_1} & -\frac{k_1+k_1k_2^2-k_1^2-1}{2k_1k_2}\end{bmatrix}$$

Hence

$$\begin{aligned}J(k)&=x^T(0)\,P(k)\,x(0)\\&=\begin{bmatrix}1&0\end{bmatrix}\begin{bmatrix}-\frac{k_1-k_1^2+k_1^3-k_2^2}{2k_1k_2} & -\frac{k_1^2+1}{2k_1}\\ -\frac{k_1^2+1}{2k_1} & -\frac{k_1+k_1k_2^2-k_1^2-1}{2k_1k_2}\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\end{aligned}$$

Therefore

$$\boxed{J(k)=-\frac{1}{2k_1k_2}\left(k_1^3-k_1^2+k_1-k_2^2\right)}$$

For $k_1=k_2=k$, the above becomes

$$\begin{aligned}J(k)&=-\frac{\left(k^3-2k^2+k\right)}{2k^2}\\&=-\frac{\left(k^2-2k+1\right)}{2k}\end{aligned}$$

Or

$$\boxed{J(k)=-\frac{1}{2k}(k-1)^2}$$

Now we find the optimal J^*. Since

$$\frac{dJ(k)}{dk}=\frac{(k-1)^2}{2k^2}-\frac{(2k-2)}{2k}$$

Then $\frac{dJ(k)}{dk}=0$ gives

$$k=1,-1$$

Since k must be negative for stable system, we pick

$$\boxed{k^*=-1}$$

And

$$\frac{d^2J(k)}{dk^2}=\frac{(k-1)^2}{k^3}-\frac{2(1-k)}{k^2}-\frac{1}{k}$$

At $k^*=-1$

$$\frac{d^2J(k)}{dk^2}=1>0$$

Hence this is a minimum. Therefore

$$J^*=-\frac{1}{2k}(k-1)^2\bigg|_{k=-1}$$

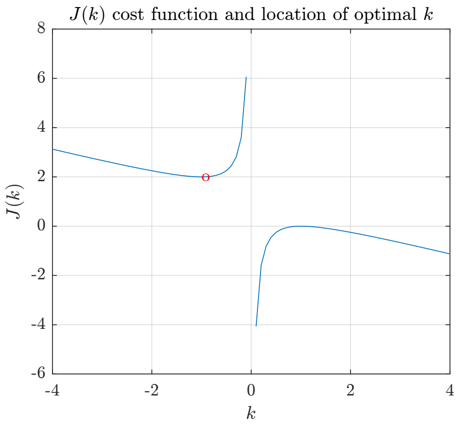
Hence

$$\boxed{J^*=2}$$

J^* do not get to zero. (same as in the class problem we did without λu^Tu term. I thought we will get $J^*=0$ now since this I thought it was the reason for using λu^Tu. I hope I did not make mistake, but do not see where if I did. Below is a plot of $J(k)$.

```
clear k;
close all;
reset(0);
set(groot,'defaulttextinterpreter','Latex');
set(groot, 'defaultAxesTickLabelInterpreter','Latex');
set(groot, 'defaultLegendInterpreter','Latex');
f=@(k) (-1./(2*k).*(k-1).^2)
k=-4:.1:4;
plot(k,f(k));
text(-1,f(-1),'o','color','red')
title('$J(k)$ cost function and location of optimal $k$');
xlabel('$k$'); ylabel('$J(k)$');
grid;
```

.



At $k = 1$ then $J(1) = 0$, but we can not use $k = 1$ since this will make the system not stable. The system now using $k^* = -1$ becomes

$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ k_1 & k_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

To verify it is stable. Since

$$|(\lambda I - A_c)| = \lambda^2 + \lambda + 1$$

The roots are

$$-\frac{1}{2} \pm \frac{1}{2} i\sqrt{3}$$

Hence the system is stable since real part of roots are negative. If we had used $k = 1$, the roots will be $-0.618, 1.618$, and the system would have been unstable.

4.3.5.3 Part(c)

From last part, we obtained P

$$P = \begin{bmatrix} -\frac{k_1 - k_1^2 + k_1^3 - k_2^2}{2k_1 k_2} & -\frac{k_1^2+1}{2k_1} \\ -\frac{k_1^2+1}{2k_1} & -\frac{k_1 + k_1 k_2^2 - k_1^2 - 1}{2k_1 k_2} \end{bmatrix}$$

When $k_1 = k_2 = k$ the above becomes

$$P = \begin{bmatrix} \frac{-k + 2k^2 - k^3}{2k^2} & -\frac{k^2+1}{2k} \\ -\frac{k^2+1}{2k} & \frac{1 - k - k^3 + k^2}{2k^2} \end{bmatrix}$$

Now since $x(0)$ is random variable, then

$$
\begin{aligned}
J(k) &= E\left(x^T(0)\,Px(0)\right)\\
&= E\left(\begin{bmatrix} x_1(0) & x_2(0)\end{bmatrix}\begin{bmatrix}\frac{-k+2k^2-k^3}{2k^2} & -\frac{k^2+1}{2k}\\ -\frac{k^2+1}{2k} & \frac{1-k-k^3+k^2}{2k^2}\end{bmatrix}\begin{bmatrix}x_1(0)\\ x_2(0)\end{bmatrix}\right)\\
&= E\left(-\frac{1}{2k^2}\left(k^3x_1^2(0)+2k^3x_1(0)\,x_2(0)+k^3x_2^2(0)-2k^2x_1^2(0)-k^2x_2^2(0)+kx_1^2(0)+2kx_1(0)\,x_2(0)+kx_2^2(0)-x_2^2(0)\right)\right)
\end{aligned}
\tag{1}
$$

Let $E(x_1(0))=\bar{x}_1$ and $E(x_2(0))=\bar{x}_2$ Then

$$
J(k)=-\frac{1}{2k^2}\left(k^3\bar{x}_1^2+2k^3\bar{x}_1\bar{x}_2+k^3\bar{x}_2^2-2k^2\bar{x}_1^2-k^2\bar{x}_2^2+k\bar{x}_1^2+2k\bar{x}_1\bar{x}_2+k\bar{x}_2^2-\bar{x}_2^2\right)
$$

But $E(x_1(0))=0$, hence $\bar{x}_1=0$ and similarly $\bar{x}_2=0$, but $\bar{x}_1^2=\frac{1}{3}$ since

$$
\int_{-1}^{1}x^2p(x)\,dx=\frac{1}{2}\int_{-1}^{1}x^2dx=\frac{1}{2}\left(\frac{x^3}{3}\right)_{-1}^{1}=\frac{1}{3}
$$

Similarly $\bar{x}_2^2=\frac{1}{3}$ and $\bar{x}_1\bar{x}_2=0$ (since i.i.d, then $E(x_1(0)\,x_2(0))=E(x_1(0))\,E(x_2(0))=0$. Using these values of expectations, Eq (1) becomes

$$
J(k)=-\frac{1}{2k^2}\left(k^3\frac{1}{3}+k^3\frac{1}{3}-2k^2\frac{1}{3}-k^2\frac{1}{3}+k\frac{1}{3}+k\frac{1}{3}-\frac{1}{3}\right)
$$

Or

$$
\boxed{J(k)=\frac{-2k^3+3k^2-2k+1}{6k^2}}\tag{2}
$$

To find the optimal:

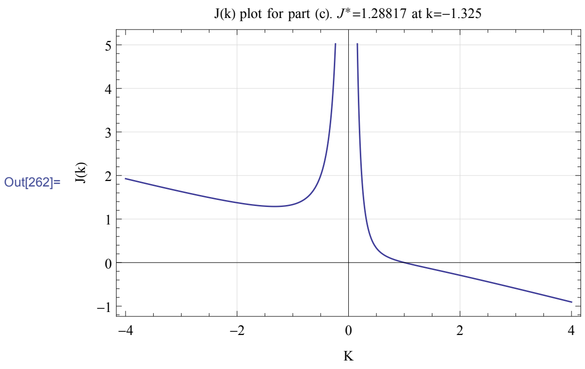
$$
\frac{dJ(k)}{dk}=-\frac{1}{3}-\frac{1}{3k^3}+\frac{1}{3k^2}
$$

$\frac{dJ(k)}{dk}=0$ gives 3 roots. The only one which is real and negative (the other two are complex) is

$$
\boxed{k^*=-1.325}
$$

At this k^*, we check $\frac{d^2J(k)}{dk^2}$ and find it is $0.611>0$, hence J is minimum at k^*. The value J^* at k^* is found to be from substituting k^* in (2)

$$
\boxed{J^*=1.28817}
$$



We now check if the system is stable. (it should be, since $k^*<1$). The system now is

$$
\begin{aligned}
\begin{bmatrix}x_1'\\ x_2'\end{bmatrix}&=\begin{bmatrix}0 & 1\\ k_1 & k_2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}\\
&=\begin{bmatrix}0 & 1\\ -1.325 & -1.325\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}
\end{aligned}
$$

Hence

$$
|(\lambda I-A_c)|=\lambda^2+1.325\lambda+1.325
$$

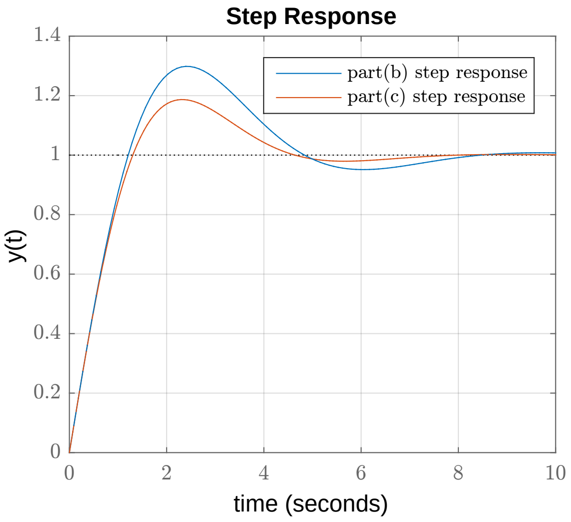
The roots are

$$-0.6625 \pm i0.941$$

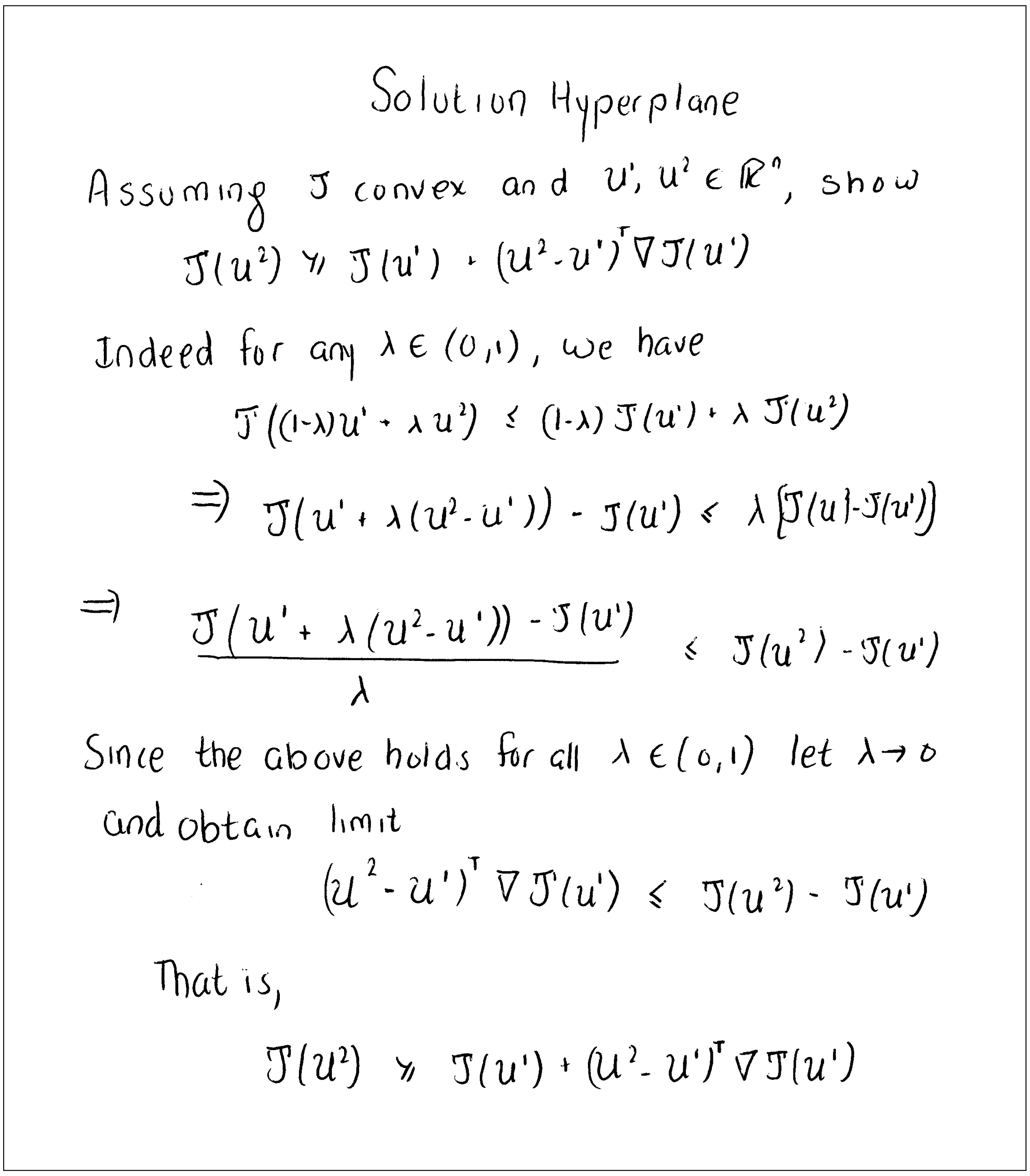
The system is stable since real part of roots are negative. The following is the step response for system in part(b) and part(c) to compare.

```
%show step responses
close all;
figure();
close all
A   = [0 1;-1 -1];
B   = [1;0]
sys = ss(A,B,[1 0],[0])
step(sys)
hold on;

A   = [0 1;-1.325 -1.325];
B   = [1;0]
sys = ss(A,B,[1 0],[0])
step(sys)
legend('part(b) step response','part(c) step response')
xlabel('time');
ylabel('y(t)');
grid
```

.



4.3.6 HW 3 key solution



Solution Hyperplane

Assuming J convex and $u^1, u^2 \in \mathbb{R}^n$, show

$$J(u^2) \geqslant J(u^1) + (u^2 - u^1)^T \nabla J(u^1)$$

Indeed for any $\lambda \in (0,1)$, we have

$$J((1-\lambda)u^1 + \lambda u^2) \leqslant (1-\lambda) J(u^1) + \lambda J(u^2)$$

$$\Rightarrow J(u^1 + \lambda(u^2 - u^1)) - J(u^1) \leqslant \lambda [J(u) - J(u^1)]$$

$$\Rightarrow \frac{J(u^1 + \lambda(u^2 - u^1)) - J(u^1)}{\lambda} \leqslant J(u^2) - J(u^1)$$

Since the above holds for all $\lambda \in (0,1)$ let $\lambda \to 0$ and obtain limit

$$(u^2 - u^1)^T \nabla J(u^1) \leqslant J(u^2) - J(u^1)$$

That is,

$$J(u^2) \geqslant J(u^1) + (u^2 - u^1)^T \nabla J(u^1)$$

Barmish

ECE 719 – Solution Eigenvalue

With $M(q)$ being an $n \times n$ symmetric matrix having entries which are convex in q, a counterexample can be given. Reader: If you proved this case, you should identify where in proof you are in error. Hint: Consider a 2×2 matrix, and notice $x^T M(q)x$ may not be convex with negative x_i. Henceforth, we assume $M(q)$ is linear in q. This makes $x^T M(q)x$ linear and hence convex. Now, we want to show that the maximum eigenvalue

$$J(q) \doteq \lambda_{max}[M(q)]$$

is a convex function of q. To establish this desired convexity, we first view $\mathcal{X} = \{x \in \mathbf{R}^n : ||x|| = 1\}$ as an *index set.* Now, we define an indexed collection of convex functions as follows: For each $x \in \mathcal{X}$, let

$$J_x(q) \doteq x^T M(q)x.$$

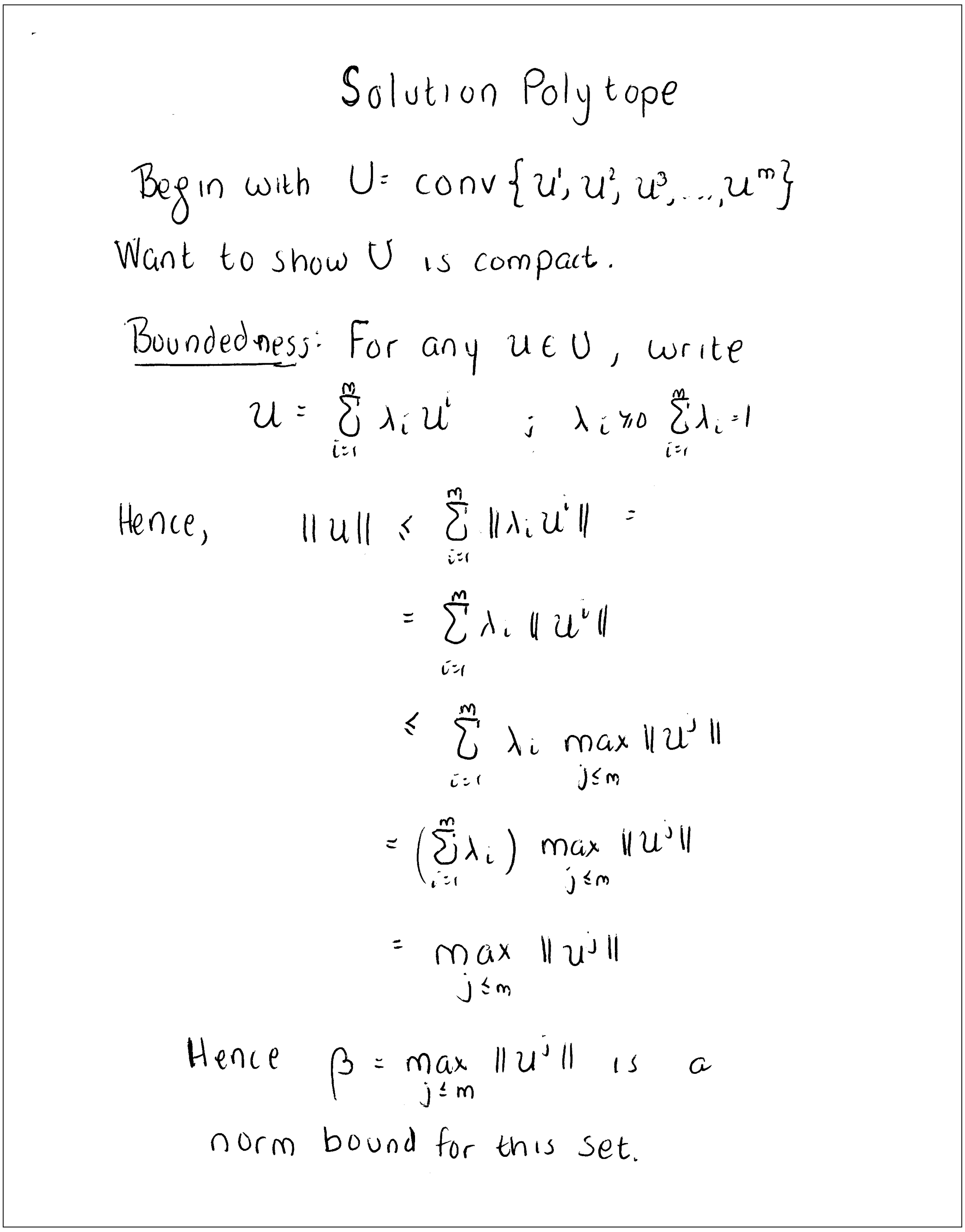
Since this function is linear, hence convex, in q, J_x is convex. Recalling from matrix algebra that

$$\lambda_{max}[M(q)] = \max_{||x||=1} x^T M(q)x,$$

it follows that

$$J(q) = \max_{x \in \mathcal{X}} J_x(q)$$

Now, using the class result that the supremum over an indexed collection of convex functions is convex, we conclude that $J(q)$ is convex.



Solution Polytope

Begin with $U = \text{conv}\{u^1, u^2, u^3, \dots, u^m\}$

Want to show U is compact.

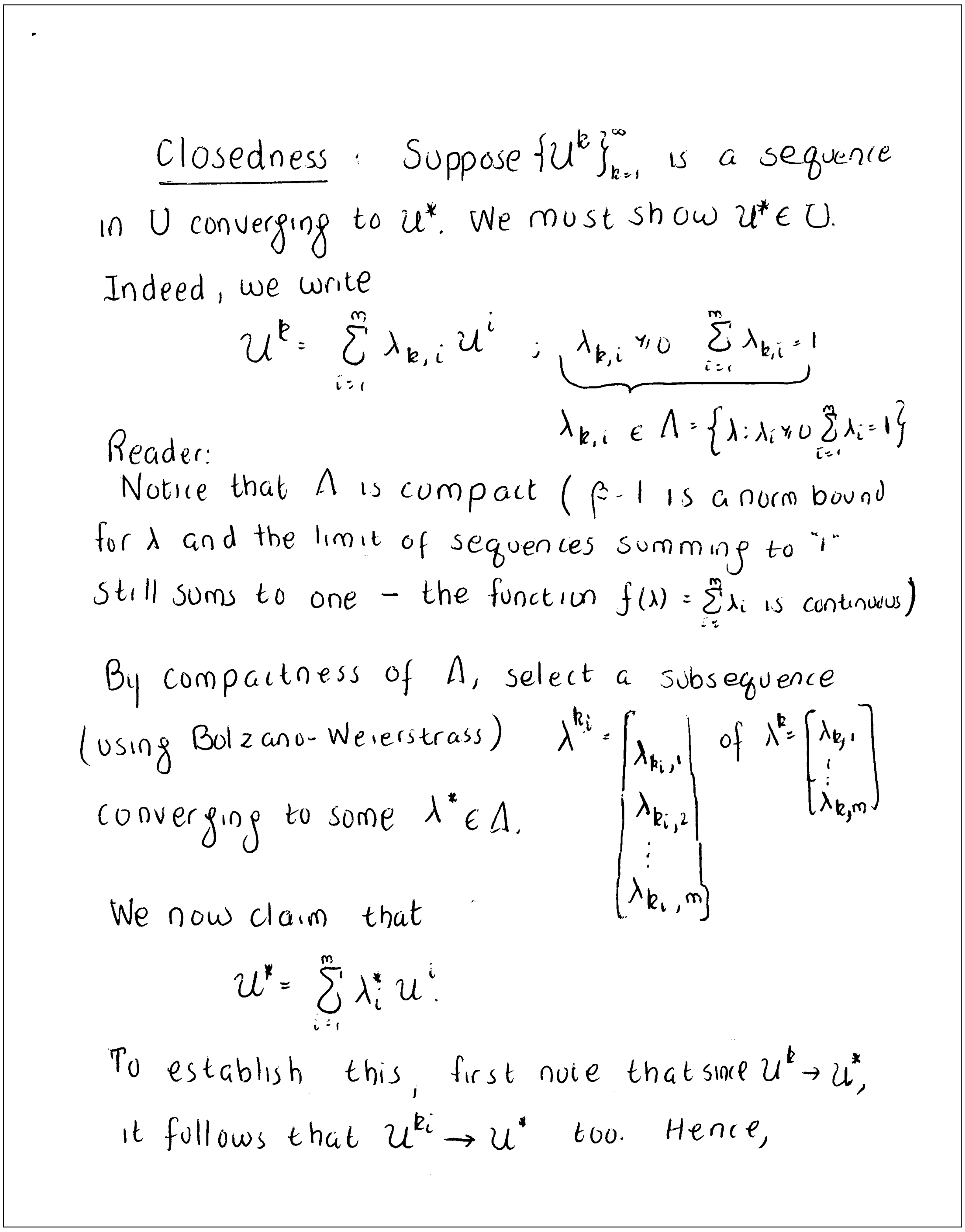
Boundedness: For any $u \in U$, write

$$u = \sum_{i=1}^{m} \lambda_i u^i \quad ; \quad \lambda_i \geq 0 \quad \sum_{i=1}^{m} \lambda_i = 1$$

Hence,

$$\begin{aligned} \|u\| &\leq \sum_{i=1}^{m} \|\lambda_i u^i\| = \\ &= \sum_{i=1}^{m} \lambda_i \|u^i\| \\ &\leq \sum_{i=1}^{m} \lambda_i \max_{j \leq m} \|u^j\| \\ &= \left(\sum_{i=1}^{m} \lambda_i\right) \max_{j \leq m} \|u^j\| \\ &= \max_{j \leq m} \|u^j\| \end{aligned}$$

Hence $\beta = \max_{j \leq m} \|u^j\|$ is a norm bound for this set.



Closedness : Suppose $\{u^k\}_{k=1}^{\infty}$ is a sequence in U converging to u^*. We must show $u^* \in U$.

Indeed, we write

$$u^k = \sum_{i=1}^{m} \lambda_{k,i} u^i \; ; \; \underbrace{\lambda_{k,i} \geq 0 \quad \sum_{i=1}^{m} \lambda_{k,i} = 1}_{\lambda_{k,i} \in \Lambda = \{\lambda : \lambda_i \geq 0 \ \sum_{i=1}^{m} \lambda_i = 1\}}$$

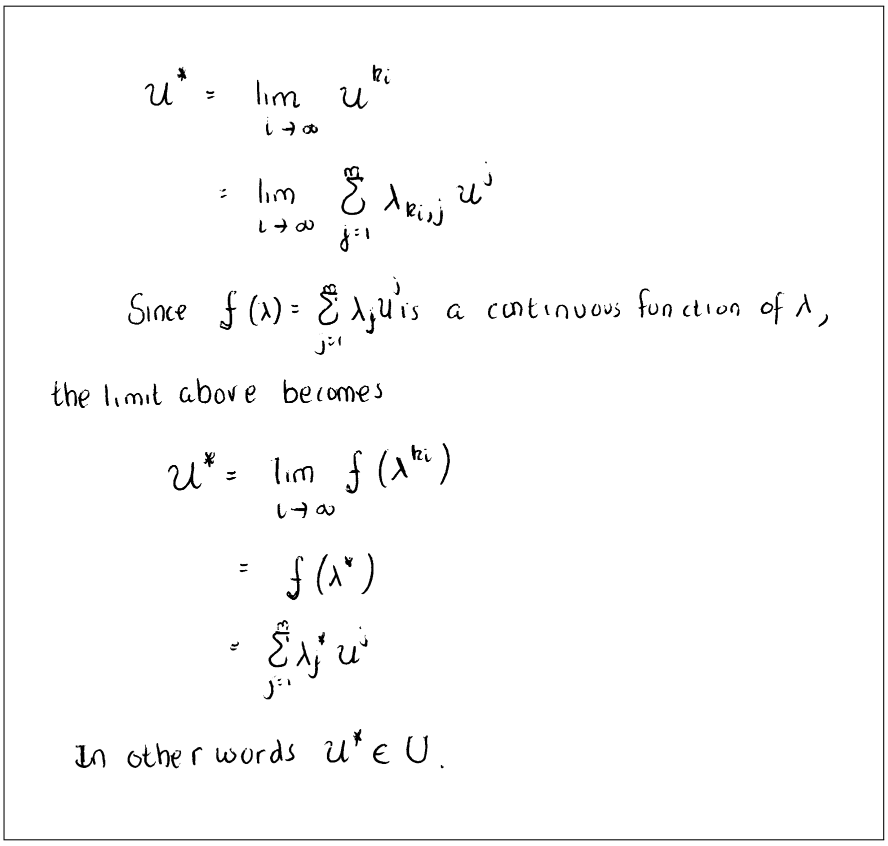
Reader:
Notice that Λ is compact (ℓ-1 is a norm bound for λ and the limit of sequences summing to "1" still sums to one - the function $f(\lambda) = \sum_{i=1}^{m} \lambda_i$ is continuous)

By compactness of Λ, select a subsequence (using Bolzano-Weierstrass) $\lambda^{k_i} = \begin{bmatrix} \lambda_{k_i,1} \\ \lambda_{k_i,2} \\ \vdots \\ \lambda_{k_i,m} \end{bmatrix}$ of $\lambda^k = \begin{bmatrix} \lambda_{k,1} \\ \vdots \\ \lambda_{k,m} \end{bmatrix}$ converging to some $\lambda^* \in \Lambda$.

We now claim that

$$u^* = \sum_{i=1}^{m} \lambda_i^* u^i.$$

To establish this, first note that since $u^k \to u^*$, it follows that $u^{k_i} \to u^*$ too. Hence,



$$u^* = \lim_{i \to \infty} u^{k_i}$$

$$= \lim_{i \to \infty} \sum_{j=1}^{m} \lambda_{k_i,j} u^j$$

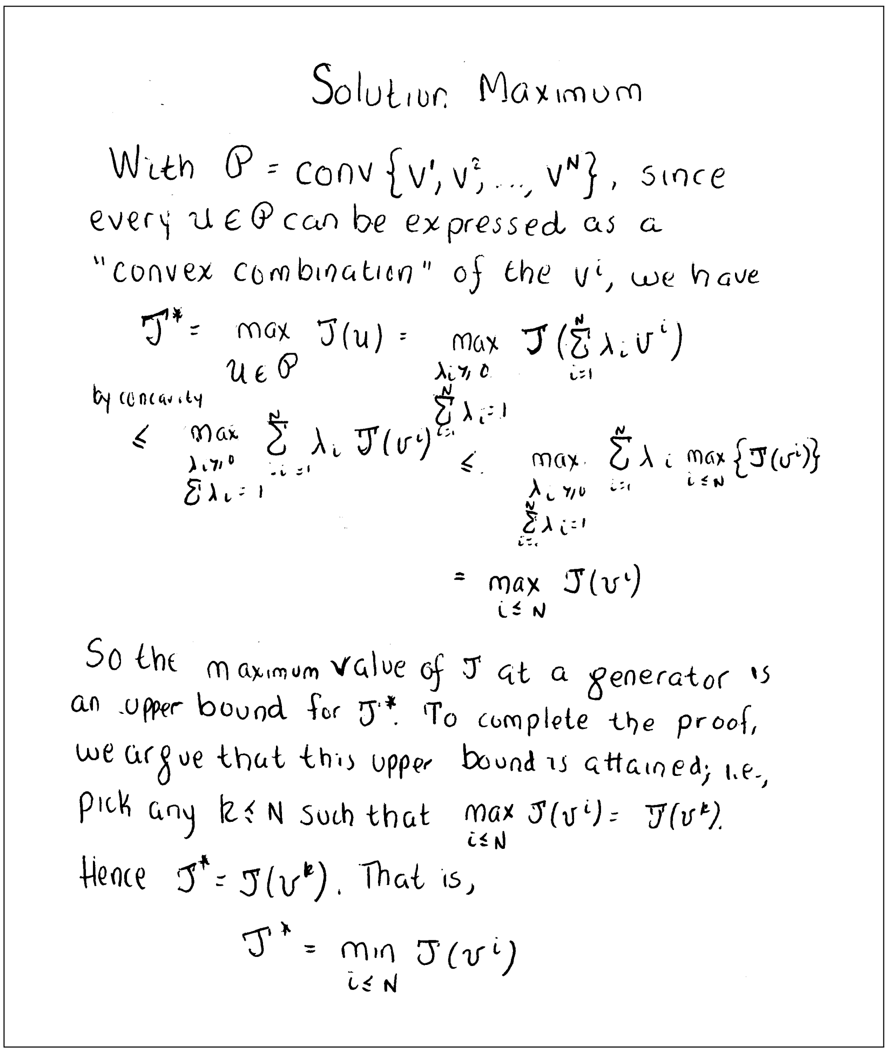
Since $f(\lambda) = \sum_{j=1}^{m} \lambda_j u^j$ is a continuous function of λ, the limit above becomes

$$u^* = \lim_{i \to \infty} f(\lambda^{k_i})$$

$$= f(\lambda^*)$$

$$= \sum_{j=1}^{m} \lambda_j^* u^j$$

In other words $u^* \in U$.



Solution Maximum

With $\mathcal{P} = \text{conv}\{v^1, v^2, \ldots, v^N\}$, since every $u \in \mathcal{P}$ can be expressed as a "convex combination" of the v^i, we have

$$J^* = \max_{u \in \mathcal{P}} J(u) = \max_{\substack{\lambda_i \geq 0 \\ \sum_{i=1}^N \lambda_i = 1}} J\left(\sum_{i=1}^{N} \lambda_i v^i\right)$$

by concavity

$$\leq \max_{\substack{\lambda_i \geq 0 \\ \sum \lambda_i = 1}} \sum_{i=1}^{N} \lambda_i J(v^i) \leq \max_{\substack{\lambda_i \geq 0 \\ \sum_{i=1}^N \lambda_i = 1}} \sum_{i=1}^{N} \lambda_i \max_{i \leq N} \{J(v^i)\}$$

$$= \max_{i \leq N} J(v^i)$$

So the maximum value of J at a generator is an upper bound for J^*. To complete the proof, we argue that this upper bound is attained; i.e., pick any $k \leq N$ such that $\max_{i \leq N} J(v^i) = J(v^k)$. Hence $J^* = J(v^k)$. That is,

$$J^* = \min_{i \leq N} J(v^i)$$

With

$$J(K) = \int_0^\infty x^T(t)x(t) + \lambda u^T(t)u(t)dt,$$

we substitute $u = Kx$ and obtain

$$J(K) = \int_0^\infty x^T(t)[I + \lambda K^T K]x(t)dt.$$

Now expressing the integrand as the exact differential

$$x^T(t)[I + \lambda K^T K]x(t) = -d(x^T P x),$$

we substitute the dynamics into the expression above and proceed exactly as in class. This yields Lyapunov equation

$$\bar{A}^T P + P\bar{A} = -(I + \lambda K^T K)$$

to be solved for P.

$$\bar{A} = A + BK = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix}[k \quad k] \quad k_1 = k_2 = k$$

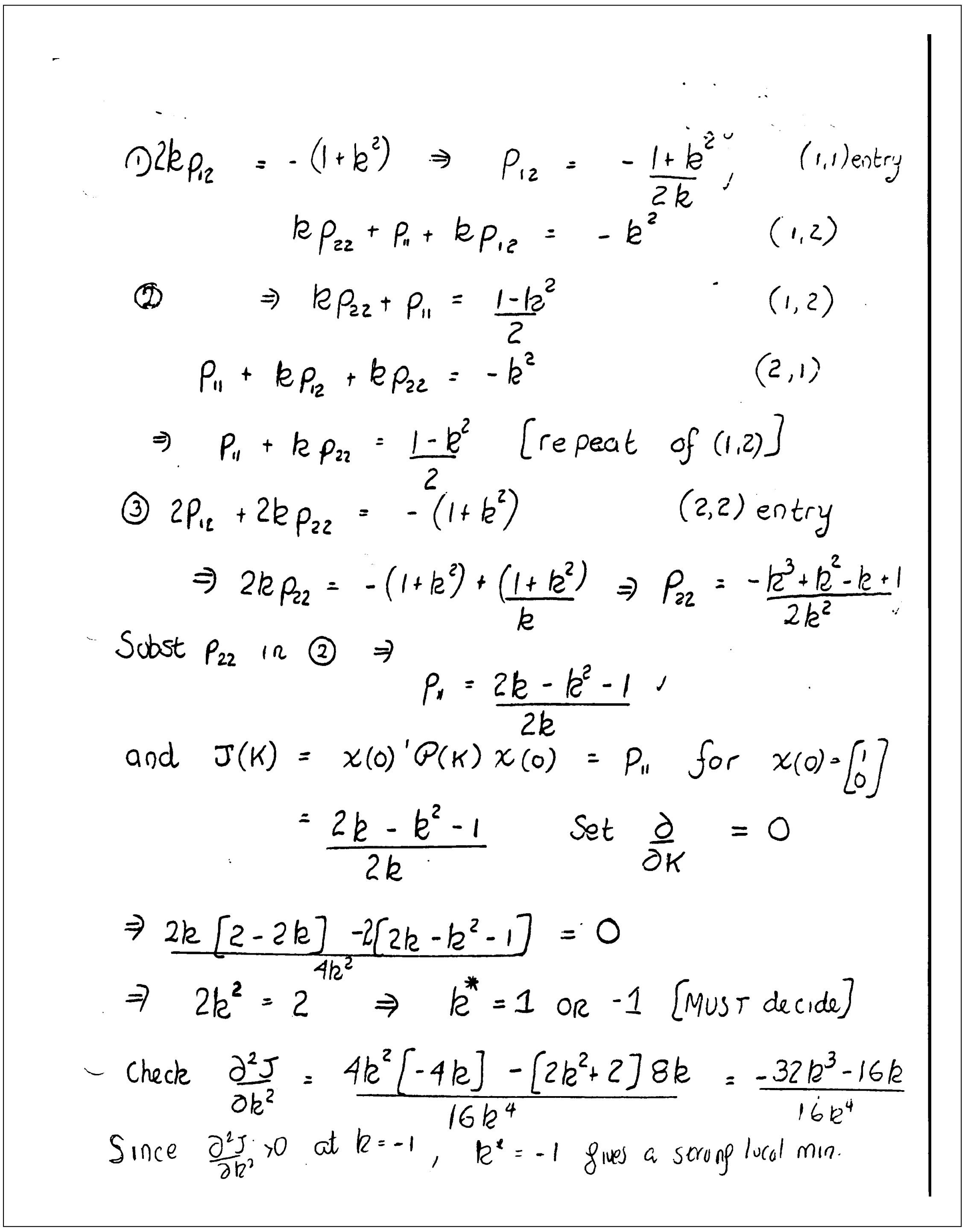
$$= \begin{bmatrix}0 & 1\\ k & k\end{bmatrix}. \text{ Evaluate } \underset{(\lambda = 1)}{I + KK'} = \begin{bmatrix}1+k^2 & k^2\\ k^2 & 1+k^2\end{bmatrix}$$

Look at

$$\begin{bmatrix}0 & k\\ 1 & k\end{bmatrix}\begin{bmatrix}p_{11} & p_{12}\\ p_{12} & p_{22}\end{bmatrix} + \begin{bmatrix}p_{11} & p_{12}\\ p_{12} & p_{22}\end{bmatrix}\begin{bmatrix}0 & 1\\ k & k\end{bmatrix} = -\begin{bmatrix}1+k^2 & k^2\\ k^2 & 1+k^2\end{bmatrix}$$

See next page

Reader: Please note that solution to follow is for $x(0) = \begin{bmatrix}1\\ 0\end{bmatrix}$. Same ideas apply for $x(0) = \begin{bmatrix}1\\ 1\end{bmatrix}$



① $2kp_{12} = -(1+k^2) \Rightarrow p_{12} = -\frac{1+k^2}{2k}$ ✓ (1,1) entry

$kp_{22} + p_{11} + kp_{12} = -k^2$ (1,2)

② $\Rightarrow kp_{22} + p_{11} = \frac{1-k^2}{2}$ (1,2)

$p_{11} + kp_{12} + kp_{22} = -k^2$ (2,1)

$\Rightarrow p_{11} + kp_{22} = \frac{1-k^2}{2}$ [repeat of (1,2)]

③ $2p_{12} + 2kp_{22} = -(1+k^2)$ (2,2) entry

$\Rightarrow 2kp_{22} = -(1+k^2) + \frac{(1+k^2)}{k} \Rightarrow p_{22} = -\frac{k^3+k^2-k+1}{2k^2}$

Subst p_{22} in ② $\Rightarrow$

$p_{11} = \frac{2k-k^2-1}{2k}$ ✓

and $J(K) = x(0)'P(K)x(0) = p_{11}$ for $x(0) = \begin{bmatrix}1\\0\end{bmatrix}$

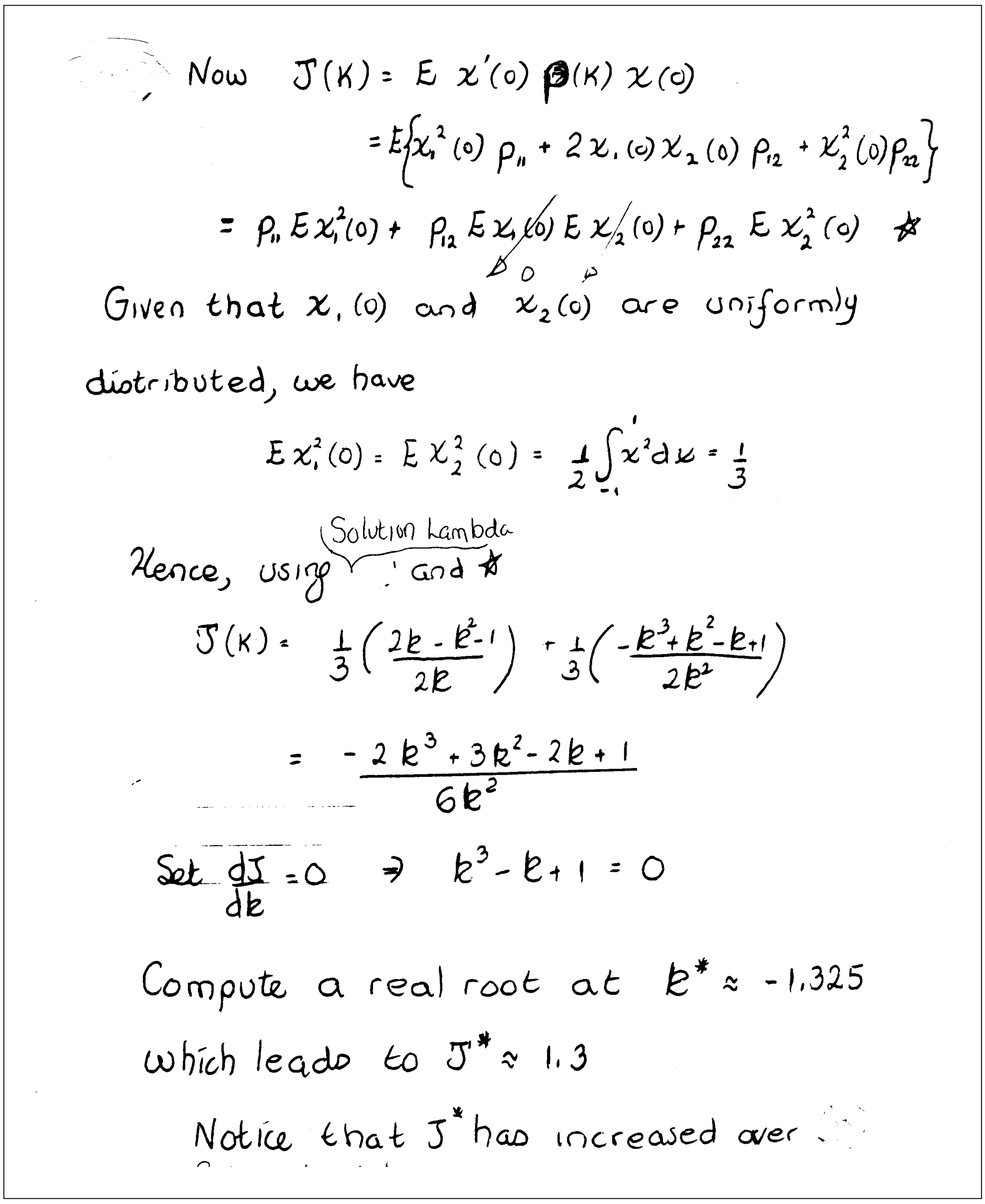
$= \frac{2k-k^2-1}{2k}$ Set $\frac{\partial}{\partial K} = 0$

$\Rightarrow \frac{2k[2-2k] - 2[2k-k^2-1]}{4k^2} = 0$

$\Rightarrow 2k^2 = 2 \Rightarrow k^* = 1$ or -1 [MUST decide]

Check $\frac{\partial^2 J}{\partial k^2} = \frac{4k^2[-4k] - [2k^2+2]8k}{16k^4} = \frac{-32k^3-16k}{16k^4}$

Since $\frac{\partial^2 J}{\partial k^2} > 0$ at $k=-1$, $k^* = -1$ gives a strong local min.



Now $J(K) = E\, x'(0)\, P(K)\, x(0)$

$$= E\left\{x_1^2(0)\, p_{11} + 2 x_1(0) x_2(0)\, p_{12} + x_2^2(0) p_{22}\right\}$$

$$= p_{11} E x_1^2(0) + p_{12} \underset{0}{\cancel{E x_1(0)}}\, \underset{0}{\cancel{E x_2(0)}} + p_{22} E x_2^2(0) \quad ★$$

Given that $x_1(0)$ and $x_2(0)$ are uniformly distributed, we have

$$E x_1^2(0) = E x_2^2(0) = \frac{1}{2}\int_{-1}^{1} x^2 dx = \frac{1}{3}$$

Hence, using $\overbrace{\quad}^{\text{Solution Lambda}}$ and ★

$$J(K) = \frac{1}{3}\left(\frac{2k - k^2 - 1}{2k}\right) + \frac{1}{3}\left(\frac{-k^3 + k^2 - k + 1}{2k^2}\right)$$

$$= \frac{-2k^3 + 3k^2 - 2k + 1}{6k^2}$$

Set $\frac{dJ}{dk} = 0 \Rightarrow k^3 - k + 1 = 0$

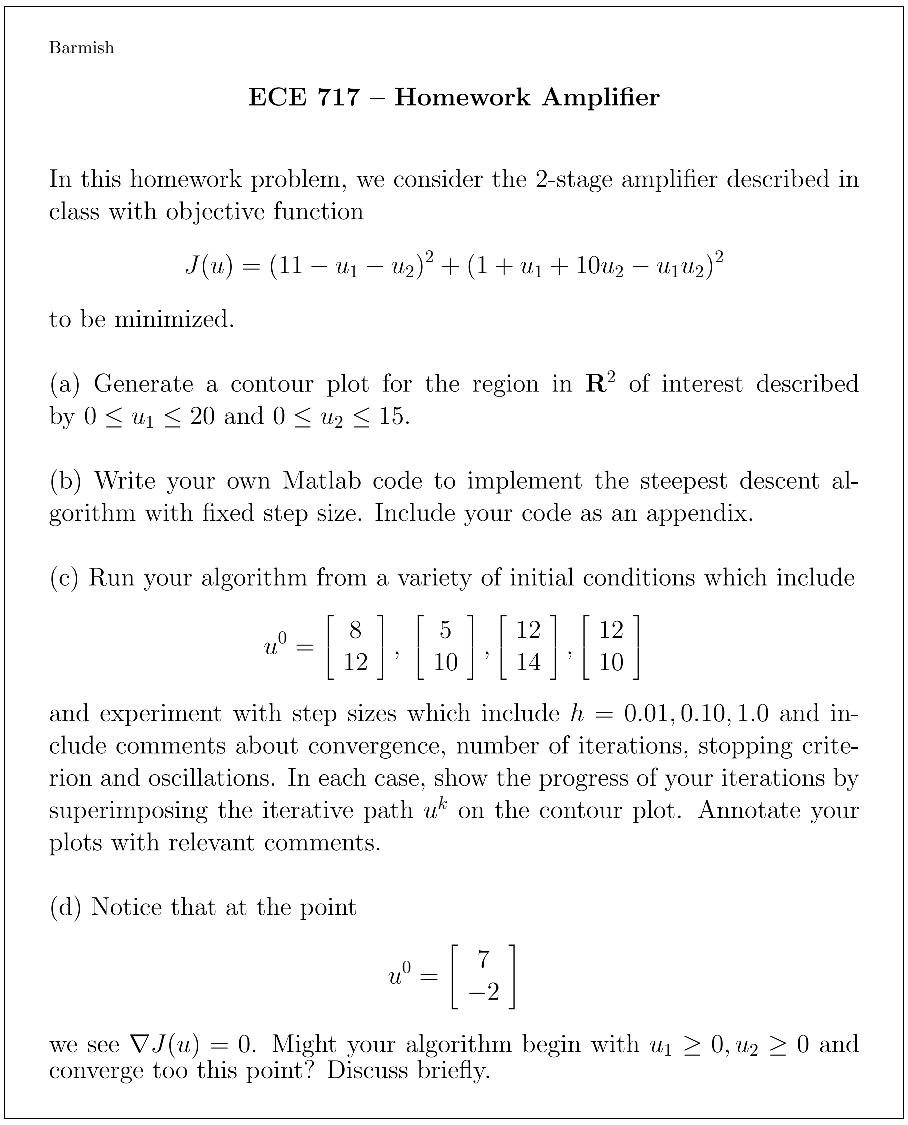
Compute a real root at $k^* \approx -1.325$

which leads to $J^* \approx 1.3$

Notice that J^* has increased over

4.4 HW 4

4.4.1 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 4.1: problem 1 description

4.4.1.1 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.

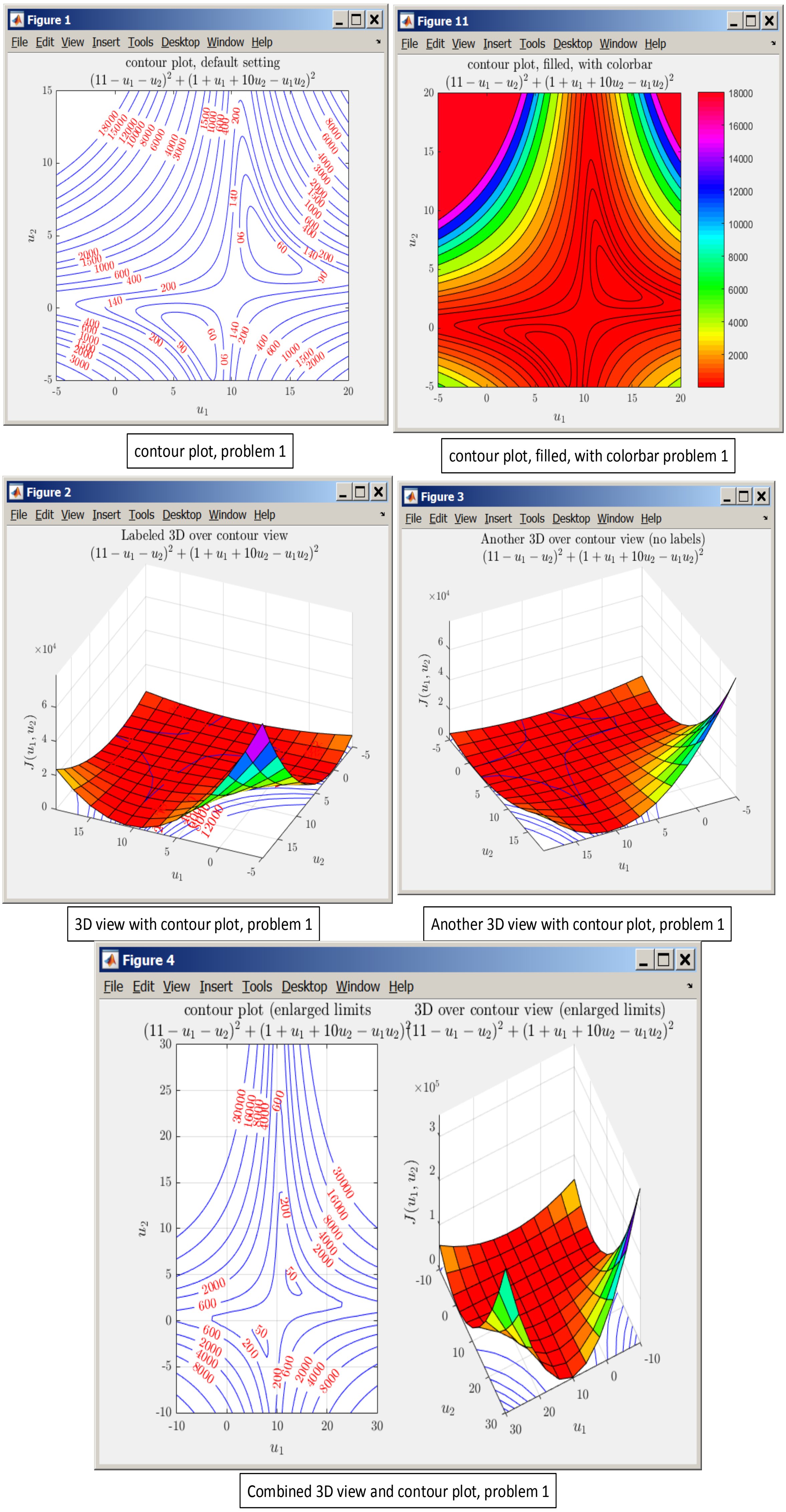


Figure 1
File Edit View Insert Tools Desktop Window Help
contour plot, default setting
$(11-u_1-u_2)^2+(1+u_1+10u_2-u_1u_2)^2$
u_1
u_2
-5 0 5 10 15 20
18000 15000 12000 10000 8000 6000 4000 3000 2000 1500 1000 600 400 200 140 90 60
Figure 11
File Edit View Insert Tools Desktop Window Help
contour plot, filled, with colorbar
$(11-u_1-u_2)^2+(1+u_1+10u_2-u_1u_2)^2$
u_1
u_2
2000 4000 6000 8000 10000 12000 14000 16000 18000
Figure 2
File Edit View Insert Tools Desktop Window Help
Labeled 3D over contour view
$(11-u_1-u_2)^2+(1+u_1+10u_2-u_1u_2)^2$
$\times10^4$
$J(u_1,u_2)$
u_1
u_2
Figure 3
File Edit View Insert Tools Desktop Window Help
Another 3D over contour view (no labels)
$(11-u_1-u_2)^2+(1+u_1+10u_2-u_1u_2)^2$
$\times10^4$
$J(u_1,u_2)$
u_1
u_2
Figure 4
File Edit View Insert Tools Desktop Window Help
contour plot (enlarged limits
3D over contour view (enlarged limits)
$(11-u_1-u_2)^2+(1+u_1+10u_2-u_1u_2)^2$
$(11-u_1-u_2)^2+(1+u_1+10u_2-u_1u_2)^2$
u_1
u_2
-10 0 10 20 30
30000 16000 8000 4000 2000 600 200 50
$\times10^5$
$J(u_1,u_2)$

contour plot, problem 1

contour plot, filled, with colorbar problem 1

3D view with contour plot, problem 1

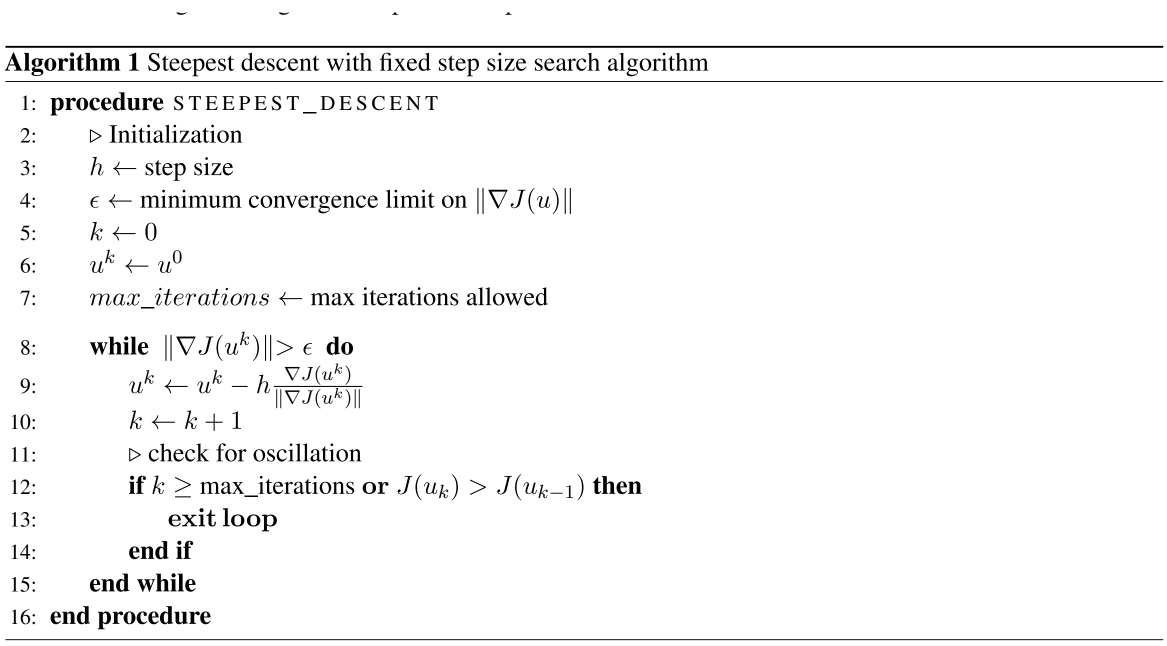
Another 3D view with contour plot, problem 1

Combined 3D view and contour plot, problem 1

Figure 4.2: Matlab output for part(a) problem 1, HW4

4.4.1.2 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 4.3: Steepest descent with fixed step size search algorithm

4.4.1.3 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 4.1: 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`

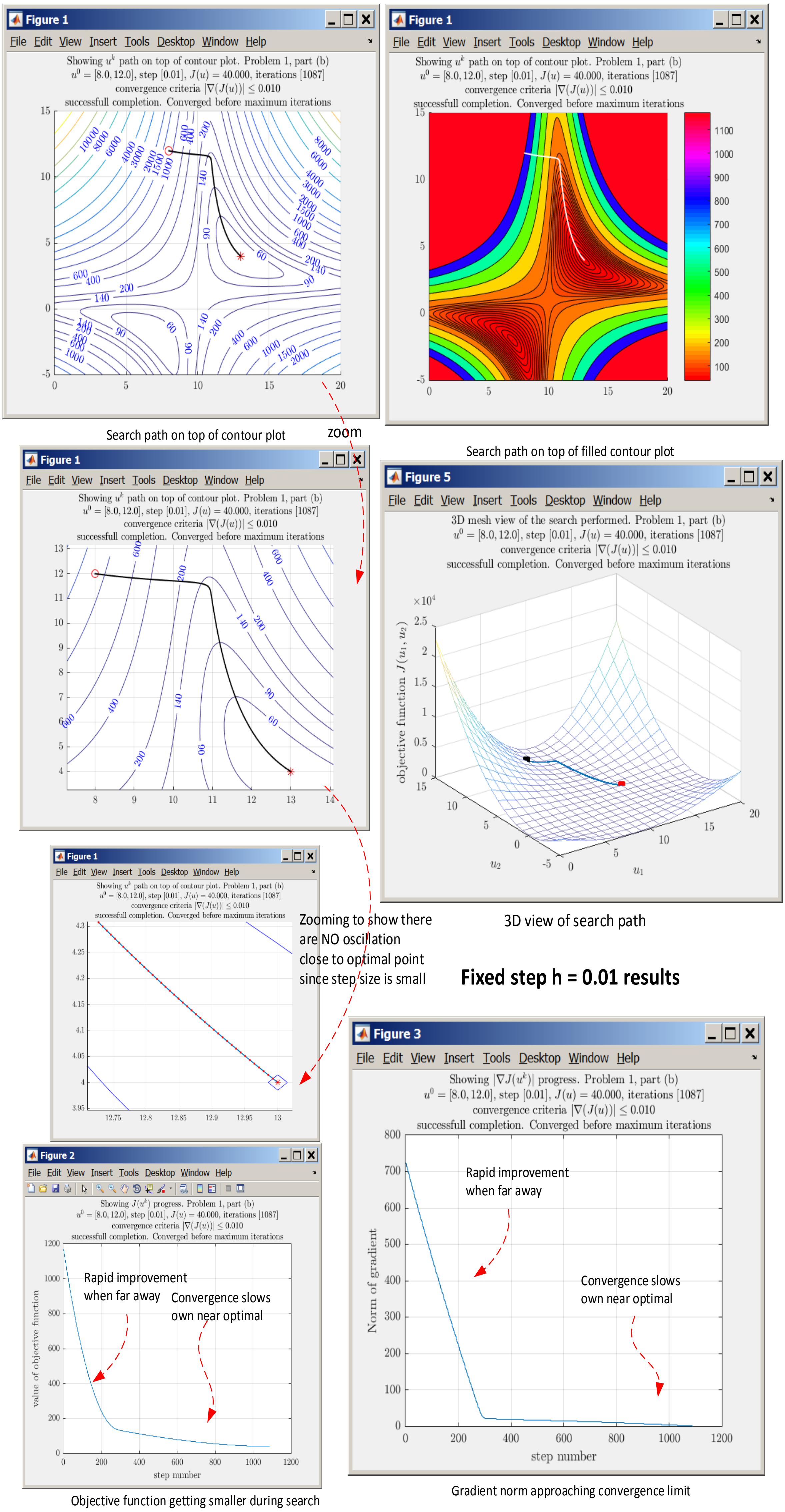


Figure 1

Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.01]$, $J(u) = 40.000$, iterations $[1087]$
convergence criteria $|\nabla(J(u))| \leq 0.010$
successfull completion. Converged before maximum iterations

Search path on top of contour plot

zoom

Figure 1

Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.01]$, $J(u) = 40.000$, iterations $[1087]$
convergence criteria $|\nabla(J(u))| \leq 0.010$
successfull completion. Converged before maximum iterations

Search path on top of filled contour plot

Figure 1

Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.01]$, $J(u) = 40.000$, iterations $[1087]$
convergence criteria $|\nabla(J(u))| \leq 0.010$
successfull completion. Converged before maximum iterations

Figure 5

3D mesh view of the search performed. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.01]$, $J(u) = 40.000$, iterations $[1087]$
convergence criteria $|\nabla(J(u))| \leq 0.010$
successfull completion. Converged before maximum iterations

Figure 1

Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.01]$, $J(u) = 40.000$, iterations $[1087]$
convergence criteria $|\nabla(J(u))| \leq 0.010$
successfull completion. Converged before maximum iterations

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

3D view of search path

Fixed step h = 0.01 results

Figure 3

Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.01]$, $J(u) = 40.000$, iterations $[1087]$
convergence criteria $|\nabla(J(u))| \leq 0.010$
successfull completion. Converged before maximum iterations

Rapid improvement when far away

Convergence slows own near optimal

Figure 2

Showing $J(u^k)$ progress. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.01]$, $J(u) = 40.000$, iterations $[1087]$
convergence criteria $|\nabla(J(u))| \leq 0.010$
successfull completion. Converged before maximum iterations

Rapid improvement when far away

Convergence slows own near optimal

Objective function getting smaller during search

Gradient norm approaching convergence limit

File Edit View Insert Tools Desktop Window Help
objective function $J(u_1, u_2)$
$\times 10^4$
u_1
u_2
value of objective function
step number
Norm of gradient
step number

Figure 4.4: Result for using step 0.01 starting from (8,12)

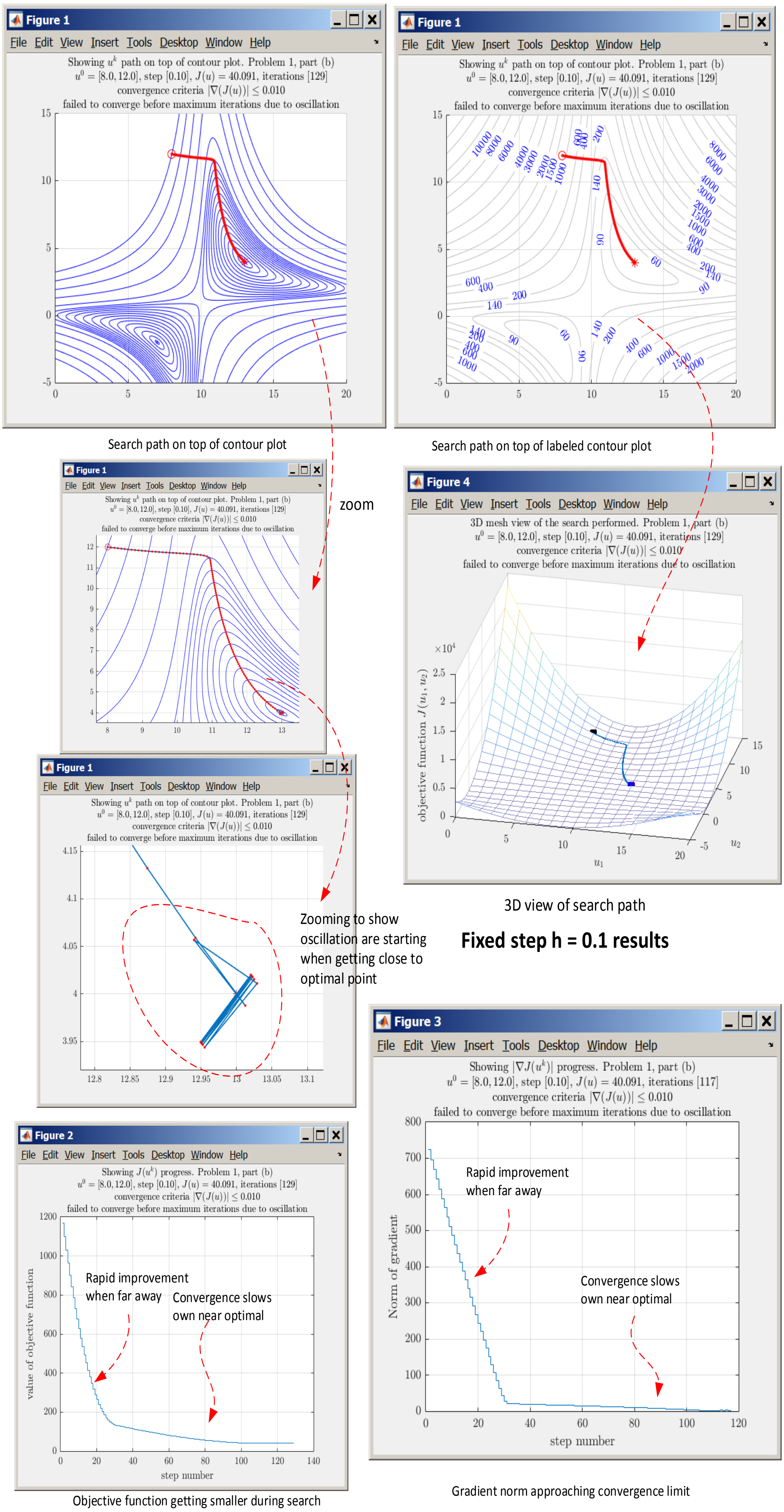


Figure 1
File Edit View Insert Tools Desktop Window Help
Figure 4
Figure 2
Figure 3
objective function $J(u_1,u_2)$
$\times 10^4$
u_1
u_2
value of objective function
Norm of gradient
step number

Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.10]$, $J(u) = 40.091$, iterations $[129]$
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Search path on top of contour plot

Search path on top of labeled contour plot

zoom

Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.10]$, $J(u) = 40.091$, iterations $[129]$
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

3D mesh view of the search performed. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.10]$, $J(u) = 40.091$, iterations $[129]$
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

3D view of search path

Zooming to show oscillation are starting when getting close to optimal point

Fixed step h = 0.1 results

Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.10]$, $J(u) = 40.091$, iterations $[117]$
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Rapid improvement when far away

Convergence slows own near optimal

Showing $J(u^k)$ progress. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[0.10]$, $J(u) = 40.091$, iterations $[129]$
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Rapid improvement when far away

Convergence slows own near optimal

Objective function getting smaller during search

Gradient norm approaching convergence limit

Figure 4.5: Result for using step 0.1 starting from (8,12)

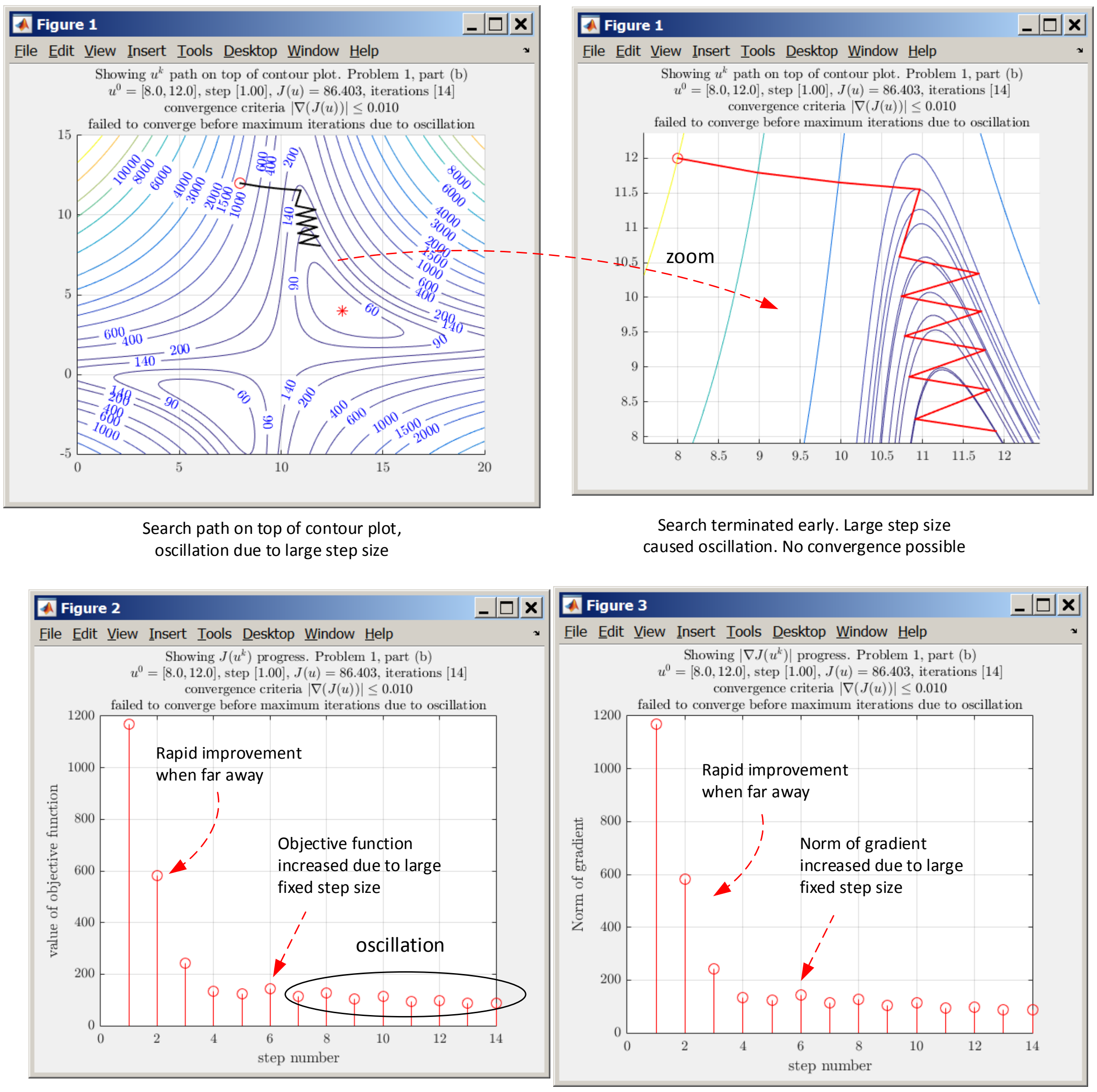


Figure 1

Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[1.00]$, $J(u) = 86.403$, iterations [14]
convergence criteria $|\nabla(J(u))| \leq 0.010$
failed to converge before maximum iterations due to oscillation

Figure 2

Showing $J(u^k)$ progress. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[1.00]$, $J(u) = 86.403$, iterations [14]
convergence criteria $|\nabla(J(u))| \leq 0.010$
failed to converge before maximum iterations due to oscillation

Figure 3

Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)
$u^0 = [8.0, 12.0]$, step $[1.00]$, $J(u) = 86.403$, iterations [14]
convergence criteria $|\nabla(J(u))| \leq 0.010$
failed to converge before maximum iterations due to oscillation

File Edit View Insert Tools Desktop Window Help
zoom
Rapid improvement when far away
Objective function increased due to large fixed step size
oscillation
Norm of gradient increased due to large fixed step size
value of objective function
Norm of gradient
step number

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

Table 4.2: Starting point [5,10]

h	steps to converge	comments

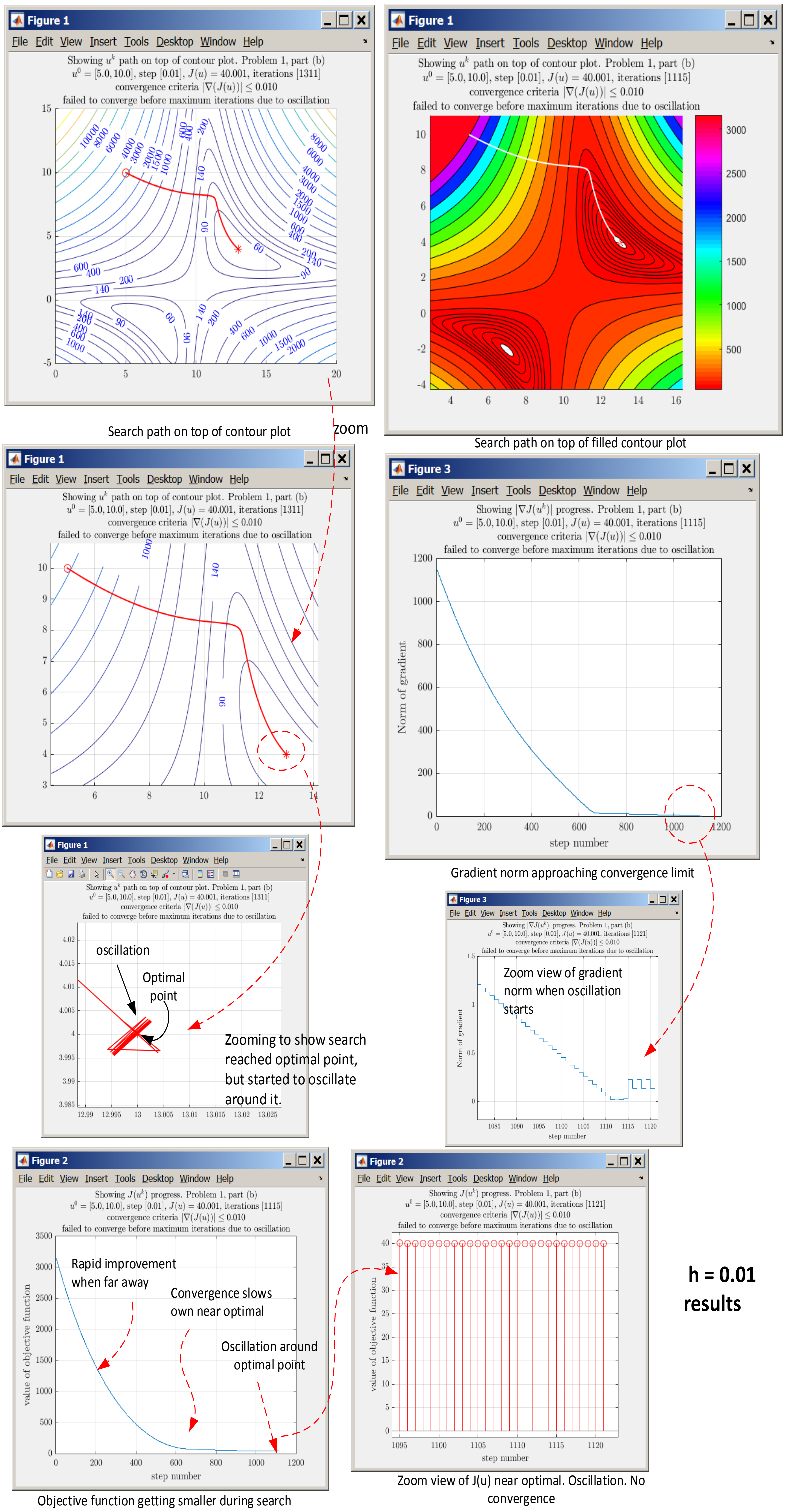
0.01	1311	Search reached optimal point (13, 4), but skipped over it and started to oscillate back and forth around the optimal point due to using large fixed step size. Below are the last few values of $J(u)$ recorded showing this.
		`K>> levelSets(end-10:end)`
		`.....`
		`40.000714475783`
		`40.0002484312073`
		`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`

Continued on next page

Table 4.2 – 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 $\lvert\nabla(J(u))\rvert$
		`K>> gradientNormTol(end-6:end)`
		`....`
		`16.1440020280613`
		`17.487837406306`
		`16.092991548592`
		`17.4442963174089`



Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [5.0, 10.0]$, step [0.01], $J(u) = 40.001$, iterations [1311]
convergence criteria $|\nabla(J(u))| \leq 0.010$
failed to converge before maximum iterations due to oscillation

Search path on top of contour plot

zoom

Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [5.0, 10.0]$, step [0.01], $J(u) = 40.001$, iterations [1115]
convergence criteria $|\nabla(J(u))| \leq 0.010$
failed to converge before maximum iterations due to oscillation

Search path on top of filled contour plot

Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)
$u^0 = [5.0, 10.0]$, step [0.01], $J(u) = 40.001$, iterations [1115]
convergence criteria $|\nabla(J(u))| \leq 0.010$
failed to converge before maximum iterations due to oscillation

Gradient norm approaching convergence limit

oscillation

Optimal point

Zooming to show search reached optimal point, but started to oscillate around it.

Zoom view of gradient norm when oscillation starts

Showing $J(u^k)$ progress. Problem 1, part (b)
$u^0 = [5.0, 10.0]$, step [0.01], $J(u) = 40.001$, iterations [1115]
convergence criteria $|\nabla(J(u))| \leq 0.010$
failed to converge before maximum iterations due to oscillation

Rapid improvement when far away

Convergence slows own near optimal

Oscillation around optimal point

Objective function getting smaller during search

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

h = 0.01 results

Figure 1
Figure 3
Figure 2
File Edit View Insert Tools Desktop Window Help
Norm of gradient
step number
value of objective function

Figure 4.7: Result for using step 0.01 starting from (5,10)

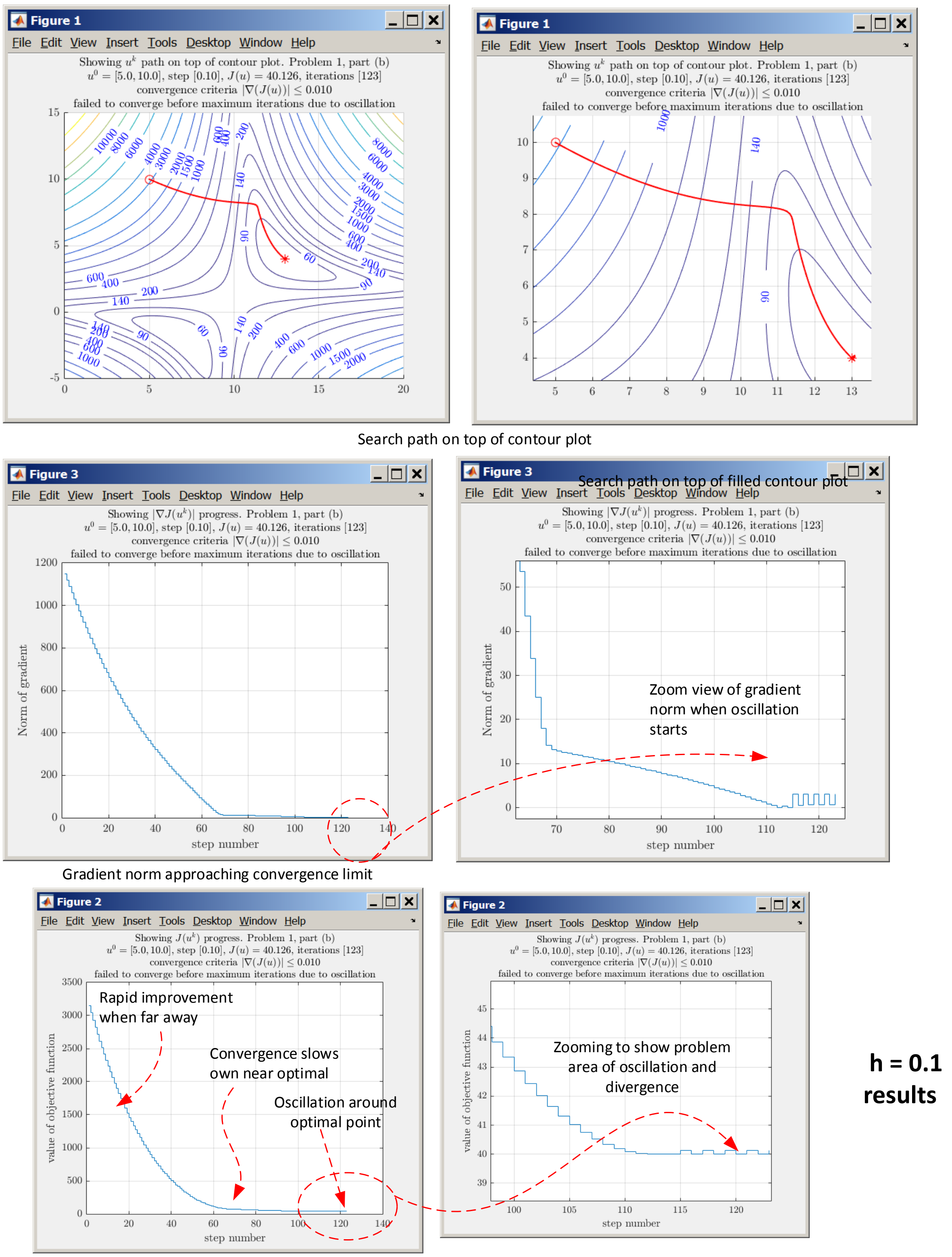


Figure 1
File Edit View Insert Tools Desktop Window Help
Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [5.0, 10.0]$, step $[0.10]$, $J(u) = 40.126$, iterations $[123]$
convergence criteria $|\nabla(J(u))| \leq 0.010$
failed to converge before maximum iterations due to oscillation
Search path on top of filled contour plot
Figure 3
Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)
Norm of gradient
step number
Zoom view of gradient norm when oscillation starts
Figure 2
Showing $J(u^k)$ progress. Problem 1, part (b)
value of objective function
Rapid improvement when far away
Convergence slows own near optimal
Oscillation around optimal point
Zooming to show problem area of oscillation and divergence
h = 0.1 results

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

Figure 4.8: Result for using step 0.1 starting from (5,10)

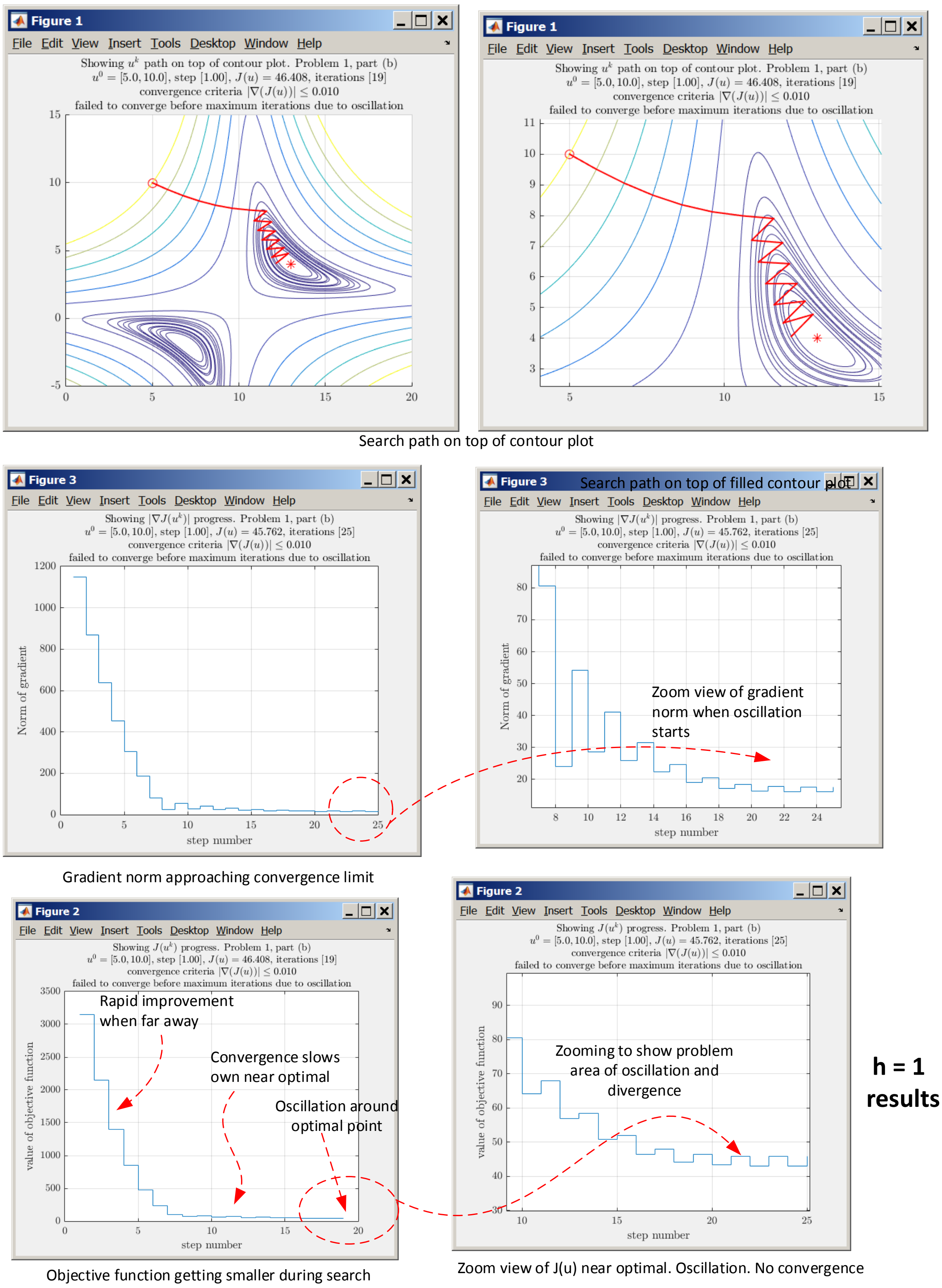


Figure 1

Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [5.0, 10.0]$, step [1.00], $J(u) = 46.408$, iterations [19]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Figure 1

Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [5.0, 10.0]$, step [1.00], $J(u) = 46.408$, iterations [19]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Search path on top of contour plot

Figure 3

Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)
$u^0 = [5.0, 10.0]$, step [1.00], $J(u) = 45.762$, iterations [25]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Gradient norm approaching convergence limit

Figure 3 Search path on top of filled contour plot

Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)
$u^0 = [5.0, 10.0]$, step [1.00], $J(u) = 45.762$, iterations [25]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Zoom view of gradient norm when oscillation starts

Figure 2

Showing $J(u^k)$ progress. Problem 1, part (b)
$u^0 = [5.0, 10.0]$, step [1.00], $J(u) = 46.408$, iterations [19]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Rapid improvement when far away

Convergence slows own near optimal

Oscillation around optimal point

Objective function getting smaller during search

Figure 2

Showing $J(u^k)$ progress. Problem 1, part (b)
$u^0 = [5.0, 10.0]$, step [1.00], $J(u) = 45.762$, iterations [25]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Zooming to show problem area of oscillation and divergence

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

h = 1 results

Figure 4.9: Result for using step 1 starting from (5,10)

Table 4.3: 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`

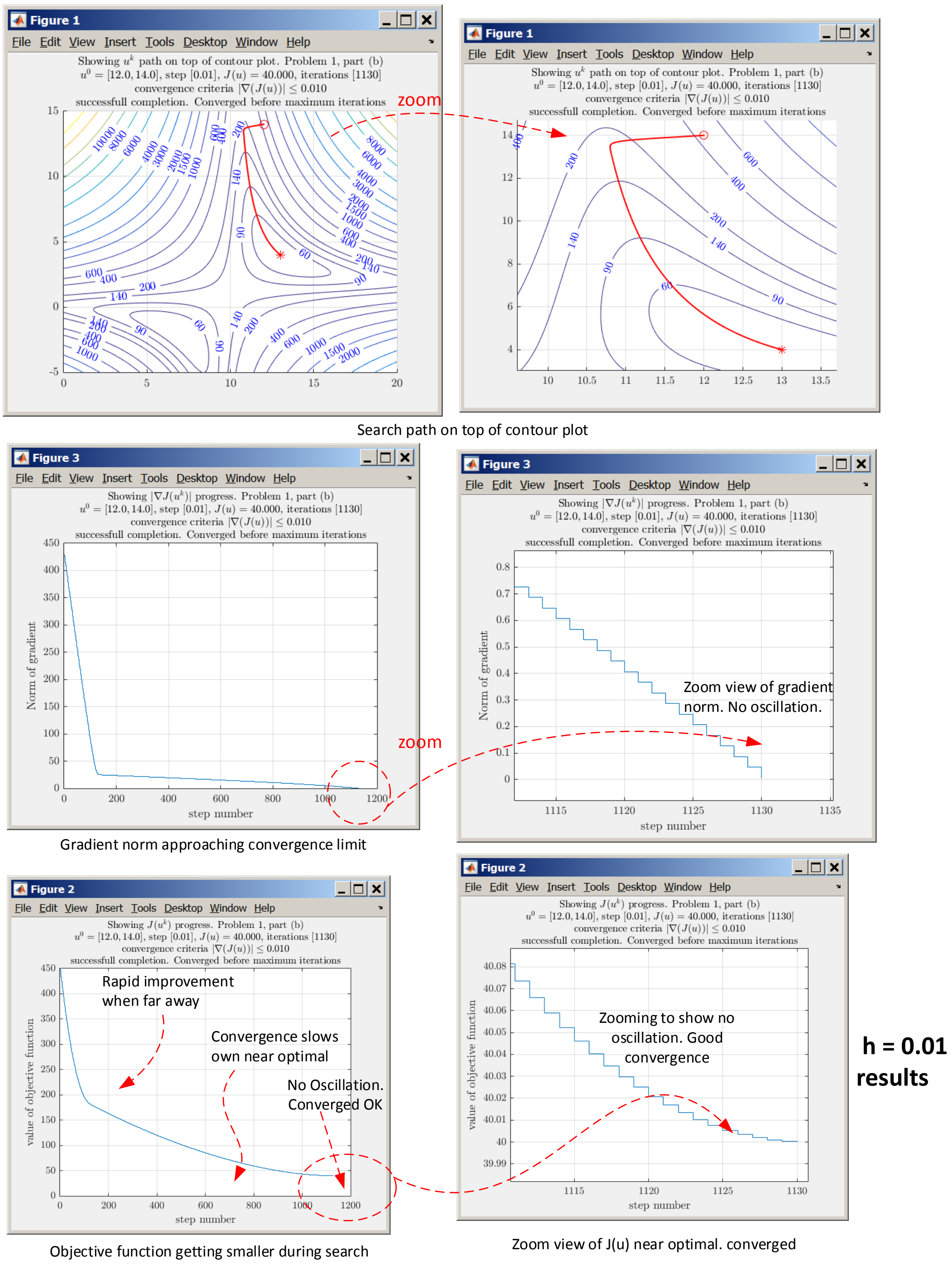


Figure 1

Showing u^k path on top of contour plot. Problem 1, part (b)

$u^0 = [12.0, 14.0]$, step $[0.01]$, $J(u) = 40.000$, iterations [1130]

convergence criteria $|\nabla(J(u))| \le 0.010$

successfull completion. Converged before maximum iterations

zoom

Search path on top of contour plot

Figure 3

Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)

$u^0 = [12.0, 14.0]$, step $[0.01]$, $J(u) = 40.000$, iterations [1130]

convergence criteria $|\nabla(J(u))| \le 0.010$

successfull completion. Converged before maximum iterations

zoom

Zoom view of gradient norm. No oscillation.

Gradient norm approaching convergence limit

Figure 2

Showing $J(u^k)$ progress. Problem 1, part (b)

$u^0 = [12.0, 14.0]$, step $[0.01]$, $J(u) = 40.000$, iterations [1130]

convergence criteria $|\nabla(J(u))| \le 0.010$

successfull completion. Converged before maximum iterations

Rapid improvement when far away

Convergence slows own near optimal

No Oscillation. Converged OK

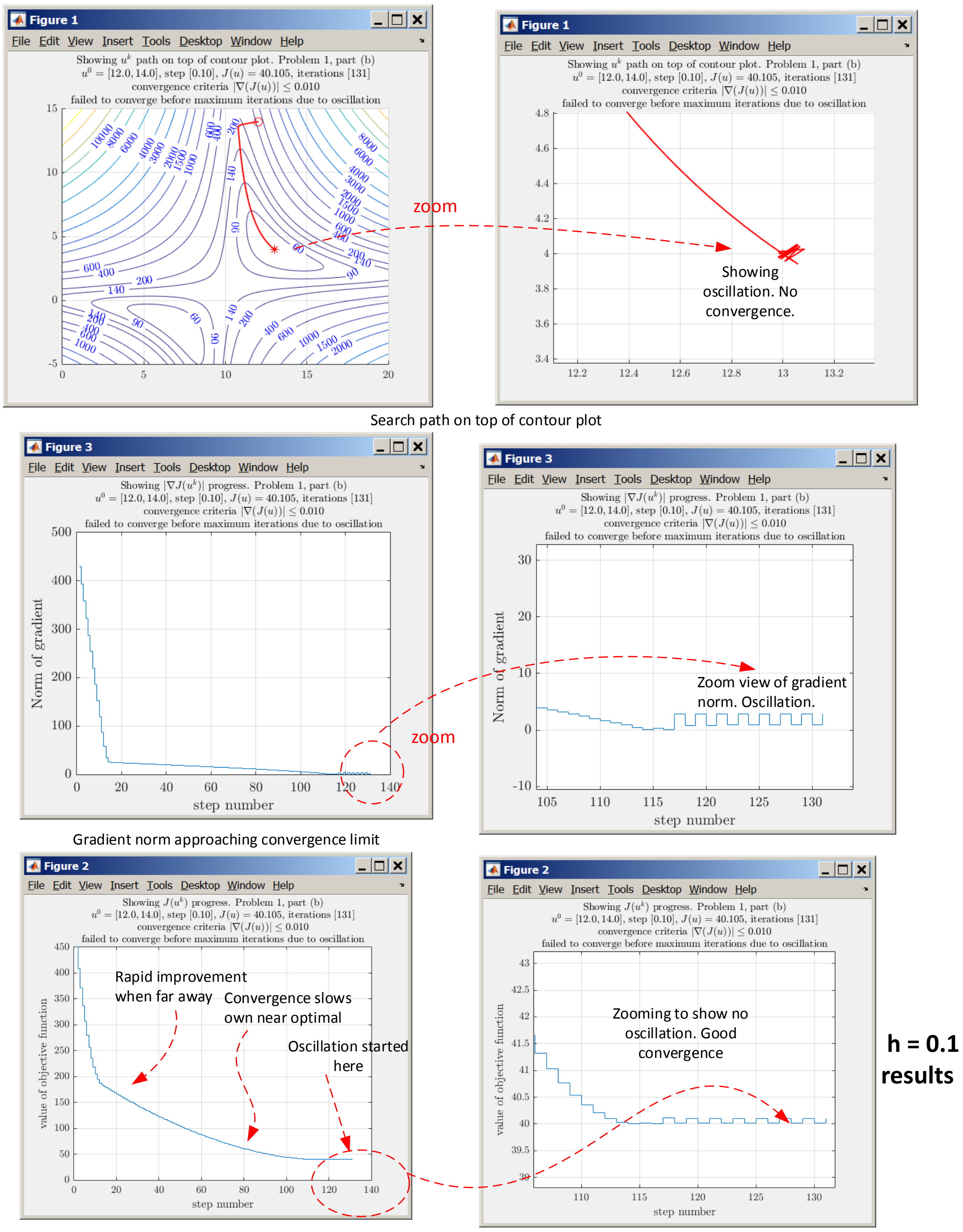
Objective function getting smaller during search

Zooming to show no oscillation. Good convergence

Zoom view of J(u) near optimal. converged

h = 0.01 results

Figure 4.10: Result for using step 0.01 starting from (12, 14)



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 detected

Figure 4.11: Result for using step 0.1 starting from (12, 14)

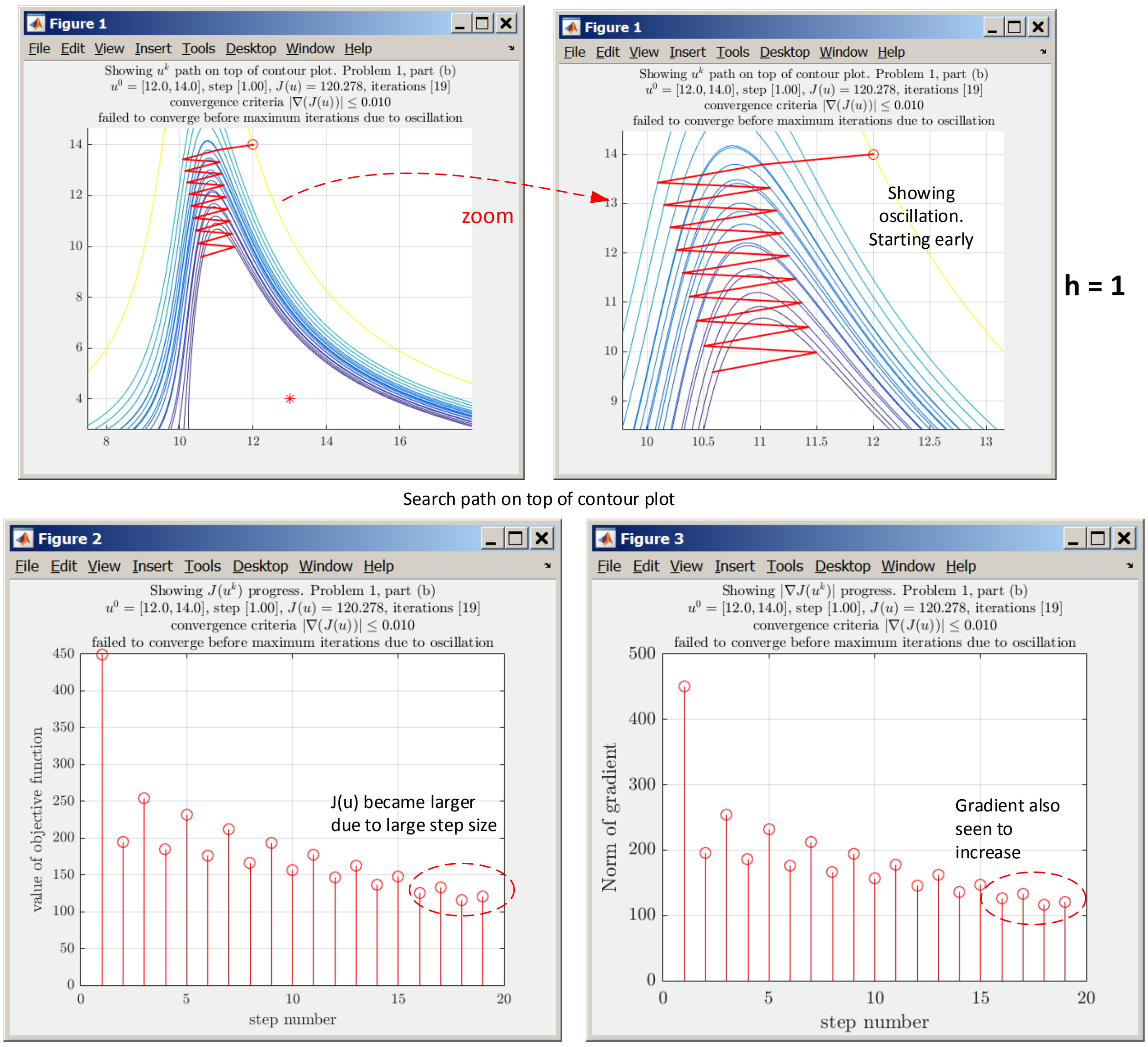


Figure 1

Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [12.0, 14.0]$, step $[1.00]$, $J(u) = 120.278$, iterations $[19]$
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Showing oscillation. Starting early

h = 1

Search path on top of contour plot

Figure 2

Showing $J(u^k)$ progress. Problem 1, part (b)
$u^0 = [12.0, 14.0]$, step $[1.00]$, $J(u) = 120.278$, iterations $[19]$
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

J(u) became larger due to large step size

Objective function. Oscillation detected

Figure 3

Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)
$u^0 = [12.0, 14.0]$, step $[1.00]$, $J(u) = 120.278$, iterations $[19]$
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Gradient also seen to increase

Norm of gradient.

Figure 4.12: Result for using step 1 starting from (12,14)

Table 4.4: 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 $\lvert\nabla(J(u))\rvert$
		`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 $\lvert\nabla(J(u))\rvert$ 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 $\lvert\nabla(J(u))\rvert$ showing it is diverging.
		`K>> gradientNormTol(end-6:end)`
		`....`
		`16.7542019931462`
		`17.5230111072761`
		`16.7540613766743`
		`17.5229596031784`
		`16.7540287643191`

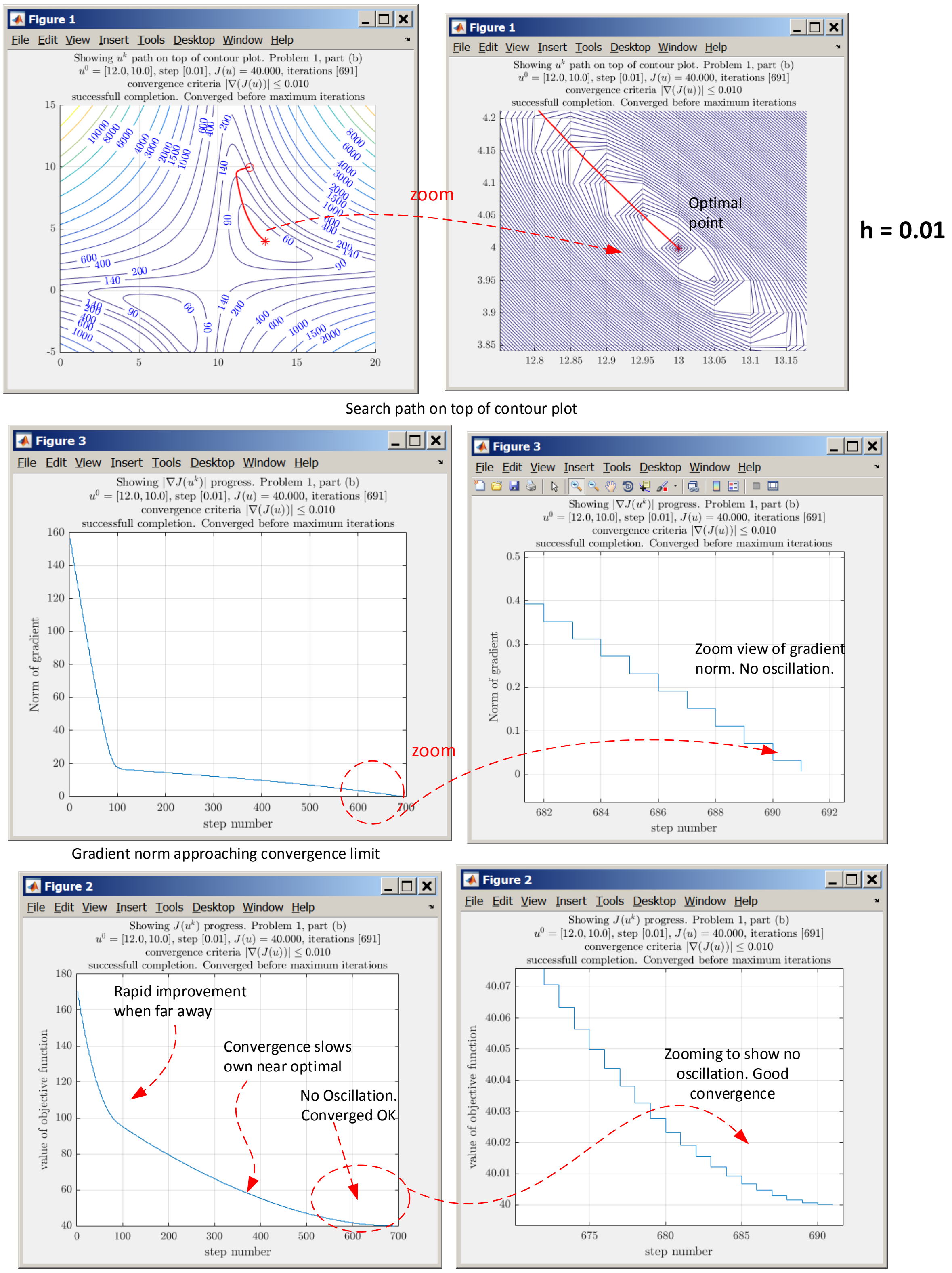


Figure 1
File Edit View Insert Tools Desktop Window Help
Showing u^k path on top of contour plot. Problem 1, part (b)
u^0 = [12.0, 10.0], step [0.01], $J(u)$ = 40.000, iterations [691]
convergence criteria $|\nabla(J(u))| \leq 0.010$
successfull completion. Converged before maximum iterations
Figure 3
Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)
Norm of gradient
step number
Figure 2
Showing $J(u^k)$ progress. Problem 1, part (b)
value of objective function
step number

zoom

Optimal point

h = 0.01

Search path on top of contour plot

zoom

Zoom view of gradient norm. No oscillation.

Gradient norm approaching convergence limit

Rapid improvement when far away

Convergence slows own near optimal

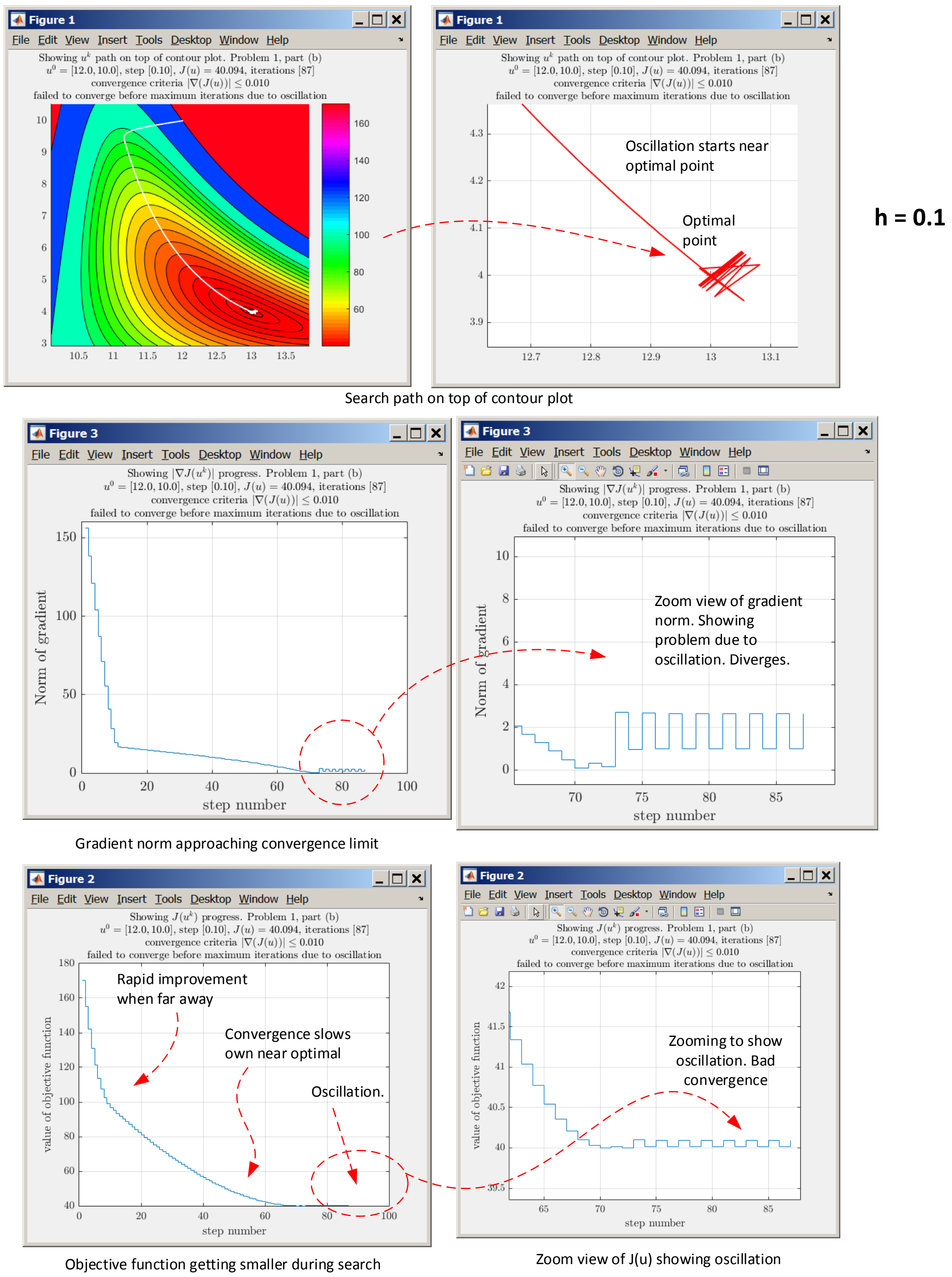
No Oscillation. Converged OK

Zooming to show no oscillation. Good convergence

Objective function getting smaller during search

Zoom view of J(u) near optimal. converged

Figure 4.13: Result for using step 0.01 starting from (12, 10)



Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [12.0, 10.0]$, step $[0.10]$, $J(u) = 40.094$, iterations [87]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Oscillation starts near optimal point

Optimal point

h = 0.1

Search path on top of contour plot

Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)
$u^0 = [12.0, 10.0]$, step $[0.10]$, $J(u) = 40.094$, iterations [87]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Zoom view of gradient norm. Showing problem due to oscillation. Diverges.

Gradient norm approaching convergence limit

Showing $J(u^k)$ progress. Problem 1, part (b)
$u^0 = [12.0, 10.0]$, step $[0.10]$, $J(u) = 40.094$, iterations [87]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Rapid improvement when far away

Convergence slows own near optimal

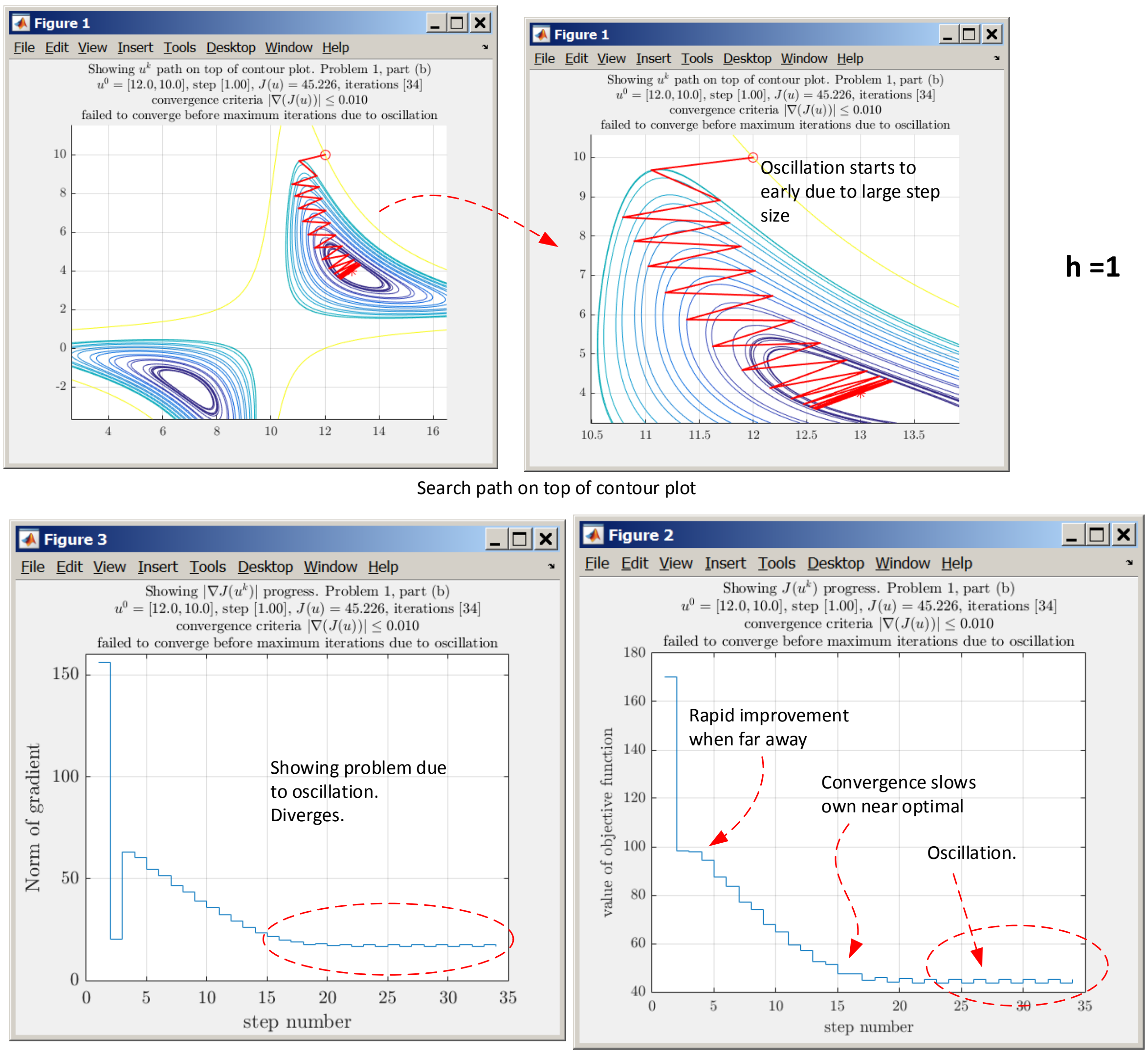
Oscillation.

Zooming to show oscillation. Bad convergence

Objective function getting smaller during search

Zoom view of J(u) showing oscillation

Figure 4.14: Result for using step 0.1 starting from (12,10)



Showing u^k path on top of contour plot. Problem 1, part (b)
$u^0 = [12.0, 10.0]$, step [1.00], $J(u) = 45.226$, iterations [34]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Oscillation starts to early due to large step size

h =1

Search path on top of contour plot

Showing $|\nabla J(u^k)|$ progress. Problem 1, part (b)
$u^0 = [12.0, 10.0]$, step [1.00], $J(u) = 45.226$, iterations [34]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Showing problem due to oscillation. Diverges.

Gradient norm showing divergence

Showing $J(u^k)$ progress. Problem 1, part (b)
$u^0 = [12.0, 10.0]$, step [1.00], $J(u) = 45.226$, iterations [34]
convergence criteria $|\nabla(J(u))| \le 0.010$
failed to converge before maximum iterations due to oscillation

Rapid improvement when far away

Convergence slows own near optimal

Oscillation.

Objective function getting smaller showing oscillation early on

Figure 4.15: Result for using step 0.1 starting from (12, 10)

4.4.1.4 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

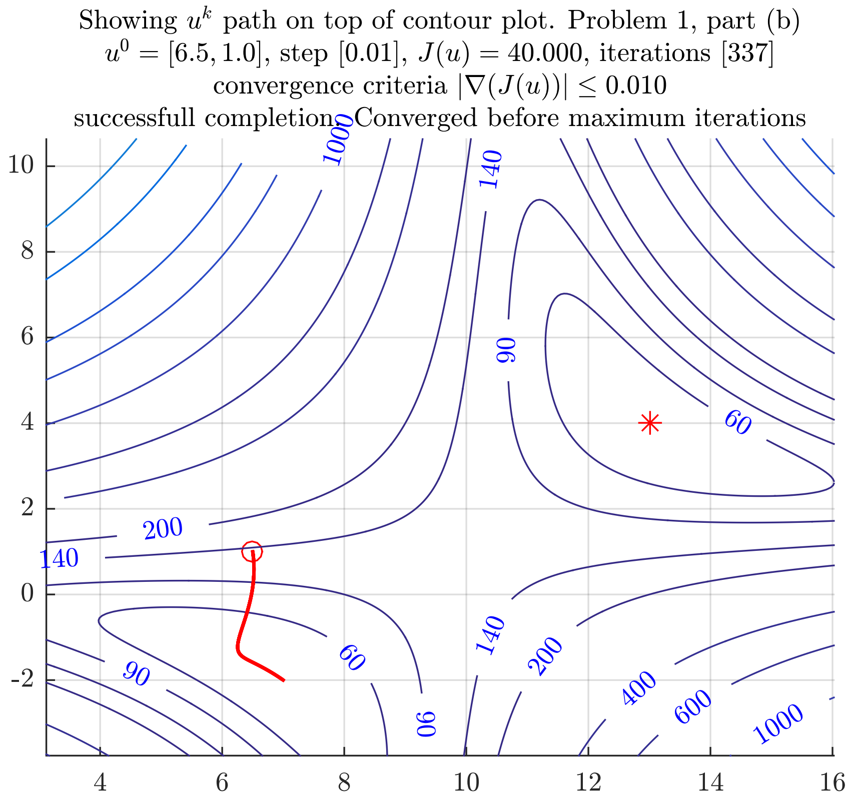


Figure 4.16: Converging to (7, −2) using step size 0.01 starting from (6.5, 1)

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.

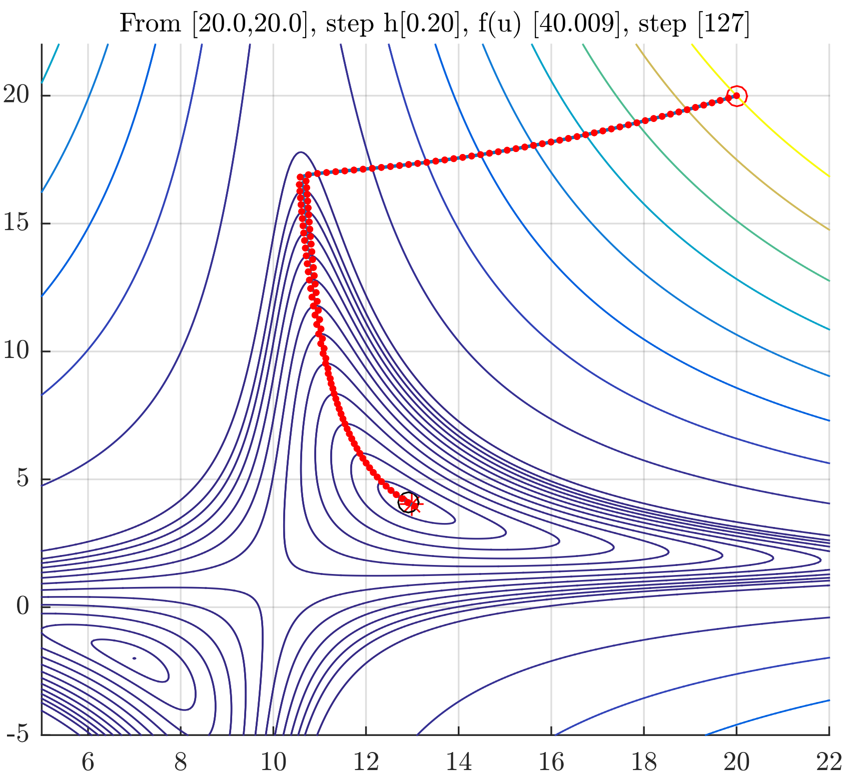
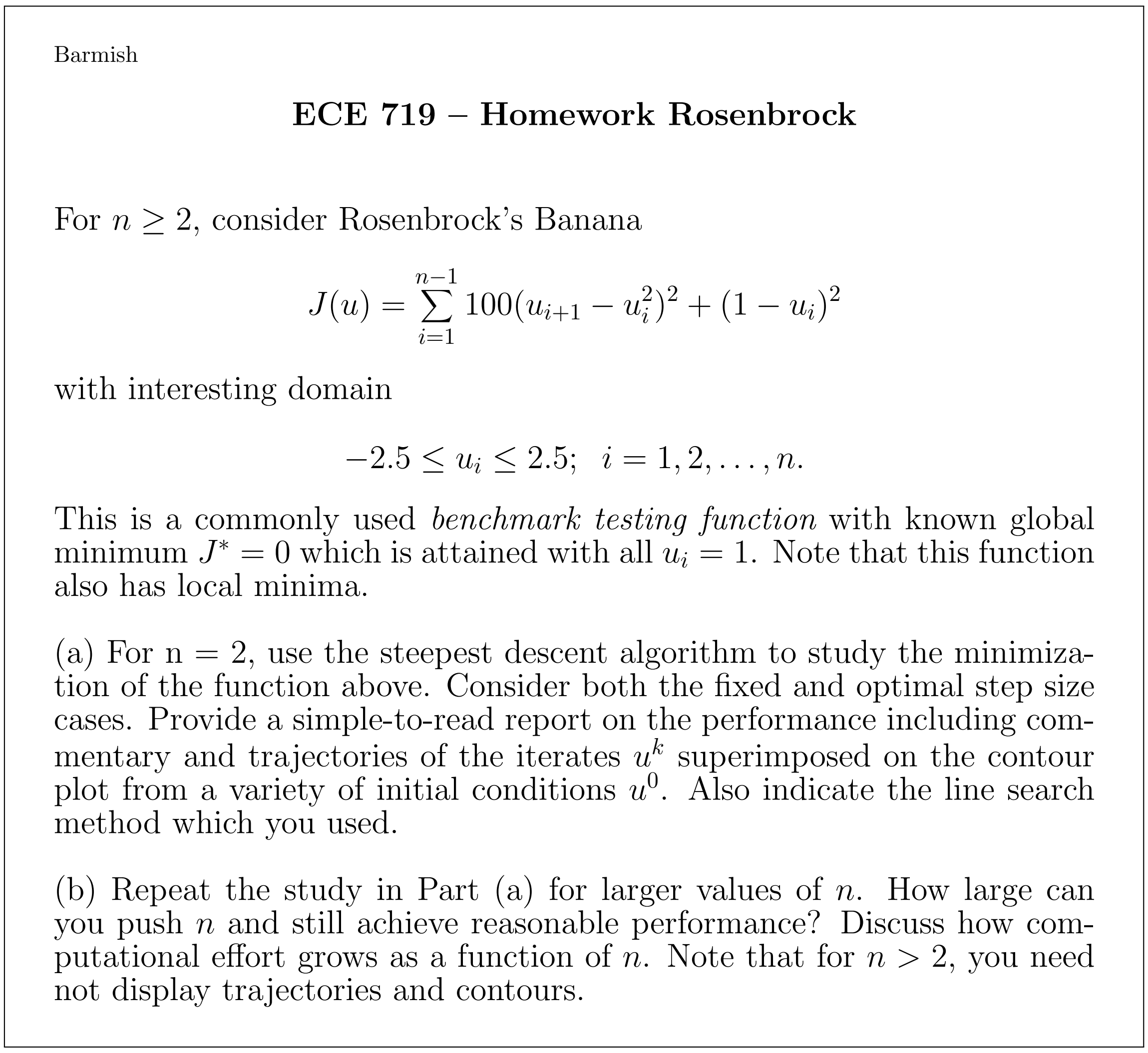


Figure 4.17: Missing $u^* = [7, -2]$ when starting too far it. Starting from $(20, 20)$ using step size 0.01

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.

4.4.2 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 4.18: problem 2 description

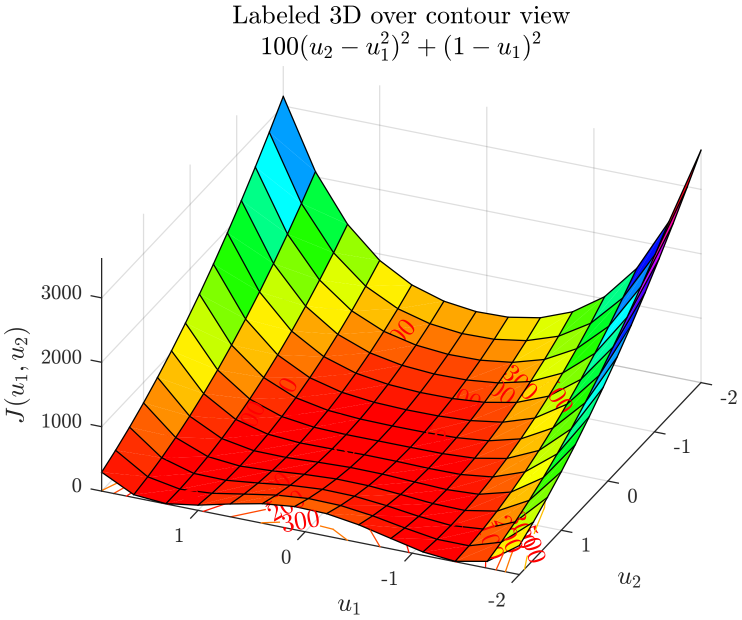


Figure 4.19: 3D view of $J(u)$

4.4.2.1 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
    $H \leftarrow$ maximum step size
    $max\_iterations \leftarrow$ max iterations allowed
    $\epsilon \leftarrow$ minimum convergence limit on $\|\nabla J(u)\|$
    $k \leftarrow 0$
    $u^k \leftarrow u^0$
    while $\|\nabla J(u^k)\| > \epsilon$ do
        ▷ do line search
        $h^* \leftarrow$ call golden_section$(H, J(u))$ to find optimal $h^*$ of function $\tilde{J}(h^*) = J(u^k - h^* \nabla J(u^k))$
        $u^k \leftarrow u^k - h^* \frac{\nabla J(u^k)}{\|\nabla J(u^k)\|}$
        $k \leftarrow k + 1$
        ▷ detect oscillation
        if $k \geq$ max_iterations or $J(u_k) > J(u_{k-1})$ then
            exit loop
        end if
    end while
end procedure
```

Figure 4.20: 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

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

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 $J^* = 0$.

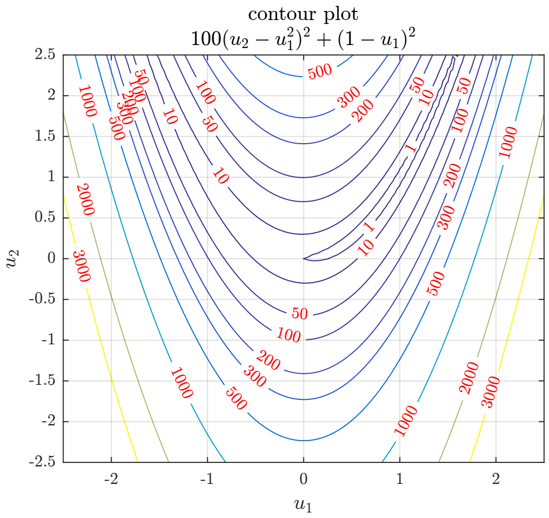


Figure 4.21: Contour $J(u)$

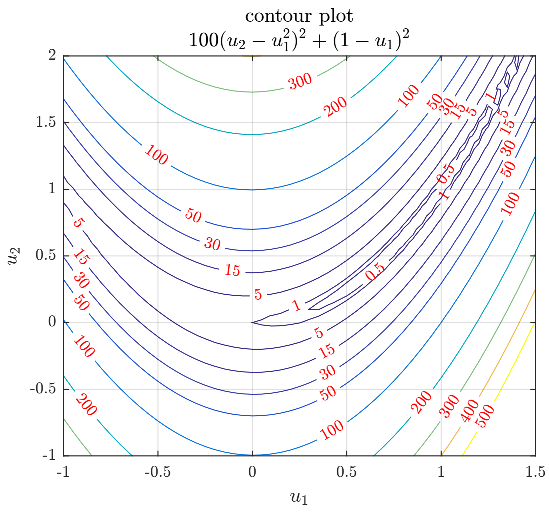


Figure 4.22: Zooming on Contour $J(u)$

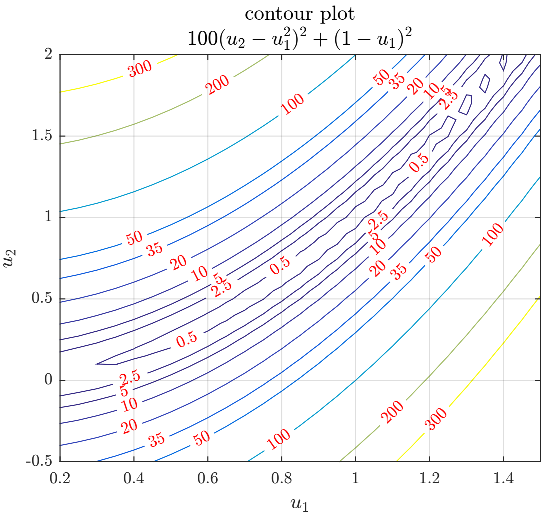


Figure 4.23: More zooming. Contour $J(u)$

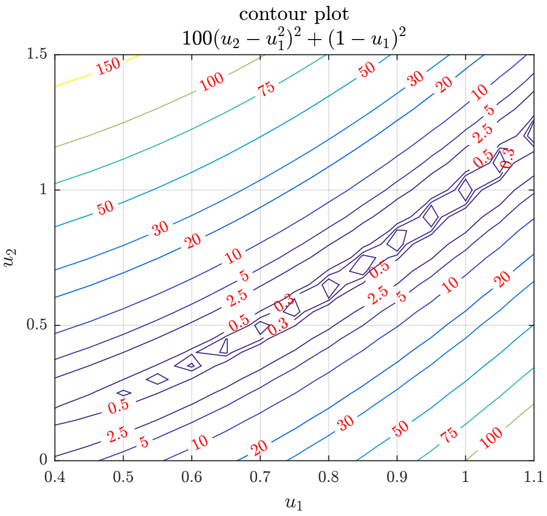


Figure 4.24: More zooming. Contour $J(u)$

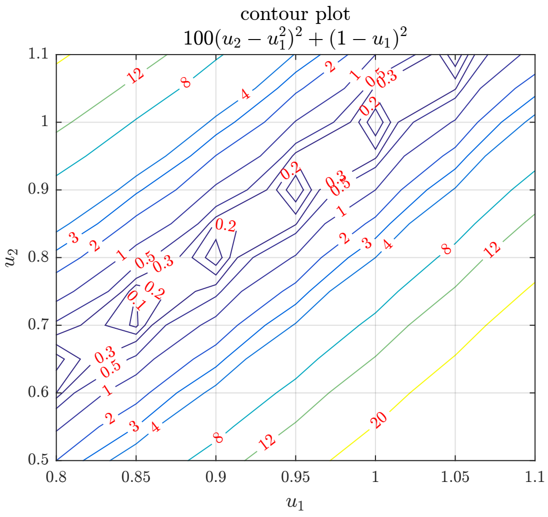


Figure 4.25: More zooming on Contour $J(u)$

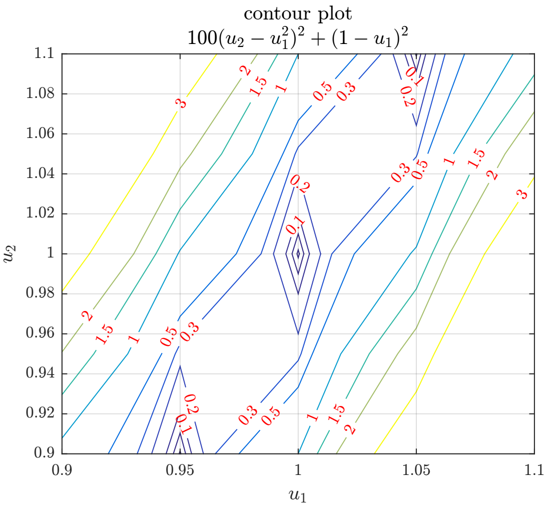


Figure 4.26: More zooming on Contour $J(u)$

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.

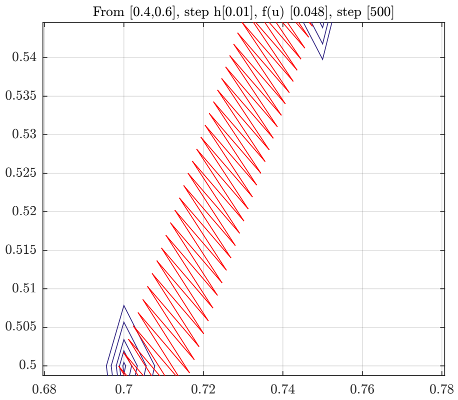


Figure 4.27: Zoom view of search when inside valley, showing the small steps and zig-zag pattern

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.

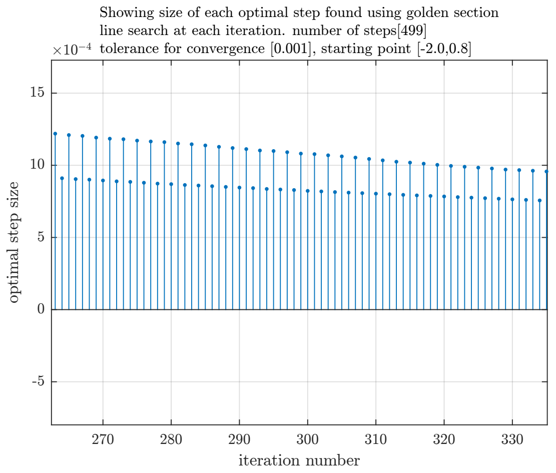


Figure 4.28: Showing how optimal step size changes at each iteration during typical search

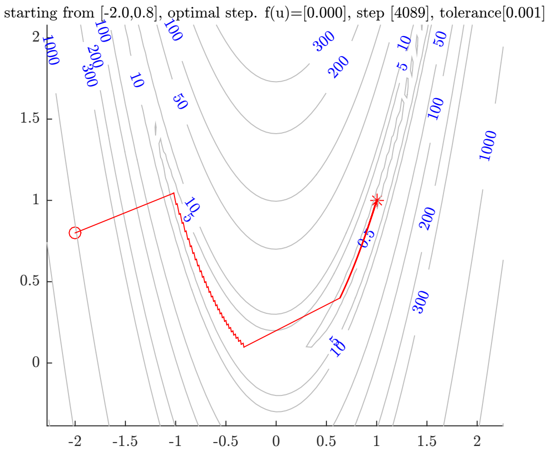


Figure 4.29: Typical search pattern using optimal step size from arbitrary starting point

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 4.5: 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.

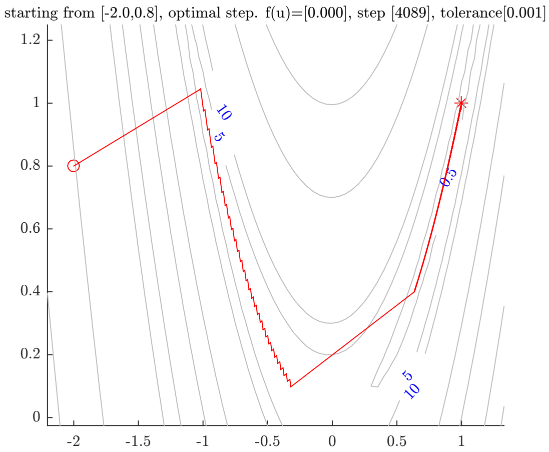


Figure 4.30: path of u^k using optimal step starting from $(-2, 0.8)$

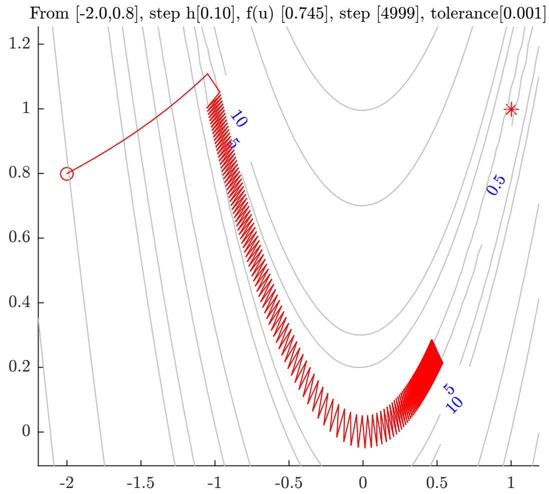


Figure 4.31: path of u^k using fixed step $h = 0.1$ starting from $(-2, 0.8)$

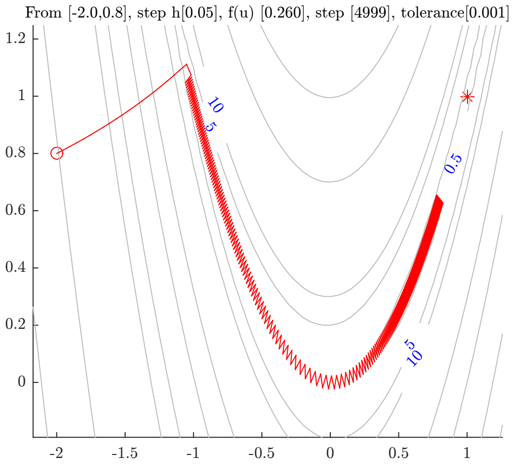


Figure 4.32: u^k path, fixed step $h = 0.05$ from $(-2, 0.8)$

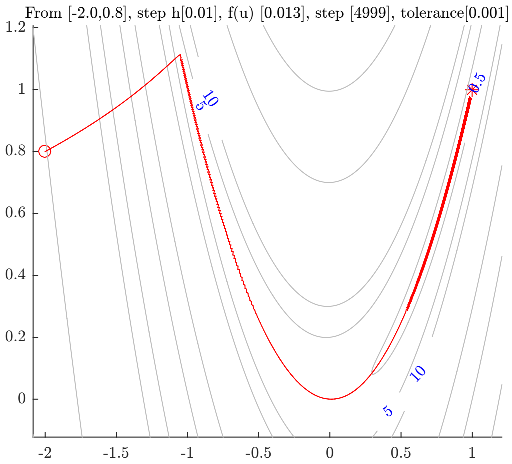


Figure 4.33: u^k path, fixed step $h = 0.01$ from $(-2, 0.8)$

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 4.6: 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.

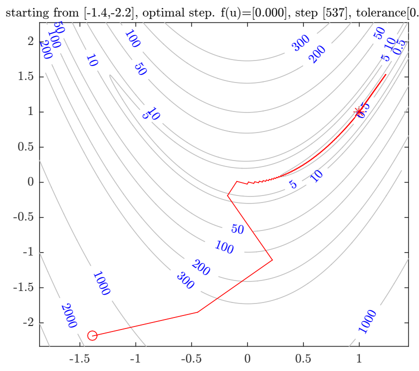


Figure 4.34: u^k path, optimal step from at $(-1.4, -2)$

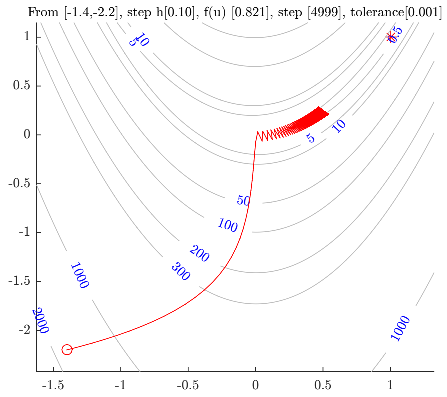


Figure 4.35: u^k path, fixed step $h = 0.1$ from $(-1.4, -2)$

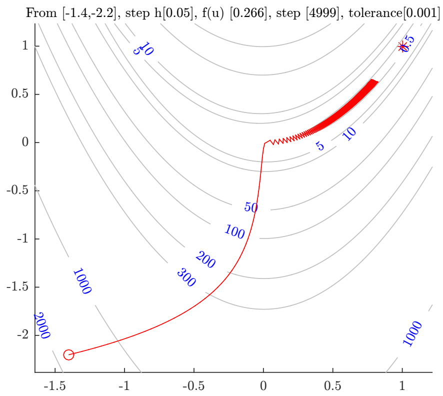


Figure 4.36: u^k path, fixed step $h = 0.05$ from $(-1.4, -2)$

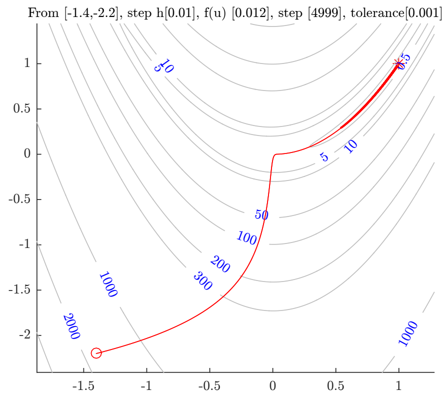


Figure 4.37: u^k path, fixed step $h = 0.01$ from $(-1.4, -2.0)$

4.4.2.2 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
```

.

4.4.2.3 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.

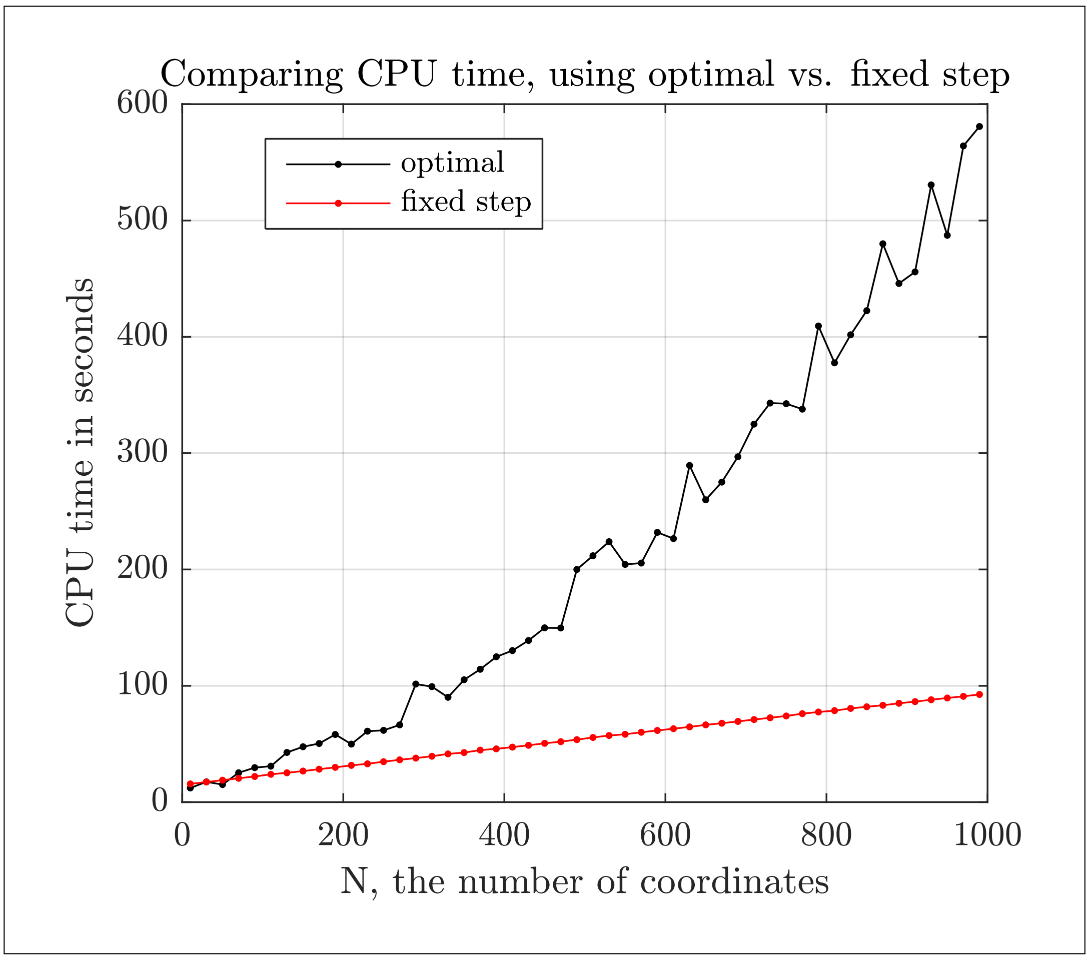


Figure 4.38: Comparing CPU time, optimal step and fixed step

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.

4.4.3 Source code listing

4.4.3.1 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
```

4.4.3.2 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
```

4.4.3.3 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');
```

```matlab

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
```

4.4.3.4 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
```

4.4.3.5 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
```

4.4.3.6 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
```

4.4.3.7 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);

```

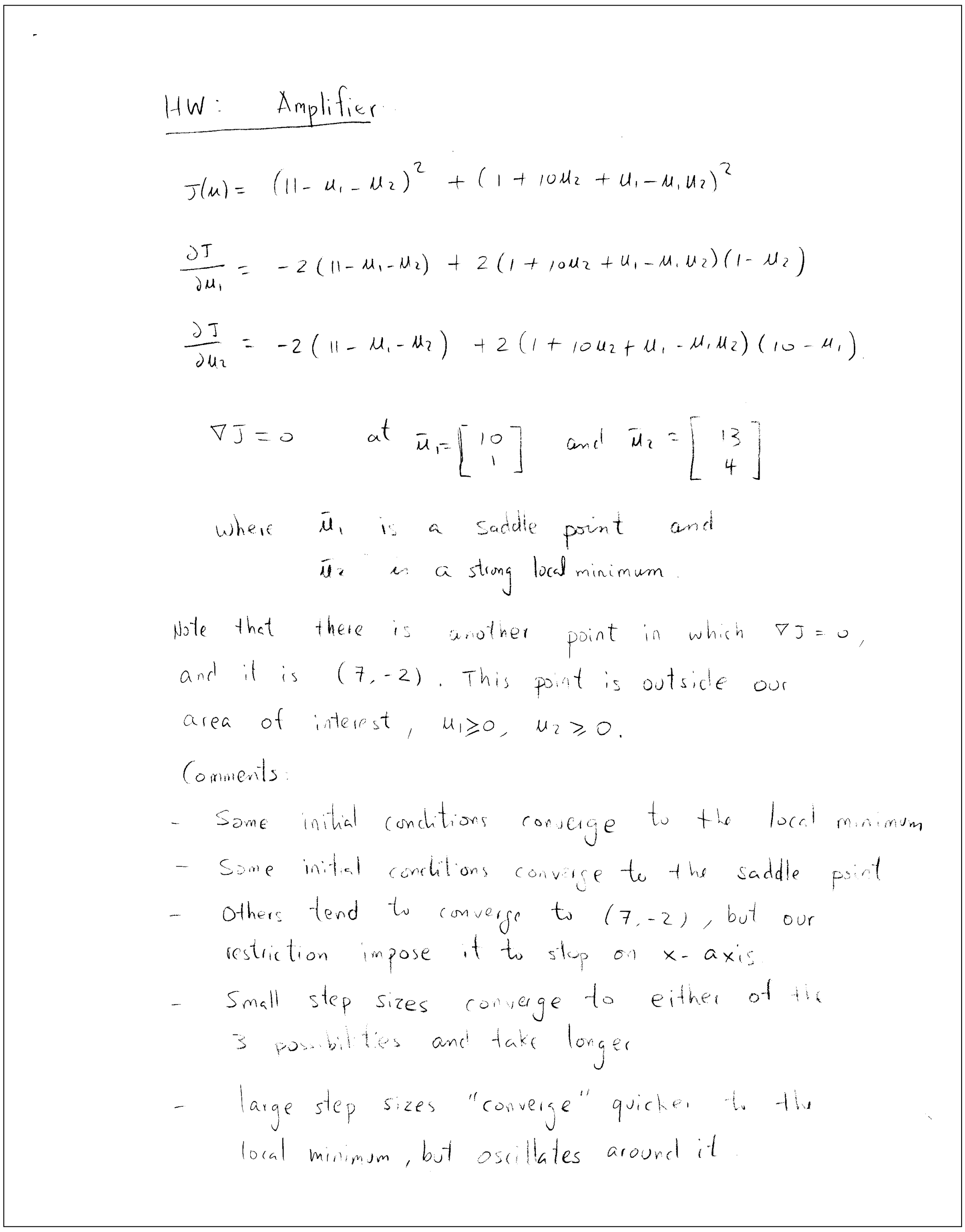
```
format long g;
tic;
[status,pts,levelSets, gradientNormTol,steps] = ...
                                     nma_steepest_descent(opt);
time_used = toc;
fprintf('\nCPU time %3.6f\n',time_used);

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
```

4.4.3.8 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
```

4.4.4 HW 4 key solution



HW: Amplifier

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

$$\frac{\partial J}{\partial u_1} = -2(11 - u_1 - u_2) + 2(1 + 10u_2 + u_1 - u_1 u_2)(1 - u_2)$$

$$\frac{\partial J}{\partial u_2} = -2(11 - u_1 - u_2) + 2(1 + 10u_2 + u_1 - u_1 u_2)(10 - u_1)$$

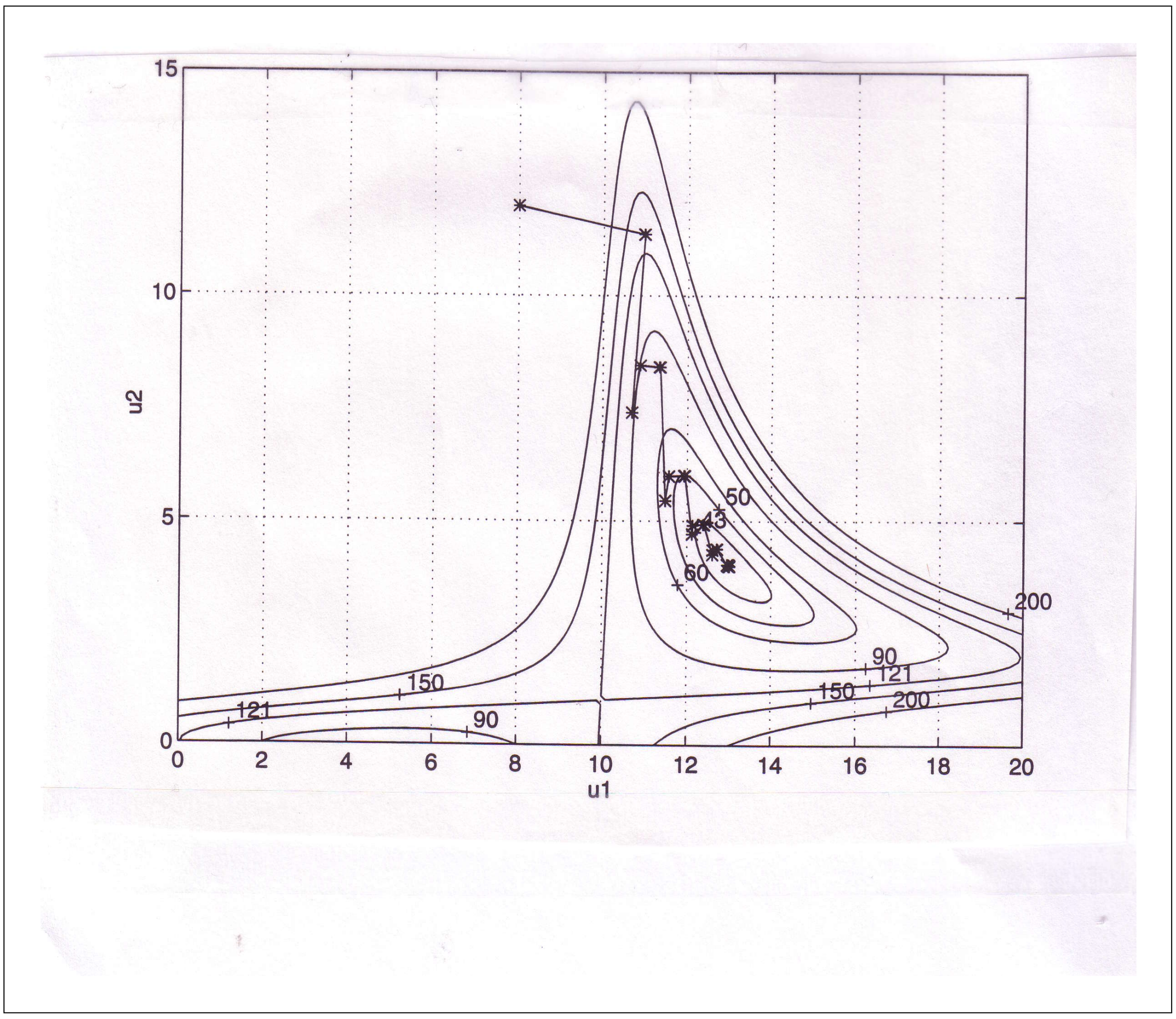
$\nabla J = 0$ at $\bar{u}_1 = \begin{bmatrix} 10 \\ 1 \end{bmatrix}$ and $\bar{u}_2 = \begin{bmatrix} 13 \\ 4 \end{bmatrix}$

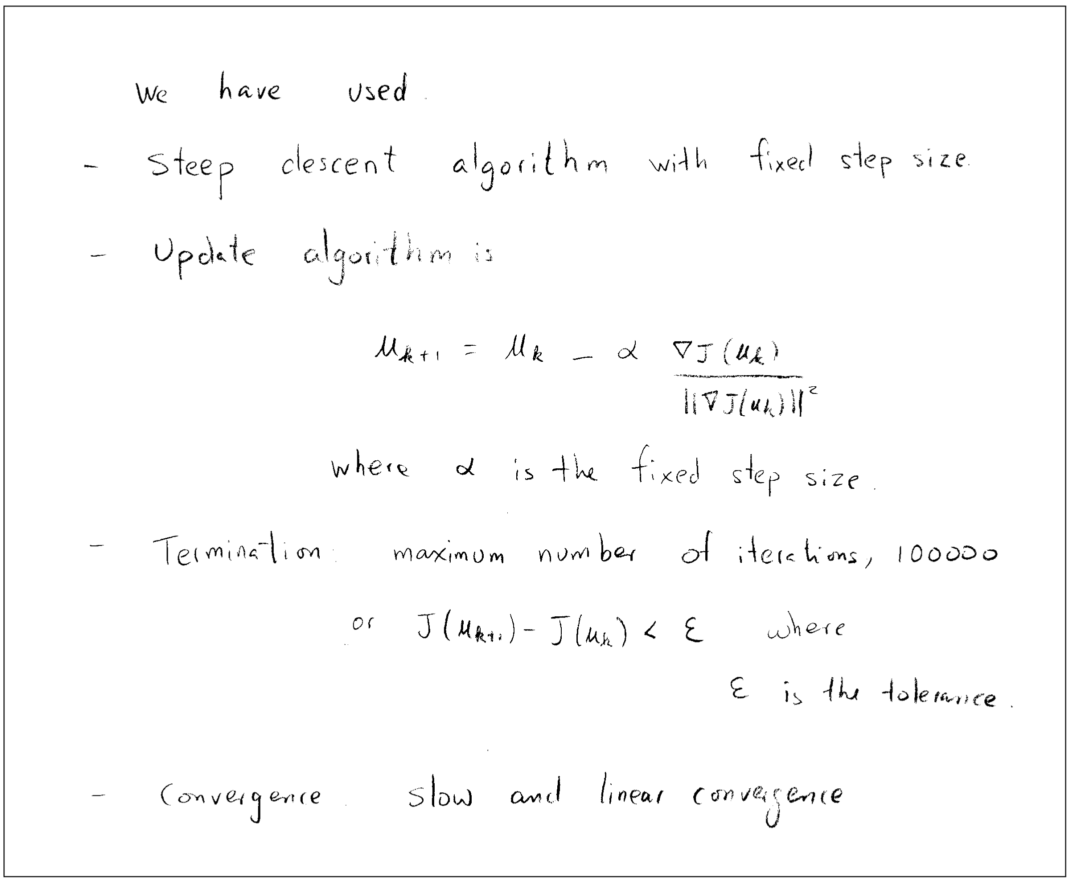
where $\bar{u}_1$ is a saddle point and $\bar{u}_2$ is a strong local minimum.

Note that there is another point in which $\nabla J = 0$, and it is $(7, -2)$. This point is outside our area of interest, $u_1 \geq 0$, $u_2 \geq 0$.

Comments:

- Some initial conditions converge to the local minimum
- Some initial conditions converge to the saddle point
- Others tend to converge to $(7, -2)$, but our restriction impose it to stop on x-axis.
- Small step sizes converge to either of the 3 possibilities and take longer
- large step sizes "converge" quicker to the local minimum, but oscillates around it





We have used.

- Steep descent algorithm with fixed step size.
- Update algorithm is

$$u_{k+1} = u_k - \alpha \frac{\nabla J(u_k)}{\|\nabla J(u_k)\|^2}$$

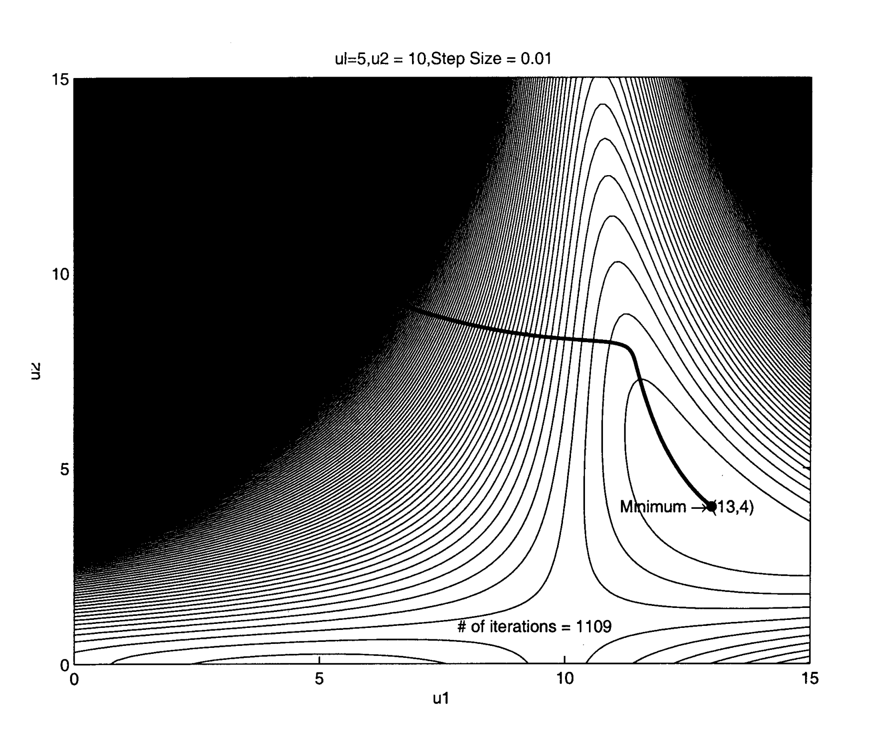
where α is the fixed step size.

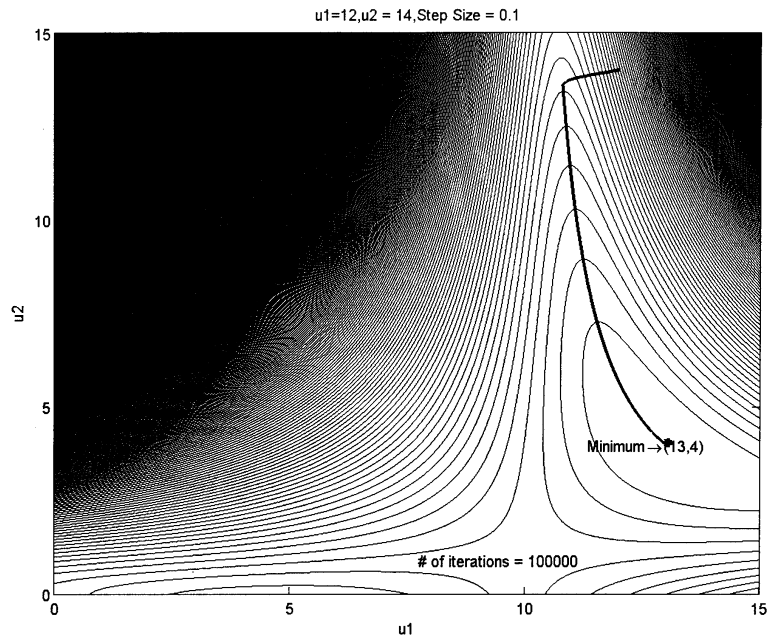
- Termination: maximum number of iterations, 100000

or $J(u_{k+1}) - J(u_k) < \varepsilon$ where

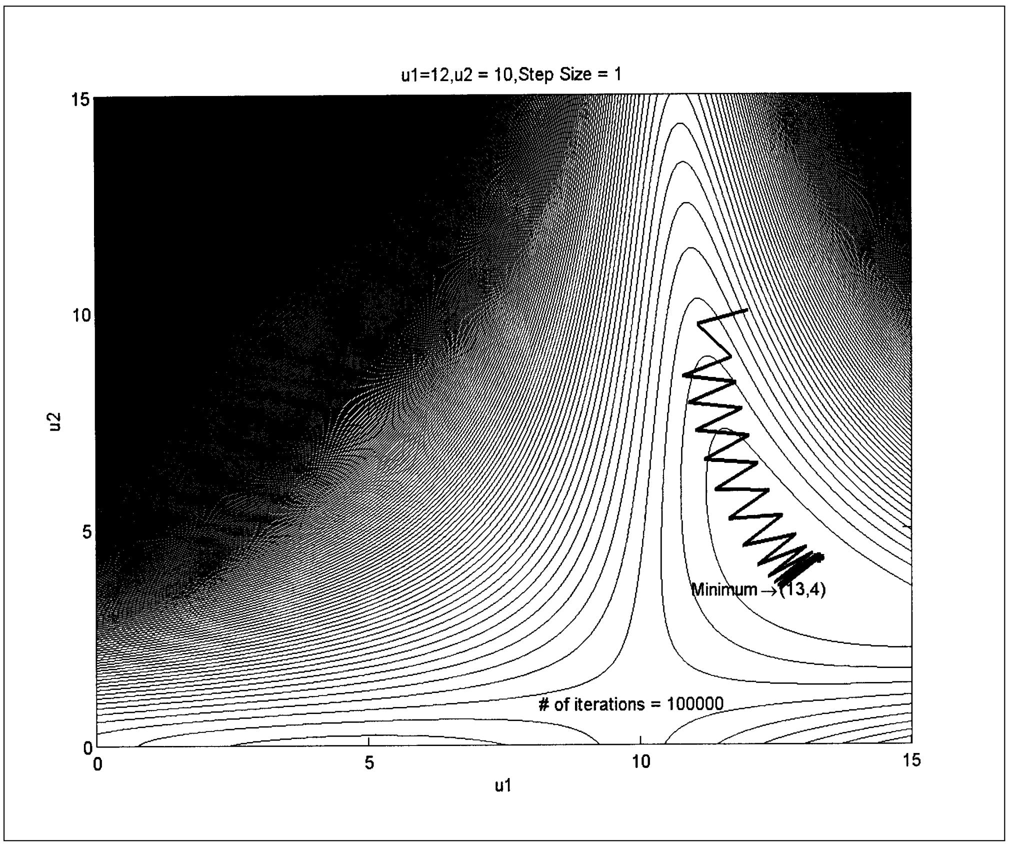
ε is the tolerance.

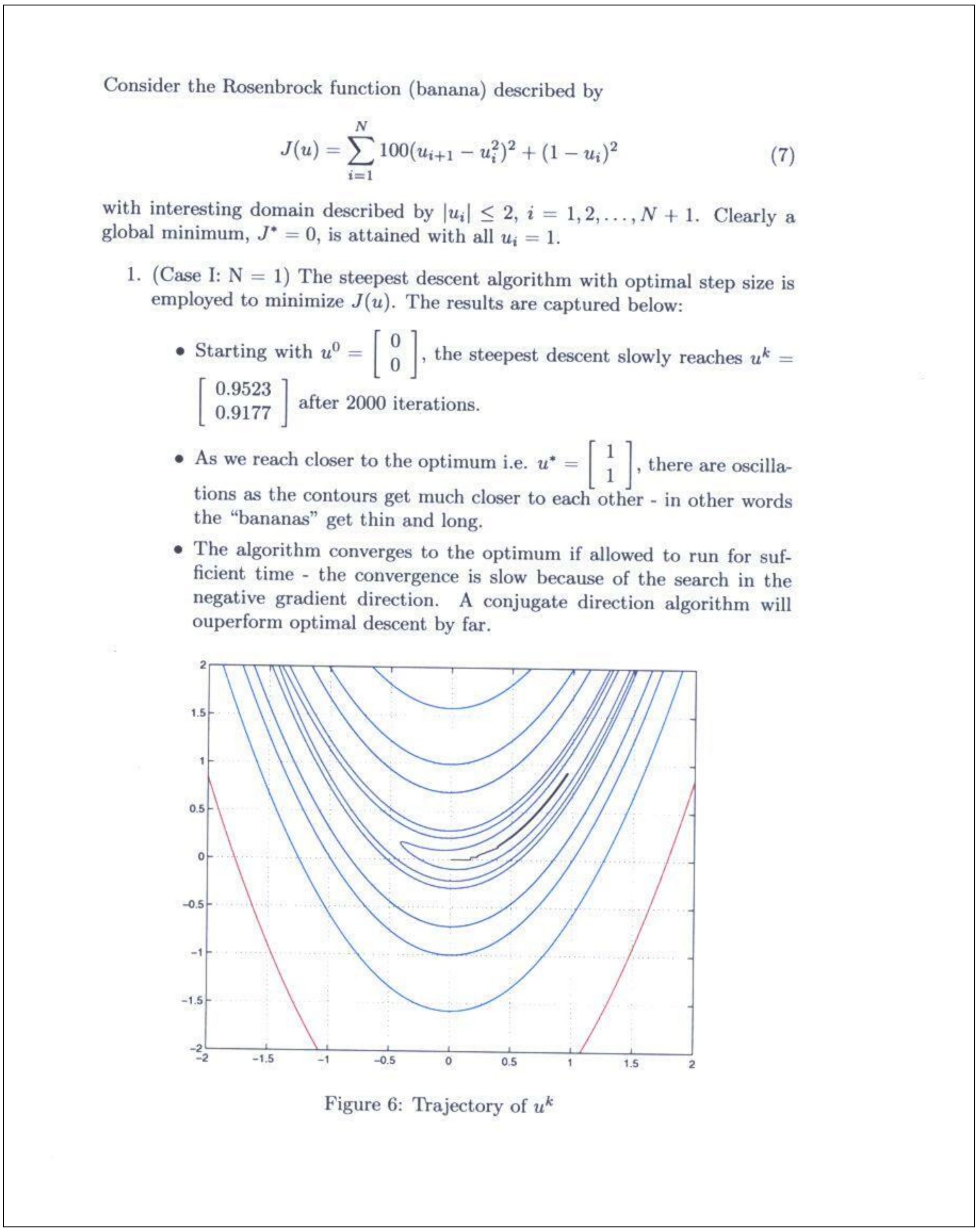
- Convergence: slow and linear convergence





Since the tolerance is smaller than the step size, the algorithm stopped at the maximum number of iterations. It oscillates around the minimum.


u1=12,u2 = 10,Step Size = 1
15
10
5
0
u2
0
5
10
15
u1
Minimum→(13,4)
# of iterations = 100000



Consider the Rosenbrock function (banana) described by

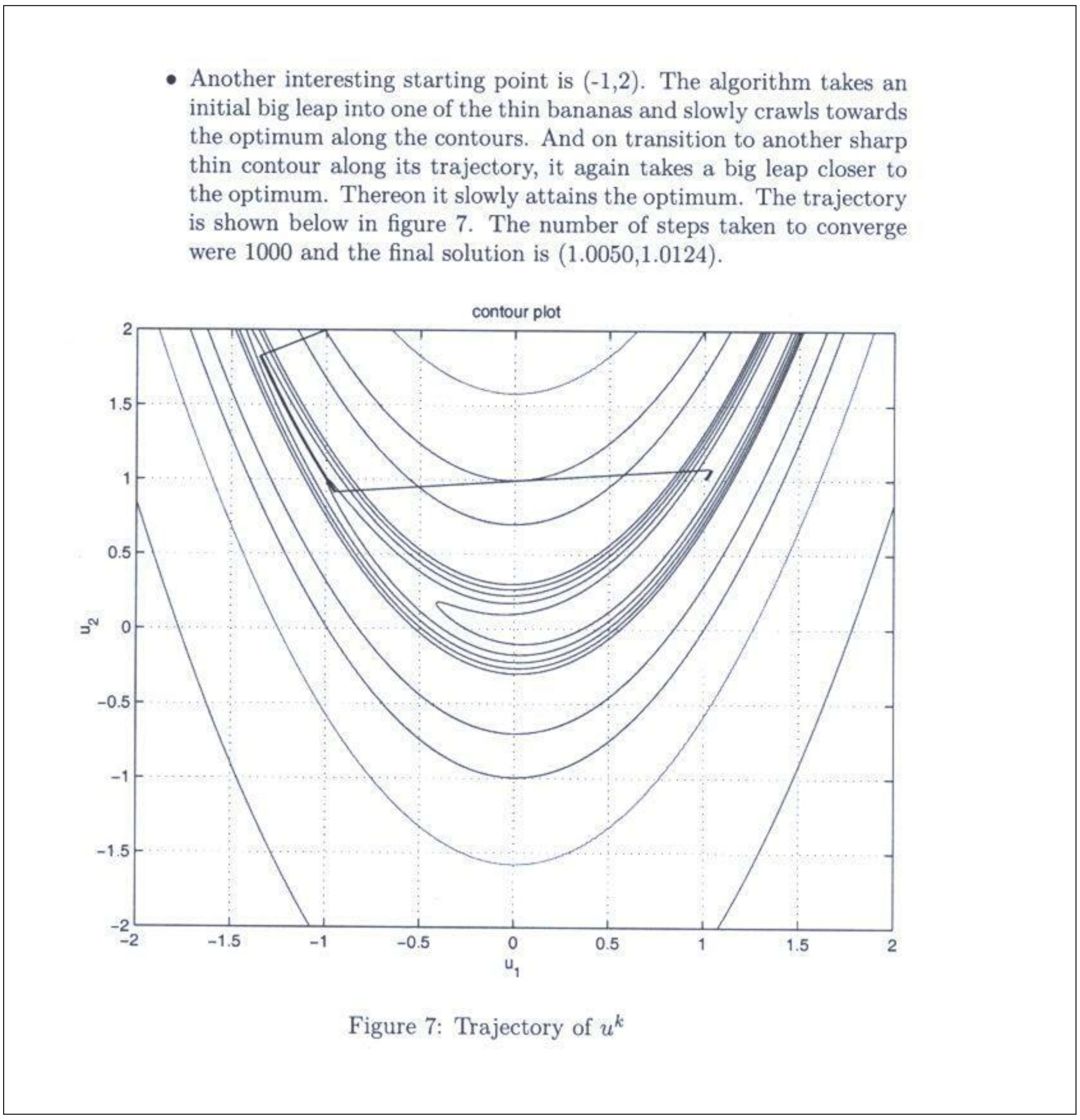
$$J(u) = \sum_{i=1}^{N} 100(u_{i+1} - u_i^2)^2 + (1 - u_i)^2 \tag{7}$$

with interesting domain described by $|u_i| \leq 2$, $i = 1, 2, \ldots, N+1$. Clearly a global minimum, $J^* = 0$, is attained with all $u_i = 1$.

1. (Case I: N = 1) The steepest descent algorithm with optimal step size is employed to minimize $J(u)$. The results are captured below:

   - Starting with $u^0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, the steepest descent slowly reaches $u^k = \begin{bmatrix} 0.9523 \\ 0.9177 \end{bmatrix}$ after 2000 iterations.
   - As we reach closer to the optimum i.e. $u^* = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, there are oscillations as the contours get much closer to each other - in other words the "bananas" get thin and long.
   - The algorithm converges to the optimum if allowed to run for sufficient time - the convergence is slow because of the search in the negative gradient direction. A conjugate direction algorithm will ouperform optimal descent by far.

Figure 6: Trajectory of u^k

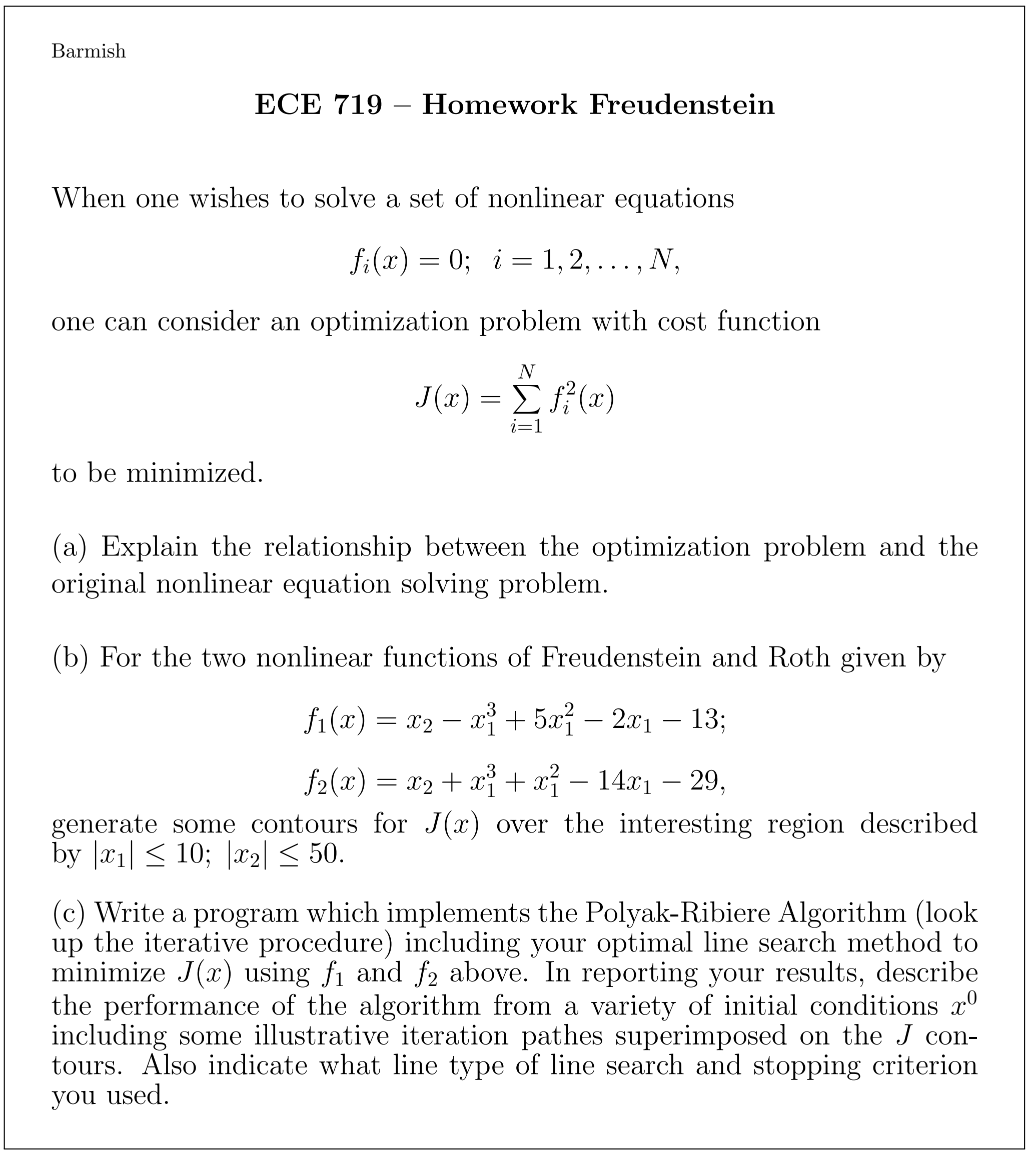


- Another interesting starting point is (-1,2). The algorithm takes an initial big leap into one of the thin bananas and slowly crawls towards the optimum along the contours. And on transition to another sharp thin contour along its trajectory, it again takes a big leap closer to the optimum. Thereon it slowly attains the optimum. The trajectory is shown below in figure 7. The number of steps taken to converge were 1000 and the final solution is (1.0050,1.0124).

Figure 7: Trajectory of u^k

4.5 HW 5

4.5.1 Problem 1



Barmish

ECE 719 – Homework Freudenstein

When one wishes to solve a set of nonlinear equations

$$f_i(x) = 0; \quad i = 1, 2, \ldots, N,$$

one can consider an optimization problem with cost function

$$J(x) = \sum_{i=1}^{N} f_i^2(x)$$

to be minimized.

(a) Explain the relationship between the optimization problem and the original nonlinear equation solving problem.

(b) For the two nonlinear functions of Freudenstein and Roth given by

$$f_1(x) = x_2 - x_1^3 + 5x_1^2 - 2x_1 - 13;$$

$$f_2(x) = x_2 + x_1^3 + x_1^2 - 14x_1 - 29,$$

generate some contours for $J(x)$ over the interesting region described by $|x_1| \leq 10$; $|x_2| \leq 50$.

(c) Write a program which implements the Polyak-Ribiere Algorithm (look up the iterative procedure) including your optimal line search method to minimize $J(x)$ using f_1 and f_2 above. In reporting your results, describe the performance of the algorithm from a variety of initial conditions x^0 including some illustrative iteration pathes superimposed on the J contours. Also indicate what line type of line search and stopping criterion you used.

Figure 4.39: problem 1 description

4.5.1.1 part (a)

Let $f_i(x) : \Re^n \to \Re$. We want to solve

$$f_i(x) = 0 \qquad i = 1, 2, \cdots N \tag{1}$$

Which means finding x^* which makes value of $f_i(x^*)$ be zero. If we consider the vector $F(x)$ of functions $f_i(x)$

$$F(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_N(x) \end{pmatrix}$$

Then the square of the Euclidean norm of $F(x)$ is

$$\|F(x)\|^2 = \sum_{i=1}^{N} f_i^2(x)$$

The minimum value of $\|F(x)\|$ is zero since it is a norm. Which is the same as $\|F(x)\|^2 = 0$.

This means $F(x) = 0$ occurs when $\|F(\boldsymbol{x})\|^2 = 0$. So the solution to (1) is the same $\boldsymbol{x}^*$ as finding the minimizer $\boldsymbol{x}^*$ which makes $\|F(\boldsymbol{x})\|^2$ minimum.

Therefore minimizing $J(x) = \|F(\boldsymbol{x})\|^2 = \sum_{i=1}^N f_i^2(\boldsymbol{x})$ will give the solution to (1). This is similar to finding least squares solution to set of linear equations, except now the set of equations $F(\boldsymbol{x})$ are non-linear in x.

4.5.1.2 part b

$$f_1(x) = x_2 - x_1^3 + 5x_1^2 - 2x_1 - 13$$
$$f_2(x) = x_2 + x_1^3 + x_1^2 - 14x_1 - 29$$

Hence

$$\begin{aligned} J(x) &= f_1^2(x) + f_2^2(x) \\ &= \left(x_2 - x_1^3 + 5x_1^2 - 2x_1 - 13\right)^2 + \left(x_2 + x_1^3 + x_1^2 - 14x_1 - 29\right)^2 \\ &= 2x_1^6 - 8x_1^5 + 2x_1^4 - 80x_1^3 + 12x_1^2x_2 + 12x_1^2 - 32x_1x_2 + 864x_1 + 2x_2^2 - 84x_2 + 1010 \qquad (1) \end{aligned}$$

$J(x)$ is non-linear function. The above is the $\|F(\boldsymbol{x})\|^2$ where now $F(\boldsymbol{x}) = \begin{pmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \end{pmatrix}$. We will use optimization to find the solution $\boldsymbol{x}$ to $F(\boldsymbol{x}) = 0$ by finding the minimizer of (1). The solution will turn out to be

$$\boldsymbol{x}^* = (x_1 = 4, x_2 = 5)$$

At this point $J(x^*) = 0$ and also $f_1(x^*) = 0$ and $f_2(x^*) = 0$. So the above is the true solution to $f_i(x) = 0$. But there is also another local minimum close to it located at

$$\boldsymbol{x} = (x_1 = -0.8968, x_2 = 11.4128)$$

where here $J(x) = 48.98$ and not zero. At this second local minimum, the corresponding values for f_i are $f_1(\boldsymbol{x}) = 4.949$ and $f_2(\boldsymbol{x}) = -4.949$. These were found by running the conjugate gradient algorithm with Polyak-Ribiere stepping as given below.

The following is contour plot of the full range given in the problem statement, showing there are two local minimums, one around point $x_1 = 4.5, x_2 = 8$ and another around $x_1 = -1.3, x_2 = 10$

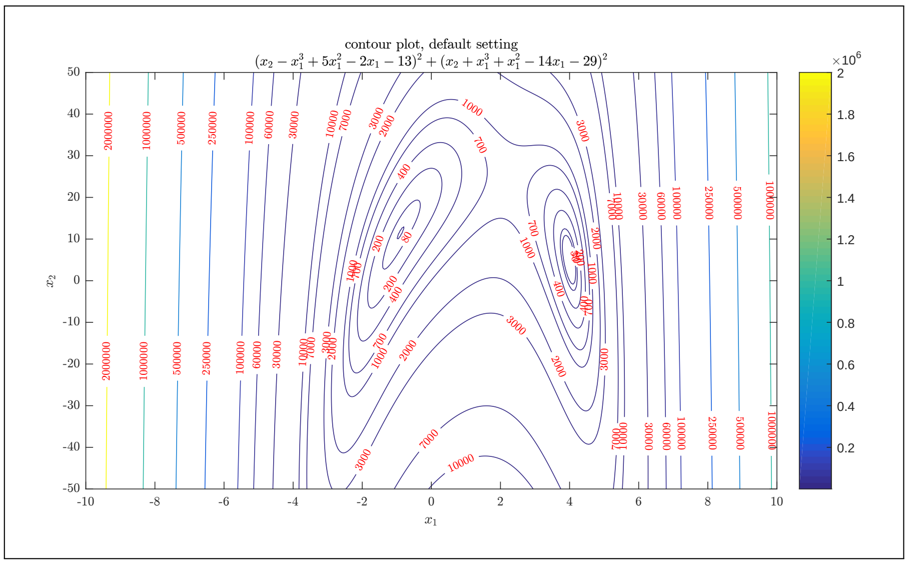


Figure 4.40: contour plot, full range

This is zoomed version of the above to show more clearly the area around the variations

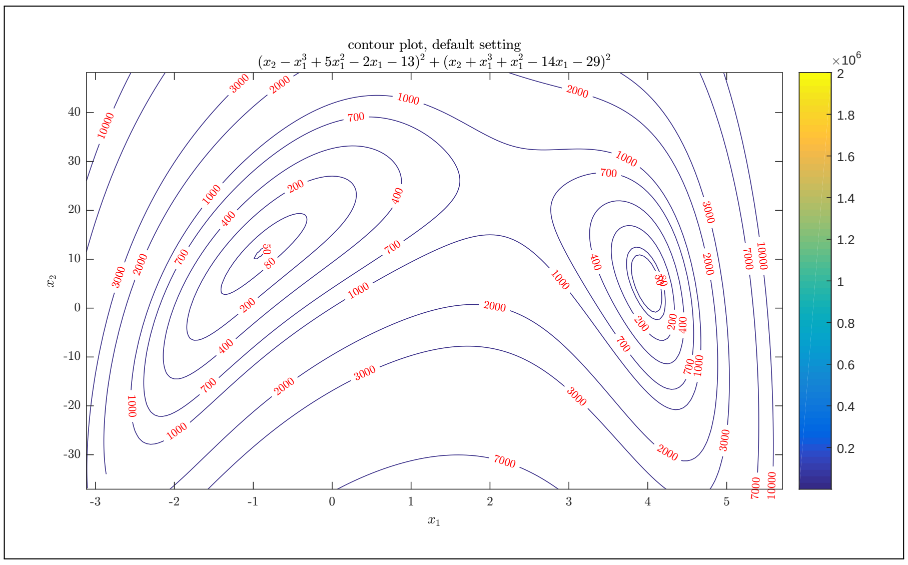


Figure 4.41: contour plot, zoomed version

This is filled contour version of the above.

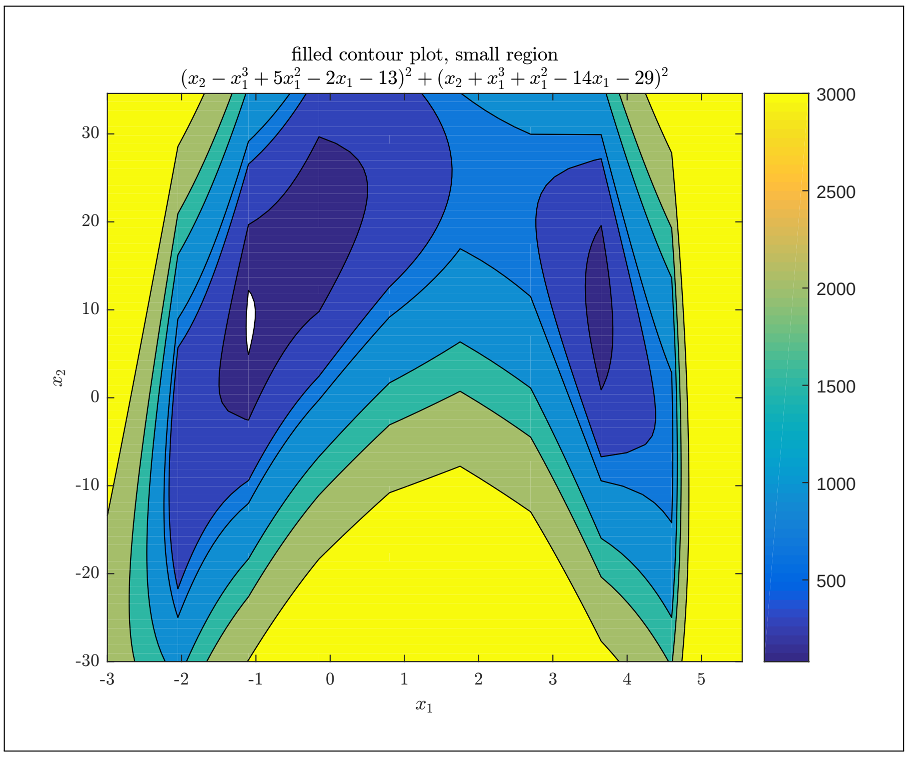


Figure 4.42: contour plot, filled zoomed version

This is 3D plot of the function $J(x)$

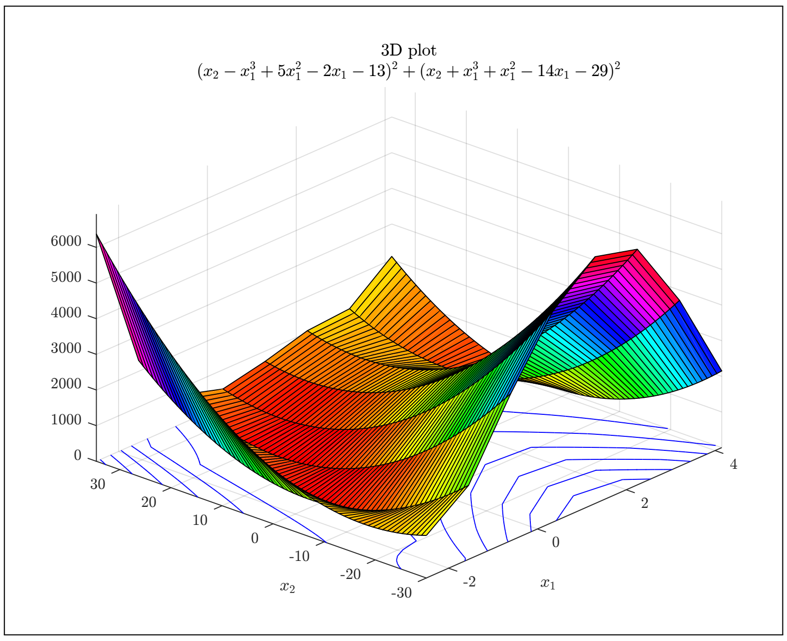


Figure 4.43: contour plot, filled zoomed version

4.5.1.3 part(c)

A Matlab program is given in the appendix which implements Polyalk-Ribiere (and it also supports Fletcher-Reeves). The result is given below, with discussion following each result. One result also shows an interesting difference found between Polyalk-Ribiere and Fletcher-Reeves when starting from some random found u^0 point. In all runs, Matlab fminsearch was also used to compare the minimizer found. In some cases, this algorithm found the same minimum as Matlab's fminsearch, and in other cases it did not.

The result shows each point u^k visited with the value of β_k and α_k found, and the value of the objective function and the gradient at each step.

For the line search, golden section search was used to find α_k with maximum step size $H_{\max} = 1$. The stopping criteria used for all these runs is $|\nabla J(u)| \leq 0.001$.

4.5.1.3.1 The algorithm The following is the outline of general algorithm expressed as pseudo code.

Algorithm 1 Conjugate gradient using Polyalk-Ribiere or Fletcher-Reeves

```
procedure CONJUGATE_GRADIENT
    ▷ Initialization
    $\epsilon \leftarrow$ minimum convergence limit on $\|\nabla J(u)\|$
    $k \leftarrow 0$
    $u \leftarrow u^0$
    $max\_iterations \leftarrow$ max iterations allowed
    $g_{current} \leftarrow \nabla J(u)$
    $v_{current} \leftarrow -g_{current}$
    while $\|g_{current}\| > \epsilon$ do
        ▷ do line search, using golden section, maximum step size is one
        $\alpha \leftarrow \min_\alpha \tilde{J}(\alpha) = J(u + \alpha\, v_{current})$
        $g_{previous} \leftarrow g_{current}$
        $u \leftarrow u + \alpha v_{current}$
        $g_{current} \leftarrow \nabla J(u)$
        if Fletcher-Reeves then
            $\beta \leftarrow \frac{\|g_{current}\|^2}{\|g_{previous}\|^2}$
        else if Polyalk-Ribiere then
            $\beta \leftarrow \frac{g_{current}^T (g_{current} - g_{previous})}{\|g_{previous}\|^2}$
        end if
        $v_{current} \leftarrow -g_{current} + \beta\, v_{current}$
    end while
end procedure
```

Figure 4.44: Conjugate gradient using Polyalk-Ribiere or Fletcher-Reeves

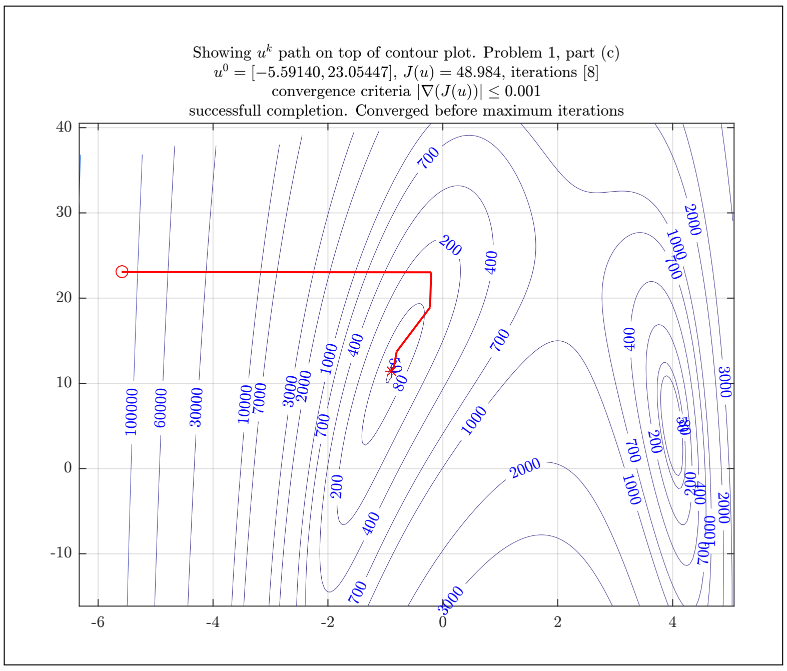


Figure 4.45: test case 1, problem 1, part c

4.5.1.3.2 Test 1 starting from $(-5.49, 23.05)$ Matlab fminsearch found $J(-0.8968, 11.4128) = 48.9843$. This program found $J(-0.8968, 11.4128) = 48.9843$. since $u^* = (4, 5)$ we see that the search did not find u^* but found the other local minimum near it, since the search was started from a point closer to the second one. Also we see Matlab fminsearch result matched our result. So this is good. It took 8 steps. We also notice that $\nabla J\left(x^k\right)$ increased at one step (step 3) during the search. This is indication that this is not a quadratic function (which we already know this), but $\nabla J\left(x^k\right)$ started to decrease again after that.

k	x^k	$J\left(x^k\right)$	$\left\|\nabla J\left(x^k\right)\right\|$	α_k	β_k
1	$(-5.59, 23.05)$	129231.19	116683.427	0.000046	−0.000003
2	$(-0.20, 23.028)$	122.99	15.12	0.273	86.62
3	$(-0.2227, 18.9)$	91.83	140.56	0.003958	0.3535
4	$(-0.8, 13.72)$	52.15	39.059	0.00394	0.4082
5	$(-0.855, 11.885)$	49.16	12.1855	0.00236	0.05
6	$(-0.896, 11.436)$	48.98	0.583	0.00238	0.0094
7	$(-0.8968, 11.4129)$	48.98	0.00545	0.002095	0.00123
8	$(-0.8968, 11.4128)$	48.98	0.000007	0.000000	0.000000

4.5.1.3.3 Test 2 starting from $(5.8, 35.89)$

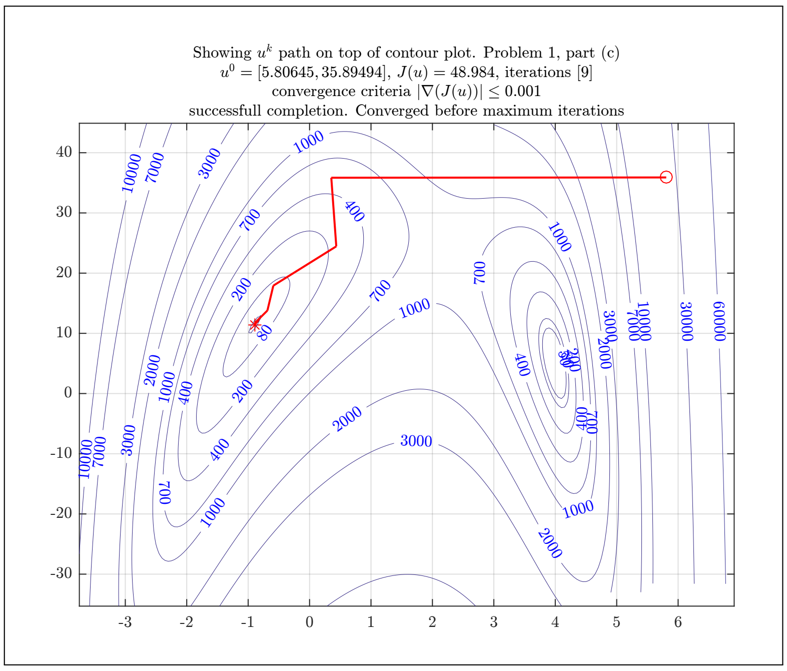


Figure 4.46: test case 2, problem 1, part c

Matlab fminsearch found $J(-0.8968, 11.4128) = 48.9843$. This program found $J(-0.8968, 11.4128) = 48.9843$. since $u^* = (4, 5)$ we see again that the search did not find u^* but found the other local minimum. Also we see Matlab fminsearch result matched our result. So this is good. It took 9 steps. We also notice that $\nabla J\left(x^k\right)$ increased at one step (step 3) during the search.

k	x^k	$J\left(x^k\right)$	$\left\|\nabla J\left(x^k\right)\right\|$	α_k	β_k
1	$(5.806, 35.895)$	24303.88	32066.72	0.000170	0.000011
2	$(0.353, 35.847)$	520.53	49.582	0.2299	36.96
3	$(0.435, 24.448)$	238.41	301.265	0.00356	0.28
4	$(-0.59, 17.92)$	71.95	69.317	0.0079	1.3363
5	$(-0.686, 13.789)$	53.18	52.828	0.0028	0.1788
6	$(-0.883, 11.7991)$	49.08	8.1969	0.00295	0.06401
7	$(-0.895, 11.4284)$	48.98	0.4961	0.00193	0.00629
8	$(-0.896801, 11.412913)$	48.98	0.003106	0.002634	0.000101
9	$(-0.896805, 11.412779)$	48.98	0.000000	0.000000	0.000000

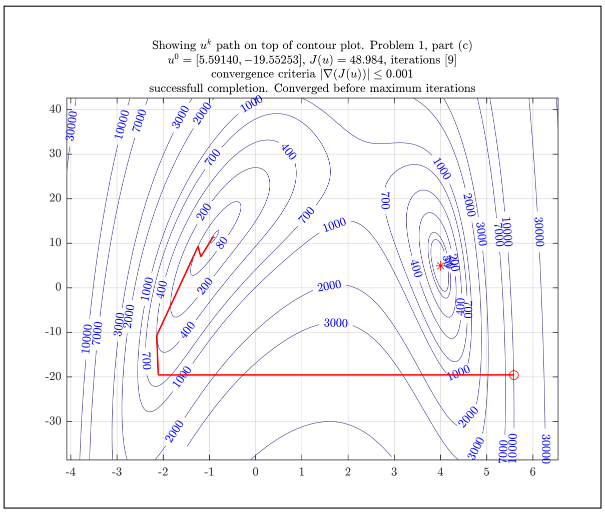


Figure 4.47: test case 3, problem 1, part c

4.5.1.3.4 Test 3 starting from (5.59, −19.55)

Matlab fminsearch found the true minimum $J(4, 5) = 0$. This program did not do as well, and went for the second local minimum at $J(-0.8968, 11.4128) = 48.9843$, which has the corresponding solution $f_1(x) = 4.9490, f_2(x) = -4.9490$.

One surprising thing to note, is that Matlab fminsearch uses simplex method according to the help. But this problem is not linear. It turns out that Matlab fminsearch uses a modified version of simplex method, called the Nelder-Mead simplex (direct search). It seems to do better than algorithm implemented in this problem. But in the next test case, we will see that this algorithm evens the score with Matlab's and in the next test case it is we who will do better.

It took 9 steps. Again as before, $\nabla J\left(x^k\right)$ increased at one step during the search (at step 3 also).

k	x^k	$J\left(x^k\right)$	$\left\lvert\nabla J\left(x^k\right)\right\rvert$	α_k	β_k
1	$(5.5914, -19.553)$	10150.82	19380.2	0.000397	−0.000023
2	$(-2.107, -19.566)$	585.39	41.5373	0.2127	423.717
3	$(-2.142, -10.731)$	401.83	854.79	0.001138	−0.213
4	$(-1.247, 9.303)$	79.34	264.025	0.000619	−0.14389
5	$(-1.1875, 6.974)$	57.85	46.1832	0.00822	−0.2476
6	$(-0.9218, 11.436)$	49.30	24.1288	0.001065	−0.02258
7	$(-0.9046, 11.29)$	48.99	0.56707	0.0389	−0.0926
8	$(-0.8969, 11.4131)$	48.98	0.05981	0.001129	−0.000415
9	$(-0.896806, 11.412772)$	48.98	0.000025	0.000000	0.000000

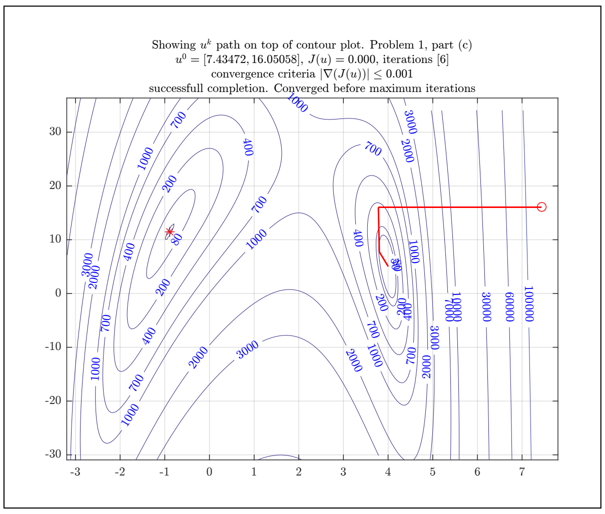


Figure 4.48: test case 4, problem 1, part c

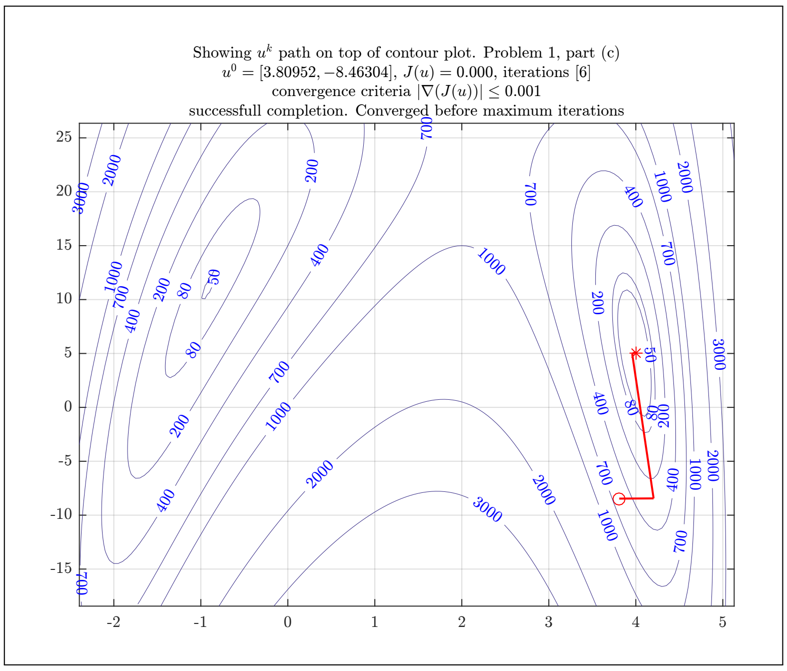
4.5.1.3.5 Test 4 starting from $(7.43472, 16.05058)$

Matlab fminsearch here did not find the true minimum $J(4,5) = 0$ while this algorithm did.

It took 6 steps only. Again as before, $\nabla J\left(x^k\right)$ increased at one step during the search (at step 2).

k	x^k	$J\left(x^k\right)$	$\left\lvert\nabla J\left(x^k\right)\right\rvert$	α_k	β_k
1	$(7.435, 16.05)$	143368.39	143787.679	0.000025	0.000002
2	$(3.78, 16.04)$	172.57	30.799	0.2696	200.218
3	$(3.84, 7.74)$	44.74	435.9738	0.000433	0.00797
4	$(3.999778, 5.066770)$	0.01	3.455556	0.001349	0.009287
5	$(3.999989, 5.000154)$	0.00	0.031875	0.000336	−0.000038
6	$(4.000000, 5.000000)$	0.00	0.000001	0.000000	0.000000

4.5.1.3.6 Test 5 starting from $(3.809, -8.46)$



Showing u^k path on top of contour plot. Problem 1, part (c)
$u^0 = [3.80952, -8.46304]$, $J(u) = 0.000$, iterations [6]
convergence criteria $|\nabla(J(u))| \leq 0.001$
successfull completion. Converged before maximum iterations

Figure 4.49: test case 5, problem 1, part c

Here both Matlab fminsearch and this algorithm, found the true minimum.

It took 6 steps only. Again as before, $\nabla J\left(x^k\right)$ increased at one step during the search (at step 3).

k	x^k	$J\left(x^k\right)$	$\left\lvert\nabla J\left(x^k\right)\right\rvert$	α_k	β_k
1	(3.809524, −8.463035)	580.29	1386.28	0.000285	0.000784
2	(4.204487, −8.444322)	267.86	40.2297	0.33335	14.037757
3	(3.958554, 4.969723)	3.20	150.186235	0.000278	−0.009224
4	(3.997429, 5.127561)	0.02	1.447732	0.022782	0.258685
5	(4.000078, 5.000400)	0.00	0.316225	0.000273	0.000175
6	(4.000000, 5.000004)	0.00	0.000057	0.000000	0.000000

4.5.1.3.7 Test 6 (first strange one) starting from (6.63594, −14.29961)

This test case and the second one are pathological cases, in the sense that this algorithm did find the true minimum, but the path taken headed first to the second local minimum and was very close to it, before turning and going to the true minimum at (4, −5). At this time, I am not able to explain this and more time needed to investigate. It does however find the true minimum eventually, so this is good result even if the path taken looks very strange compared to all the other tests above. The main difference between this test case and the last ones, is that here the objective function $J(x)$ increased at one point during the search (at step 6 as shown below).

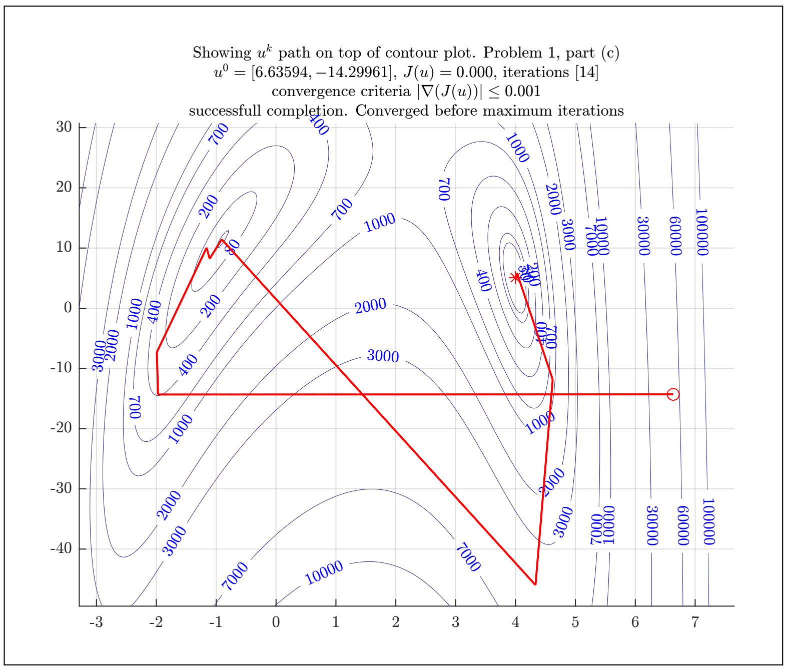


Figure 4.50: test case 6, Polyak-Ribiere, problem 1, part c

Here both Matlab fminsearch and this algorithm, found the true minimum.

It took 14 steps. Here $\nabla J\left(x^k\right)$ increased and decreased more than one time during the search.

k	x^k	$J\left(x^k\right)$	$\left\lvert\nabla J\left(x^k\right)\right\rvert$	α_k	β_k
1	(6.63594, −14.29961)	52701.77	67823.57	0.000127	0.000002
2	(−1.970382, −14.321801)	393.95	31.646	0.221630	385.651
3	(−1.991739, −7.308131)	282.95	621.5254	0.001424	−0.217509
4	(−1.159618, 10.073263)	67.36	199.4501	0.000696	−0.132737
5	(−1.109423, 8.218728)	53.56	31.5552	0.009140	−0.241872
6	(−0.908598, 11.459289)	49.08	13.239	0.662964	22576.86
7	(4.328897, −45.933286)	4299.56	1995.887	0.000000	−0.009991
8	(4.333267, −45.980644)	4299.50	1975.7426	0.001732	3.309271
9	(4.620182, −11.840013)	883.28	2743.2211	0.000271	−0.051462
10	(4.024522, 5.862406)	3.99	149.444	0.000338	−0.154415
11	(4.012227, 4.726667)	0.22	28.564	0.000532	−0.010341
12	(4.000032, 5.002778)	0.00	0.299	0.000518	−0.004453
13	(4.000001, 4.999987)	0.00	0.00134	0.000550	0.001374
14	(4.000000, 5.000000)	0.00	0.000002	0.000000	0.000000

The above was re-run again, starting from the same u^0, but now using Fletcher-Reeves formula. The result was surprising. Now the algorithm did not show the strange path as above, however, it also did not find the true minimum at $(4, -5)$ and instead went for the second local minimum as shown below. This shows, at least in this test, that Polyak-Ribiere formula did a better job, even though it took more steps.

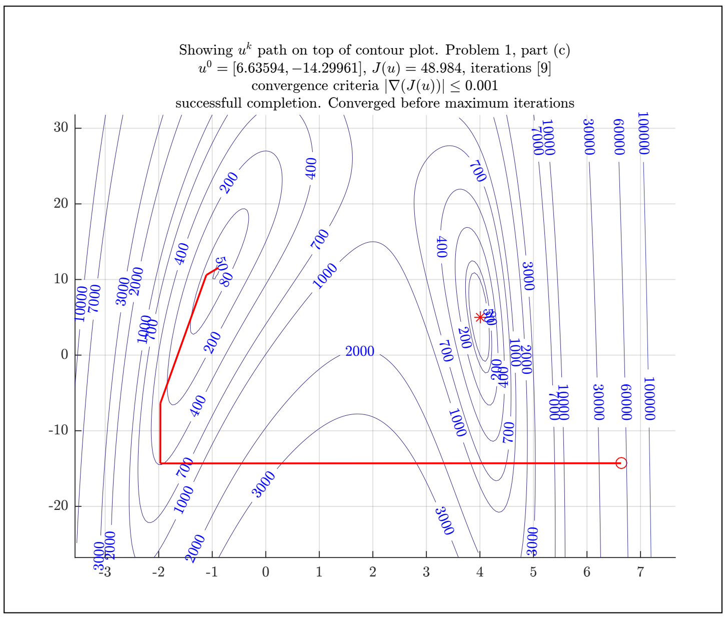


Figure 4.51: test case 6, using Fletcher-Reeves, problem 1, part c

k	x^k	$J\left(x^k\right)$	$\left\lvert\nabla J\left(x^k\right)\right\rvert$	α_k	β_k
1	$(6.63594, -14.29961)$	52701.77	67823.57	0.000127	0.000000
2	$(-1.970382, -14.321801)$	393.95	31.646	0.2519	393.1282
3	$(-1.968895, -6.351520)$	267.83	627.462	0.001362	0.07595
4	$(-1.111164, 10.592228)$	63.10	172.916	0.001001	0.000010
5	$(-0.890514, 11.528839)$	48.99	0.5567	0.001137	0.03014
6	$(-0.889896, 11.528704)$	48.99	0.0967	0.888831	202.77
7	$(-0.893567, 11.441590)$	48.99	1.3763	0.001469	0.000012
8	$(-0.896818, 11.412476)$	48.98	0.00481	0.001101	0.002681
9	$(-0.896823, 11.412476)$	48.98	0.000249	0.000000	0.000000

4.5.1.3.8 Test 7 (second strange one) starting from $(0.5837, -46.595)$

This test case also showed difference between Polyak-Ribiere and Fletcher-Reeves.

With Polyak-Ribiere, it found the same minimum as Matlab fminsearch using a strange path where $J(u)$ did increase at one point before decreasing again.

However, it did a better job than Fletcher-Reeves.

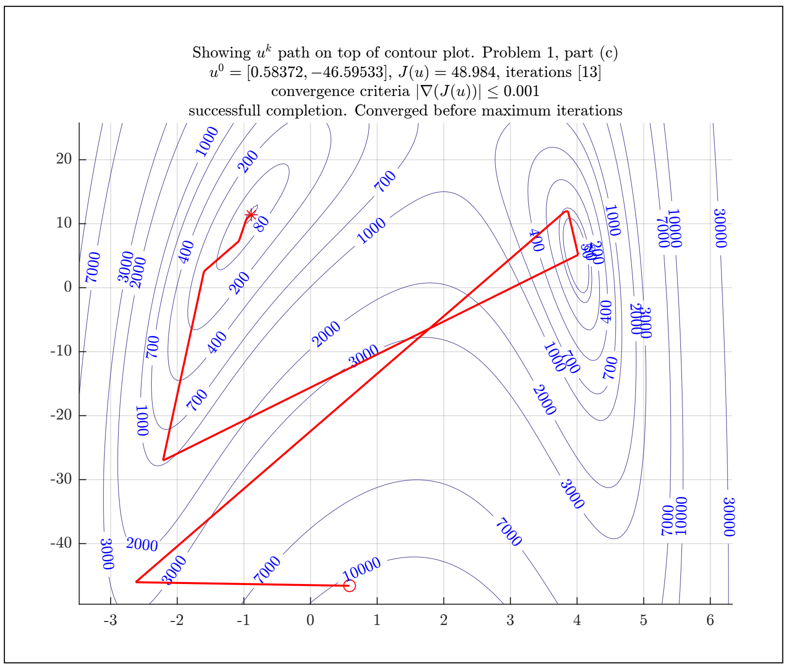


Figure 4.52: test case 7, Polyak-Ribiere, problem 1, part c

Here both Matlab fminsearch and this algorithm did not find the true minimum.

It took 13 steps. Here $\nabla J\left(x^k\right)$ increased and decreased more than one time during the search. Also $J(u)$ increased during the search before decreasing again.

k	x^k	$J\left(x^k\right)$	$\left\lvert\nabla J\left(x^k\right)\right\rvert$	α_k	β_k
1	$(0.583720, -46.595330)$	10438.38	1656.968	0.001966	0.003862
2	$(-2.624791, -46.035172)$	2450.06	103.007	0.564906	2.002
3	$(3.819328, 11.909200)$	72.20	148.967	0.000257	-0.1194
4	$(3.863288, 11.957784)$	69.31	28.774	0.163101	7.727
5	$(4.016286, 5.132002)$	0.67	70.2157	0.098572	18.327
6	$(-2.188772, -26.898544)$	959.12	336.852	0.000017	-0.183
7	$(-2.213801, -26.997880)$	958.16	255.113	0.025963	3.785
8	$(-1.599015, 2.551447)$	123.88	387.315	0.001099	0.179
9	$(-1.074840, 7.277357)$	57.59	60.1637	0.004433	0.517
10	$(-0.961876, 10.715217)$	49.45	22.635	0.001854	-0.0502
11	$(-0.895537, 11.456508)$	48.99	1.2014	0.002193	-0.01161
12	$(-0.896851, 11.412270)$	48.98	0.014138	0.002174	0.000948
13	$(-0.896805, 11.412779)$	48.98	0.000013	0.000000	0.000000

4.5.1.4 Appendix. Source code for problem 1

```
function nma_HW5_problem_1_part_b
%Solves problem 1, part b, HW5
%ECE 719, UW Madison, Spring 2016
%
%

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

%reset(0);
xlimits   = [-10 10];  %x limits, for plotting, change as needed
ylimits   = [-50 50]; %y limits, for plotting, change as needed
myTitle   = ...
 '$$(x_2-x_1^3+5x_1^2-2x_1-13)^2+(x_2+x_1^3+x_1^2-14x_1-29)^2$$';
[u1,u2,z] = nma_makeContourData(0.05,xlimits,ylimits);

figure(1);
v =[50 80 200 400 700 10^3 2*10^3 3*10^3 7*10^3 10^4 ...
           3*10^4 6*10^4 10^5 2.5*10^5 ...
           5*10^5 10^6 2*10^6];
[C,h]    = contour(u1,u2,z,v);
colorbar
%[C,h]    = contour(u1,u2,z);

clabel(C,h,v,'Fontsize',7,'interpreter','Latex','Color','red');
nma_setMyLabels('$$x_1$$','$$x_2$$',...
    {'\makebox[4in][c]{contour plot, default setting}',...
    sprintf('\\makebox[4in][c]{%s}',myTitle)});

figure();
xlimits   = [-3 6];  %x limits, for plotting, change as needed
ylimits   = [-30 35]; %y limits, for plotting, change as needed
[X,Y,Z] = nma_makeContourData(.95,xlimits,ylimits);
v        =[80 300 700 900 1500 2000 3000];
[C,h]    = contourf(X,Y,Z,v);
colorbar;
nma_setMyLabels('$$x_1$$','$$x_2$$',...
    {'\makebox[4in][c]{filled contour plot, small region}',...
    sprintf('\\makebox[4in][c]{%s}',myTitle)});

```

```
figure();
xlimits   = [-2.5 5];  %x limits, for plotting, change as needed
ylimits   = [-30 30]; %y limits, for plotting, change as needed
[X,Y,Z] = nma_makeContourData(.95,xlimits,ylimits);
%surfl(X,Y,Z);
surf(X,Y,Z);
colormap(hsv);
%view([154,46]);
hold on;
contour(X,Y,Z,'Linecolor',[0 0 1]);
nma_setMyLabels('$$x_1$$','$$x_2$$',...
    {'\makebox[4in][c]{surf plot}',...
    sprintf('\\makebox[4in][c]{%s}',myTitle)});

end
```

```
function nma_HW5_problem_1_part_c()
%finds the min value of
%
%    J(x) = f1^2 + f2^2 where
%    f1 = x2-x1^3+5*x1^2-2x1-13
%    f2=x2+x1^3+x1^2-14x1-29
%
% over range x1=-10..10 and x2=-50..50 using steepest descent
%
% ECE 719, Spring 2016
% Matlab 2015a
%Nasser M. Abbasi

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

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

%paramters, change as needed
%select the algorithm to use. Either 'conjugate gradient'
%or 'steepest descent'
METHOD       = 'conjugate gradient';

DO_GUI       = false;    %set to true to get input 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.
xlimits      = [-20 20];   %x limits, for plotting
ylimits      = [-90 90];   %y limits, for plotting
del          = 0.05;     %grid size, used for making meshgrid
fixed_levels = [50 80 200 400 700 10^3 2*10^3 3*10^3 ...
                7*10^3 10^4 3*10^4 ...
                6*10^4 10^5 2.5*10^5 5*10^5 10^6 2*10^6];
CONTOUR_LINES_AUTO = 'fix'; %set to 'auto', to see matlab contour
%                      %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
%optimization function
%opt.u = [6.63594;-14.29961];    %starting guess x-coordinate
```

```
%selection of tough ones: This gives very different
%                               result from fletcher and polak.
%opt.u  = [6.63594;-14.29961];   %starting guess x-coordinate
opt.u = [0.58372;-46.59533];   %starting guess x-coordinate

opt.MAX_ITER        = 10^4; %maximum iterations allowed
opt.STEP_SIZE       = -1;  %step size. set to -1 to use optimal
opt.objectiveFunc   = @objectiveFunc; %see function definition
opt.gradientFunc    = @gradientFunc;  %see function definition
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,u2,z] = nma_makeContourData(del,xlimits,ylimits);
figure();
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

%Find the minumum using Matlab build-in, in order to
%compare with in plot
optimalValue = fminsearch(opt.objectiveFunc, opt.u);
objectiveAtOptimal = objectiveFunc(optimalValue);
fprintf('Matlab found J(%5.4f,%5.4f)=%5.4f\n',optimalValue(1),...
                            optimalValue(2),objectiveAtOptimal);

%mark location of minimum found by fminsearch on plot
hold on;
plot(optimalValue(1),optimalValue(2),'*r');

%plot starting point
plot(opt.u(1),opt.u(2),'or');
xlim(xlimits); ylim(ylimits);
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

%make the call to find search path.
if strcmp(METHOD,'steepest descent')
   [status , pts,levelSets, gradientNormTol,steps] = ...
                                     nma_steepest_descent(opt);
else
   [status,pts,levelSets, gradientNormTol,steps,betaK]=...
                                        nma_polyak_ribiere(opt);
end
fprintf('HW5 found J(%5.4f,%5.4f)=%5.4f\n',pts(end,1),...
                                     pts(end,2),levelSets(end));

%check if search was success or not.
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
```

```matlab

%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);
      clabel(C,h,'Fontsize',8,'interpreter','Latex',...
                                                'Color','blue');
    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 )
        line([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],...
                                   'LineWidth',1,'Color','red');
        %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 (c)',...
           opt,pts,k,status),'FontSize', 8);
    end
end
title(format_plot_title(...
'Showing $u^k$ path on top of contour plot. Problem 1, part (c)',...
 opt,pts,size(pts,1),status),'FontSize', 8);

```

```
%plot J(x) changes
figure();
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');

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

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

%Plot betaK
if strcmp(METHOD,'conjugate gradient')
figure();
stem(betaK);
grid;
title(format_plot_title(...
        'Showing $\beta(k)$ progress. Problem 1, part (c)',...
    opt,pts,size(pts,1),status),'FontSize', 8);
xlabel('step number'); ylabel('$\beta_k$');
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
end

%Plot alpha
if strcmp(METHOD,'conjugate gradient')
figure();
stem(steps);
grid;
title(format_plot_title(...
        'Showing $\alpha(k)$ progress. Problem 1, part (c)',...
    opt,pts,size(pts,1),status),'FontSize', 8);
xlabel('step number'); ylabel('$\alpha_k$');
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
end

format long g;
fprintf('x1\t\t\tx2\t\t\tJ(x1,x2)\tgrad(J(u))\talpha\t\tbeta\n');
for i=1:length(steps)
    fprintf('%7.6f\t%7.6f\t%5.2f\t%7.6f\t%7.6f\t%7.6f\n',...
              pts(i,1),pts(i,2),...
              levelSets(i),gradientNormTol(i),steps(i),betaK(i));
end

f1 =@(X1,X2) X2-X1^3+5*X1^2-2*X1-13;
f2 = @(X1,X2) X2+X1^3+X1.^2-14*X1-29;

fprintf('f1(x)=%5.4f,f1(x)=%5.4f\n',...
         f1(pts(end,1),pts(end,2)),f2(pts(end,1),pts(end,2)));
```

```
end
%-------------------------
%Evaluate J(u) at u
function f = objectiveFunc(u)
X1  = u(1);
X2  = u(2);
f1 = X2-X1.^3+5*X1.^2-2*X1-13;
f2 = X2+X1.^3+X1.^2-14*X1-29;
f = f1.^2+f2.^2;
end
%---------------------
%Evaluate grad(J(u)) at u
function g = gradientFunc(u)
x1  = u(1);
x2  = u(2);
g1=2*(3*x1^2 + 2*x1 - 14)*(x1^3 + x1^2 - 14*x1 + x2 - 29) +...
 2*(3*x1^2 - 10*x1 + 2)*(x1^3 - 5*x1^2 + 2*x1 - x2 + 13);
g2=12*x1^2 - 32*x1 + 4*x2 - 84;
g=[g1;g2];
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]{$u^0=[%6.5f,%6.5f]$,  $J(u)=%3.3f$, iterations [$%d$]}',...
  opt.u(1),opt.u(2),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 (not used, for practice)
function g = hessian_func(u)
x1  = u(1);
x2 = u(2);

g11=2*(6*x1 - 10)*(x1^3 - 5*x1^2 + 2*x1 - x2 + 13) + ...
    2*(3*x1^2 - 10*x1 + 2)^2 ...
    + 2*(3*x1^2 + 2*x1 - 14)^2 + ...
    2*(6*x1 + 2)*(x1^3 + x1^2 - 14*x1 + x2 - 29);
g12=24*x1 - 32;
g21=24*x1 - 32;
g22=4;
g=[g11,g12;g21,g22];
end
```

```
function [status,pointsFound,levelSets,gradientNormTol,steps,betaK]= ...
      nma_polyak_ribiere(opt)
% This function performs conjugate gradient search
% starting from a point looking for point which minimizes a
% function. Supports multi-variable function. It needs handle
% of the funtion and handle to the gradient. It reurns all
% points visited in the search. This supports Fletcher-Reeves
% and Polyalk-Ribiere
%
% 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.
% 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 return 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.
% stop_on_oscillation  flag. Set to true to stop when objective
%                 function detected to be increasing. Else set
%                 to false if you do not want to stop when J(u)
%                 increases at any point
%
% 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 contain 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 These are the alpha_k
%                  values.
% betaK            vector contains values of beta found at
%                  each step
%
% by Nasser M. Abbasi  ECE 719, UW Madison, HW 5

%pre-allocate data for use in the main loop below
N               = size(opt.u,1);
fLambda         = @(alpha,u,s) opt.objectiveFunc(u+alpha*s);

```

```
%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);
    betaK           = zeros(opt.MAX_ITER,1);
end

% initialize counters before main loop
k                    = 1;
currentPoint         = opt.u;
keep_running         = true;
status               = 0;
steps_in_oscillation = 0;
last_level           = 0;
current_grad         = opt.gradientFunc(currentPoint);
current_v            = -current_grad;

while keep_running

    update_accumlate();

    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_convergence();

    if keep_running

       %A   = opt.hessian(currentPoint);
       %lam = - dot(current_grad, current_v)/...
       %                              (current_v.'*A*current_v);

       %do not use norm on current_v here!
       alpha = nma_golden_section(...
        fLambda,currentPoint,current_v,0,1,sqrt(eps('double')));

       % make step towards min
       currentPoint = currentPoint + alpha* current_v;
       last_grad    = current_grad;
       current_grad = opt.gradientFunc(currentPoint);

       %fletcher
       %beta = norm(current_grad)^2/norm(last_grad)^2;

       %polyak
       beta = (current_grad.' * (current_grad-last_grad))/...
                                               norm(last_grad)^2;

       current_v = -current_grad + beta * current_v;
       if opt.accumulate
          steps(k) = alpha;
          betaK(k) = beta;
       end

       k = k + 1;
   end
end
%done. Chop data to correct number of steps used before returning
```

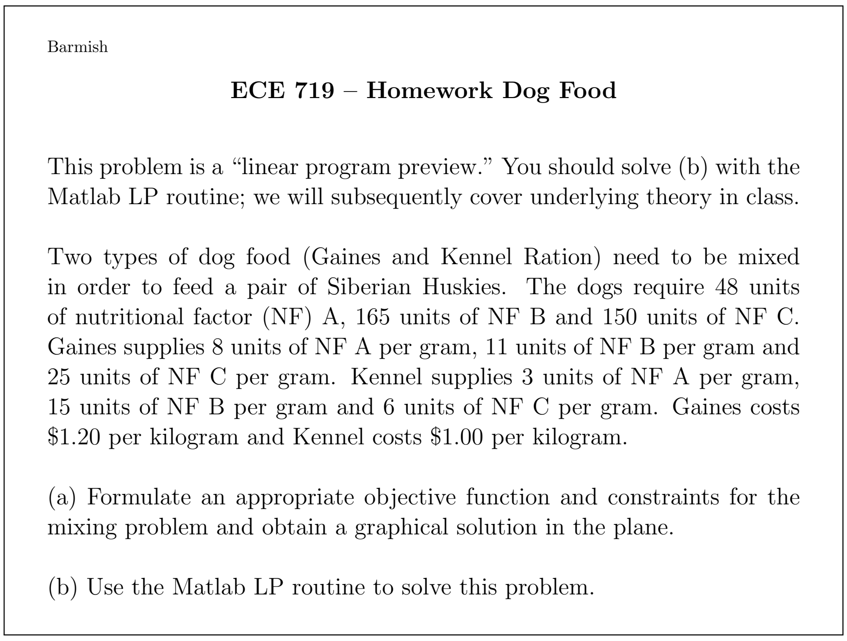
```
if opt.accumulate
    pointsFound     = pointsFound(1:k,:);
    levelSets       = levelSets(1:k);
    gradientNormTol = gradientNormTol(1:k);
    steps           = steps(1:k);
    betaK           = betaK(1:k);
else
    pointsFound     = currentPoint ;
    levelSets       = current_level;
    gradientNormTol = current_grad_norm;
    steps           = k;
    betaK           = beta;
end

    %----------------------------
    %internal function. Check if still need to keep iterating
    function check_convergence()
     % 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.
            keep_running = false;
            if steps_in_oscillation>0
                status = 1;
            else
                if k == opt.MAX_ITER
                    status= 2;
                end
            end
        end
    end
    %----------------------------------------------
    %internal
    function  update_accumlate()
        if k>1
            last_level = current_level;
        end

        current_level    = norm(opt.objectiveFunc(currentPoint));
        current_grad_norm = norm(current_grad);

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

```
%------------
%helper function to set plot attributes.
function nma_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 [X1,X2,Z] = nma_makeContourData(del,xlimits,ylimits)

x1      = xlimits(1):del:xlimits(2);
x2      = ylimits(1):del:ylimits(2);
[X1,X2] = meshgrid(x1,x2);
f1 = X2-X1.^3+5*X1.^2-2*X1-13;
f2 = X2+X1.^3+X1.^2-14*X1-29;
Z = f1.^2+f2.^2;
end
```

4.5.2 Problem 2



Barmish

ECE 719 – Homework Dog Food

This problem is a "linear program preview." You should solve (b) with the Matlab LP routine; we will subsequently cover underlying theory in class.

Two types of dog food (Gaines and Kennel Ration) need to be mixed in order to feed a pair of Siberian Huskies. The dogs require 48 units of nutritional factor (NF) A, 165 units of NF B and 150 units of NF C. Gaines supplies 8 units of NF A per gram, 11 units of NF B per gram and 25 units of NF C per gram. Kennel supplies 3 units of NF A per gram, 15 units of NF B per gram and 6 units of NF C per gram. Gaines costs $1.20 per kilogram and Kennel costs $1.00 per kilogram.

(a) Formulate an appropriate objective function and constraints for the mixing problem and obtain a graphical solution in the plane.

(b) Use the Matlab LP routine to solve this problem.

Figure 4.53: problem 2 description

4.5.2.1 part a

We need to minimize the cost of $48A + 165B + 150C$ by finding the optimal mix (quantities) of A, B, C obtained from Gaines and Kennel supply as shown in the following diagram

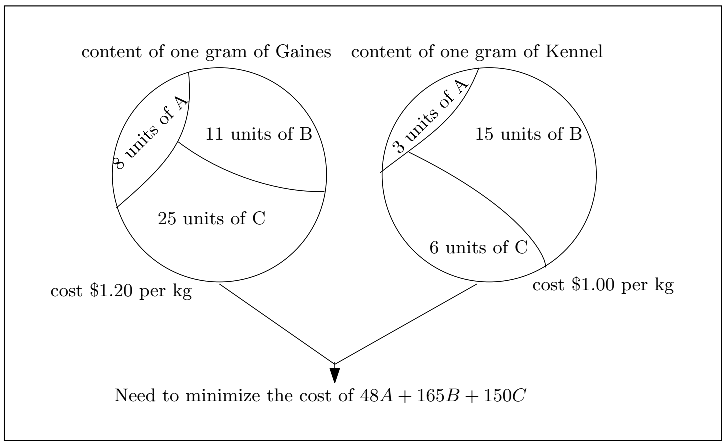


Figure 4.54: problem 2, part a

Let the amount (in grams) from Gaines be u_1 and let the amount of grams from Kennel be u_2. Therefore, we need to minimize

$$J(u) = (0.0012)\,u_1 + (0.001)\,u_2 \tag{1}$$

Since this is the cost. Since there are 8 units of A in each gram from Gaines, and there are 3 units of A in each gram from Kennel, then we have the first restriction which is

$$8u_1 + 3u_2 \geq 48$$

Similarly, we find for B and C the following

$$\begin{aligned} 11u_1 + 15u_2 &\geq 165 \\ 25u_1 + 6u_2 &\geq 150 \end{aligned}$$

Convert to equality, and now use x instead of u since now we are converting to standard form

$$8x_1 + 3x_2 - x_3 = 48$$

Similarly, we find for B and C the following

$$\begin{aligned} 11x_1 + 15x_2 - x_4 &= 165 \\ 25x_1 + 6x_2 - x_5 &= 150 \end{aligned}$$

Now we write the above in the standard form

$$\begin{aligned} &\min c^T x \\ &Ax = b \end{aligned}$$

Or

$$\min \begin{bmatrix} 0.0012 & 0.001 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

$$\begin{bmatrix} 8 & 3 & -1 & 0 & 0 \\ 11 & 15 & 0 & -1 & 0 \\ 25 & 6 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 48 \\ 165 \\ 150 \end{bmatrix}$$

A graphical solution was found by plotting the three constraints. Since the extreme point must be at a vertex of the feasible region, we see that it is at $u^* = (4, 8)$ which is confirmed using Matlab LP in the second part.

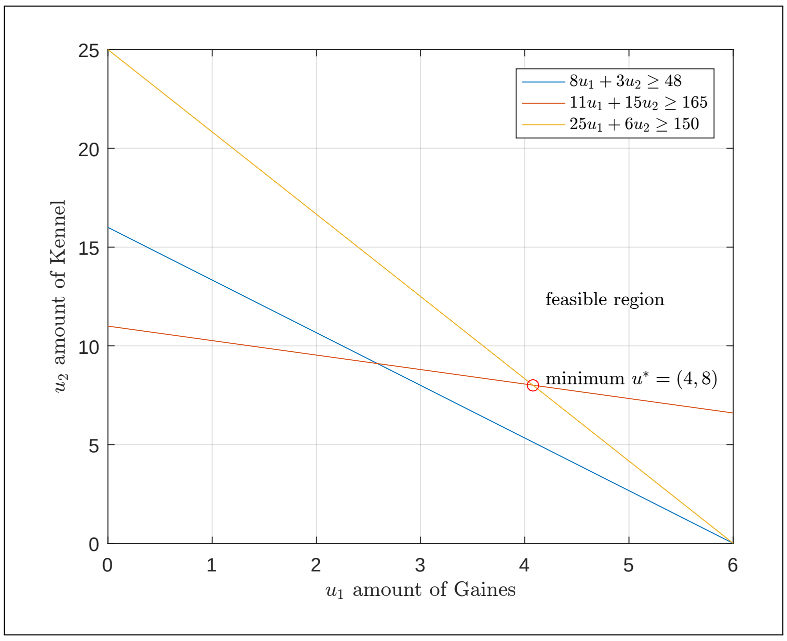


Figure 4.55: Graphical solution

Now contour lines are added to the above plot. Since the objective function is linear, the contour will be straight line, showing how $J(u)$ increases. The smallest value of $J(u)$ level set line which touches the first vertex of the feasible region will be the optimal point. Here is the result of the above plot, with contour lines added:

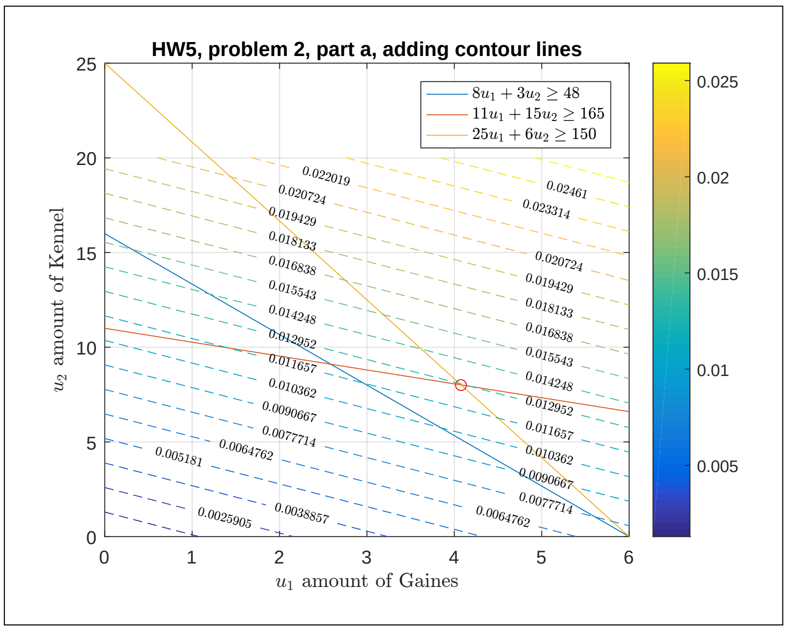


Figure 4.56: Graphical solution with contour lines added

```
clear; close;
x=0:6;
plot( x,(48-8*x)/3);
hold on;
plot( x,(165-11*x)/15);
hold on;
plot( x,(150-25*x)/6);
h=legend('$8u_1+3u_2\geq 48$','$11u_1+15u_2\geq 165$',...
        '$25u_1+6u_2\geq 150$');
set(h,'Interpreter','latex')
xlabel('$u_1$ amount of Gaines','Interpreter','latex');
ylabel('$u_2$ amount of Kennel','Interpreter','latex');
plot(4.077,8.0097,'ro');
text(4.2,8.3,'minimum $u^{\ast}=(4.077,8.0097)$',...
     'Interpreter','latex');
text(4.2,12.3,'feasible region','Interpreter','latex');
grid

%add contour lines
x1      = 0:0.1:6;
x2      = 0:0.1:20;
[X1,X2] = meshgrid(x1,x2);
Z = 0.0012*X1+0.001*X2;
[C,h]=contour(X1,X2,Z,20,'--');
clabel(C,h,'Fontsize',7,'interpreter','Latex');
title('HW5, problem 2, part a, adding contour lines')
colorbar
```

.

4.5.2.2 Part b

Matlab linprog was used to solve the above to find $x_1, x_2, x_3, x_4, x_5, x_6$. Here is the result for x^*

$$
\begin{aligned}
x_1 &= 4.0777\\
x_2 &= 8.0097\\
x_3 &= 8.6505\\
x_4 &= 0\\
x_5 &= 0
\end{aligned}
$$

Mapping this back to u, we see that $u_1 = x_1$ and $u_2 = x_2$. Hence the minimum cost in dollars is from (1)

$$
\begin{aligned}
J(\boldsymbol{u}) &= (0.0012)\,u_1 + (0.001)\,u_2\\
&= (0.0012)\,4.0777 + (0.001)\,8.0097\\
&= 0.01290\,3
\end{aligned}
$$

The above is the cost of 4 grams from Gaines and 8 grams from Kennel. The above basically says to buy twice as much from Kennel as from Gaines.

4.5.2.3 Source code for problem 2

```
c1=0.0012;
c2=0.001;
f=[c1,c2,0,0,0];
A=[8,3,-1,0,0;
   11,15,0,-1,0;
   25,6,0,0,-1];
b=[48,165,150];
[X,FVAL,EXITFLAG,OUTPUT]=linprog(f,[],[],A,b,zeros(size(f)),[])
```

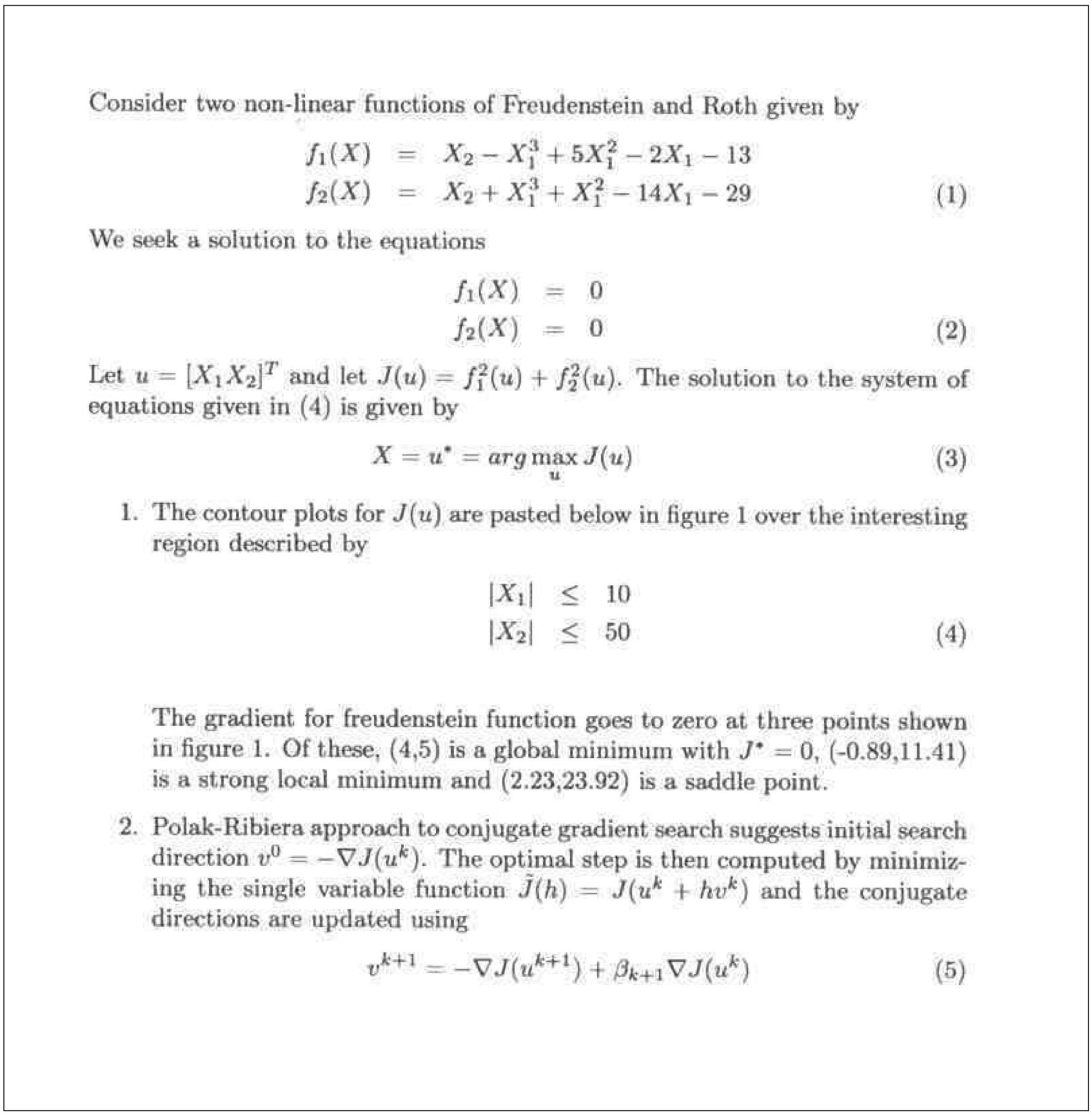
.

Result of above run

```
c1=0.0012;
```

```
c2=0.001;
f=[c1,c2,0,0,0];
A=[8,3,-1,0,0;
11,15,0,-1,0;
25,6,0,0,-1];
b=[48,165,150];
[X,FVAL,EXITFLAG,OUTPUT]=linprog(f,[],[],A,b,zeros(size(f)),[])
Optimization terminated.
X =
4.0777
8.0097
8.6505
0.0000
0.0000
FVAL =
0.0129
EXITFLAG =
1
OUTPUT =
iterations: 6
algorithm: 'interior-point-legacy'
cgiterations: 0
message: 'Optimization terminated.'
constrviolation: 8.5265e-14
firstorderopt: 4.0665e-10
```

4.5.3 HW 5 key solution



Consider two non-linear functions of Freudenstein and Roth given by

$$\begin{aligned} f_1(X) &= X_2 - X_1^3 + 5X_1^2 - 2X_1 - 13 \\ f_2(X) &= X_2 + X_1^3 + X_1^2 - 14X_1 - 29 \end{aligned} \tag{1}$$

We seek a solution to the equations

$$\begin{aligned} f_1(X) &= 0 \\ f_2(X) &= 0 \end{aligned} \tag{2}$$

Let $u = [X_1 X_2]^T$ and let $J(u) = f_1^2(u) + f_2^2(u)$. The solution to the system of equations given in (4) is given by

$$X = u^* = arg \max_u J(u) \tag{3}$$

1. The contour plots for $J(u)$ are pasted below in figure 1 over the interesting region described by

$$\begin{aligned} |X_1| &\leq 10 \\ |X_2| &\leq 50 \end{aligned} \tag{4}$$

The gradient for freudenstein function goes to zero at three points shown in figure 1. Of these, (4,5) is a global minimum with $J^* = 0$, (-0.89,11.41) is a strong local minimum and (2.23,23.92) is a saddle point.

2. Polak-Ribiera approach to conjugate gradient search suggests initial search direction $v^0 = -\nabla J(u^k)$. The optimal step is then computed by minimizing the single variable function $\tilde{J}(h) = J(u^k + hv^k)$ and the conjugate directions are updated using

$$v^{k+1} = -\nabla J(u^{k+1}) + \beta_{k+1} \nabla J(u^k) \tag{5}$$

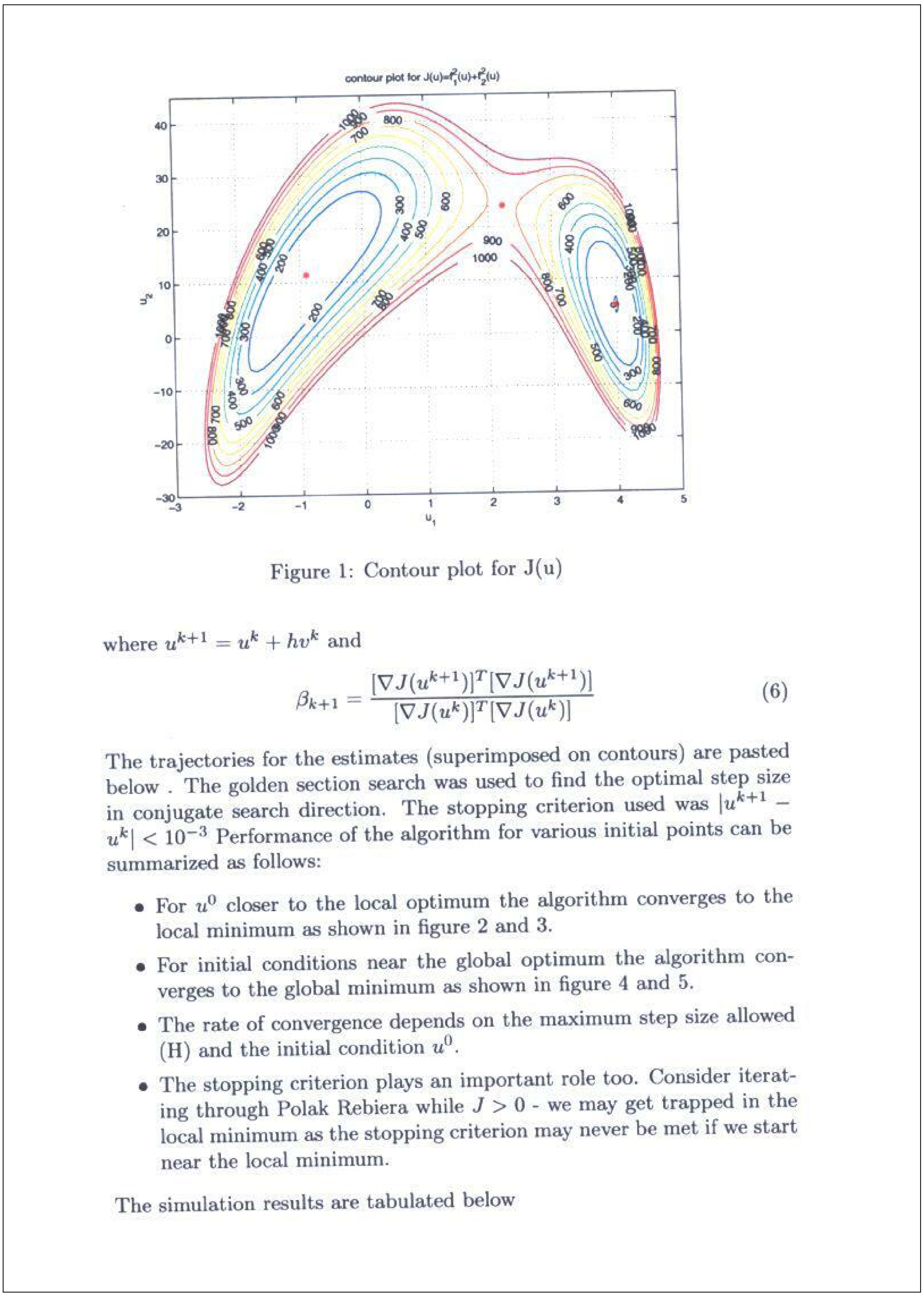

contour plot for J(u)=f$_1^2$(u)+f$_2^2$(u)
u$_2$
u$_1$
40
30
20
10
0
−10
−20
−30
−3
−2
−1
0
1
2
3
4
5
200
300
400
500
600
700
800
900
1000

Figure 1: Contour plot for J(u)

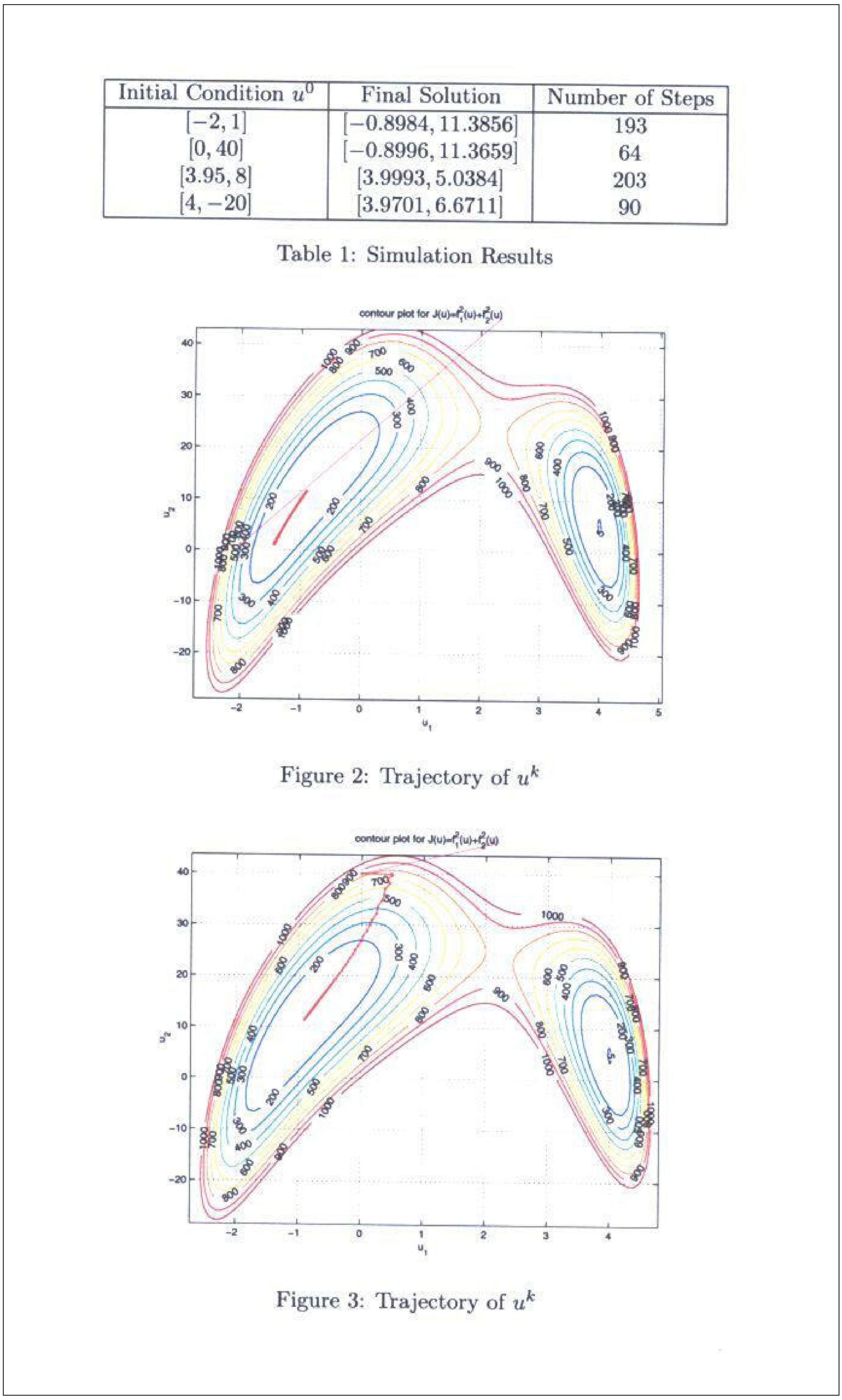
where $u^{k+1} = u^k + hv^k$ and

$$\beta_{k+1} = \frac{[\nabla J(u^{k+1})]^T[\nabla J(u^{k+1})]}{[\nabla J(u^k)]^T[\nabla J(u^k)]} \tag{6}$$

The trajectories for the estimates (superimposed on contours) are pasted below . The golden section search was used to find the optimal step size in conjugate search direction. The stopping criterion used was $|u^{k+1} - u^k| < 10^{-3}$ Performance of the algorithm for various initial points can be summarized as follows:

- For u^0 closer to the local optimum the algorithm converges to the local minimum as shown in figure 2 and 3.
- For initial conditions near the global optimum the algorithm converges to the global minimum as shown in figure 4 and 5.
- The rate of convergence depends on the maximum step size allowed (H) and the initial condition u^0.
- The stopping criterion plays an important role too. Consider iterating through Polak Rebiera while $J > 0$ - we may get trapped in the local minimum as the stopping criterion may never be met if we start near the local minimum.

The simulation results are tabulated below



Initial Condition u^0	Final Solution	Number of Steps
$[-2, 1]$	$[-0.8984, 11.3856]$	193
$[0, 40]$	$[-0.8996, 11.3659]$	64
$[3.95, 8]$	$[3.9993, 5.0384]$	203
$[4, -20]$	$[3.9701, 6.6711]$	90

Table 1: Simulation Results

contour plot for J(u)=f_1^2(u)+f_2^2(u)
u_2
u_1
40
30
20
10
0
-10
-20
-2
-1
0
1
2
3
4
5
200
300
400
500
600
700
800
900
1000

Figure 2: Trajectory of u^k

contour plot for J(u)=f_1^2(u)+f_2^2(u)
u_2
u_1
40
30
20
10
0
-10
-20
-2
-1
0
1
2
3
4
200
300
400
500
600
700
800
900
1000

Figure 3: Trajectory of u^k

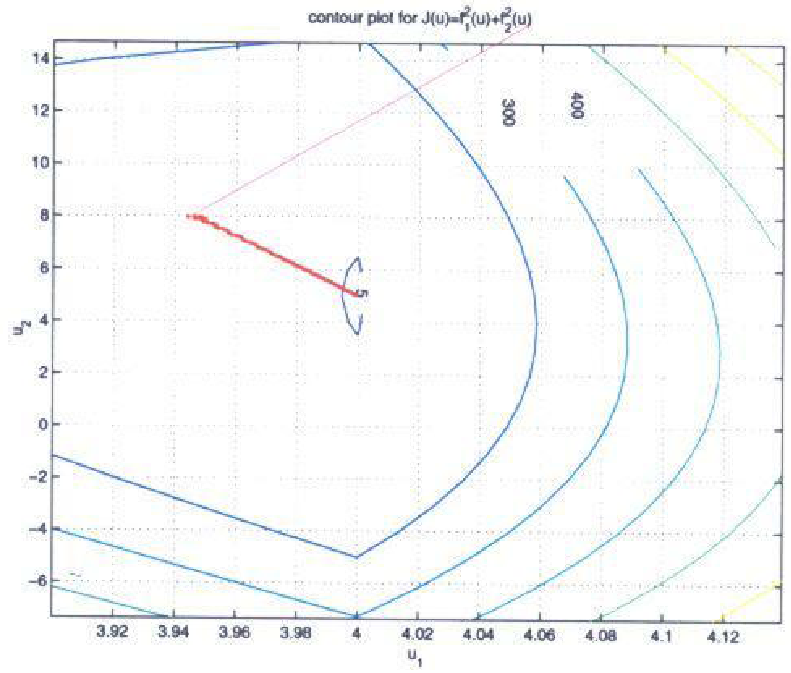


Figure 4: Trajectory of u^k

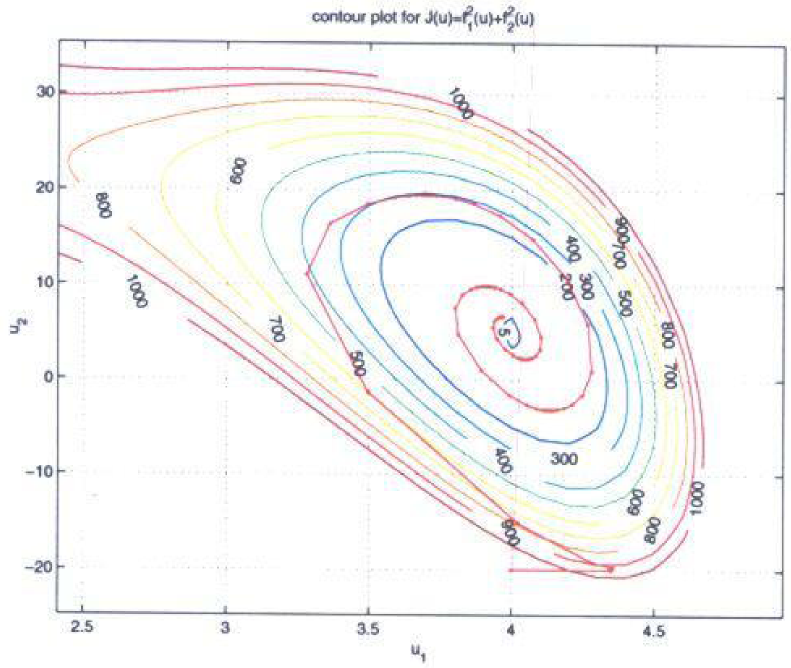


Figure 5: Trajectory of u^k

Barmish

ECE 719 – Solution Dog Food

Taking variables $u_1 = x_1 =$ number of grams of Gaines and $u_2 = x_2 =$ number of grams of Kennel Ration, we obtain inequalities for the three nutrient factors as follows:

$$8x_1 + 3x_2 \;\geq\; 48;$$

$$11x_1 + 15x_2 \;\geq\; 165;$$

$$25x_1 + 6x_2 \;\geq\; 150.$$

In addition, converting kilogram cost of food to cost per gram, we obtain objective function

$$J = 0.0012x_1 + 0.001x_2.$$

Now, to obtain a standard form LP, we add surplus variables and obtain the triple (A, b, c) given in Matlab form by

$$A = [8\;\;3\;\;-1\;\;0\;\;0\;\;0;\;11\;\;15\;\;0\;\;-1\;\;0;\;\;25\;\;6\;\;0\;\;0\;\;-1];$$

$$b = [48;\;165;\;150];$$

$$c = [0.0012;\;0.001;\;0;\;0;\;0].$$

Now running Matlab, the optimal solution is obtained as

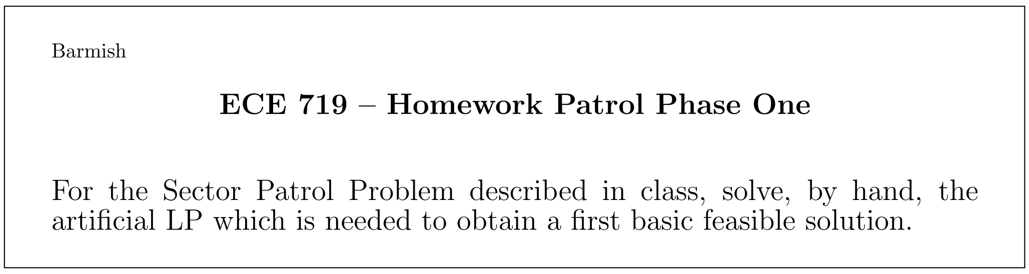
$$u_1^* \approx 4.1; \quad u_2^* \approx 8.01$$

with remaining surplus variables being zero and optimal cost in dollars per day being

$$J^* \approx 0.0129.$$

4.6 HW 6

4.6.1 Problem 1



Barmish

ECE 719 – Homework Patrol Phase One

For the Sector Patrol Problem described in class, solve, by hand, the artificial LP which is needed to obtain a first basic feasible solution.

Figure 4.57: problem 1 description

The patrol problem is given by

$$
\begin{aligned}
u_i &\geq 0\\
2u_1+2u_2 &\leq 0\\
2u_1+2u_2 &\geq 0\\
u_2 &\geq 1.5u_1
\end{aligned}
$$

And the objective function which we want to minimize is

$$J(u)=\frac{1}{30}u_1+\frac{1}{15}u_2$$

The above is the raw LP. We convert it to standard LP by introducing slack and surplus variable and rename the variables to x_i from u_i. Therefore the first table of the initial phase is

	x_1	x_2	x_3	x_4	x_5	b
row 1	2	2	1	0	0	10
row 2	2	2	0	−1	0	4
row 3	−1.5	1	0	0	−1	0
$J(x)$	$\frac{1}{30}$	$\frac{1}{15}$	0	0	0	0

4.6.1.1 Phase one

Since there are two surplus variables (these are the ones associated with −1 entries), we have to start with phase one LP. If there were no surplus variables (i.e. only slack variables), then we go directly to phase two. Phase one is only needed when there are surplus variables. We introduce two new artificial variables y_1, y_2 and an artificial objective $J(y)$ function which we want to minimize over y to zero,

$$\min_{y\geq 0, x\geq 0} J(y)=y_1+y_2$$

The first table of first phase is (where we now use the artificial objective function $J(y)$ in place of $J(x)$.)

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	b
row 1	2	2	1	0	0	0	0	10
row 2	2	2	0	−1	0	1	0	4
row 3	−1.5	1	0	0	−1	0	1	0
row 4, $J(y)$	0	0	0	0	0	1	1	0

We start by making last row canonical (this means we need to zero out last row entries under y_1, y_2 columns). Doing row(4) = row(4)-row(2) gives

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	b
row 1	2	2	1	0	0	0	0	10
row 2	2	2	0	−1	0	1	0	4
row 3	−1.5	1	0	0	−1	0	1	0
row 4, $J(y)$	−2	−2	0	1	0	0	1	−4

To zero out last row under y_2, row(4)=row(4)-row(3) applied to the above gives

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	b
row 1	2	2	1	0	0	0	0	10
row 2	2	2	0	−1	0	1	0	4
row 3	−1.5	1	0	0	−1	0	1	0
row 4, $J(y)$	−0.5	−3	0	1	1	0	0	−4

We see that the artificial basic feasible solution $J(y)$ is not optimal, since there are negative values on the last row. second column has the largest negative value in the last row, at −3. So we need to move this column in the basis vectors. To decide on the pivot row we look at ratio of $\frac{b}{\text{second column}}$ which gives $\min\left\{\frac{10}{2},\frac{4}{2},\frac{0}{1}\right\}=0$ which is associated with the third row. So the third row is the pivot row. Now we need to zero out all other entries in the second column. To make (1,2) zero: row 1 = row(1) - $(2\times\text{row}(3))$ gives

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	b
row 1	5	0	1	0	2	0	−2	10
row 2	2	2	0	−1	0	1	0	4
row 3 (pivot)	−1.5	1	0	0	−1	0	1	0
row 4, $J(y)$	−0.5	−3	0	1	1	0	0	−4

To make (2,2) zero, row (2) = row(2) - 2 row(3) gives

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	b
row 1	5	0	1	0	2	0	−2	10
row 2	5	0	0	−1	2	1	−2	4
row 3 (pivot)	−1.5	1	0	0	−1	0	1	0
row 4, $J(y)$	−0.5	−3	0	1	1	0	0	−4

Finally to make (4,2) entry zero, row(4)=row(4)+3 row(3) gives

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	b
row 1	5	0	1	0	2	0	−2	10
row 2	5	0	0	−1	2	1	−2	4
row 3 (pivot)	−1.5	1	0	0	−1	0	1	0
row 4, $J(y)$	−5	0	0	1	−2	0	3	−4

We see that the artificial basic feasible solution is still not optimal since there is negative values on last row. The most negative is in first column. This is the column to move in. Taking the ratio of $\frac{b}{\text{first column}}$ gives $\min\left\{\frac{10}{5},\frac{4}{5}\right\}=\frac{4}{5}$ (we do not divide by negative entries). This minimum is associated with second row. So the second row is the pivot row. We start by normalizing the second row (the pivot row) so that entry (2,1) is one (it is 5 now). Row(2) = row(2)/5

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	b
row 1	5	0	1	0	2	0	−2	10
row 2 (pivot)	1	0	0	$-\frac{1}{5}$	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{-2}{5}$	$\frac{4}{5}$
row 3	−1.5	1	0	0	−1	0	1	0
row 4, $J(y)$	−5	0	0	1	−2	0	3	−4

To make (1,1) entry zero, then row(1)=row(1)-5 row(2)

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	b
row 1	0	0	1	1	0	−1	0	6
row 2 (pivot)	1	0	0	$-\frac{1}{5}$	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{-2}{5}$	$\frac{4}{5}$
row 3	−1.5	1	0	0	−1	0	1	0
row 4, $J(y)$	−5	0	0	1	−2	0	3	−4

To make (1,3) zero, then row(3)=row(3)+1.5 row(2)

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	b
row 1	0	0	1	1	0	−1	0	6
row 2 (pivot)	1	0	0	$-\frac{1}{5}$	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{-2}{5}$	$\frac{4}{5}$
row 3	0	1	0	$-\frac{3}{10}$	$-\frac{2}{5}$	$\frac{3}{10}$	$\frac{2}{5}$	$\frac{6}{5}$
row 4, $J(y)$	−5	0	0	1	−2	0	3	−4

To make (4,1) zero, row(4)=row(4)+5 row(2)

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	b
row 1	0	0	1	1	0	−1	0	6
row 2 (pivot)	1	0	0	$-\frac{1}{5}$	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{-2}{5}$	$\frac{4}{5}$
row 3	0	1	0	$\frac{-3}{10}$	$\frac{-2}{5}$	$\frac{3}{10}$	$\frac{2}{5}$	$\frac{6}{5}$
row 4, $J(y)$	0	0	0	0	0	1	1	0

We have driven $J(y)$ to zero with no positive entries in last row. This completes phase one. Now we remove the last row and also remove the y_1, y_2 columns from the above table, and put back the $J(x)$ in its place in last row. This next tableau is the starting of phase 2.

4.6.1.2 Phase 2

	x_1	x_2	x_3	x_4	x_5	b
row 1	0	0	1	1	0	6
row 2	1	0	0	$-\frac{1}{5}$	$\frac{2}{5}$	$\frac{4}{5}$
row 3	0	1	0	$\frac{-3}{10}$	$\frac{-2}{5}$	$\frac{6}{5}$
$J(x)$	$\frac{1}{30}$	$\frac{1}{15}$	0	0	0	0

We see that all entries in the last row positive, therefore phase two is now complete. There is nothing to do in phase two. All the hard work was done in phase one. This is special case and we were lucky. Note: The text book says that we should now zero out the last row so that zeros appear under the basis columns. But I find this not needed, since it does not change the optimal x^*. We can always calculate $J(x)$ once we know x^*, and there is no need to have $J(x)$ show up in the bottom right corner of the tableau really. Therefore, I did not do this extra step as not needed.

Now we read out the solution from the above tableau

$$x = \begin{bmatrix} \frac{4}{5} \\ \frac{6}{5} \\ 6 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 1.2 \\ 6 \\ 0 \\ 0 \end{bmatrix}$$

To find the corresponding $J^*(u)$, since $x_1 = u_1$ and $x_2 = u_2$ and $J(u) = \frac{1}{30}u_1 + \frac{1}{15}u_2$, then

$$\begin{aligned} J^* &= \frac{1}{30}(0.8) + \frac{1}{15}(1.2) \\ &= 0.10667 \end{aligned}$$

4.6.1.3 Verification

Using Matlab linprog

```
function nma_HW6_problem_1
%Solves first problem, HW 6, ECE 719
%Nasser M. Abbasi

f=[1/30,1/15,0,0,0];
A=[2,2,1,0,0;
   2,2,0,-1,0;
   -1.5,1,0,0,-1];
b=[10,4,0];
[X,FVAL,EXITFLAG,OUTPUT]=linprog(f,[],[],A,b,zeros(size(f)),[])

end
```

The output from the above is

```
Optimization terminated.

X =
0.8000
1.2000
6.0000
0.0000
0.0000

FVAL =
0.1067

EXITFLAG =
1

OUTPUT =
iterations: 7
algorithm: 'interior-point-legacy'
cgiterations: 0
message: 'Optimization terminated.'
constrviolation: 8.8818e-16
firstorderopt: 5.0706e-12
```

Using my own `nma_simple.m` which prints all the intermediate tableau and solutions x during the search

```
f=[1/30,1/15,0,0,0];
A=[2,2,1,0,0;
   2,2,0,-1,0;
   -1.5,1,0,0,-1];
b=[10,4,0];
nma_simplex(A,b,f,true)
```

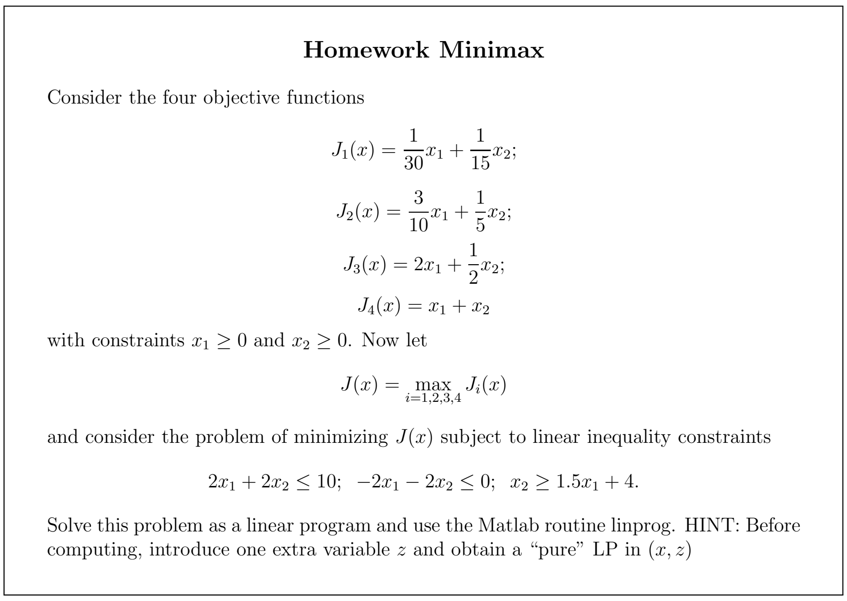
.

The output from the above is

```
>>>>Current tableau [phase one]
    2.0000    2.0000    1.0000         0         0    1.0000         0         0   10.0000
    2.0000    2.0000         0   -1.0000         0         0    1.0000         0    4.0000
   -1.5000    1.0000         0         0   -1.0000         0         0    1.0000         0
         0         0         0         0         0    1.0000    1.0000    1.0000         0
***********************
Current tableau [phase one]
    2.0000    2.0000    1.0000         0         0    1.0000         0         0   10.0000
    2.0000    2.0000         0   -1.0000         0         0    1.0000         0    4.0000
   -1.5000    1.0000         0         0   -1.0000         0         0    1.0000         0
   -2.5000   -5.0000   -1.0000    1.0000    1.0000         0         0         0         0

pivot row is 3
current basic feasible solution is
     0
     0
     0
     0
     0
    10
     4
     0
**********************
Current tableau [phase one]
    5.0000         0    1.0000         0    2.0000    1.0000         0   -2.0000   10.0000
    5.0000         0         0   -1.0000    2.0000         0    1.0000   -2.0000    4.0000
   -1.5000    1.0000         0         0   -1.0000         0         0    1.0000         0
  -10.0000         0   -1.0000    1.0000   -4.0000         0         0    5.0000         0

pivot row is 2
current basic feasible solution is
    0.8000
    1.2000
         0
         0
         0
    6.0000
         0
         0
**********************
Current tableau [phase one]
         0         0    1.0000    1.0000         0    1.0000   -1.0000         0    6.0000
    1.0000         0         0   -0.2000    0.4000         0    0.2000   -0.4000    0.8000
         0    1.0000         0   -0.3000   -0.4000         0    0.3000    0.4000    1.2000
         0         0   -1.0000   -1.0000         0         0    2.0000    1.0000    8.0000

pivot row is 1
current basic feasible solution is
    0.8000
    1.2000
    6.0000
         0
         0
         0
         0
         0
**********************
Current tableau [phase one]
         0         0    1.0000    1.0000         0    1.0000   -1.0000         0    6.0000
    1.0000         0         0   -0.2000    0.4000         0    0.2000   -0.4000    0.8000
         0    1.0000         0   -0.3000   -0.4000         0    0.3000    0.4000    1.2000
         0         0         0         0         0    1.0000    1.0000    1.0000   14.0000
**********************
Current tableau [phase two]
         0         0    1.0000    1.0000         0    6.0000
    1.0000         0         0   -0.2000    0.4000    0.8000
         0    1.0000         0   -0.3000   -0.4000    1.2000
    0.0333    0.0667         0         0         0         0

ans =
         0         0    1.0000    1.0000         0    6.0000
    1.0000         0         0   -0.2000    0.4000    0.8000
         0    1.0000         0   -0.3000   -0.4000    1.2000
    0.0333    0.0667         0         0         0         0
```

Which gives same answer.

4.6.2 Problem 2



Homework Minimax

Consider the four objective functions

$$J_1(x) = \frac{1}{30}x_1 + \frac{1}{15}x_2;$$

$$J_2(x) = \frac{3}{10}x_1 + \frac{1}{5}x_2;$$

$$J_3(x) = 2x_1 + \frac{1}{2}x_2;$$

$$J_4(x) = x_1 + x_2$$

with constraints $x_1 \geq 0$ and $x_2 \geq 0$. Now let

$$J(x) = \max_{i=1,2,3,4} J_i(x)$$

and consider the problem of minimizing $J(x)$ subject to linear inequality constraints

$$2x_1 + 2x_2 \leq 10; \quad -2x_1 - 2x_2 \leq 0; \quad x_2 \geq 1.5x_1 + 4.$$

Solve this problem as a linear program and use the Matlab routine linprog. HINT: Before computing, introduce one extra variable z and obtain a "pure" LP in (x, z)

Figure 4.58: problem 2 description

4.6.2.1 Initial Graphical view

This section shows different views of the problem. In the next section, the solution itself is given. Under each plot, the small code used to generate the plot is shown. First, the feasibility region given by the constraints is plotted.

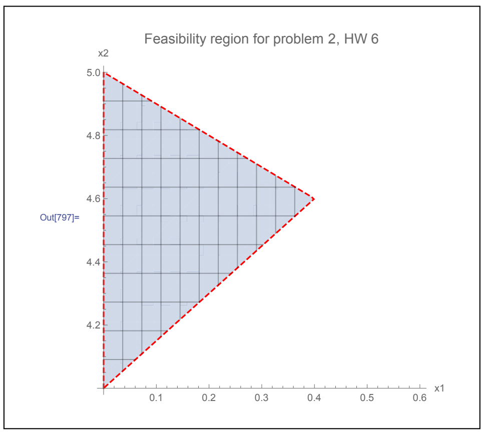


Figure 4.59: Region defined by constraints

```
RegionPlot[
2 x1 + 2 x2 <= 10 && -2 x1 - 2 x2 <= 0 && x2 >= 1.5 x1 + 4 && x1 >= 0 &&
x2 >= 0, {x1, 0, 0.6}, {x2, 4, 5}, BoundaryStyle -> {Red, Dashed},
Mesh -> 10, AxesLabel -> {x1, x2}, Axes -> True, Frame -> None,
PlotLabel -> Style["Feasibility region for problem 2, HW 6", 14]]
```

.

This plot shows each constraint, superimposed on top of the feasibility region.

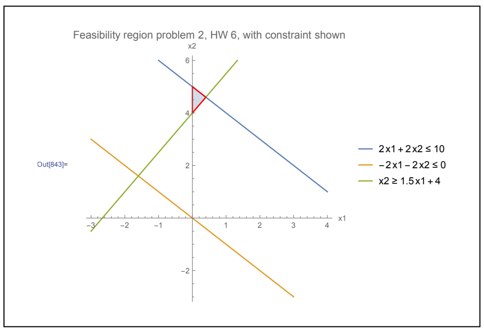


Figure 4.60: Region shown with constraints superimposed

```
p1 = RegionPlot[
2 x1 + 2 x2 <= 10 && -2 x1 - 2 x2 <= 0 && x2 >= 1.5 x1 + 4 && x1 >= 0 &&
x2 >= 0, {x1, 0, 0.6}, {x2, 4, 5}, BoundaryStyle -> {Red}, Mesh -> None,
AxesLabel -> {x1, x2}, Axes -> True, Frame -> None];
p2 = ContourPlot[{2 x1 + 2 x2 == 10, -2 x1 - 2 x2 == 0,
x2 == 1.5 x1 + 4}, {x1, -3, 4}, {x2, -3, 6},
PlotLegends -> {2 x1 + 2 x2 <= 10, -2 x1 - 2 x2 <= 0, x2 >= 1.5 x1 + 4}];
Show[p2, p1, AxesLabel -> {x1, x2}, Axes -> True, Frame -> None,
PlotRange -> All,
PlotLabel ->
Style["Feasibility region problem 2, HW 6, with constraint shown", 14]]
```

.

The following is 3D plot, showing the four objective functions

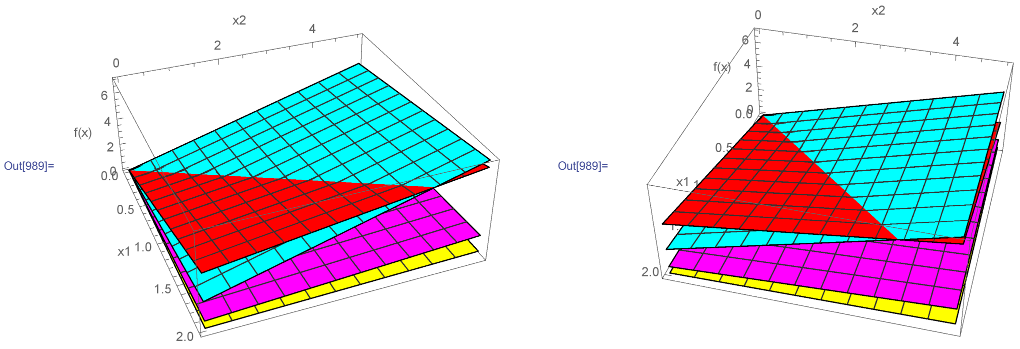


Figure 4.61: 3D plot of the four objective functions

```
funs = {1/30 x1 + 1/15 x2,3/10 x1 + 1/5 x2, 2 x1 + 1/2 x2,x1 + x2};
mesh = 10
g1 = Plot3D[funs[[1]], {x1, 0, 2}, {x2, 0, 5}, PlotStyle -> Yellow,
Mesh -> mesh, Lighting -> {{"Ambient", White}}];
g2 = Plot3D[funs[[2]], {x1, 0, 2}, {x2, 0, 5},PlotStyle -> Magenta,
Mesh -> mesh, Lighting -> {{"Ambient", White}}];
g3 = Plot3D[funs[[3]], {x1, 0, 2}, {x2, 0, 5},PlotStyle -> Red,
                   Mesh -> mesh, Lighting -> {{"Ambient", White}}];
g4 = Plot3D[funs[[4]], {x1, 0, 2}, {x2, 0, 5}, PlotStyle -> Cyan,
                  Mesh -> mesh, Lighting -> {{"Ambient", White}}];
Show[g1, g2, g3, g4, AxesLabel -> {x1, x2, "f(x)"}, PlotRange -> All,
                                         SphericalRegion -> True]
```

.

The following is same 3D plot, but now with the constraints added

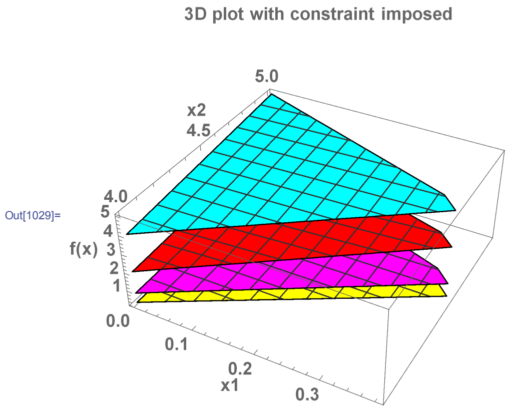


Figure 4.62: 3D plot of the four objective functions with constraint imposed

```
funs = {1/30 x1 + 1/15 x2, 3/10 x1 + 1/5 x2,2 x1 + 1/2 x2,x1 + x2};
mesh = 10;
reg = RegionFunction ->
```

```
Function[{x, y},
x >= 0 && y >= 0 && 2 x + 2 y <= 10 &&
                           -2 x - 2 y <= 0 && y >= 1.5 x + 4];
g1 = Plot3D[funs[[1]], {x1, 0, 2}, {x2, 0, 5},
         PlotStyle -> Yellow, Mesh -> mesh,
         Lighting -> {{"Ambient", White}}, Evaluate@reg];

g2 = Plot3D[funs[[2]], {x1, 0, 2}, {x2, 0, 5},
           PlotStyle -> Magenta, Mesh -> mesh,
           Lighting -> {{"Ambient", White}}, Evaluate@reg];

g3 = Plot3D[funs[[3]], {x1, 0, 2}, {x2, 0, 5},
       PlotStyle -> Red, Mesh -> mesh,
       Lighting -> {{"Ambient", White}}, Evaluate@reg];

g4 = Plot3D[funs[[4]], {x1, 0, 2}, {x2, 0, 5},
       PlotStyle -> Cyan, Mesh -> mesh,
       Lighting -> {{"Ambient", White}}, Evaluate@reg];

Show[g1, g2, g3, g4, AxesLabel -> {x1, x2, "f(x)"},
        PlotRange -> All, SphericalRegion -> True,
        PlotLabel -> Style["3D plot with constraint imposed", 14],
        BaseStyle -> {Bold, 14}]
```

.

The solution found is now added the above plot., which is $x_1 = 0, x_2 = 4$ with the min max value of $J(u) = 4$ marked with small red point below.

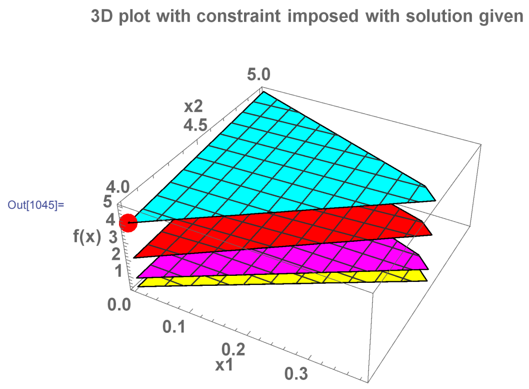


Figure 4.63: 3D plot of the four objective functions with constraint imposed with optimal solution

```
funs = {1/30 x1 + 1/15 x2, 3/10 x1 + 1/5 x2, 2 x1 + 1/2 x2, x1 + x2};
mesh = 10;
reg = RegionFunction ->
Function[{x, y},
x >= 0 && y >= 0 && 2 x + 2 y <= 10 && -2 x - 2 y <= 0 && y >= 1.5 x + 4];
g1 = Plot3D[funs[[1]], {x1, 0, 2}, {x2, 0, 5}, PlotStyle -> Yellow,
Mesh -> mesh, Lighting -> {{"Ambient", White}}, Evaluate@reg];
g2 = Plot3D[funs[[2]], {x1, 0, 2}, {x2, 0, 5}, PlotStyle -> Magenta,
Mesh -> mesh, Lighting -> {{"Ambient", White}}, Evaluate@reg];
g3 = Plot3D[funs[[3]], {x1, 0, 2}, {x2, 0, 5}, PlotStyle -> Red, Mesh -> mesh,
Lighting -> {{"Ambient", White}}, Evaluate@reg];
g4 = Plot3D[funs[[4]], {x1, 0, 2}, {x2, 0, 5}, PlotStyle -> Cyan,
Mesh -> mesh, Lighting -> {{"Ambient", White}}, Evaluate@reg];
Show[g1, g2, g3, g4,
Graphics3D[{Red, PointSize[.05], Point[{0, 4, 4}]}, Axes -> True],
AxesLabel -> {x1, x2, "f(x)"}, PlotRange -> All, SphericalRegion -> True,
PlotLabel ->
Style["3D plot with constraint imposed with solution given", 14],
BaseStyle -> {Bold, 14}]
```

.

4.6.2.2 Solution using Matlab linprog

The first step is to convert this multi-objective minimax problem with constraints, to a pure linear programming problem so that Matlab linprog can be used. Introducing extra variable z the problem can be written as

$$
\begin{aligned}
\min_{z} \quad & \tilde{J}(z) = z \\
s.t. \quad & z \geq J_i(x) \qquad i = 1 \cdots 4 \\
& 2x_1 + 2x_2 \leq 10 \\
& -2x_1 - 2x_2 \leq 0 \\
& x_2 \geq 1.5x_1 + 4
\end{aligned}
$$

Using the $J_i(x)$ given, the above becomes

$$
\begin{aligned}
\min_{z} \quad & \tilde{J}(z) = z \\
s.t. \quad & \frac{1}{30}x_1 + \frac{1}{15}x_2 - z \leq 0 \\
& \frac{3}{30}x_1 + \frac{1}{5}x_2 - z \leq 0 \\
& 2x_1 + \frac{1}{2}x_2 - z \leq 0 \\
& x_1 + x_2 - z \leq 0 \\
& 2x_1 + 2x_2 \leq 10 \\
& -2x_1 - 2x_2 \leq 0 \\
& x_2 \geq 1.5x_1 + 4
\end{aligned}
$$

To use Matlab linprog, we first convert the above to the standard form. We do this by introducing slack and surplus variables. The above becomes (added tilde on top of the x to make it clear which one variables are the original raw LP variables, and which is ones are the new variables).

$$
\begin{aligned}
\min_{z} \quad & \tilde{J}(z) = z \\
s.t. \quad & \frac{1}{30}x_1 + \frac{1}{15}x_2 - z + \tilde{x}_3 = 0 \\
& \frac{3}{30}x_1 + \frac{1}{5}x_2 - z + \tilde{x}_4 = 0 \\
& 2x_1 + \frac{1}{2}x_2 - z + \tilde{x}_5 = 0 \\
& x_1 + x_2 - z + \tilde{x}_6 = 0 \\
& 2x_1 + 2x_2 + \tilde{x}_7 = 10 \\
& -2x_1 - 2x_2 + \tilde{x}_8 = 0 \\
& -1.5x_1 + x_2 - \tilde{x}_9 = 4
\end{aligned}
$$

For $x, \tilde{x} \geq 0$. Hence $Ax = b$ becomes

$$
\begin{pmatrix}
\frac{1}{30} & \frac{1}{15} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
\frac{3}{30} & \frac{1}{5} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\
2 & \frac{1}{2} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\
2 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
-2 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
-1.5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0
\end{pmatrix}
\begin{pmatrix}
x_1 \\ x_2 \\ \tilde{x}_3 \\ \tilde{x}_4 \\ \tilde{x}_5 \\ \tilde{x}_6 \\ \tilde{x}_7 \\ \tilde{x}_8 \\ \tilde{x}_9 \\ z
\end{pmatrix}
=
\begin{pmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 10 \\ 0 \\ 4
\end{pmatrix}
$$

And $\min c^T x$ becomes

$$\min \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \tilde{x}_3 \\ \tilde{x}_4 \\ \tilde{x}_5 \\ \tilde{x}_6 \\ \tilde{x}_7 \\ \tilde{x}_8 \\ \tilde{x}_9 \\ z \end{pmatrix}$$

Therefore, using the above, we can write the first tableau table. Then use Matlab to solve it.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	z	b
row 1	$\frac{1}{30}$	$\frac{1}{15}$	1	0	0	0	0	0	0	−1	0
row 2	$\frac{3}{30}$	$\frac{1}{5}$	0	1	0	0	0	0	0	−1	0
row 3	2	$\frac{1}{2}$	0	0	1	0	0	0	0	−1	0
row 4	1	1	0	0	0	1	0	0	0	−1	0
row 5	2	2	0	0	0	0	1	0	0	0	10
row 6	−2	−2	0	0	0	0	0	1	0	0	0
row 7	−1.5	1	0	0	0	0	0	0	−1	0	4
$\tilde{J}(x,z)$	0	0	0	0	0	0	0	0	0	1	0

Below is the Matlab code used to solve the above. The solution is

$$\boxed{\begin{pmatrix} x_1 \\ x_2 \\ \tilde{x}_3 \\ \tilde{x}_4 \\ \tilde{x}_5 \\ \tilde{x}_6 \\ \tilde{x}_7 \\ \tilde{x}_8 \\ \tilde{x}_9 \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 3.7333 \\ 3.2 \\ 2 \\ 0 \\ 2 \\ 8 \\ 0 \\ 4 \end{pmatrix}}$$

And minimum of the maximum of $J_i(x)$ is

$$\boxed{\min \max J_i(x) = 4}$$

```
function nma_HW6_problem_2
%Solves second problem, HW 6, ECE 719
%Nasser M. Abbasi

f = [0 0 0 0 0 0 0 0 0 1];
A = [1/30  1/15 1  0  0  0  0  0  0  -1;
    1/30  1/5  0  1  0  0  0  0  0  -1;
    2     1/2  0  0  1  0  0  0  0  -1;
    1     1    0  0  0  1  0  0  0  -1;
    2     2    0  0  0  0  1  0  0   0;
```

```
   -2   -2    0  0  0  0  0  1  0   0;
  - 1.5  1    0  0  0  0  0  0  -1  0];
b=[0 0 0 0 10 0 4];

[X,FVAL,EXITFLAG,OUTPUT]=linprog(f,[],[],A,b,zeros(size(f)),[])
end
```

The output from the above is

```
>> nma_HW6_problem_2
Optimization terminated.
X =
0.0000
4.0000
3.7333
3.2000
2.0000
0.0000
2.0000
8.0000
0.0000
4.0000
FVAL =
4.0000
EXITFLAG =
1
OUTPUT =
iterations: 7
algorithm: 'interior-point-legacy'
cgiterations: 0
message: 'Optimization terminated.'
constrviolation: 1.4211e-14
firstorderopt: 7.8638e-13
```

This was also solved using Mathematica. Here is the result, which confirms the above as well. This command is from Mathematica help, and used it to apply to this problem:

```
SetAttributes[FindMinMax, HoldAll];
FindMinMax[{f_Max, cons_}, vars_, opts___?OptionQ] :=
With[{res = iFindMinMax[{f, cons}, vars, opts]}, res /; ListQ[res]];
iFindMinMax[{ff_Max, cons_}, vars_, opts___?OptionQ] :=
Module[{z, res, f = List @@ ff},
res = FindMinimum[{z, (And @@ cons) && (And @@ Thread[z >= f])},
Append[Flatten[{vars}, 1], z], opts];
If[ListQ[res], {z /. res[[2]], Thread[vars -> (vars /. res[[2]])]}]];
FindMinMax[{Max[{1/30 x + 1/15 y, 3/10 x + 1/5 y, 2 x + 1/2 y, x + y}], {x >=
0, y >= 0, 2 x + 2 y <= 10, -2 x - 2 y <= 0, y >= 1.5 x + 4}}, {x, y}]
```

.

And the output is

```
{4., {x -> 0., y -> 4.}}
```

Which is the same.

4.6.3 Problem 3

Barmish

ECE 719 – Homework Diet

The table on the page to follow comes from a bestseller of the sixties, "Let's Eat to Keep Fit," by Adelle Davis. Let u_i, per labelling in the table, be the number of measures (quarts, cups cubes, etc.) of food i to be consumed. Now formulate a standard form linear programming problem to minimize the total cost of daily feeding while satisfying the following conditions:

(i) Minimum and maximum daily allowances should be met; see table.

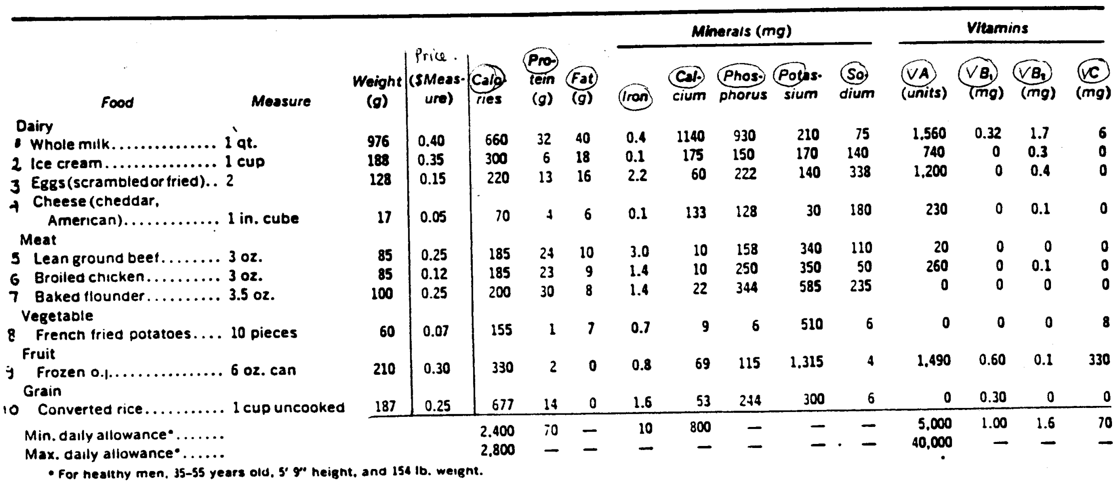
(ii) Daily potassium intake should be within 15% of sodium intake.

(ii) Daily calcium intake should be at least 75% of daily phosphorous intake.

(a) Solve the LP above, indicating both the optimal solution and optimal cost.

(b) Davis recommends that greater amounts of fat in the diet should be accompanied by greater vitamin B intake. To incorporate such a constraint let the amount of B_1 and B_2 be γ(the amount of fat). Now, with this added constraint, re-run your LP for various values of γ.

Summarize all your work with appropriate commentary.



			Price ·				Minerals (mg)					Vitamins			
Food	Measure	Weight (g)	($Measure)	Calories	Protein (g)	Fat (g)	Iron	Calcium	Phosphorus	Potassium	Sodium	VA (units)	VB_1 (mg)	VB_2 (mg)	VC (mg)
Dairy															
1 Whole milk...............	1 qt.	976	0.40	660	32	40	0.4	1140	930	210	75	1,560	0.32	1.7	6
2 Ice cream................	1 cup	188	0.35	300	6	18	0.1	175	150	170	140	740	0	0.3	0
3 Eggs(scrambled or fried)..	2	128	0.15	220	13	16	2.2	60	222	140	338	1,200	0	0.4	0
4 Cheese (cheddar, American).............	1 in. cube	17	0.05	70	4	6	0.1	133	128	30	180	230	0	0.1	0
Meat															
5 Lean ground beef........	3 oz.	85	0.25	185	24	10	3.0	10	158	340	110	20	0	0	0
6 Broiled chicken..........	3 oz.	85	0.12	185	23	9	1.4	10	250	350	50	260	0	0.1	0
7 Baked flounder..........	3.5 oz.	100	0.25	200	30	8	1.4	22	344	585	235	0	0	0	0
Vegetable															
8 French fried potatoes....	10 pieces	60	0.07	155	1	7	0.7	9	6	510	6	0	0	0	8
Fruit															
9 Frozen o.j................	6 oz. can	210	0.30	330	2	0	0.8	69	115	1,315	4	1,490	0.60	0.1	330
Grain															
10 Converted rice...........	1 cup uncooked	187	0.25	677	14	0	1.6	53	244	300	6	0	0.30	0	0
Min. daily allowance*.......				2,400	70	—	10	800	—	—	—	5,000	1.00	1.6	70
Max. daily allowance*......				2,800	—	—	—	—	—	—	—	40,000	—	—	—

* For healthy men, 35–55 years old, 5′ 9″ height, and 154 lb. weight.

Figure 4.64: problem 3 description

4.6.3.1 Part(a)

The variables are given in this table

variable	description	cost per unit	calories	potassium (mg)	sodium (mg)	calcium (mg)	phosphorus (mg)
u_1	whole milk	0.4	660	210	75	1140	930
u_2	ice cream	0.35	300	170	140	175	150
u_3	Eggs	0.15	220	140	338	60	222
u_4	Cheese	0.05	70	30	180	133	128
u_5	Beef	0.25	185	340	110	10	158
u_6	Broiled chicken	0.12	185	350	50	10	250
u_7	Baked vegetable	0.25	200	585	235	22	344
u_8	French fried	0.07	155	510	6	9	6
u_9	Frozen Grain	0.30	330	1315	4	69	115
u_{10}	Rice	0.25	677	300	6	53	244

Since the goal is to minimize the total cost of daily feeding, then

$$J(u) = 0.4u_1 + 0.35u_2 + 0.15u_3 + 0.05u_4 + 0.25u_5 + 0.12u_6 + 0.25u_7 + 0.07u_8 + 0.30u_9 + 0.25u_{10}$$

To be able to express the constraints, we need to first convert the minerals to same units used in $J(u)$. For example, one u_1 is one qt, which is 976 grams. The mineral calcium is 1.14 gram, therefore in units of u_1 it becomes $\frac{1.114}{976}u_1$, the same for all other minerals. Hence the above table becomes (where now each mineral is fraction of unit)

variable	description	cost per unit	calories	potassium)	sodium (mg)	calcium (mg)	phosphorus (mg)
u_1	whole milk	0.4	660	$\frac{0.210}{976}$	$\frac{0.075}{976}$	$\frac{1.140}{976}$	$\frac{0.930}{976}$
u_2	ice cream	0.35	300	$\frac{0.17}{188}$	$\frac{0.14}{188}$	$\frac{0.175}{188}$	$\frac{0.15}{188}$
u_3	Eggs	0.15	220	$\frac{0.14}{128}$	$\frac{0.338}{128}$	$\frac{0.06}{128}$	$\frac{0.222}{128}$
u_4	Cheese	0.05	70	$\frac{0.03}{17}$	$\frac{0.18}{17}$	$\frac{0.133}{17}$	$\frac{0.128}{17}$
u_5	Beef	0.25	185	$\frac{0.34}{85}$	$\frac{0.11}{85}$	$\frac{0.01}{85}$	$\frac{0.158}{85}$
u_6	Broiled chicken	0.12	185	$\frac{0.35}{85}$	$\frac{0.05}{85}$	$\frac{0.01}{85}$	$\frac{0.25}{85}$
u_7	Baked vegetable	0.25	200	$\frac{0.585}{100}$	$\frac{0.235}{100}$	$\frac{0.022}{100}$	$\frac{0.344}{100}$
u_8	French fried	0.07	155	$\frac{0.51}{60}$	$\frac{0.006}{60}$	$\frac{0.009}{60}$	$\frac{0.006}{60}$
u_9	Frozen Grain	0.30	330	$\frac{1.315}{210}$	$\frac{0.004}{210}$	$\frac{0.069}{210}$	$\frac{0.115}{210}$
u_{10}	Rice	0.25	677	$\frac{0.3}{187}$	$\frac{0.006}{187}$	$\frac{0.053}{187}$	$\frac{0.244}{187}$

Now we formulate each constraint. Let A be the total daily calories given by

$$A = 660u_1 + 300u_2 + 220u_3 + 70u_4 + 185u_5 + 185u_6 + 200u_7 + 155u_8 + 330u_9 + 677u_{10}$$

Hence constraint (i) in the problem becomes

$$\begin{aligned} A &\geq 2400 \\ A &\leq 2800 \end{aligned}$$

For formulating constraint (ii), let B be the daily potassium and C be the daily sodium

$$\begin{aligned} B &= \frac{0.210}{976}u_1 + \frac{0.17}{188}u_2 + \frac{0.14}{128}u_3 + \frac{0.03}{17}u_4 + \frac{0.34}{85}u_5 + \frac{0.35}{85}u_6 + \frac{0.585}{100}u_7 + \frac{0.51}{60}u_8 + \frac{1.315}{210}u_9 + \frac{0.3}{187}u_{10} \\ C &= \frac{0.075}{976}u_1 + \frac{0.14}{188}u_2 + \frac{0.338}{128}u_3 + \frac{0.18}{17}u_4 + \frac{0.11}{85}u_5 + \frac{0.05}{85}u_6 + \frac{0.235}{100}u_7 + \frac{0.006}{60}u_8 + \frac{0.004}{210}u_9 + \frac{0.006}{187}u_{10} \end{aligned}$$

Hence constraint (ii) in the problem becomes

$$\begin{aligned} B &\leq 1.15C \\ B &\geq 0.85C \end{aligned}$$

For formulating constraint (iii), let D be the daily calcium and E be the daily phosphorus

$$\begin{aligned} D &= \frac{1.140}{976}u_1 + \frac{0.175}{188}u_2 + \frac{0.06}{128}u_3 + \frac{0.133}{17}u_4 + \frac{0.01}{85}u_5 + \frac{0.01}{85}u_6 + \frac{0.022}{100}u_7 + \frac{0.009}{60}u_8 + \frac{0.069}{210}u_9 + \frac{0.053}{187}u_{10} \\ E &= \frac{0.930}{976}u_1 + \frac{0.15}{188}u_2 + \frac{0.222}{128}u_3 + \frac{0.128}{17}u_4 + \frac{0.158}{85}u_5 + \frac{0.25}{85}u_6 + \frac{0.344}{100}u_7 + \frac{0.006}{60}u_8 + \frac{0.115}{210}u_9 + \frac{0.244}{187}u_{10} \end{aligned}$$

Hence constraint (iii) in the problem becomes

$$D \geq 0.75E$$

Therefore the LP problem is

$$\begin{aligned} \min \quad & J(u) \\ s.t. \quad & A \geq 2400 \\ & A \leq 2800 \\ & B \leq 1.15C \\ & B \geq 0.85C \\ & D \geq 0.75E \end{aligned}$$

Converting to standard form and using x_i instead of u_i the above becomes

$$\begin{aligned} \min \quad & J(x) = 0.4x_1 + 0.35x_2 + 0.15x_3 + 0.05x_4 + 0.25x_5 + 0.12x_6 \\ & +0.25x_7 + 0.07x_8 + 0.30x_9 + 0.25x_{10} \\ \text{subject to} \quad & A - \tilde{x}_{11} = 2400 \\ & A + \tilde{x}_{12} = 2800 \\ & B + \tilde{x}_{13} - 1.15C = 0 \\ & B - \tilde{x}_{14} - 0.85C = 0 \\ & D - \tilde{x}_{15} - 0.75E = 0 \end{aligned}$$

In full form, the above is

$$\begin{aligned} \min \quad & J(x) = 0.4x_1 + 0.35x_2 + 0.15x_3 + 0.05x_4 + 0.25x_5 + 0. \\ s.t. \quad & 660x_1 + 300x_2 + 220x_3 + 70x_4 + 185x_5 + 185x_6 + 200 \\ & 660x_1 + 300x_2 + 220x_3 + 70x_4 + 185x_5 + 185x_6 + 200 \\ & \frac{0.210}{976}x_1 + \frac{0.17}{188}x_2 + \frac{0.14}{128}x_3 + \frac{0.03}{17}x_4 + \frac{0.34}{85}x_5 + \frac{0.35}{85}x_6 + \frac{0.585}{100}x_7 + \frac{0.51}{60}x_8 + \frac{1.315}{210}x_9 + \frac{0.3}{187}x_{10} + \tilde{x}_{13} - 1.15 \Big(\\ & \frac{0.210}{976}x_1 + \frac{0.17}{188}x_2 + \frac{0.14}{128}x_3 + \frac{0.03}{17}x_4 + \frac{0.34}{85}x_5 + \frac{0.35}{85}x_6 + \frac{0.585}{100}x_7 + \frac{0.51}{60}x_8 + \frac{1.315}{210}x_9 + \frac{0.3}{187}x_{10} - \tilde{x}_{14} - 0.85 \Big(\\ & \frac{1.140}{976}x_1 + \frac{0.175}{188}x_2 + \frac{0.06}{128}x_3 + \frac{0.133}{17}x_4 + \frac{0.01}{85}x_5 + \frac{0.01}{85}x_6 + \frac{0.022}{100}x_7 + \frac{0.009}{60}x_8 + \frac{0.069}{210}x_9 + \frac{0.053}{187}x_{10} - \tilde{x}_{15} - 0.75 \Big(\end{aligned}$$

Simplifying gives

$$\begin{aligned} \min \quad & J(x) = 0.4x_1 + 0.35x_2 + 0.15x_3 + 0.05x_4 + 0.25x_5 + 0.12x_6 + 0.25 \\ s.t. \quad & 660x_1 + 300x_2 + 220x_3 + 70x_4 + 185x_5 + 185x_6 + 200x_7 + 155x_8 + \\ & 660x_1 + 300x_2 + 220x_3 + 70x_4 + 185x_5 + 185x_6 + 200x_7 + 155x_8 + \\ & 1.268 \times 10^{-4}x_1 + 4.787 \times 10^{-5}x_2 - 1.943 \times 10^{-3}x_3 - 1.041 \times 10^{-2}x_4 + 2.512 \times 10^{-3}x_5 + 3.441 \times 10^{-3}x_6 + 3. \\ & 1.499 \times 10^{-4}x_1 + 2.713 \times 10^{-4}x_2 - 1.151 \times 10^{-3}x_3 - 7.235 \times 10^{-3}x_4 + 0.003x_5 + 3.618 \times 10^{-3}x_6 + 3.853 \times \\ & 4.534 \times 10^{-4}x_1 + 3.325 \times 10^{-4}x_2 - 8.32 \times 10^{-4}x_3 + 2.176 \times 10^{-3}x_4 - 1.277 \times 10^{-3}x_5 - 2.088 \times 10^{-3}x_6 - \end{aligned}$$

Therefore $c^T x$ becomes

$$\min c^T x = \begin{pmatrix} 0.4 & 0.35 & 0.15 & 0.05 & 0.25 & 0.12 & 0.25 & 0.07 & 0.3 & 0.25 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ \tilde{x}_{11} \\ \tilde{x}_{12} \\ \tilde{x}_{13} \\ \tilde{x}_{14} \\ \tilde{x}_{15} \end{pmatrix}$$

subject to $Ax = b$

$$\begin{pmatrix} 600 & 300 & 220 & 70 & 185 & 185 & 200 & 55 & 330 & 677 & -1 & 0 & 0 & 0 & 0 \\ 600 & 300 & 220 & 70 & 185 & 185 & 200 & 55 & 330 & 677 & 0 & 1 & 0 & 0 & 0 \\ 1.267\times 10^{-4} & 4.787\times 10^{-5} & -1.943\times 10^{-3} & -1.041\times 10^{-2} & 2.512\times 10^{-3} & 3.441\times 10^{-3} & 3.148\times 10^{-3} & 8.385\times 10^{-3} & 0.006 & 1.567\times 10^{-3} & 0 & 0 & 1 & 0 & 0 \\ 1.498\times 10^{-4} & 2.712\times 10^{-4} & -1.150\times 10^{-3} & -7.235\times 10^{-3} & 0.003 & 3.617\times 10^{-3} & 3.852\times 10^{-3} & 8.415\times 10^{-3} & 6.245\times 10^{-3} & 1.577\times 10^{-3} & 0 & 0 & 0 & -1 & 0 \\ 4.534\times 10^{-4} & 3.324\times 10^{-4} & -8.320\times 10^{-4} & 2.176\times 10^{-3} & -1.276\times 10^{-3} & -2.088\times 10^{-3} & -0.002 & 7.5\times 10^{-5} & -8.214\times 10^{-5} & -6.952\times 10^{-4} & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ \tilde{x}_{11} \\ \tilde{x}_{12} \\ \tilde{x}_{13} \\ \tilde{x}_{14} \\ \tilde{x}_{15} \end{pmatrix} = \begin{pmatrix} 2400 \\ 2800 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The above is now solved using Matlab linprog. The following is the result, followed by the Matlab code used.

$$x_{optimal} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ \tilde{x}_{11} \\ \tilde{x}_{12} \\ \tilde{x}_{13} \\ \tilde{x}_{14} \\ \tilde{x}_{15} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1.0792 \\ 0 \\ 0 \\ 0 \\ 0.2889 \\ 0 \\ 3.41 \\ 0 \\ 400 \\ 0.0035 \\ 0 \\ 0 \end{pmatrix}$$

In terms of raw u_i variables, the above says to buy [2]

$$\boxed{\text{1.0792 cube of cheese, and 2.889 pieces of french fried and 3.41 rice cups}}$$

[2] This seems to meet the requirements, but it does not look like a healthy diet actually?

4.6.3.1.1 Verification:

Calories from the above is

$$(1.0792)(70) + (0.2889)(155) + (3.41)(677) = 2428.9$$

Which is within daily allowance OK. Potassium from the above is

$$(1.0792)\left(\frac{0.03}{17}\right) + (0.2889)\left(\frac{0.51}{60}\right) + (3.41)\left(\frac{0.3}{187}\right) = 9.831 \text{ mg}$$

While sodium from the above is

$$(1.0792)\left(\frac{0.18}{17}\right) + (0.2889)\left(\frac{0.006}{60}\right) + (3.41)\left(\frac{0.006}{187}\right) = 10.1565 \text{ mg}$$

Hence Potassium is within 15% of sodium OK. Daily calcium from above is

$$(1.0792)\left(\frac{0.133}{17}\right) + (0.2889)\left(\frac{0.009}{60}\right) + (3.41)\left(\frac{0.053}{187}\right) = 9.453 \text{ mg}$$

While daily phosphorus is

$$(1.0792)\left(\frac{0.128}{17}\right) + (0.2889)\left(\frac{0.006}{60}\right) + (3.41)\left(\frac{0.244}{187}\right) = 12.6 \text{ mg}$$

Hence Daily calcium is at least 75% of daily phosphorus OK. All verified OK. The corresponding optimal daily cost is

$$\boxed{J^* = 0.9267 \text{ dollars}}$$

```
function nma_HW6_problem_3
%Solves third problem, HW 6, ECE 719
%Nasser M. Abbasi

c = [0.4 0.35 0.15 0.05 0.25 0.12 0.25 0.07 0.3 0.25 0 0 0 0 0];
A = [600  300 220 70 185 185 200 55 330 677 -1 0 0 0 0;
     600  300 220 70 185 185 200 55 330 677 0 1 0 0 0;
     1.2679*10^(-4),4.7872*10^(-5),-1.9430*10^(-3),  ...
     -1.0412*10^(-2),...
     2.5118*10^(-3),3.4412*10^(-3),3.1475*10^(-3),8.385*10^(-3),...
     0.00624,1.5674*10^(-3),0, 0, 1, 0, 0;
     1.4985*10^(-4),2.7128*10^(-4),-1.1508*10^(-3),...
     -7.2353*10^(-3),...
     0.0029,3.6176*10^(-3),3.8525*10^(-3),8.415*10^(-3),...
     6.2457*10^(-3),...
     1.577*10^(-3),0, 0, 0, -1, 0;
     4.5338*10^(-4),3.3245*10^(-4),-8.3203*10^(-4),...
      2.1765*10^(-3),...
     -1.2765*10^(-3),-2.0882*10^(-3),-0.00236,7.5*10^(-5),...
     -8.2143*10^(-5),...
     -6.9519*10^(-4), 0, 0, 0, 0, -1];
b=[2400 2800 0 0 0];
options = optimset('LargeScale','off','Simplex','on');
[X,FVAL,EXITFLAG,OUTPUT]=linprog(c,[],[],A,b,zeros(size(c)),...
                                                  [],[],options)
end
```

The output from the above is

```
Optimization terminated.
X =
0
0
0
1.0792
0
0
0
0.2889
0
3.4100
```

```
0
400.0000
0.0035
0
0
FVAL =
0.9267
EXITFLAG =
1
OUTPUT =
iterations: 0
algorithm: 'simplex'
cgiterations: []
message: 'Optimization terminated.'
constrviolation: 4.3368e-19
firstorderopt: 5.5511e-17
```

4.6.3.2 part(b)

Now we add a new constraint, which is $B_1 + B_2 = \gamma(fat)$. In terms of u_i, this becomes

$$B_1 = \frac{0.00032}{976}u_1 + \frac{0.0006}{210}u_9 + \frac{0.0003}{187}u_{10}$$

And

$$B_2 = \frac{0.0017}{976}u_1 + \frac{0.0003}{188}u_2 + \frac{0.0004}{128}u_3 + \frac{0.0001}{17}u_4 + \frac{0.0001}{85}u_6 + \frac{0.0001}{210}u_9$$

And

$$fat = \frac{40}{976}u_1 + \frac{18}{188}u_2 + \frac{16}{128}u_3 + \frac{6}{17}u_4 + \frac{10}{85}u_5 + \frac{9}{85}u_6 + \frac{8}{100}u_7 + \frac{7}{60}u_8$$

Hence the new constraint is

$$(B_1 + B_2) - \gamma\left(fat\right) = 0$$

Converting to standard LP form, the above becomes

$$\left(2.06976\times 10^{-6} - \frac{5}{122}\gamma\right)x_1 + \left(1.59574\times 10^{-6} - \frac{9}{94}\gamma\right)x_2 + \left(3.125\times 10^{-6} - \frac{1}{8}\gamma\right)x_3 + \left(5.88235\times 10^{-6} - \frac{6}{17}\gamma\right)x_4$$
$$-\frac{2}{17}\gamma x_5 + \left(1.17647\times 10^{-6} - \frac{9}{85}\gamma\right)x_6 - \frac{2}{25}\gamma x_7 - \frac{7}{60}\gamma x_8 + 3.33333\times 10^{-6}x_9 + 1.604\times 10^{-6}\gamma x_{10} = 0$$

Now the new $Ax = b$ becomes ($c^T x$ do not change from part(a) and remain the same).

$$\begin{pmatrix} 600 & 300 & 220 & 70 & 185 & 185 & 200 & 55 & 330 & 677 & -1 & 0 & 0 & 0 & 0 \\ 600 & 300 & 220 & 70 & 185 & 185 & 200 & 55 & 330 & 677 & 0 & 1 & 0 & 0 & 0 \\ 1.268\times 10^{-4} & 4.787\times 10^{-5} & -1.943\times 10^{-3} & -1.041\times 10^{-2} & 2.512\times 10^{-3} & 3.441\times 10^{-3} & 3.148\times 10^{-3} & 8.385\times 10^{-3} & 0.006 & 1.567\times 10^{-3} & 0 & 0 & 1 & 0 & 0 \\ 1.499\times 10^{-4} & 2.713\times 10^{-4} & -1.151\times 10^{-3} & -7.235\times 10^{-3} & 0.003 & 3.618\times 10^{-3} & 3.853\times 10^{-3} & 8.415\times 10^{-3} & 6.2467\times 10^{-3} & 1.577\times 10^{-3} & 0 & 0 & 0 & -1 & 0 \\ 4.534\times 10^{-4} & 3.325\times 10^{-4} & -8.32\times 10^{-4} & 2.177\times 10^{-3} & -1.277\times 10^{-3} & -2.088\times 10^{-3} & -0.003 & 7.5\times 10^{-5} & -8.214\times 10^{-5} & -6.952\times 10^{-4} & 0 & 0 & 0 & 0 & -1 \\ 2.06976\times 10^{-6} - 0.03279\gamma & \left(1.59574\times 10^{-6} - 0.031915\gamma\right) & \left(3.125\times 10^{-6} - 0.1016\gamma\right) & \left(5.88235\times 10^{-6} - 0.2353\gamma\right) & -0.28235\gamma & \left(1.17647\times 10^{-6} - 0.27058\gamma\right) & -0.3\gamma & -0.01667\gamma & \left(3.33333\times 10^{-6} - 0.00953\gamma\right) & -0.0749\gamma & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ \tilde{x}_{11} \\ \tilde{x}_{12} \\ \tilde{x}_{13} \\ \tilde{x}_{14} \\ \tilde{x}_{15} \end{pmatrix} = \begin{pmatrix} 2400 \\ 2800 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The above was solve in Matlab for different values of γ.For $\gamma = 0.00001$, the optimal x was

$$x_{optimal} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ \tilde{x}_{11} \\ \tilde{x}_{12} \\ \tilde{x}_{13} \\ \tilde{x}_{14} \\ \tilde{x}_{15} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 4.2150 \\ 6.8174 \\ 0 \\ 0 \\ 3.0043 \\ 0 \\ 1.0022 \\ 0 \\ 400 \\ 0 \\ 0.0161 \\ 0 \end{pmatrix}$$

With optimal $J^* = 2.376$ dollars. For $\gamma = 0.00002$, the optimal x was

$$x_{optimal} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ \tilde{x}_{11} \\ \tilde{x}_{12} \\ \tilde{x}_{13} \\ \tilde{x}_{14} \\ \tilde{x}_{15} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2.0750 \\ 0 \\ 0 \\ 0 \\ 1.1779 \\ 0 \\ 3.2348 \\ 0 \\ 400 \\ 0.0067 \\ 0 \\ 0.0024 \end{pmatrix}$$

With optimal $J^{*} = 0.9949$ dollars. For $\gamma = 0.00003$, the optimal x was

$$x_{optimal} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ \tilde{x}_{11} \\ \tilde{x}_{12} \\ \tilde{x}_{13} \\ \tilde{x}_{14} \\ \tilde{x}_{15} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1.0712 \\ 0 \\ 0 \\ 0 \\ 0.2078 \\ 0.1119 \\ 3.3629 \\ 0 \\ 400 \\ 0.0034 \\ 0 \\ 0 \end{pmatrix}$$

With optimal $J^{*} = 0.9424$ dollars. For $\gamma = 0.00004$, the optimal x was

$$x_{optimal} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ \tilde{x}_{11} \\ \tilde{x}_{12} \\ \tilde{x}_{13} \\ \tilde{x}_{14} \\ \tilde{x}_{15} \end{pmatrix} = \begin{pmatrix} 0.3431 \\ 0 \\ 0 \\ 0.8519 \\ 0 \\ 0 \\ 0 \\ 0.7094 \\ 2.8071 \\ 0 \\ 0 \\ 400 \\ 0 \\ 0.0027 \\ 0 \end{pmatrix}$$

With optimal $J^{*} = 1.0944$ dollars.When going more than $\gamma = 0.00004$, say $\gamma = 0.00005$, Matlab was not able to obtain solution using simplex method. The message was "Exiting: The constraints are overly stringent; no feasible starting point found"

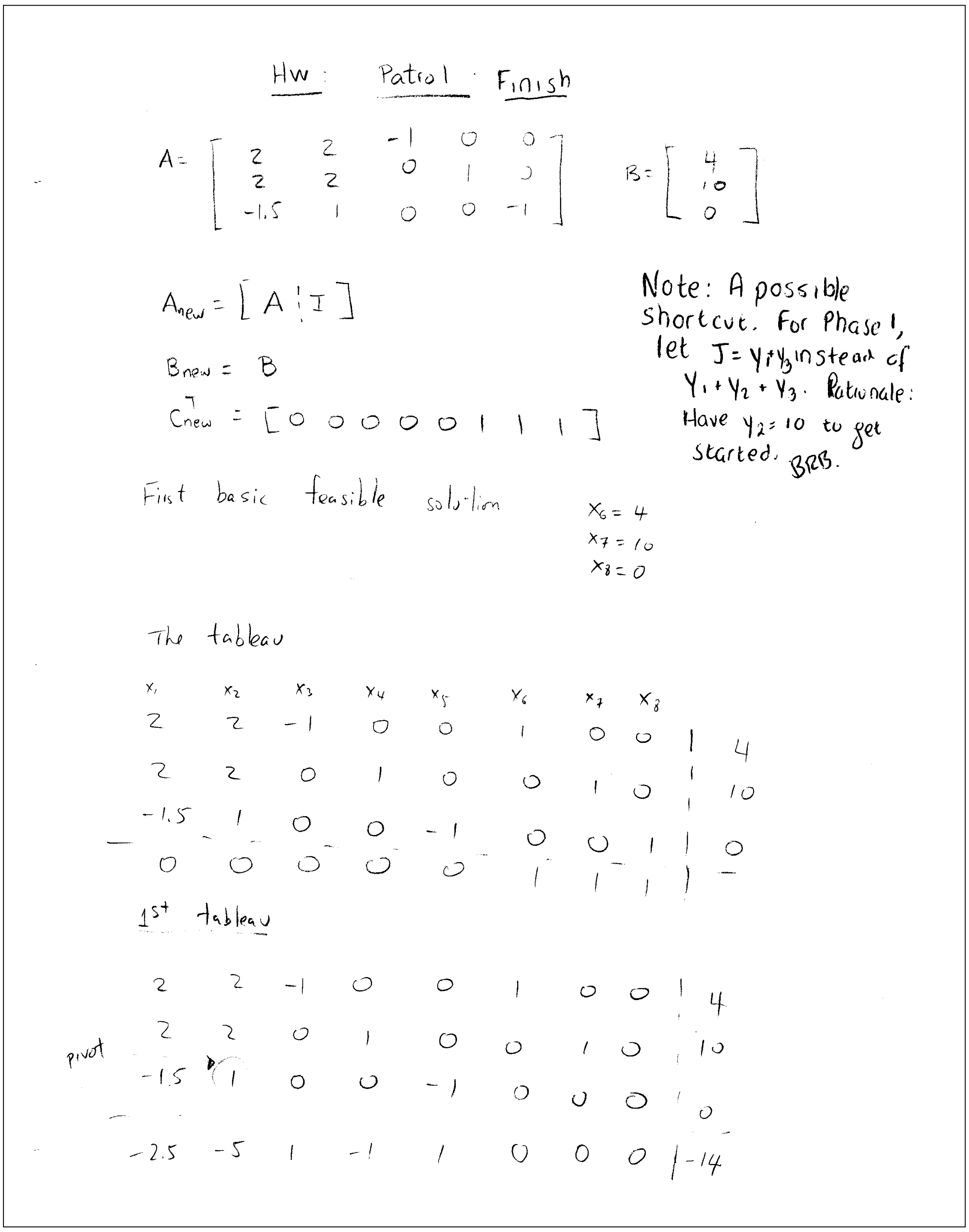
```
Exiting: The constraints are overly stringent;
            no feasible starting point found.
X =
0
0
0
0
0
0
0
0
0
0
-2400
```

```
2800
0
0
0
FVAL =
0
EXITFLAG =
-2
OUTPUT =
iterations: 0
algorithm: 'simplex'
cgiterations: []
message: 'Exiting: The constraints are overly stringent;
                             no feasible starting\U{2026}'
constrviolation: 2400
firstorderopt: 0.4000
```

When going smaller than $\gamma = 0.00001$ same problem was seen. Out of these γ values, $\gamma = 0.00003$ gave the least cost of 0.9424 dollars.

```
function nma_HW6_problem_3_part_b
%Solves third problem, HW 6, ECE 719
%Nasser M. Abbasi

lam=0.00001;
c = [0.4 0.35 0.15 0.05 0.25 0.12 0.25 0.07 0.3 0.25 0 0 0 0 0];
A = [600  300 220 70 185 185 200 55 330 677 -1 0 0 0 0;
     600  300 220 70 185 185 200 55 330 677 0 1 0 0 0;
     1.2679*10^(-4),4.7872*10^(-5),-1.9430*10^(-3),  ...
     -1.0412*10^(-2),...
     2.5118*10^(-3),3.4412*10^(-3),3.1475*10^(-3),...
     8.385*10^(-3),...
     0.00624,1.5674*10^(-3),0, 0, 1, 0, 0;
     1.4985*10^(-4),2.7128*10^(-4),-1.1508*10^(-3),...
     -7.2353*10^(-3),...
     0.0029,3.6176*10^(-3),3.8525*10^(-3),8.415*10^(-3),...
     6.2457*10^(-3),...
     1.577*10^(-3),0, 0, 0, -1, 0;
     4.5338*10^(-4),3.3245*10^(-4),-8.3203*10^(-4),...
     2.1765*10^(-3),...
     -1.2765*10^(-3),-2.0882*10^(-3),-0.00236,7.5*10^(-5),...
     -8.2143*10^(-5),...
     -6.9519*10^(-4), 0, 0, 0, 0, -1;
     (2.0697*10^(-6)-5/122*lam),(1.5957*10^(-6)-9/94*lam),...
     (3.125*10^(-6)-1/8*lam),(5.8824*10^(-6)-6/17*lam),...
     (-2/17*lam),(1.1765*10^(-6)-9/85*lam),-2/25*lam,-7/60*lam,...
     3.3333*10^(-6),1.60428*10^(-6),0,0,0,0,0];
options = optimset('LargeScale','off','Simplex','on');
b=[2400 2800 0 0 0 0];
[X,FVAL,EXITFLAG,OUTPUT]=linprog(c,[],[],A,b,zeros(size(c)),...
                                                 [],[],options)
end
```

4.6.4 HW 6 key solution



Hw : Patrol · Finish

$$A = \begin{bmatrix} 2 & 2 & -1 & 0 & 0 \\ 2 & 2 & 0 & 1 & 0 \\ -1.5 & 1 & 0 & 0 & -1 \end{bmatrix} \qquad B = \begin{bmatrix} 4 \\ 10 \\ 0 \end{bmatrix}$$

$A_{new} = [\, A \mid I \,]$

$B_{new} = B$

$C_{new}^T = [\, 0 \; 0 \; 0 \; 0 \; 0 \; 1 \; 1 \; 1 \,]$

Note: A possible shortcut. For Phase 1, let $J = y_1 + y_3$ instead of $y_1 + y_2 + y_3$. Rationale: Have $y_2 = 10$ to get started. BRB.

First basic feasible solution

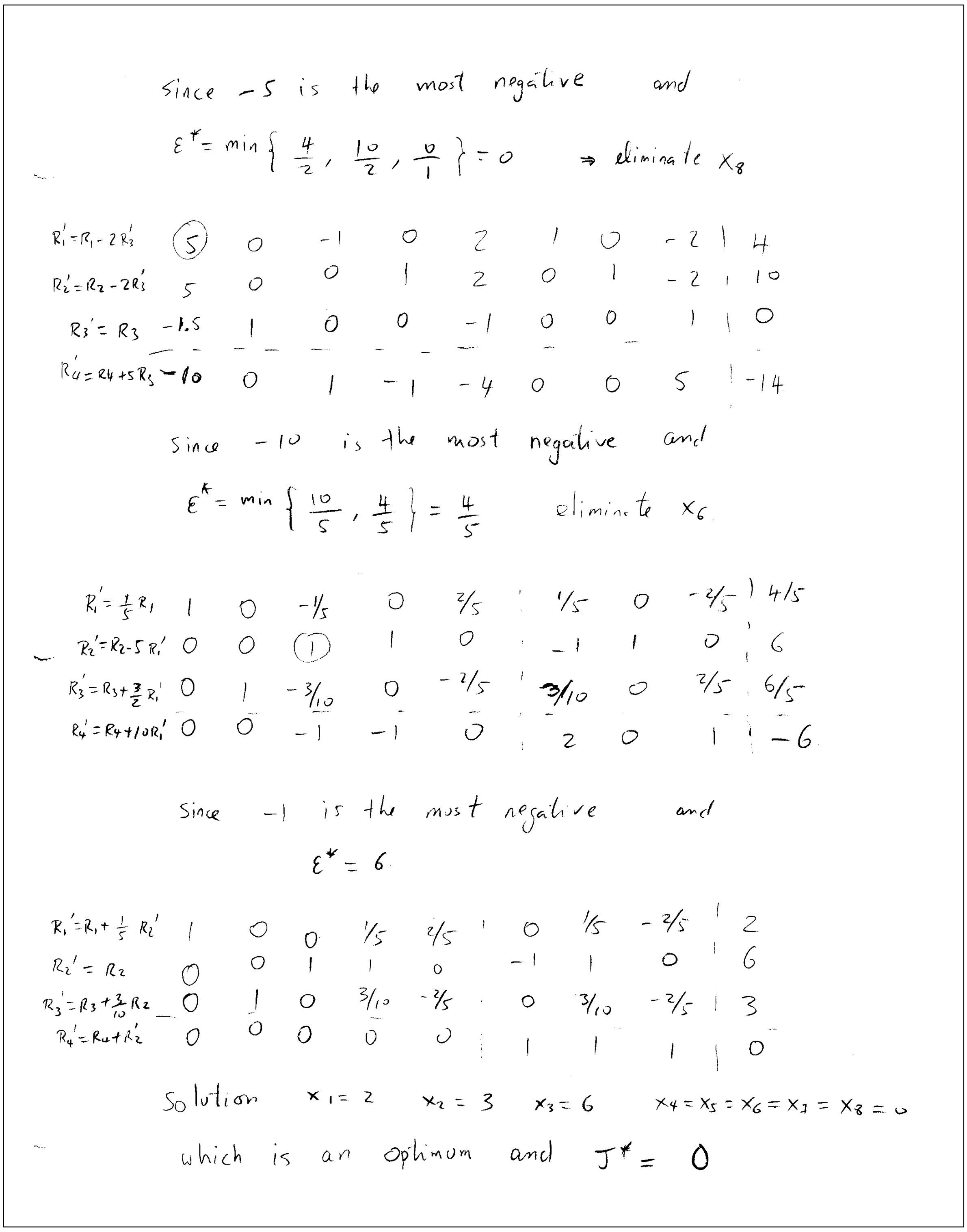
$x_6 = 4$
$x_7 = 10$
$x_8 = 0$

The tableau

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
2	2	-1	0	0	1	0	0	4
2	2	0	1	0	0	1	0	10
-1.5	1	0	0	-1	0	0	1	0
0	0	0	0	0	1	1	1	—

1st tableau

	2	2	-1	0	0	1	0	0	4
	2	2	0	1	0	0	1	0	10
pivot	-1.5	1	0	0	-1	0	0	0	0
	-2.5	-5	1	-1	1	0	0	0	-14



Since -5 is the most negative and

$\varepsilon^* = \min\left\{ \frac{4}{2}, \frac{10}{2}, \frac{0}{1} \right\} = 0$ ⇒ eliminate x_8

$$\begin{array}{l|cccccccc|c}
R_1' = R_1 - 2R_3' & \boxed{5} & 0 & -1 & 0 & 2 & 1 & 0 & -2 & 4 \\
R_2' = R_2 - 2R_3' & 5 & 0 & 0 & 1 & 2 & 0 & 1 & -2 & 10 \\
R_3' = R_3 & -1.5 & 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\
\hline
R_4' = R_4 + 5R_3 & -10 & 0 & 1 & -1 & -4 & 0 & 0 & 5 & -14
\end{array}$$

Since -10 is the most negative and

$\varepsilon^* = \min\left\{ \frac{10}{5}, \frac{4}{5} \right\} = \frac{4}{5}$ eliminate x_6.

$$\begin{array}{l|cccccccc|c}
R_1' = \frac{1}{5}R_1 & 1 & 0 & -1/5 & 0 & 2/5 & 1/5 & 0 & -2/5 & 4/5 \\
R_2' = R_2 - 5R_1' & 0 & 0 & \boxed{1} & 1 & 0 & -1 & 1 & 0 & 6 \\
R_3' = R_3 + \frac{3}{2}R_1' & 0 & 1 & -3/10 & 0 & -2/5 & 3/10 & 0 & 2/5 & 6/5 \\
\hline
R_4' = R_4 + 10R_1' & 0 & 0 & -1 & -1 & 0 & 2 & 0 & 1 & -6
\end{array}$$

Since -1 is the most negative and

$\varepsilon^* = 6$

$$\begin{array}{l|cccccccc|c}
R_1' = R_1 + \frac{1}{5}R_2' & 1 & 0 & 0 & 1/5 & 2/5 & 0 & 1/5 & -2/5 & 2 \\
R_2' = R_2 & 0 & 0 & 1 & 1 & 0 & -1 & 1 & 0 & 6 \\
R_3' = R_3 + \frac{3}{10}R_2 & 0 & 1 & 0 & 3/10 & -2/5 & 0 & 3/10 & -2/5 & 3 \\
\hline
R_4' = R_4 + R_2' & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{array}$$

Solution $x_1 = 2$ $x_2 = 3$ $x_3 = 6$ $x_4 = x_5 = x_6 = x_7 = x_8 = 0$

which is an optimum and $J^* = 0$

Solution Minimax

Based on exposition in class, we introduce a new variable z and add the following linear inequalities to the original problem:

$$c_1^T x \leq z; \;\; \frac{1}{30}x_1 + \frac{1}{15}x_2 - z \leq 0;$$

$$c_2^T x \leq z; \;\; \frac{3}{10}x_1 + \frac{1}{5}x_2 - z \leq 0;$$

$$c_3^T x \leq z; \;\; 2x_1 + \frac{1}{2}x_2 - z \leq 0;$$

$$c_4^T x \leq z; \;\; x_1 + x_2 - z \leq 0.$$

Letting $x_3 = z$ be the new variable, the "new" LP expressed in inequality form $Ax \leq b$ is described by

$$c_{new} = [0\ 0\ 1]^T;$$

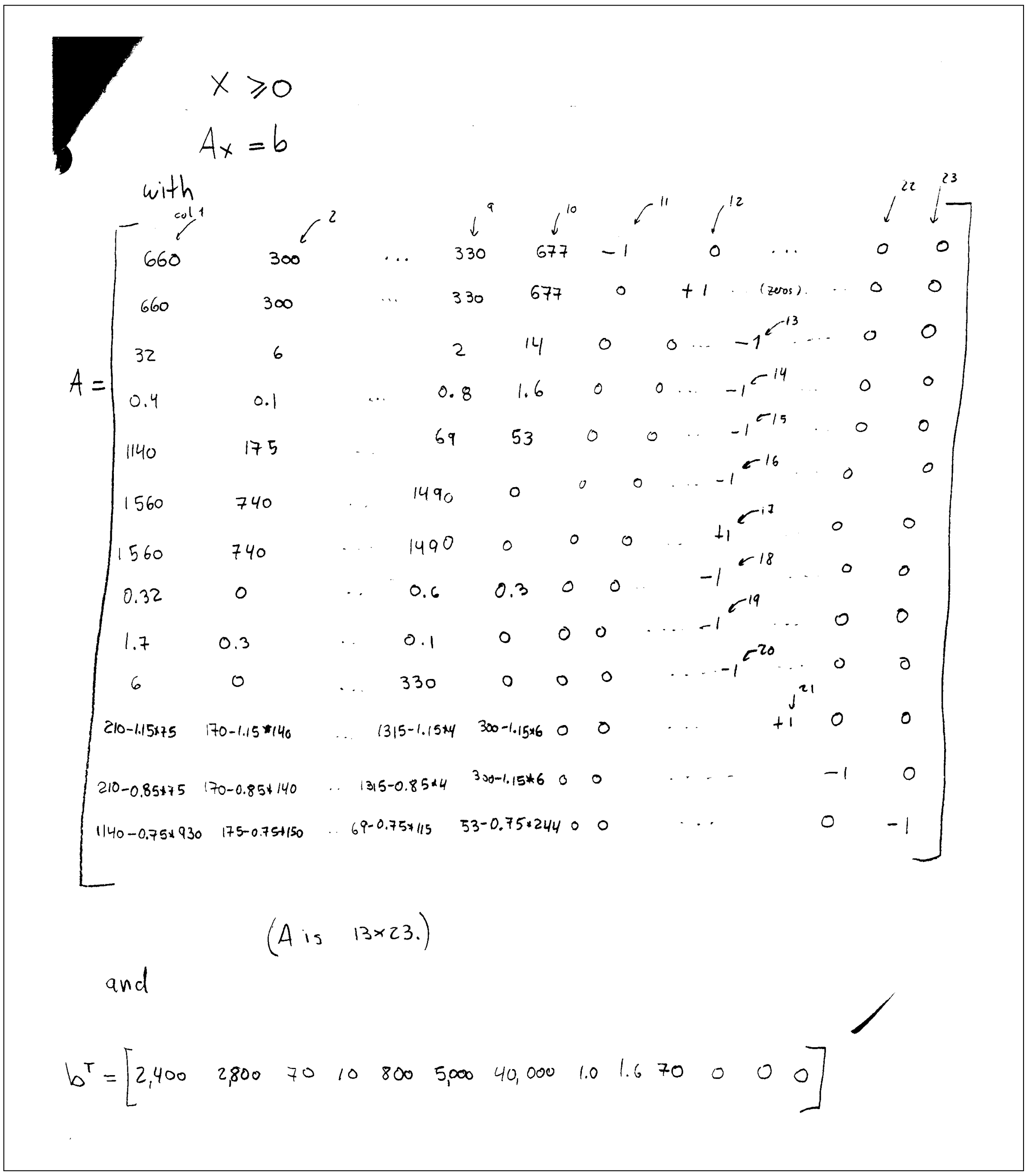
$$A_{new} = \begin{bmatrix} 2 & 2 & 0 \\ -2 & -2 & 0 \\ 1.5 & -1 & 0 \\ 1/30 & 1/15 & -1 \\ 3/10 & 1/5 & -1 \\ 2 & 1/2 & -1 \\ 1 & 1 & -1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix};$$

$$b_{new} = [10\ 0\ -4\ 0\ 0\ 0\ 0\ 0\ 0]^T.$$

Now, using linprog$(c_{new}, A_{new}, b_{new})$. we obtain optimal solution

$$u_0^* = 0; \;\; u_1^* = 4; \;\; J^* = 4.$$

Remark: The solution above is very easy to obtain by plotting the constraint region in the (x_1, x_2) plane. The vertex $x_1^* = 0, x_2^* = 4$ leads to $J^* = 4$.



$$x \geq 0$$

$$Ax = b$$

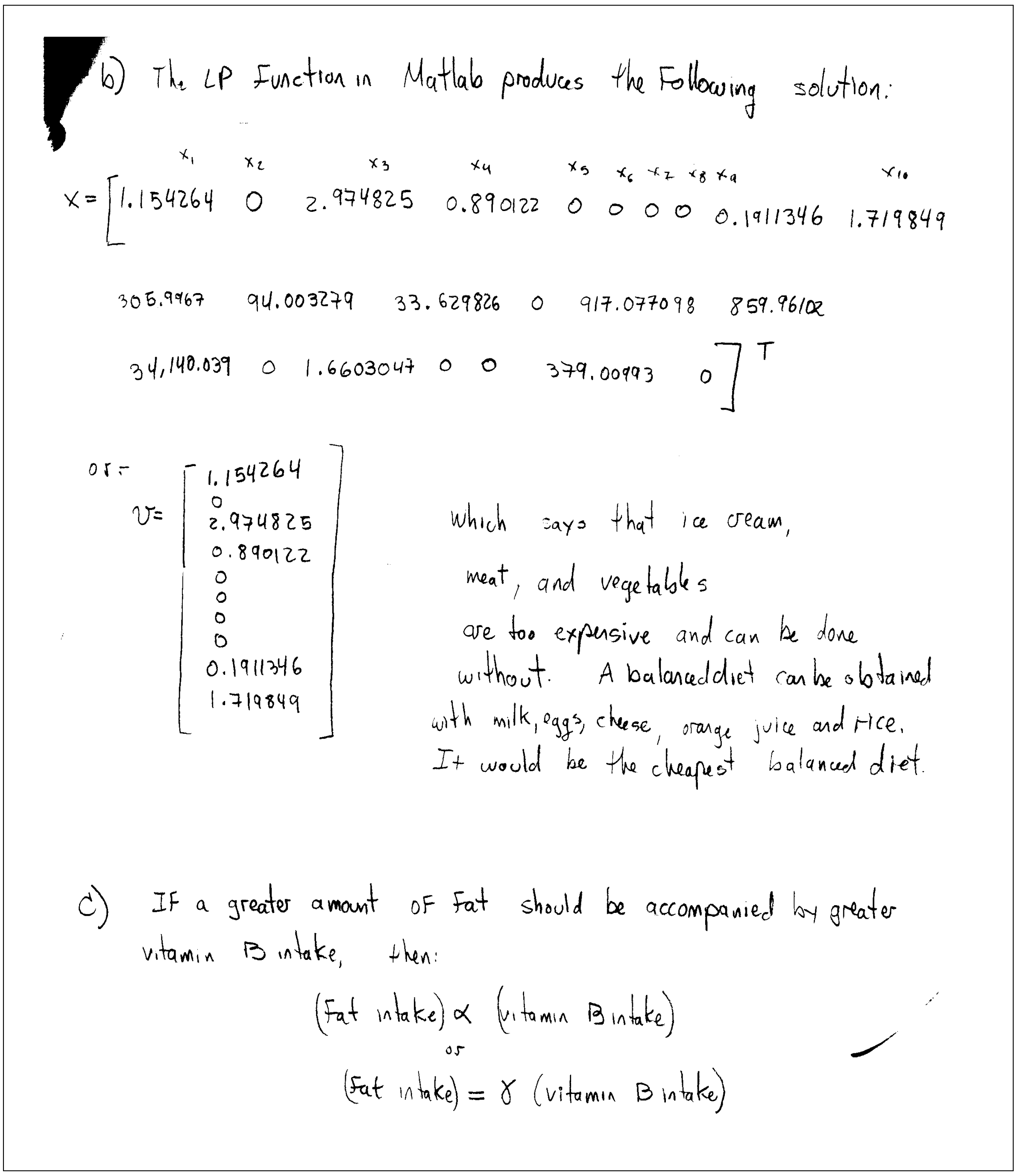
with

$$A = \begin{array}{c}\begin{array}{cccccccccccc} \text{col } 1 & 2 & & 9 & 10 & 11 & 12 & & & & 22 & 23 \end{array}\\ \left[\begin{array}{cccccccccccc}
660 & 300 & \cdots & 330 & 677 & -1 & 0 & \cdots & & & 0 & 0\\
660 & 300 & \cdots & 330 & 677 & 0 & +1 & \cdots & (\text{zeros}) & \cdots & 0 & 0\\
32 & 6 & & 2 & 14 & 0 & 0 & \cdots & \overset{13}{-1} & \cdots & 0 & 0\\
0.4 & 0.1 & \cdots & 0.8 & 1.6 & 0 & 0 & \cdots & \overset{14}{-1} & \cdots & 0 & 0\\
1140 & 175 & \cdots & 69 & 53 & 0 & 0 & \cdots & \overset{15}{-1} & \cdots & 0 & 0\\
1560 & 740 & \cdots & 1490 & 0 & 0 & 0 & \cdots & \overset{16}{-1} & \cdots & 0 & 0\\
1560 & 740 & \cdots & 1490 & 0 & 0 & 0 & \cdots & \overset{17}{+1} & & 0 & 0\\
0.32 & 0 & \cdots & 0.6 & 0.3 & 0 & 0 & \cdots & \overset{18}{-1} & \cdots & 0 & 0\\
1.7 & 0.3 & \cdots & 0.1 & 0 & 0 & 0 & \cdots & \overset{19}{-1} & \cdots & 0 & 0\\
6 & 0 & \cdots & 330 & 0 & 0 & 0 & \cdots & \overset{20}{-1} & \cdots & 0 & 0\\
210-1.15*75 & 170-1.15*140 & \cdots & 1315-1.15*4 & 300-1.15*6 & 0 & 0 & \cdots & \overset{21}{+1} & & 0 & 0\\
210-0.85*75 & 170-0.85*140 & \cdots & 1315-0.85*4 & 300-1.15*6 & 0 & 0 & \cdots & & & -1 & 0\\
1140-0.75*930 & 175-0.75*150 & \cdots & 69-0.75*115 & 53-0.75*244 & 0 & 0 & \cdots & & & 0 & -1
\end{array}\right]\end{array}$$

(A is 13×23.)

and

$$b^T = \begin{bmatrix} 2{,}400 & 2{,}800 & 70 & 10 & 800 & 5{,}000 & 40{,}000 & 1.0 & 1.6 & 70 & 0 & 0 & 0 \end{bmatrix}$$



b) The LP Function in Matlab produces the Following solution:

$$X = \begin{array}{cccccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \\ \Big[\,1.154264 & 0 & 2.974825 & 0.890122 & 0 & 0 & 0 & 0 & 0.1911346 & 1.719849 \end{array}$$

$$305.9967 \quad 94.003279 \quad 33.629826 \quad 0 \quad 917.077098 \quad 859.96102$$

$$34{,}140.039 \quad 0 \quad 1.6603047 \quad 0 \quad 0 \quad 379.00993 \quad 0\,\Big]^T$$

or:

$$v = \begin{bmatrix} 1.154264 \\ 0 \\ 2.974825 \\ 0.890122 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.1911346 \\ 1.719849 \end{bmatrix}$$

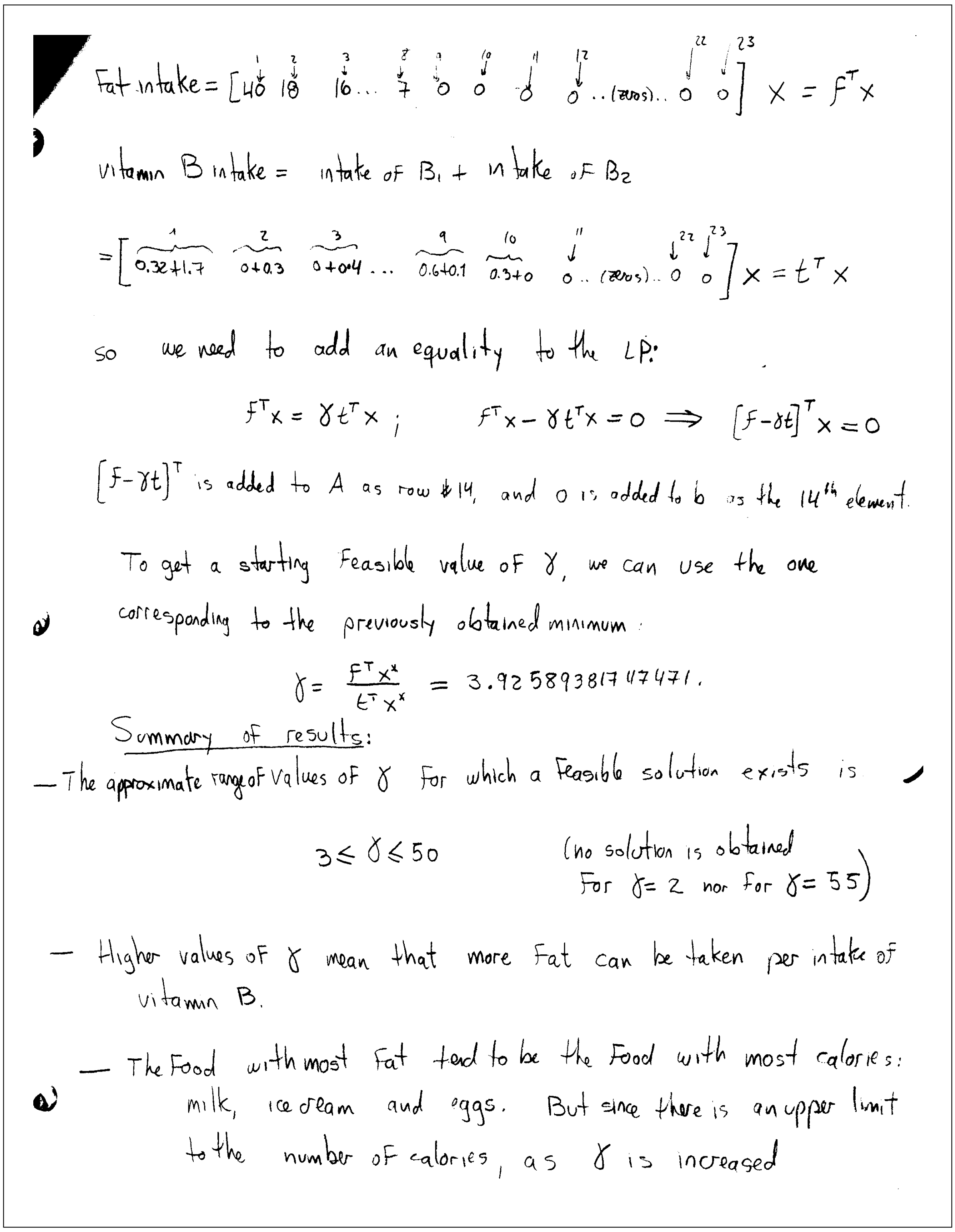
which says that ice cream, meat, and vegetables are too expensive and can be done without. A balanceddiet can be obtained with milk, eggs, cheese, orange juice and rice. It would be the cheapest balanced diet.

c) If a greater amount of Fat should be accompanied by greater vitamin B intake, then:

$$(\text{Fat intake}) \propto (\text{vitamin B intake})$$

or

$$(\text{Fat intake}) = \gamma\,(\text{vitamin B intake})$$



Fat intake $= [\overset{1}{40}\ \overset{2}{18}\ \ \overset{3}{16} \ldots \overset{8}{7}\ \ \overset{9}{0}\ \ \overset{10}{0}\ \ \overset{11}{0}\ \ \overset{12}{0} \ldots \text{(zeros)} \ldots \overset{22}{0}\ \ \overset{23}{0}]\ x = f^T x$

vitamin B intake = intake of B_1 + intake of B_2

$$= [\overbrace{0.32+1.7}^{1}\ \ \overbrace{0+0.3}^{2}\ \ \overbrace{0+0.4}^{3} \ldots \overbrace{0.6+0.1}^{9}\ \ \overbrace{0.3+0}^{10}\ \ \overset{11}{0} \ldots \text{(zeros)} \ldots \overset{22}{0}\ \ \overset{23}{0}]\ x = t^T x$$

so we need to add an equality to the LP:

$$f^T x = \gamma t^T x \ ; \qquad f^T x - \gamma t^T x = 0 \Rightarrow [f - \gamma t]^T x = 0$$

$[f-\gamma t]^T$ is added to A as row # 14, and 0 is added to b as the 14th element.

To get a starting feasible value of γ, we can use the one corresponding to the previously obtained minimum:

$$\gamma = \frac{f^T x^*}{t^T x^*} = 3.92589381747471.$$

Summary of results:

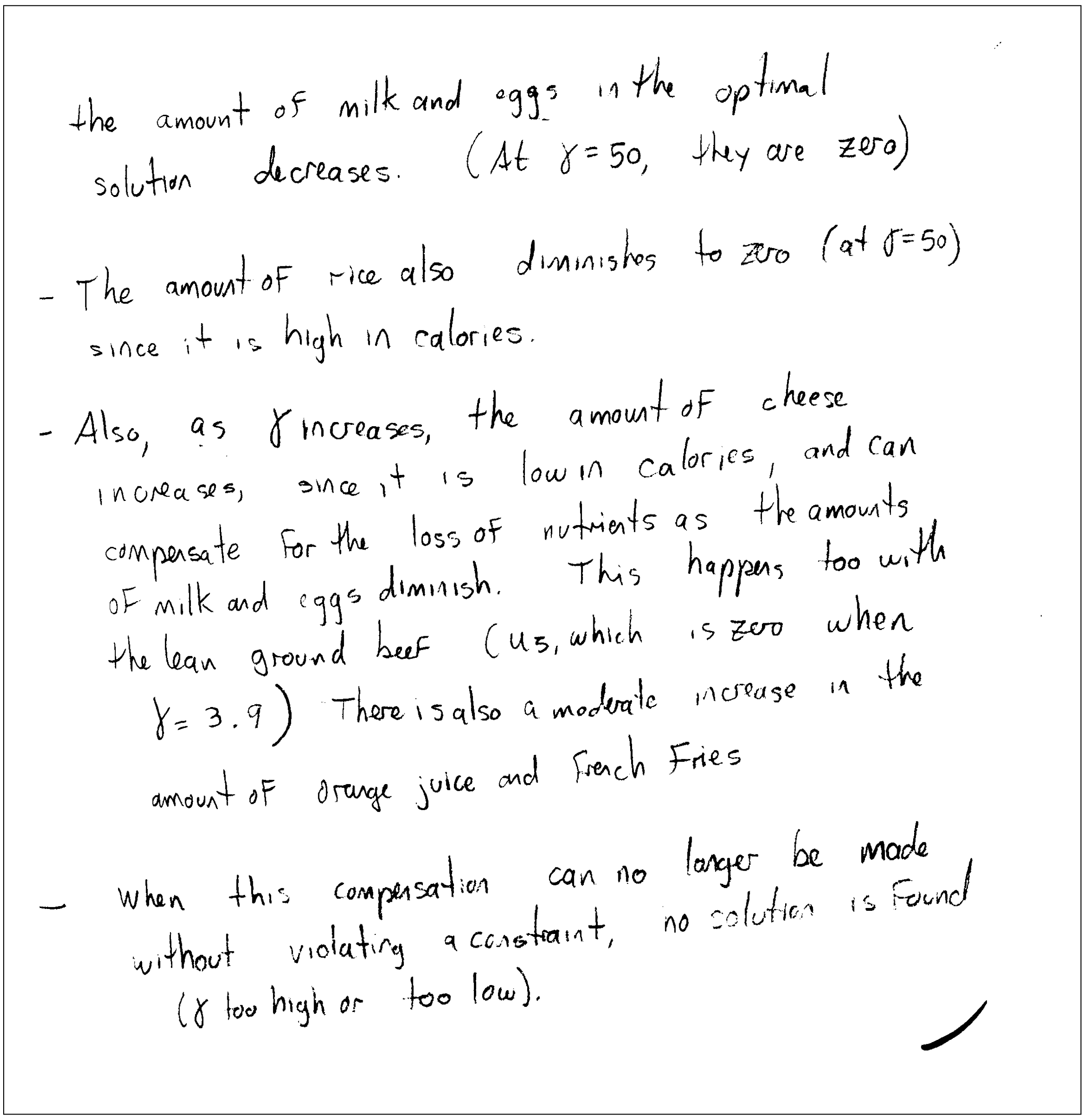
— The approximate range of values of γ for which a feasible solution exists is

$$3 \leq \gamma \leq 50$$

(no solution is obtained for $\gamma = 2$ nor for $\gamma = 55$)

— Higher values of γ mean that more fat can be taken per intake of vitamin B.

— The food with most fat tend to be the food with most calories: milk, ice cream and eggs. But since there is an upper limit to the number of calories, as γ is increased

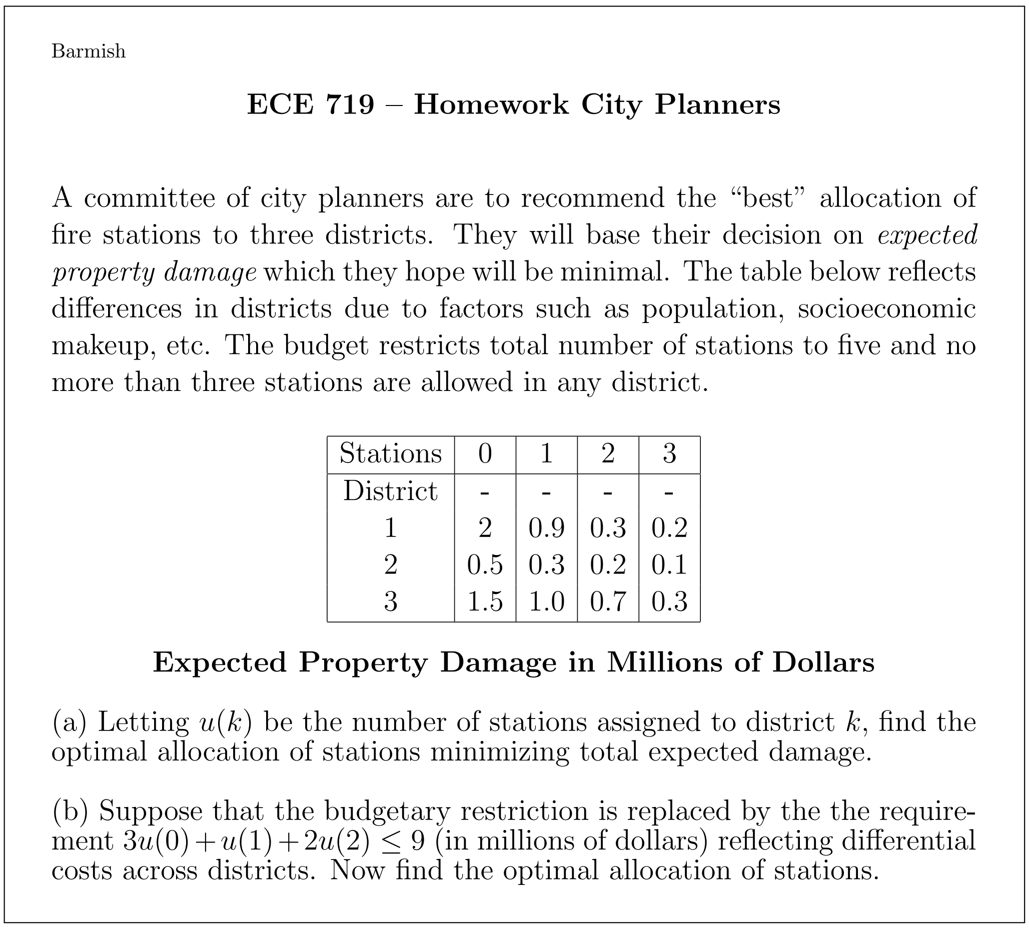


the amount of milk and eggs in the optimal solution decreases. (At $\gamma = 50$, they are zero)

- The amount of rice also diminishes to zero (at $\gamma = 50$) since it is high in calories.
- Also, as γ increases, the amount of cheese increases, since it is low in calories, and can compensate for the loss of nutrients as the amounts of milk and eggs diminish. This happens too with the lean ground beef (u_5, which is zero when $\gamma = 3.9$) There is also a moderate increase in the amount of orange juice and French Fries
- When this compensation can no longer be made, no solution is found without violating a constraint, (γ too high or too low).

4.7 HW 7

4.7.1 Problem 1



Barmish

ECE 719 – Homework City Planners

A committee of city planners are to recommend the "best" allocation of fire stations to three districts. They will base their decision on *expected property damage* which they hope will be minimal. The table below reflects differences in districts due to factors such as population, socioeconomic makeup, etc. The budget restricts total number of stations to five and no more than three stations are allowed in any district.

Stations	0	1	2	3
District	-	-	-	-
1	2	0.9	0.3	0.2
2	0.5	0.3	0.2	0.1
3	1.5	1.0	0.7	0.3

Expected Property Damage in Millions of Dollars

(a) Letting $u(k)$ be the number of stations assigned to district k, find the optimal allocation of stations minimizing total expected damage.

(b) Suppose that the budgetary restriction is replaced by the the requirement $3u(0)+u(1)+2u(2) \leq 9$ (in millions of dollars) reflecting differential costs across districts. Now find the optimal allocation of stations.

Figure 4.65: problem 1 description

4.7.1.1 part a

Before applying dynamic programming to solve the problem, the solution was first found by brute force in order to verify that the D.P. method when completed was correct. Using brute force, the optimal arrangement is found to be:

Assign 2 stations to the first district, and 3 stations to third district and no stations are assigned to the second district, for a minimum total expected property damage of 1.1 millions.

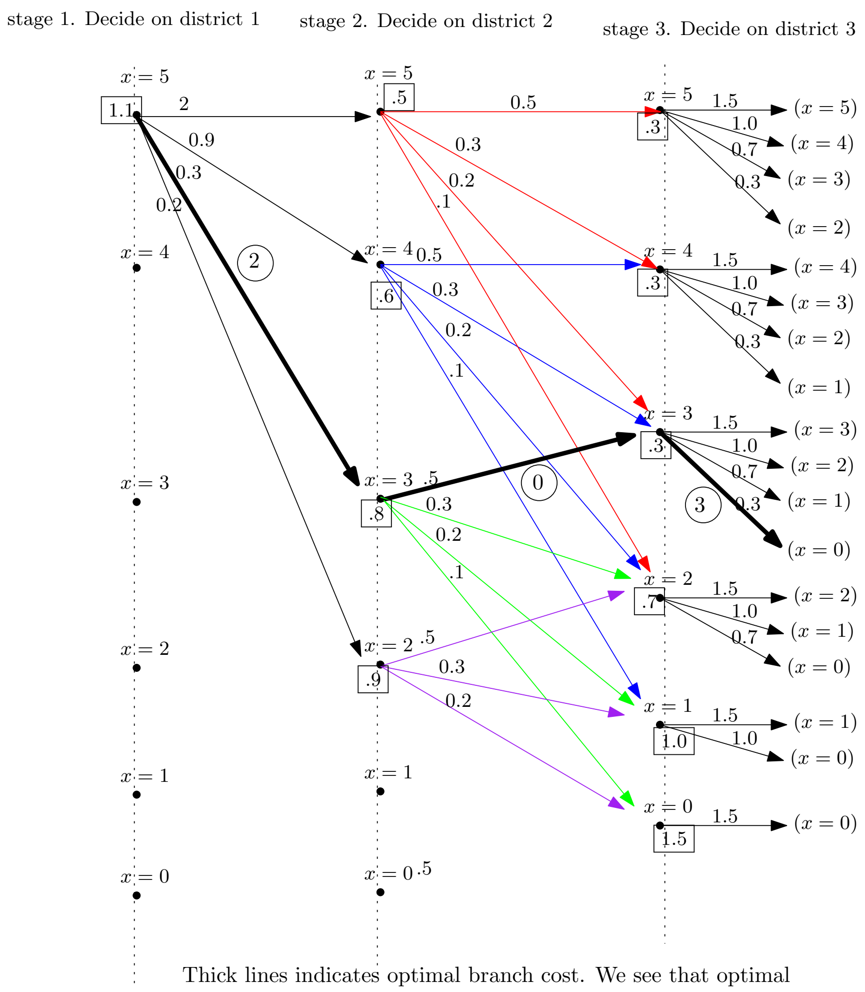
The brute force method also generated a list of all the arrangements (44 of them) and the cost of each. For reference, here is the complete table with the small Matlab code used to generated it in the appendix. After this, the graphical D.P. method called branch and bound was used to verify this result.

district 1	district 2	district 3	cost in millions
0	0	0	4.0
0	0	1	3.5
0	0	2	3.2
0	0	3	2.8
0	1	0	3.8
0	1	1	3.3
0	1	2	3.0
0	1	3	2.6
0	2	0	3.7

0	2	1	3.2
0	2	2	2.9
0	2	3	2.5
0	3	0	3.6
0	3	1	3.1
0	3	2	2.8
1	0	0	2.9
1	0	1	2.4
1	0	2	2.1
1	0	3	1.7
1	1	0	2.7
1	1	1	2.2
1	1	2	1.9
1	1	3	1.5
1	2	0	2.6
1	2	1	2.1
1	2	2	1.8
1	3	0	2.5
1	3	1	2.0
2	0	0	2.3
2	0	1	1.8
2	0	2	1.5
2	0	3	1.1*
2	1	0	2.1
2	1	1	1.6
2	1	2	1.3
2	2	0	2.0
2	2	1	1.5
2	3	0	1.9
3	0	0	2.2
3	0	1	1.7
3	0	2	1.4
3	1	0	2.0
3	1	1	1.5
3	2	0	1.9

Let the state x be the number of stations available to be assigned at each stage. For example, if we are at stage 2 and $x = 5$, this means all 5 stations are available to be assigned to district two. Stage one was the decision to decide on district one, stage two for the decision to assign for district two and the final stage, stage three is for district three. This is arbitrary, any order will give the same answer. One can decide on district three first, and then district one for example. This makes no difference to the final result.

The following diagram shows the result found which agrees with the brute force method above. The branch cost is the number above the arrow itself. The number in the small rectangle at the node, is the minimal cost of the branch leaving that node. For example, in stage three, when $x = 5$, there are 4 branches that leave that node where (0,1,2,3) stations can be assigned. The lowest cost of these branches is the one with cost 0.3 and that is what goes in the small square next to the node. This process continues moving backward. This method is the graphical equivalent to the Bellman dynamic equations and can be used when the number of states is finite and the number of decisions at each state is finite also.



Thick lines indicates optimal branch cost. We see that optimal assignment is 2 stations for district 1 and 3 stations for district 3. The numbers in the small circles is number of stations assigned in that stage. This matches the brute force method as expected

Figure 4.66: Part (a) solution using Dynamic programming

4.7.1.2 Part b

The constraint now is

$$3u(0) + u(1) + 2u(2) \leq 9 \tag{1}$$

What this means, is that 3 times the number of stations assigned to district one plus one times the number of stations assigned to district two, plus 2 times the number of stations assigned to district three, must be smaller than 9 stations in total (millions of dollars in the problem was a typo).

We see that part(a) does not meet this requirement. In part(a), we found $u(0) = 2, u(1) = 0, u(3) = 3$, which means

$$\begin{aligned} 3u(0) + u(1) + 3u(2) &= 3(2) + 0 + 2(3) \\ &= 12 \end{aligned}$$

Which is larger than 9. We need to find again $u(0), u(1), u(2)$ which still satisfies part (a) requirements of no more than 3 stations for a district and no more than total of 5 stations, but now with the additional constraint of (1) in place at the same time.

The search was repeated using brute force to first find the combinations that meet this criteria, and then the one with the minimum expected damage was selected.

Here is the new table found

district 1	district 2	district 3	cost in millions	$3u_0 + u_1 + 2u_2$
0	0	0	4.0	0
0	0	1	3.5	2
0	0	2	3.2	4
0	0	3	2.8	6
0	1	0	3.8	1
0	1	1	3.3	3
0	1	2	3.0	5
0	1	3	2.6	7
0	2	0	3.7	2
0	2	1	3.2	4
0	2	2	2.9	6
0	2	3	2.5	8
0	3	0	3.6	3
0	3	1	3.1	5
0	3	2	2.8	7
1	0	0	2.9	3
1	0	1	2.4	5
1	0	2	2.1	7
1	0	3	1.7	9
1	1	0	2.7	4
1	1	1	2.2	6
1	1	2	1.9	8
1	2	0	2.6	5
1	2	1	2.1	7
1	2	2	1.8	9
1	3	0	2.5	6
1	3	1	2.0	8
2	0	0	2.3	6
2	0	1	1.8	8
2	1	0	2.1	7
2	1	1	1.6*	9
2	2	0	2.0	8
2	3	0	1.9	9
3	0	0	2.2	9

We see that the combination with the minimum expected damage is when

> two stations are assigned to district one, and one station assigned to district two and one station assigned to district three with expected damage of 1.6 million dollars.

The following diagram illustrates the branch and bound graph with the new optimal path now highlighted in the think line.

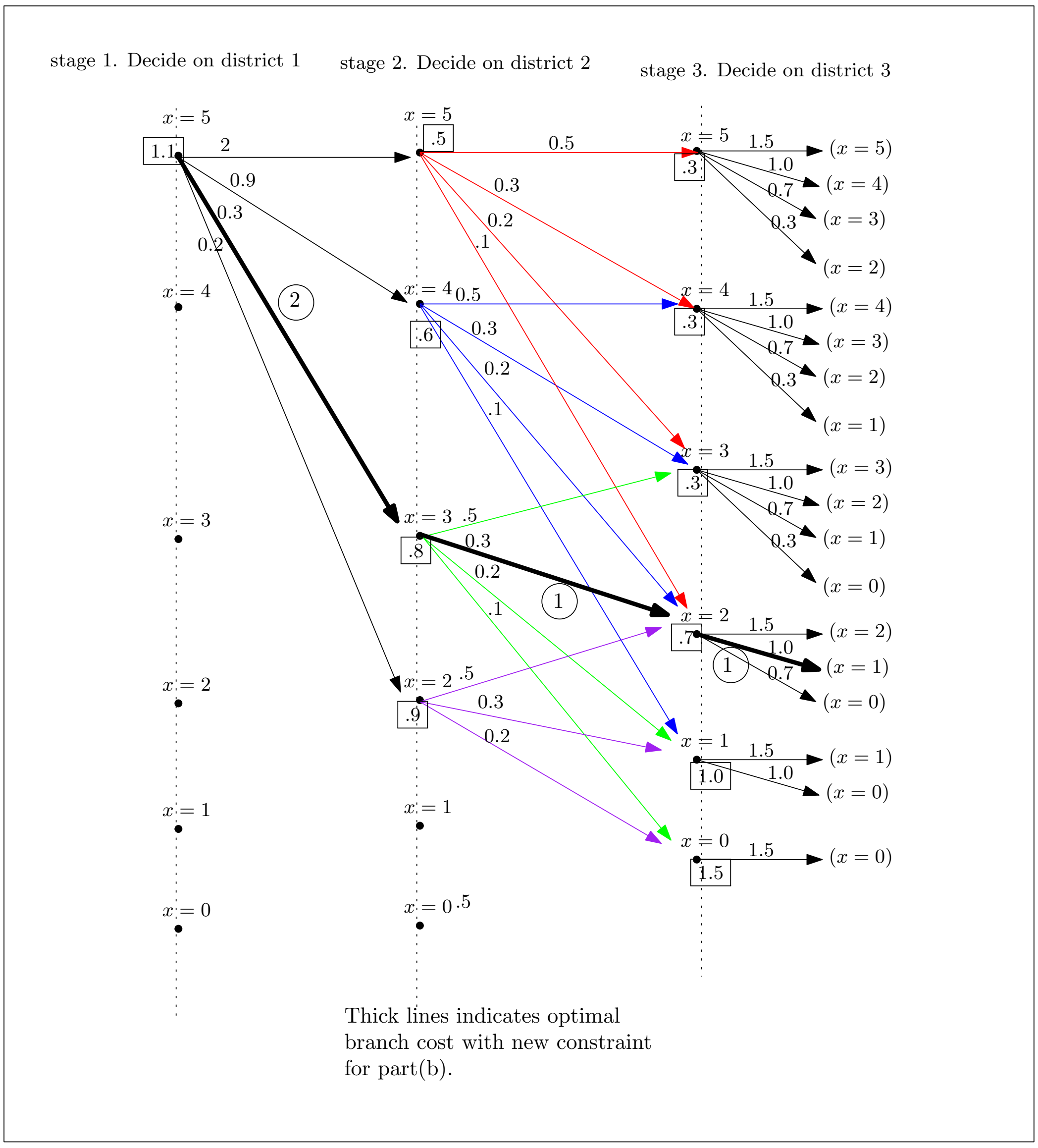

stage 1. Decide on district 1
stage 2. Decide on district 2
stage 3. Decide on district 3
x = 5
x = 4
x = 3
x = 2
x = 1
x = 0
1.1
2
0.9
0.3
0.2
2
.5
.6
.8
.9
0.5
0.3
0.2
.1
1
.3
.3
.3
.7
1.0
1.5
1.5
1.0
0.7
0.3
(x = 5)
(x = 4)
(x = 3)
(x = 2)
(x = 1)
(x = 0)
Thick lines indicates optimal branch cost with new constraint for part(b).

Figure 4.67: Part (b) solution

The code used to part(b) is in the appendix.

4.7.1.3 appendix for problem 1

code to generate the first table for part(a)

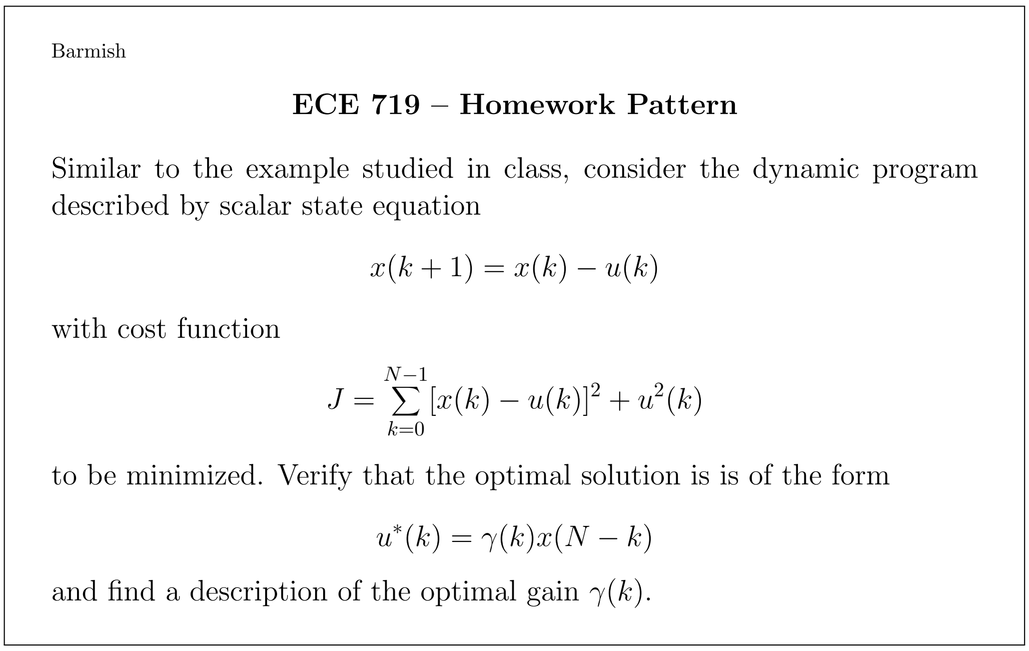
```
function nma_HW7_ECE_719_prob_1()
%find cost using brute force search, to verify DP method
%HW 7, ECE 719, APril 23,2016
%Nasser M. Abbasi
cost_table   = zeros(100,4); %place to put the cost
count_so_far = 0;
for i=0:3
    j = 0; k = 0;
    if sum([i j k])<=5
      for j=0:3
        k = 0;
        if sum([i j k])<=5
           for k = 0:3
             if sum([i j k])<=5
               count_so_far = count_so_far+1;
               fprintf('count_so_far=%d, [%d,%d,%d]\n', ...
                                          count_so_far,i,j,k);
               cost_table(count_so_far,1:3)=[i j k];
               cost_table(count_so_far,4)=find_cost([i j k]);
             end
           end
        end
      end
    end
end

for i=1:count_so_far
 fprintf('%d & %d & %d & %2.1f \\\\ \\hline\n',cost_table(i,1),...
        cost_table(i,2),cost_table(i,3),cost_table(i,4));
end

[~,J]=min( cost_table(1:count_so_far,4));
fprintf('optimal allocation is \n');
cost_table(J,:)
end
%==========================
function I=find_cost(x)
tbl=[2,.9,.3,.2;
    .5,.3,.2,.1;
    1.5,1,.7,.3];
I= tbl(1,x(1)+1) + tbl(2,x(2)+1) + tbl(3,x(3)+1);
end
```

code to generate the first table for part(b)

```
function nma_HW7_ECE_719_prob_1_part_b()
%finds lowest cost with constraint that
%3*u(0)+u(1)+2*u(2)<=9 to find optinal case using brute force,
%to verify DP method
%HW 7, ECE 719, APril 30,2016
%Nasser M. Abbasi
cost_table   = zeros(100,5); %place to put the cost
count_so_far = 0;
for i=0:3  %this is district 1
    j = 0; k = 0;
    if sum([i j k])<=5
      for j=0:3  %this is district 2
        k = 0;
        if sum([i j k])<=5
```

```
           for k = 0:3 %this is district 3
              if sum([i j k])<=5
                %check if 3*i+j+2*k <= 9 first
                if 3*i+j+2*k <=9
                   count_so_far = count_so_far+1;
                   fprintf('count_so_far=%d, [%d,%d,%d], (3*i+j+2*k=%d) \n', ...
                                  count_so_far,i,j,k,3*i+j+2*k);
                   cost_table(count_so_far,1:3)=[i j k];
                   cost_table(count_so_far,4)=find_cost([i j k]);
                   cost_table(count_so_far,5)=3*i+j+2*k;
                end
             end
           end
         end
       end
    end
end

for i=1:count_so_far
  fprintf('%d & %d & %d & %2.1f & %d \\\\ \\hline\n',...
    cost_table(i,1),cost_table(i,2),cost_table(i,3),...
    cost_table(i,4),cost_table(i,5));
end

[~,J]=min( cost_table(1:count_so_far,4));
fprintf('optimal allocation is \n');
cost_table(J,:)
end
%=========================
function I=find_cost(x)
tbl=[2,.9,.3,.2;
    .5,.3,.2,.1;
    1.5,1,.7,.3];
I= tbl(1,x(1)+1) + tbl(2,x(2)+1) + tbl(3,x(3)+1);
end
```

4.7.2 Problem 2



Barmish

ECE 719 – Homework Pattern

Similar to the example studied in class, consider the dynamic program described by scalar state equation

$$x(k+1) = x(k) - u(k)$$

with cost function

$$J = \sum_{k=0}^{N-1} [x(k) - u(k)]^2 + u^2(k)$$

to be minimized. Verify that the optimal solution is is of the form

$$u^*(k) = \gamma(k)x(N-k)$$

and find a description of the optimal gain $\gamma(k)$.

Figure 4.68: problem 2 description

The state equation is (using indices as subscripts from now on, in all that follows as it is easier to read. Therefore $x(N)$ is written as x_N and similarly for $u(N)$)

$$x_{k+1} = x_k - u_k \tag{1}$$

The cost function to minimize is

$$J=\sum_{k=0}^{N-1}\left(x_k-u_k\right)^2+u_k^2 \tag{2}$$

Now we apply Bellman dynamic equation. This diagram shows the overall process.

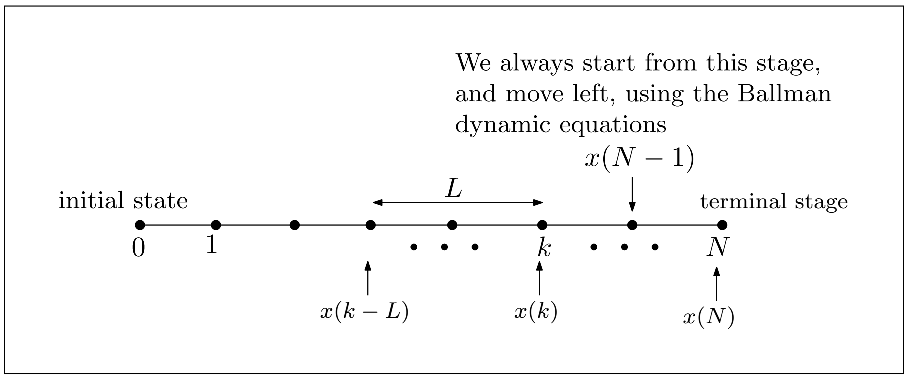

We always start from this stage, and move left, using the Ballman dynamic equations
$x(N-1)$
initial state
L
terminal stage
0
1
k
N
$x(k-L)$
$x(k)$
$x(N)$

Figure 4.69: Showing dynamic programming block diagram

The branch cost from x_{N-1} with one step to go is

$$I\left(x_{N-1},1\right)=\min_{u_{N-1}\in\Omega_{N-1}}\left\{J\left(x_{N-1},u_{N-1}\right)+\Psi\left(x_N\right)\right\}$$

$\Psi\left(x_N\right)$ is the terminal cost. Removing the terminal cost from the rest of the computation to simplify things and replacing J in the above from (2) gives

$$\begin{aligned}I\left(x_{N-1},1\right)&=\min_{u_{N-1}\in\Omega_{N-1}}\left\{J\left(x_{N-1},u_{N-1}\right)\right\}\\&=\min_{u_{N-1}\in\Omega_{N-1}}\left\{\left(x_{N-1}-u_{N-1}\right)^2+u_{N-1}^2\right\}\end{aligned} \tag{3}$$

Taking derivative in order to find optimal u_{N-1}^* results in

$$\begin{aligned}\frac{dI\left(x_{N-1},1\right)}{u_{N-1}}&=0\\2\left(x_{N-1}-u_{N-1}\right)(-1)+2u_{N-1}&=0\\4u_{N-1}-2x_{N-1}&=0\\u_{N-1}^*&=\frac{1}{2}x_{N-1}\end{aligned}$$

Using the above u_{N-1}^* in (3) gives

$$\begin{aligned}I^*\left(x_{N-1},1\right)&=\left(x_{N-1}-\frac{1}{2}x_{N-1}\right)^2+\left(\frac{1}{2}x_{N-1}\right)^2\\&=\frac{1}{2}x_{N-1}^2\end{aligned}$$

Going back one step

$$\begin{aligned}I\left(x_{N-2},2\right)&=\min_{u_{N-2}\in\Omega_{N-2}}\left\{J\left(x_{N-2},u_{N-2}\right)+I^*\left(x_{N-1},1\right)\right\}\\&=\min_{u_{N-2}\in\Omega_{N-2}}\left\{\left(x_{N-2}-u_{N-2}\right)^2+u_{N-2}^2+\frac{1}{2}x_{N-1}^2\right\}\end{aligned}$$

Before taking derivatives, we have to make all the stages to be at $N-2$, and for this we use the state equation to write x_{N-1} in terms of x_{N-2} and the above becomes

$$\begin{aligned}I\left(x_{N-2},2\right)&=\min_{u_{N-2}\in\Omega_{N-2}}\left\{\left(x_{N-2}-u_{N-2}\right)^2+u_{N-2}^2+\frac{1}{2}\left(x_{N-2}-u_{N-2}\right)^2\right\}\\&=\min_{u_{N-2}\in\Omega_{N-2}}\left\{\frac{5}{2}u_{N-2}^2-3u_{N-2}x_{N-2}+\frac{3}{2}x_{N-2}^2\right\}\end{aligned} \tag{4}$$

Now we take derivative to find optimal u_{N-2}^{*}

$$\begin{aligned}\frac{dI\left(x_{N-2},1\right)}{u_{N-2}}&=0\\5u_{N-2}-3x_{N-2}&=0\\u_{N-2}^{*}&=\frac{3}{5}x_{N-2}\end{aligned}$$

We go back to (4) and update with the optimal control found to obtain

$$\begin{aligned}I^{*}\left(x_{N-2},2\right)&=\frac{5}{2}\left(\frac{3}{5}x_{N-2}\right)^{2}-3\left(\frac{3}{5}x_{N-2}\right)x_{N-2}+\frac{3}{2}x_{N-2}^{2}\\&=\frac{3}{5}x_{N-2}^{2}\end{aligned}$$

Going back one more step

$$\begin{aligned}I\left(x_{N-3},3\right)&=\min_{u_{N-3}\in\Omega_{N-3}}\left\{J\left(x_{N-3},u_{N-3}\right)+I^{*}\left(x_{N-2},2\right)\right\}\\&=\min_{u_{N-3}\in\Omega_{N-3}}\left\{\left(x_{N-3}-u_{N-3}\right)^{2}+u_{N-3}^{2}+\frac{3}{5}x_{N-2}^{2}\right\}\end{aligned}$$

Before taking derivatives, we have to make all the states to be at $N-3$, and for this we use the state equation to write x_{N-2} in terms of x_{N-3} and the above becomes

$$\begin{aligned}I\left(x_{N-3},3\right)&=\min_{u_{N-3}\in\Omega_{N-3}}\left\{\left(x_{N-3}-u_{N-3}\right)^{2}+u_{N-3}^{2}+\frac{3}{5}\left(x_{N-3}-u_{N-3}\right)^{2}\right\}\\&=\min_{u_{N-2}\in\Omega_{N-2}}\left\{\frac{13}{5}u_{N-3}^{2}-\frac{16}{5}u_{N-3}x_{N-3}+\frac{8}{5}x_{N-3}^{2}\right\}\end{aligned}\tag{5}$$

Now we take derivative to find optimal u_{N-3}^{*}

$$\begin{aligned}\frac{dI\left(x_{N-3},3\right)}{u_{N-3}}&=0\\\frac{26}{5}u_{N-3}-\frac{16}{5}x_{N-3}&=0\\u_{N-3}^{*}&=\frac{8}{13}x_{N-3}\end{aligned}$$

We go back to (5) and update the cost to obtain

$$\begin{aligned}I^{*}\left(x_{N-3},3\right)&=\frac{13}{5}\left(\frac{8}{13}x_{N-3}\right)^{2}-\frac{16}{5}\left(\frac{8}{13}x_{N-3}\right)x_{N-3}+\frac{8}{5}x_{N-3}^{2}\\&=\frac{8}{13}x_{N-3}^{2}\end{aligned}$$

Let us do one more step backward,

$$\begin{aligned}I\left(x_{N-4},4\right)&=\min_{u_{N-4}\in\Omega_{N-4}}\left\{J\left(x_{N-4},u_{N-4}\right)+I^{*}\left(x_{N-3},3\right)\right\}\\&=\min_{u_{N-4}\in\Omega_{N-4}}\left\{\left(x_{N-4}-u_{N-4}\right)^{2}+u_{N-4}^{2}+\frac{8}{13}x_{N-3}^{2}\right\}\end{aligned}$$

Before taking derivatives, we have to make all the states to be at $N-4$, and for this we use the state equation to write x_{N-3} in terms of x_{N-4} and the above becomes

$$
\begin{aligned}
I\left(x_{N-4},4\right) &= \min_{u_{N-4}\in\Omega_{N-4}} \left\{ \left(x_{N-4}-u_{N-4}\right)^2 + u_{N-4}^2 + \frac{8}{13}\left(x_{N-4}-u_{N-4}\right)^2 \right\} \\
&= \min_{u_{N-4}\in\Omega_{N-4}} \left\{ \frac{34}{13}u_{N-4}^2 - \frac{42}{13}u_{N-4}x_{N-4} + \frac{21}{13}x_{N-4}^2 \right\} \qquad (6)
\end{aligned}
$$

Now we take derivative to find optimal u_{N-4}^*

$$
\begin{aligned}
\frac{dI\left(x_{N-4},4\right)}{u_{N-4}} &= 0 \\
\frac{68}{13}u_{N-4} - \frac{42}{13}x_{N-4} &= 0 \\
u_{N-4}^* &= \frac{21}{34}x_{N-4}
\end{aligned}
$$

We go back to (6) and update to obtain

$$
\begin{aligned}
I^*\left(x_{N-4},4\right) &= \frac{34}{13}\left(\frac{21}{34}x_{N-4}\right)^2 - \frac{42}{13}\left(\frac{21}{34}x_{N-4}\right)x_{N-4} + \frac{21}{13}x_{N-4}^2 \\
&= \frac{21}{34}x_{N-4}^2
\end{aligned}
$$

This is so much fun, so let us do one more step backward,

$$
\begin{aligned}
I\left(x_{N-5},5\right) &= \min_{u_{N-5}\in\Omega_{N-5}} \left\{ J\left(x_{N-5},u_{N-5}\right) + I^*\left(x_{N-4},4\right) \right\} \\
&= \min_{u_{N-5}\in\Omega_{N-5}} \left\{ \left(x_{N-5}-u_{N-5}\right)^2 + u_{N-5}^2 + \frac{21}{34}x_{N-4}^2 \right\}
\end{aligned}
$$

Before taking derivatives, we have to make all the states to be at $N-5$, and for this we use the state equation to write x_{N-4} in terms of x_{N-5} and the above becomes

$$
\begin{aligned}
I\left(x_{N-5},5\right) &= \min_{u_{N-5}\in\Omega_{N-5}} \left\{ \left(x_{N-5}-u_{N-5}\right)^2 + u_{N-5}^2 + \frac{21}{34}\left(x_{N-5}-u_{N-5}\right)^2 \right\} \\
&= \min_{u_{N-5}\in\Omega_{N-5}} \left\{ \frac{89}{34}u_{N-5}^2 - \frac{55}{17}u_{N-5}x_{N-5} + \frac{55}{34}x_{N-5}^2 \right\} \qquad (7)
\end{aligned}
$$

Now we take derivative to find optimal u_{N-5}^*

$$
\begin{aligned}
\frac{dI\left(x_{N-5},5\right)}{u_{N-5}} &= 0 \\
\frac{89}{17}u_{N-5} - \frac{55}{17}x_{N-5} &= 0 \\
u_{N-5}^* &= \frac{55}{89}x_{N-5}
\end{aligned}
$$

We go back to (7) and update to obtain

$$
\begin{aligned}
I^*\left(x_{N-5},5\right) &= \frac{89}{34}\left(\frac{55}{89}x_{N-5}\right)^2 - \frac{55}{17}\left(\frac{55}{89}x_{N-5}\right)x_{N-5} + \frac{55}{34}x_{N-5}^2 \\
&= \frac{55}{89}x_{N-5}^2
\end{aligned}
$$

Summary table of finding

k	$u^*(N-k)$	$I^*(x_{N-k},k)$
1	$\frac{1}{2}x_{N-1}$	$\frac{1}{2}x_{N-1}^2$
2	$\frac{3}{5}x_{N-2}$	$\frac{3}{5}x_{N-2}^2$
3	$\frac{8}{13}x_{N-3}$	$\frac{8}{13}x_{N-3}^2$
4	$\frac{21}{34}x_{N-4}^2$	$\frac{21}{34}x_{N-4}^2$
5	$\frac{55}{89}x_{N-5}$	$\frac{55}{89}x_{N-5}^2$

This is generated using bisection of the Fibonacci sequence, let $\gamma(k) = \frac{\beta(k)}{\alpha(k)}$ where[3]

$$\begin{aligned}\beta(k) &= 3\beta(k-1) - \beta(k-2)\\ \beta(0) &= 0\\ \beta(1) &= 1\end{aligned}$$

And[4]

$$\begin{aligned}\alpha(k) &= 3\alpha(k-1) - \alpha(k-2)\\ \alpha(0) &= 1\\ \alpha(1) &= 1\end{aligned}$$

Here is program which generates up to $k = 20$

```
Clear[k];
alpha[k_] := alpha[k] = If[k == 0 || k == 1, 1,
                          3*alpha[k - 1] - alpha[k - 2]]
beta[k_] := beta[k] = If[k == 0, 0,
                If[k == 1, 1, 3*beta[k - 1] - beta[k - 2]]];
Table[beta[k]/alpha[k + 1], {k, 1, 20}]
```

.

which gives

$\frac{1}{2}, \frac{3}{5}, \frac{8}{13}, \frac{21}{34}, \frac{55}{89}, \frac{144}{233}, \frac{377}{610}, \frac{987}{1597}, \frac{2584}{4181}, \frac{6765}{10946}, \frac{17711}{28657}, \frac{46368}{75025}, \frac{121393}{196418}, \frac{317811}{514229}, \frac{832040}{1346269}, \frac{2178309}{3524578}, \frac{5702887}{9227465}, \frac{14930352}{24157817}, \frac{39088169}{63245986}, \frac{102334155}{165580141}$

or numerically

$\{0.5, 0.6, 0.6153846153846154, 0.6176470588235294, 0.6179775280898876, 0.6180257510729614, 0.618032786885524$

The golden ratio is

$$\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887482036$$

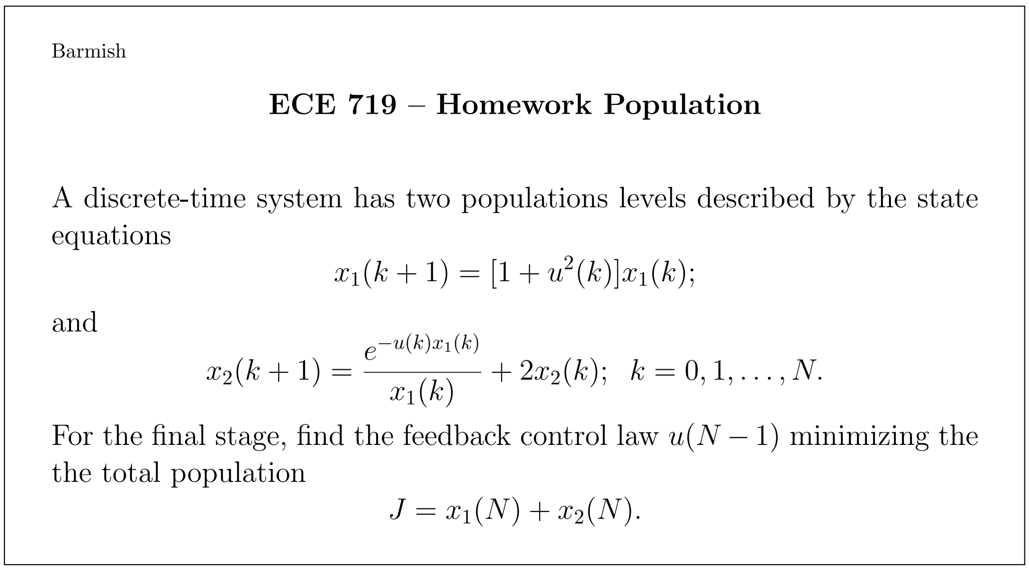
Therefore in the limit, for large k

$$\boxed{u^*(N-k) = \frac{1}{\varphi}x(N-k)}$$

4.7.3 Problem 3

[3] sequence is https://oeis.org/A001906

[4] sequence is https://oeis.org/A001519



Barmish

ECE 719 – Homework Population

A discrete-time system has two populations levels described by the state equations

$$x_1(k+1) = [1 + u^2(k)]x_1(k);$$

and

$$x_2(k+1) = \frac{e^{-u(k)x_1(k)}}{x_1(k)} + 2x_2(k); \quad k = 0, 1, \ldots, N.$$

For the final stage, find the feedback control law $u(N-1)$ minimizing the the total population

$$J = x_1(N) + x_2(N).$$

Figure 4.70: problem 3 description

Given

$$x_1(k+1) = \left[1 + u^2(k)\right] x_1(k)$$
$$x_2(k+1) = \frac{e^{-u(k)x_1(k)}}{x_1(k)} + 2x_2(k) \qquad k = 0, 1, \cdots, N$$

And the goal is to minimize the objective function at the terminal stage $J = x_1(N) + x_2(N)$. At stage $N-1$ with one step to go

$$I(x(N-1), 1) = \min_{u(N-1)\in\Omega_{N-1}} \{\Psi(x(N))\} \tag{1}$$

Where $\Psi(x(N)) = I(x(N), 0)$. Hence

$$\begin{aligned} I(x(N-1), 1) &= \min_{u(N-1)\in\Omega_{N-1}} \{x_1(N) + x_2(N)\} \\ &= \min_{u(N-1)\in\Omega_{N-1}} \left[1 + u^2(N-1)\right] x_1(N-1) + \frac{e^{-u(N-1)x_1(N-1)}}{x_1(N-1)} + 2x_2(N-1) \end{aligned}$$

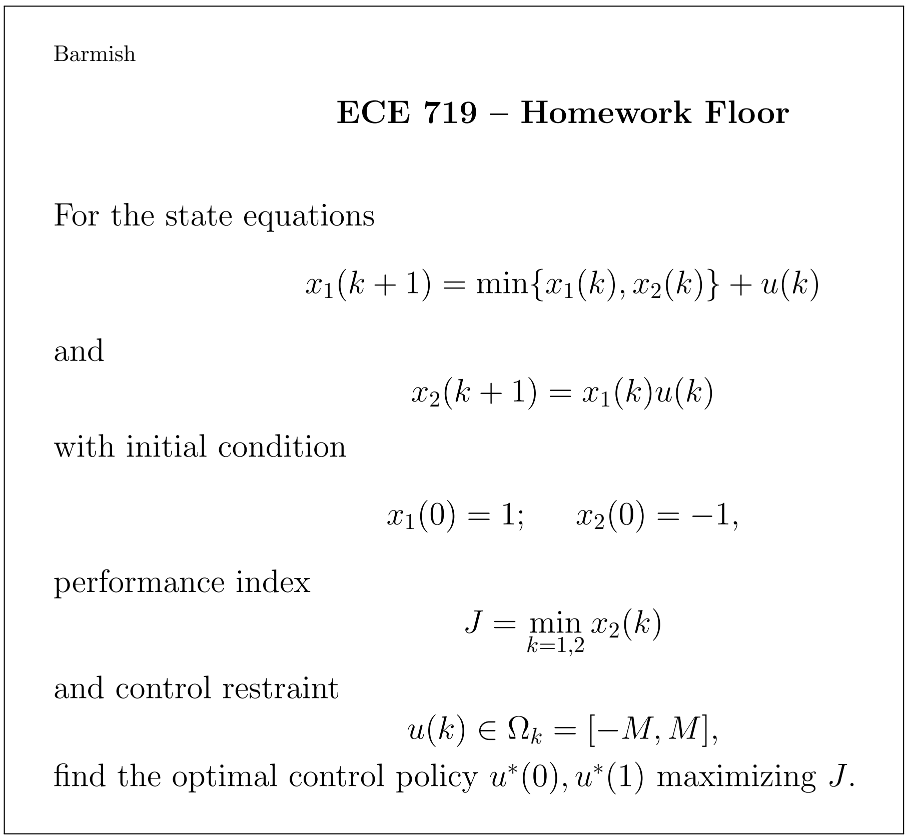
Therefore $\frac{dI(x(N-1))}{du(N-1)} = 0$ gives

$$\begin{aligned} 0 &= 2u(N-1)x_1(N-1) - e^{-u(N-1)x_1(N-1)} \\ e^{-u(N-1)x_1(N-1)} &= 2u(N-1)x_1(N-1) \end{aligned}$$

This is of the form $e^{-zx} = 2zx$ where $z \to u(N-1)$ and $x \to x_1(N-1)$, which has root at $z = \frac{0.35173}{x}$ (using Mathematica), hence the control law is

$$\boxed{u^*(N-1) = \frac{0.35173}{x_1(N-1)}}$$

4.7.4 Problem 4



Barmish

ECE 719 – Homework Floor

For the state equations

$$x_1(k+1) = \min\{x_1(k), x_2(k)\} + u(k)$$

and

$$x_2(k+1) = x_1(k)u(k)$$

with initial condition

$$x_1(0) = 1; \qquad x_2(0) = -1,$$

performance index

$$J = \min_{k=1,2} x_2(k)$$

and control restraint

$$u(k) \in \Omega_k = [-M, M],$$

find the optimal control policy $u^*(0), u^*(1)$ maximizing J.

Figure 4.71: problem 4 description

The following diagram shows the layout of the stages we need to use. There are three stages. $k = 2$ is the terminal stage, and $k = 0$ is the initial stage.

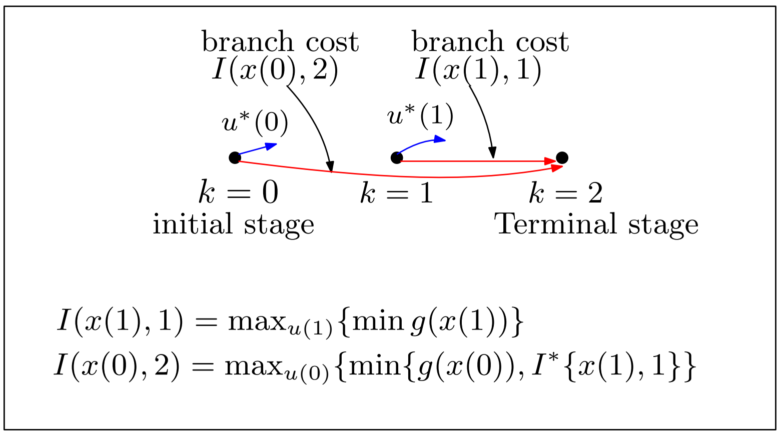


Figure 4.72: problem 4 stages

We have

$$\begin{aligned}
J &= \min_{k=1,2} x_2(k) \\
x_1(k+1) &= \min\{x_1(k), x_2(k)\} + u(k) \\
x_2(k+1) &= x_1(k)u(k) \\
x_1(0) &= 1 \\
x_2(0) &= -1
\end{aligned}$$

One step to go, where $N = 2$

$$\begin{aligned}
I(x(N-1),1) &= I(x(2-1),1) \\
&= I(x(1),1) \\
&= \max_{u(1)\in\Omega_1} \{x_2(2)\}
\end{aligned}$$

But $x_2(2) = x_1(1)u(1)$ from the state equation, hence

$$I(x(1),1) = \max_{u(1)\in\Omega_1} \{x_1(1)u(1)\}$$

We need to find $u(1)$ which maximizes $x_1(1)u(1)$. Since $u(k) \in \Omega_k = [-M, M]$ then

$$\boxed{u^*(1) = M\text{sign}(x(1))}$$

Therefore

$$\begin{aligned} I^*(x(1),1) &= x_1(1)u^*(1) \\ &= x_1(1)M \text{ sign}(x(1)) \\ &= M|x_1(1)| \end{aligned}$$

Now we go one more step backward

$$I(x(0),2) = \max_{u(0)\in\Omega_0} \min\{x_2(1), I^*(x(1),1)\}$$

But from state equation, $x_2(1) = x_1(0)u(0)$ and since $x_1(0) = 1$ then $x_2(1) = u(0)$ and the above becomes

$$I(x(0),2) = \max_{u(0)\in\Omega_0} \min\{u(0), M|x_1(1)|\} \tag{1}$$

But

$$\begin{aligned} x_1(1) &= \min\{x_1(0), x_2(0)\} + u(0) \\ &= \min\{1,-1\} + u(0) \\ &= -1 + u(0) \end{aligned}$$

Therefore (1) becomes

$$I(x(0),2) = \max_{u(0)\in\Omega_0} \min\{u(0), M|-1+u(0)|\}$$

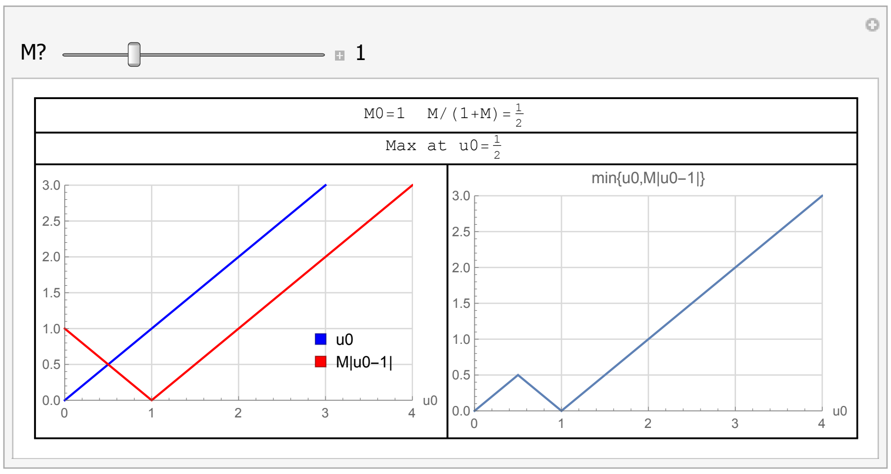
Let

$$F(u(0)) = \min\{u(0), M|-1+u(0)|\}$$

We need to find $\max_{u(0)} \min(F(u(0)))$. By making small program and adjusting M, to see all the regions, the following result was found for $u^*(0)$

M range	optimal
$0 \le M \le \sqrt{2}$	$u_0^* = \frac{M}{1+M}$
$\sqrt{2} < M$	$u_0^* = M$

The following is a plot of the small program written showing $F(u(0))$ for first case $0 \le M \le \sqrt{2}$ and another plot showing the case for $\sqrt{2} < M$. No other cases were found. The program allows one to move a slider to adjust M and it finds the maximizing u^* for $F(u)$ at each slider change.



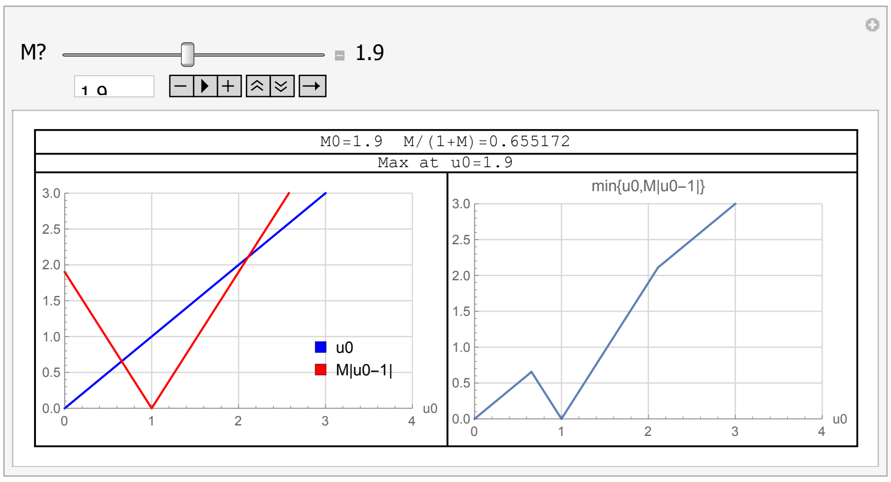
Out[55]=

M? 1

M0=1 M/(1+M)=$\frac{1}{2}$

Max at u0=$\frac{1}{2}$

Figure 4.73: case for $0 \leq M \leq \sqrt{2}$



Out[55]=

M? 1.9

1.9

M0=1.9 M/(1+M)=0.655172

Max at u0=1.9

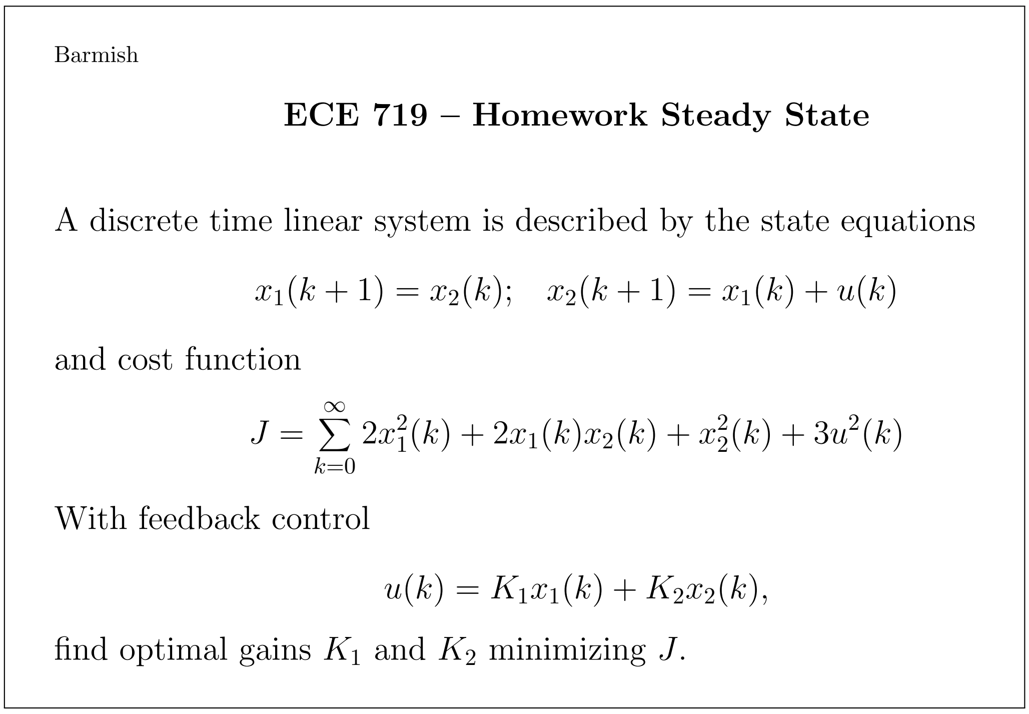
Figure 4.74: case for $M \geq \sqrt{2}$

Source code for the above

```
Manipulate[
  p0 = Plot[f[u0, M0], {u0, 0, 4}, PlotRange -> {{0, 4}, {0, 3}},
        PlotLabel -> "min{u0,M|u0-1|}",
        ImageSize -> 300, AxesLabel -> {"u0", None},
        GridLines -> Automatic, GridLinesStyle -> LightGray];

  p1 = Plot[u0, {u0, 0, 4}, PlotRange -> {{0, 4}, {0, 3}},
            PlotStyle -> Blue, ImageSize -> 300,
            AxesLabel -> {"u0", None}, GridLines -> Automatic,
            GridLinesStyle -> LightGray];

  p2 = Plot[M0*Abs[u0 - 1], {u0, 0, 4},
        PlotRange -> {{0, 4}, {0, 3}},
        PlotStyle -> Red, ImageSize -> 300];

  u0Max = ArgMax[{Min[u0, M0*Abs[u0 - 1]], u0 >= -M0 && u0 <= M0}, u0];

  p4 = Grid[{{Row[{"M0=", M0, "  M/(1+M)=", M0/(1 + M0)}],
            SpanFromLeft}, {Row[{"Max at u0=", u0Max}],
            SpanFromLeft}, {Legended[Show[p1, p2],
              Placed[SwatchLegend[{Blue, Red}, {"u0", "M|u0-1|"}],
                     {{0.7, 0.1}, {0, 0}}]], p0}},
       Frame -> All];

 p4,

 {{M0, 1, "M?"}, 0, 4, 0.01, Appearance -> "Labeled"},
Initialization :> (
  f[u0_, M0_] := Min[u0, M0*Abs[u0 - 1]]
)
]
```

.

4.7.5 Problem 5



Barmish

ECE 719 – Homework Steady State

A discrete time linear system is described by the state equations

$$x_1(k+1) = x_2(k); \quad x_2(k+1) = x_1(k) + u(k)$$

and cost function

$$J = \sum_{k=0}^{\infty} 2x_1^2(k) + 2x_1(k)x_2(k) + x_2^2(k) + 3u^2(k)$$

With feedback control

$$u(k) = K_1 x_1(k) + K_2 x_2(k),$$

find optimal gains K_1 and K_2 minimizing J.

Figure 4.75: problem 5 description

$$\begin{pmatrix} x_1(k+1) \\ x_2(k+1) \end{pmatrix} = \overbrace{\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}}^{A} \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix} + \overbrace{\begin{pmatrix} 0 \\ 1 \end{pmatrix}}^{B} u(k)$$

$$J = \sum_{k=0}^{\infty} 2x_1^2(k) + 2x_1(k)x_2(k) + x_2^2(k) + 3u^2(k)$$

Since J has the form $x^TQx + u^TRu$, then we need to find Q first. (Q is symmetric), therefore

$$\begin{aligned}
2x_1^2(k) + 2x_1(k)x_2(k) + x_2^2(k) &= \begin{pmatrix} x_1(k) & x_2(k) \end{pmatrix} \begin{pmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix} \\
&= \begin{pmatrix} x_1(k)q_{11} + x_2(k)q_{12} & x_1(k)q_{12} + x_2(k)q_{22} \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix} \\
&= x_1^2(k)q_{11} + 2q_{12}x_1(k)x_2(k) + x_2^2(k)q_{22}
\end{aligned}$$

Comparing coefficients, we see that $q_{11} = 2, q_{22} = 1, q_{12} = 1$, hence

$$Q = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$$

And R is scalar

$$R = 3$$

Therefore, the discrete algebraic Riccati equation is

$$P = APA^T - A^TPB\left(R + B^TPB\right)^{-1} B^TPA + Q \tag{1}$$

Small note: The above is what we had in lecture notes. Matlab has the above in its help pages slightly different. The first term is written as A^TPA instead of APA^T as we had.

Using Matlab

```
>> A=[0,1;1,0];B=[0;1];Q=[2,1;1,1];R=3;
>> [P,L,G] = dare(A,B,Q,R)
```

```
P =
    3.7841    1.6815
    1.6815    4.4022
```

$$P=\begin{pmatrix}3.7841 & 1.6815\\ 1.6815 & 4.4022\end{pmatrix}$$

Notice that P is symmetric as expected.

Let us check it is correct first. Substituting P in RHS of (1) gives

$$\begin{aligned}P=&\begin{pmatrix}0&1\\1&0\end{pmatrix}\begin{pmatrix}3.7841 & 1.6815\\ 1.6815 & 4.4022\end{pmatrix}\begin{pmatrix}0&1\\1&0\end{pmatrix}^T-\begin{pmatrix}0&1\\1&0\end{pmatrix}^T\begin{pmatrix}3.7841 & 1.6815\\ 1.6815 & 4.4022\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\\ &\left(3+\begin{pmatrix}0\\1\end{pmatrix}^T\begin{pmatrix}3.7841 & 1.6815\\ 1.6815 & 4.4022\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\right)^{-1}\begin{pmatrix}0\\1\end{pmatrix}^T\begin{pmatrix}3.7841 & 1.6815\\ 1.6815 & 4.4022\end{pmatrix}\begin{pmatrix}0&1\\1&0\end{pmatrix}\\ &+\begin{pmatrix}2&1\\1&1\end{pmatrix}\end{aligned}$$

The above gives $\begin{pmatrix}3.7841 & 1.6815\\ 1.6815 & 4.4022\end{pmatrix}$ which is P. Yes, it satisfies DARE. Now, using

$$\begin{aligned}u^{*}&=-\left(R+B^TPB\right)^{-1}B^TPAx\\ &=-\left(3+\begin{pmatrix}0\\1\end{pmatrix}^T\begin{pmatrix}3.7841 & 1.6815\\ 1.6815 & 4.4022\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\right)^{-1}\begin{pmatrix}0\\1\end{pmatrix}^T\begin{pmatrix}3.7841 & 1.6815\\ 1.6815 & 4.4022\end{pmatrix}\begin{pmatrix}0&1\\1&0\end{pmatrix}\begin{pmatrix}x_1(k)\\x_2(k)\end{pmatrix}\\ &=-0.59472x_1(k)-0.22716x_2(k)\end{aligned}$$

Therefore, comparing the above to $u^{*}(k)=K_1x_1(k)+K_2x_2(k)$ we see that

$$\begin{aligned}K_1&=-0.59472\\ K_2&=-0.22716\end{aligned}$$

To simulate the result under the new control law u^{*}, we need some initial condition on state. Below is small script which simulate this up to $k=20$ stages with the plot generated

```
%simulate x1 and x2 states under optimal control law
%found in problem 5, HW 7. Plot is attached

close all; clear;
A=[0,1;1,0];B=[0;1];
k1=-0.59472; k2=-0.22716;  %found using dare()

N=20;
x=zeros(2,20);
x(:,1)=[1.5;1];   %need non-zero initial state!
for i=2:N
    x(:,i)=A*x(:,i-1)+B*(k1*x(1,i-1)+k2*x(2,i-1));
end
subplot(1,2,1);
plot(1:N,x(1,:),'r',1:N,x(1,:),'r.');
title('x1 using optimal u');
grid;
subplot(1,2,2);
plot(1:N,x(2,:),'b',1:N,x(2,:),'b.');
title('x2, using optimal u');
grid;
```

.

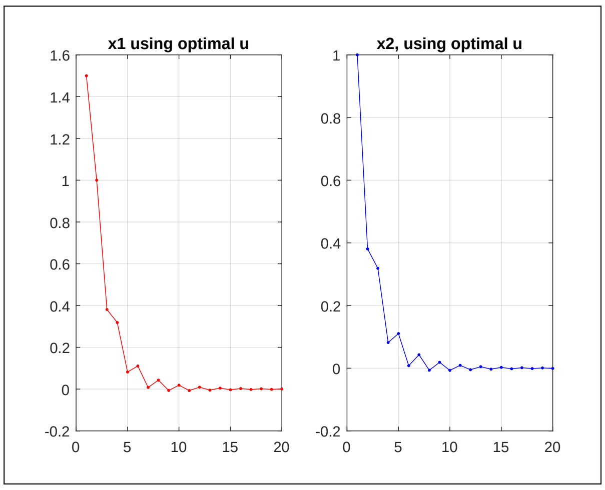
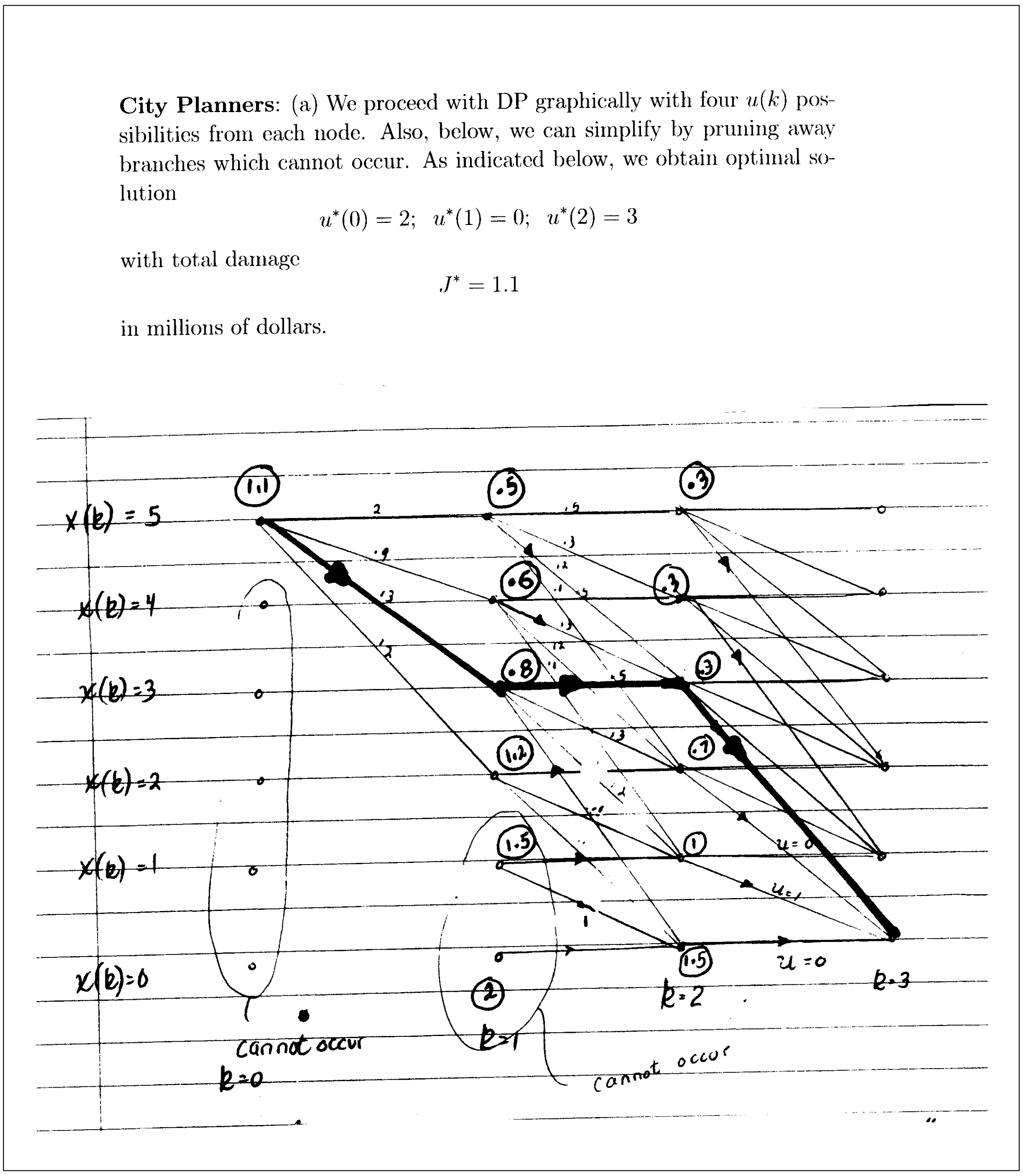


Figure 4.76: problem 5 plot

4.7.6 HW 7 key solution



City Planners: (a) We proceed with DP graphically with four $u(k)$ possibilities from each node. Also, below, we can simplify by pruning away branches which cannot occur. As indicated below, we obtain optimal solution

$$u^*(0) = 2; \quad u^*(1) = 0; \quad u^*(2) = 3$$

with total damage

$$J^* = 1.1$$

in millions of dollars.

x(k) = 5
x(k)=4
x(k)=3
x(k)=2
x(k) =1
x(k)=0
1.1
.5
.3
.6
.3
.8
.3
1.2
.7
1.5
1
1.5
2
2
.5
.9
.3
.2
u= 0
u=1
u =0
cannot occur
k=0
k=1
k=2
k=3
cannot occur

(b) In this case, the added constraint

$$3u(0) + u(1) + 2u(2) \leq 9$$

helps in pruning the state diagram. Letting $x(k)$ be the amount in millions expended up to stage k, we have

$$x(0) = 0;\ \ x(1) = x(0) + 3u(0)\ \ x(2) = x(1) + u(1)\ \ x(3) = x(2) + 2u(2)$$

with state constraints

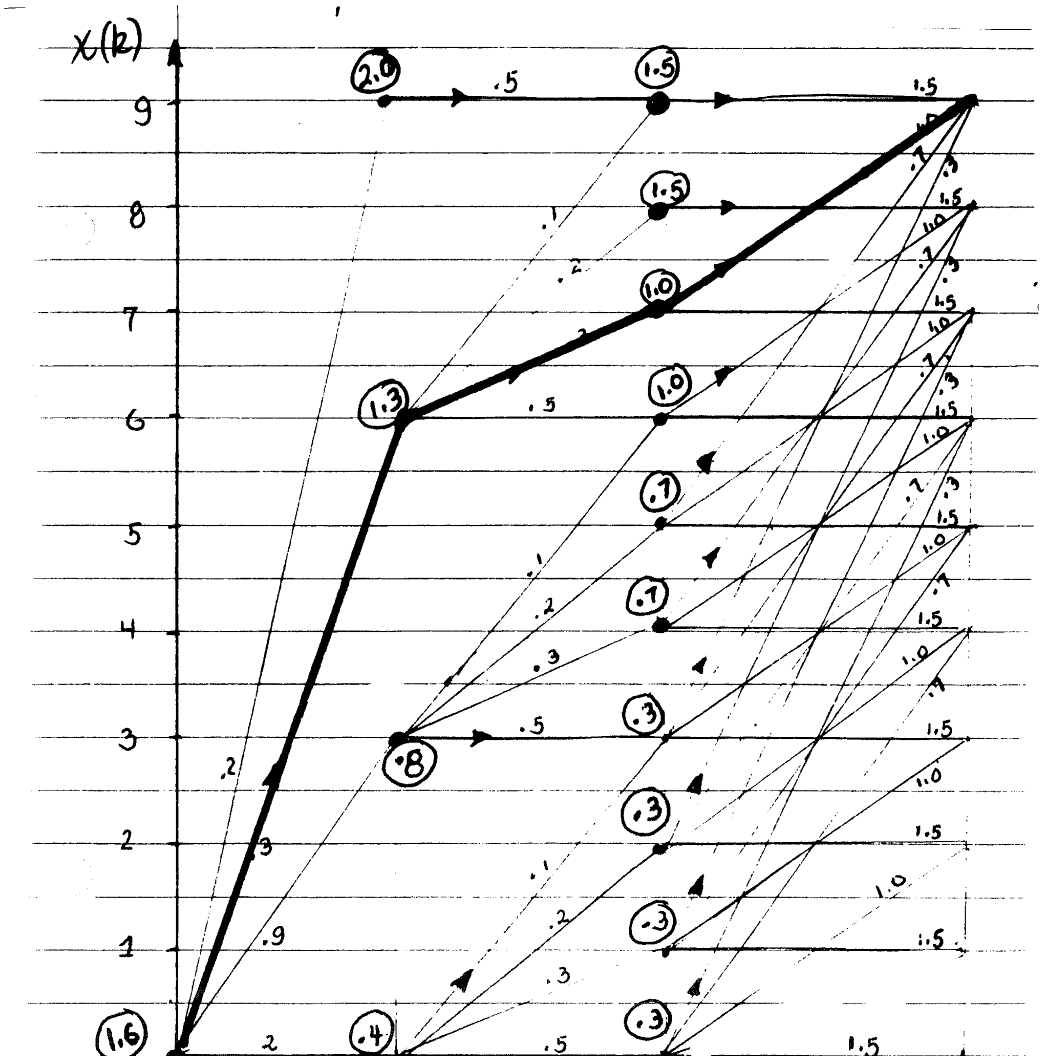
$$x(k) \leq 9;\quad u(k) \leq 3;\quad k = 1, 2, 3;$$

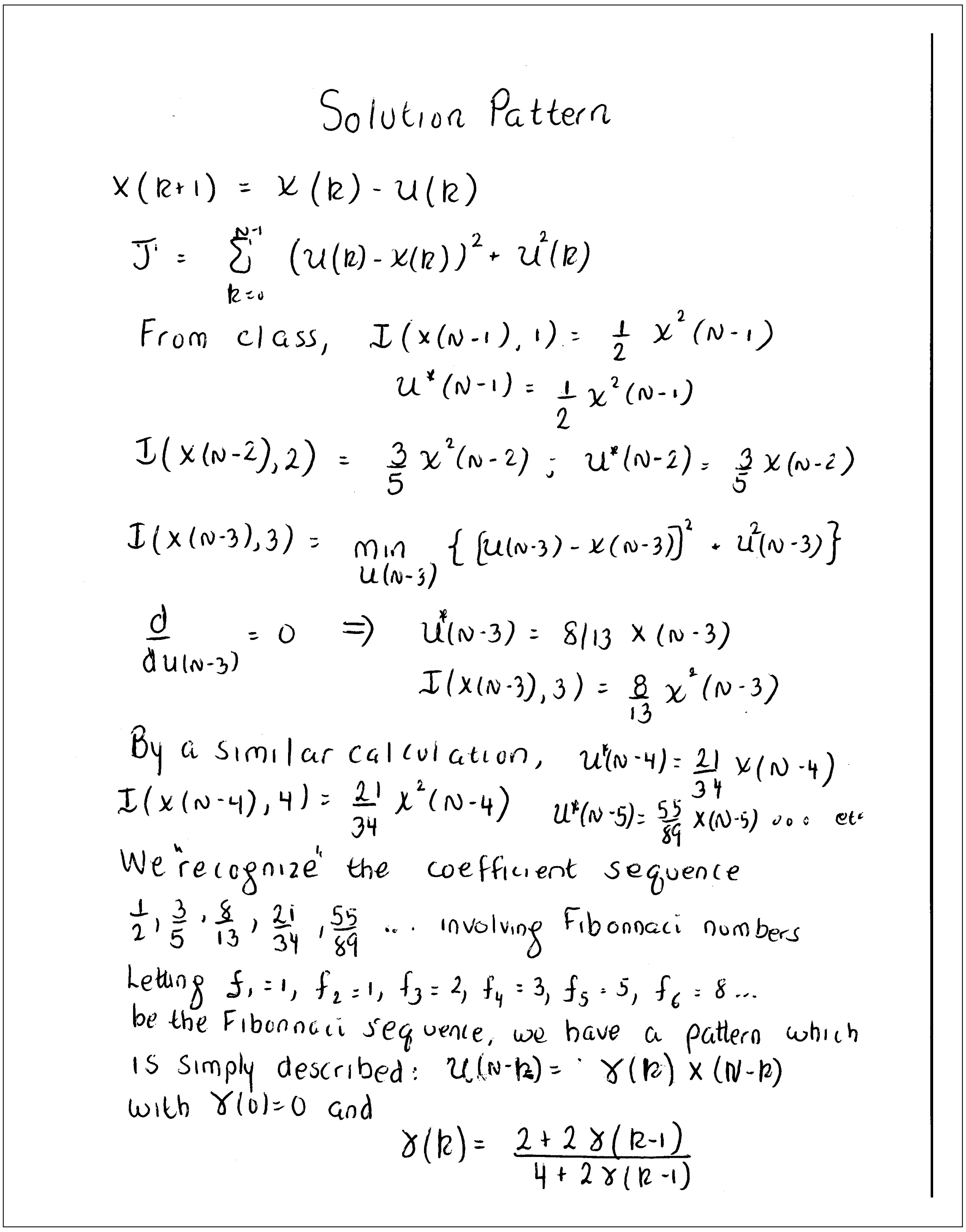
As indicated below, this leads to optimal decision

$$u^*(0) = 2;\ \ u^*(1) = 1;\ \ u^*(2) = 1$$

and associated damage in millions given by

$$J^* = 1.6.$$





Solution Pattern

$$x(k+1) = x(k) - u(k)$$

$$J = \sum_{k=0}^{N-1} (u(k) - x(k))^2 + u^2(k)$$

From class, $I(x(N-1), 1) = \frac{1}{2} x^2(N-1)$

$$u^*(N-1) = \frac{1}{2} x^2(N-1)$$

$$I(x(N-2), 2) = \frac{3}{5} x^2(N-2) \; ; \; u^*(N-2) = \frac{3}{5} x(N-2)$$

$$I(x(N-3), 3) = \min_{u(N-3)} \left\{ [u(N-3) - x(N-3)]^2 + u^2(N-3) \right\}$$

$$\frac{d}{du(N-3)} = 0 \Rightarrow u^*(N-3) = 8/13 \; x(N-3)$$

$$I(x(N-3), 3) = \frac{8}{13} x^2(N-3)$$

By a similar calculation, $u^*(N-4) = \frac{21}{34} x(N-4)$

$I(x(N-4), 4) = \frac{21}{34} x^2(N-4)$ $u^*(N-5) = \frac{55}{89} x(N-5)$ ... etc

We "recognize" the coefficient sequence

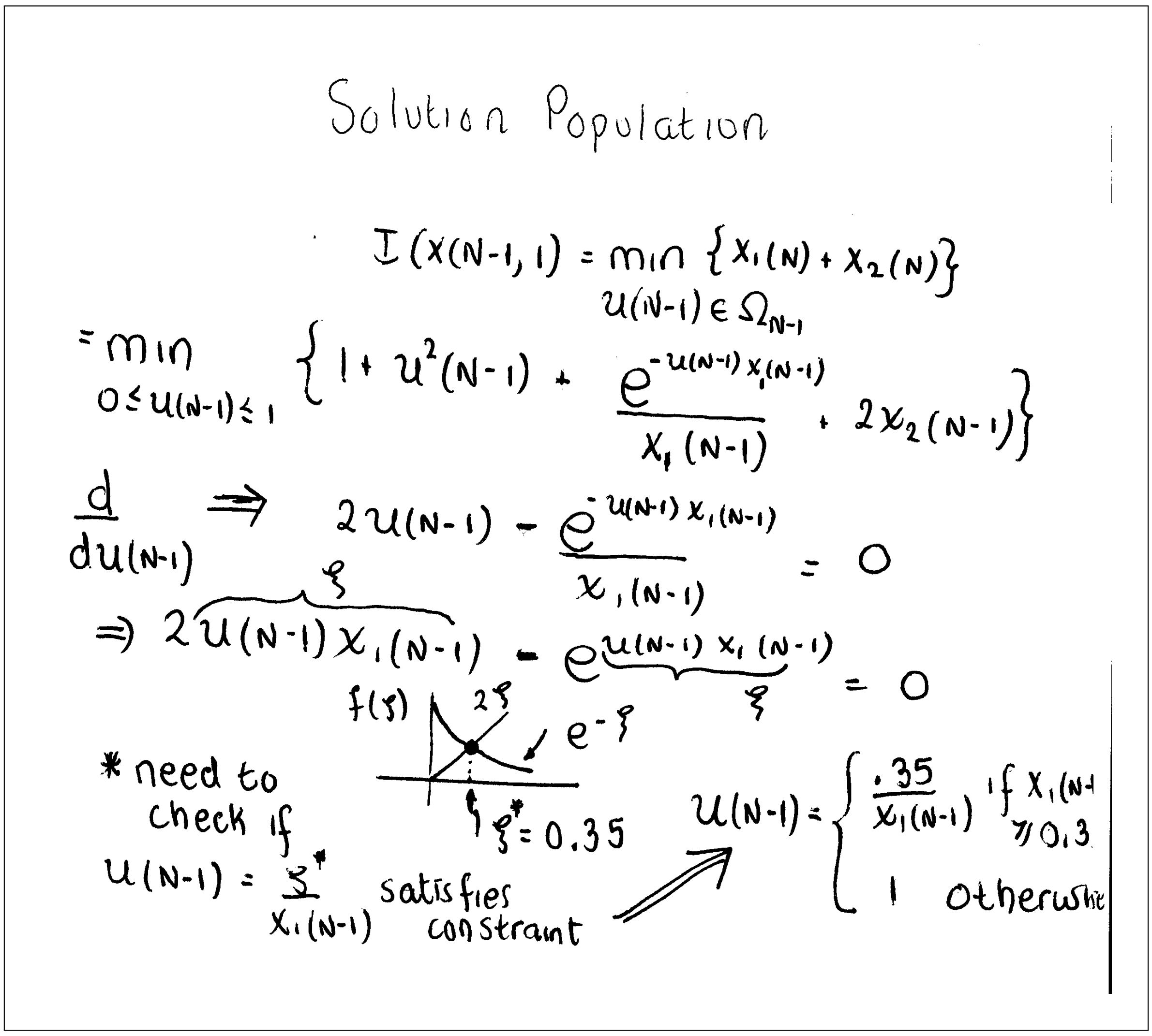
$\frac{1}{2}, \frac{3}{5}, \frac{8}{13}, \frac{21}{34}, \frac{55}{89}$ ... involving Fibonnaci numbers

Letting $f_1 = 1$, $f_2 = 1$, $f_3 = 2$, $f_4 = 3$, $f_5 = 5$, $f_6 = 8$ ...

be the Fibonnaci sequence, we have a pattern which is simply described: $u(N-k) = \gamma(k) \, x(N-k)$

with $\gamma(0) = 0$ and

$$\gamma(k) = \frac{2 + 2\gamma(k-1)}{4 + 2\gamma(k-1)}$$



Solution Population

$$I(x(N-1,1) = \min_{u(N-1)\in\Omega_{N-1}} \{x_1(N)+x_2(N)\}$$

$$= \min_{0\le u(N-1)\le 1} \left\{1+u^2(N-1)+\frac{e^{-u(N-1)x_1(N-1)}}{x_1(N-1)}+2x_2(N-1)\right\}$$

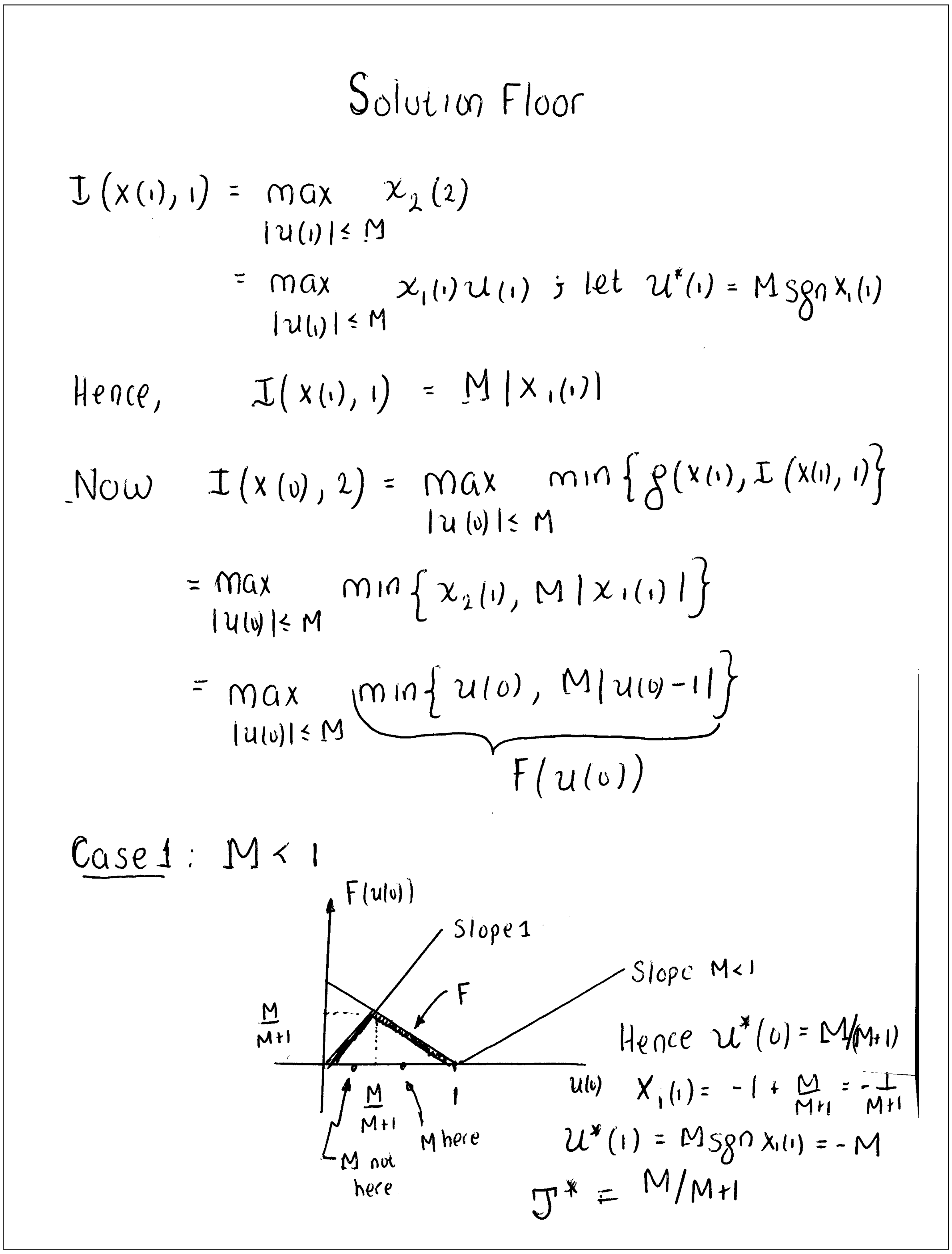
$$\frac{d}{du(N-1)} \Rightarrow 2u(N-1)-\frac{e^{-u(N-1)x_1(N-1)}}{x_1(N-1)} = 0$$

$$\Rightarrow 2\overbrace{u(N-1)x_1(N-1)}^{\xi} - e^{\underbrace{u(N-1)x_1(N-1)}_{\xi}} = 0$$

$f(\xi)$
2ξ
$e^{-\xi}$
$\xi^* = 0.35$

* need to check if $u(N-1) = \frac{\xi^*}{x_1(N-1)}$ satisfies constraint

$$u(N-1) = \begin{cases} \frac{.35}{x_1(N-1)} & \text{if } x_1(N\text{-} \ge 0.3 \\ 1 & \text{otherwis} \end{cases}$$



Solution Floor

$$I(x(1),1) = \max_{|u(1)|\le M} x_2(2)$$

$$= \max_{|u(1)|\le M} x_1(1)u(1) \text{ ; let } u^*(1) = M\,\mathrm{sgn}\,x_1(1)$$

Hence, $I(x(1),1) = M|x_1(1)|$

Now $I(x(0),2) = \max_{|u(0)|\le M} \min\{g(x(1)), I(x(1),1)\}$

$$= \max_{|u(0)|\le M} \min\{x_2(1), M|x_1(1)|\}$$

$$= \max_{|u(0)|\le M} \underbrace{\min\{u(0), M|u(0)-1|\}}_{F(u(0))}$$

Case 1 : $M < 1$

F(u(0))
Slope 1
Slope M<1
F
$\frac{M}{M+1}$
u(0)
$\frac{M}{M+1}$
1
M here
M not here

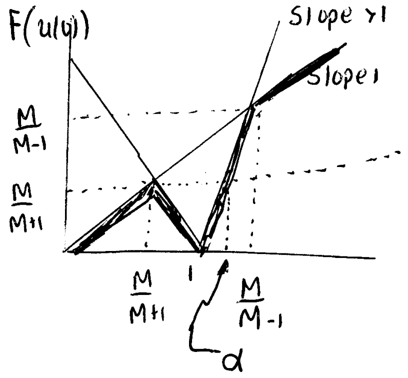
Hence $u^*(0) = M/(M+1)$

$X_1(1) = -1 + \frac{M}{M+1} = -\frac{1}{M+1}$

$u^*(1) = M\,\mathrm{sgn}\,x_1(1) = -M$

$J^* = M/M+1$

Case 2: $1 \leq M < \sqrt{2}$



For $M < \sqrt{2}$, the point $u(0) = M$ is left of α

Hence solution (idea) the same as Case 1

$u^*(0) = \overbrace{u^*(1)}^{(M/M+1)} = -M$

$J^* = \frac{M}{M+1}$

Case 3: When $M = \sqrt{2}$, as seen from figure above, obtain two solutions

First solution: Same as case 1. $\overbrace{u^*(0) = u^*(1)}^{M/M+1} = -M$

$J^* = \frac{M}{M+1}$; i.e. $u^*(0) = \frac{\sqrt{2}}{1+\sqrt{2}}$ $u^*(1) = -\sqrt{2}$

$J^* = \sqrt{2}/(1+\sqrt{2})$

Second solution $u^*(0) = \sqrt{2}$ $u^*(1) = -\sqrt{2}$

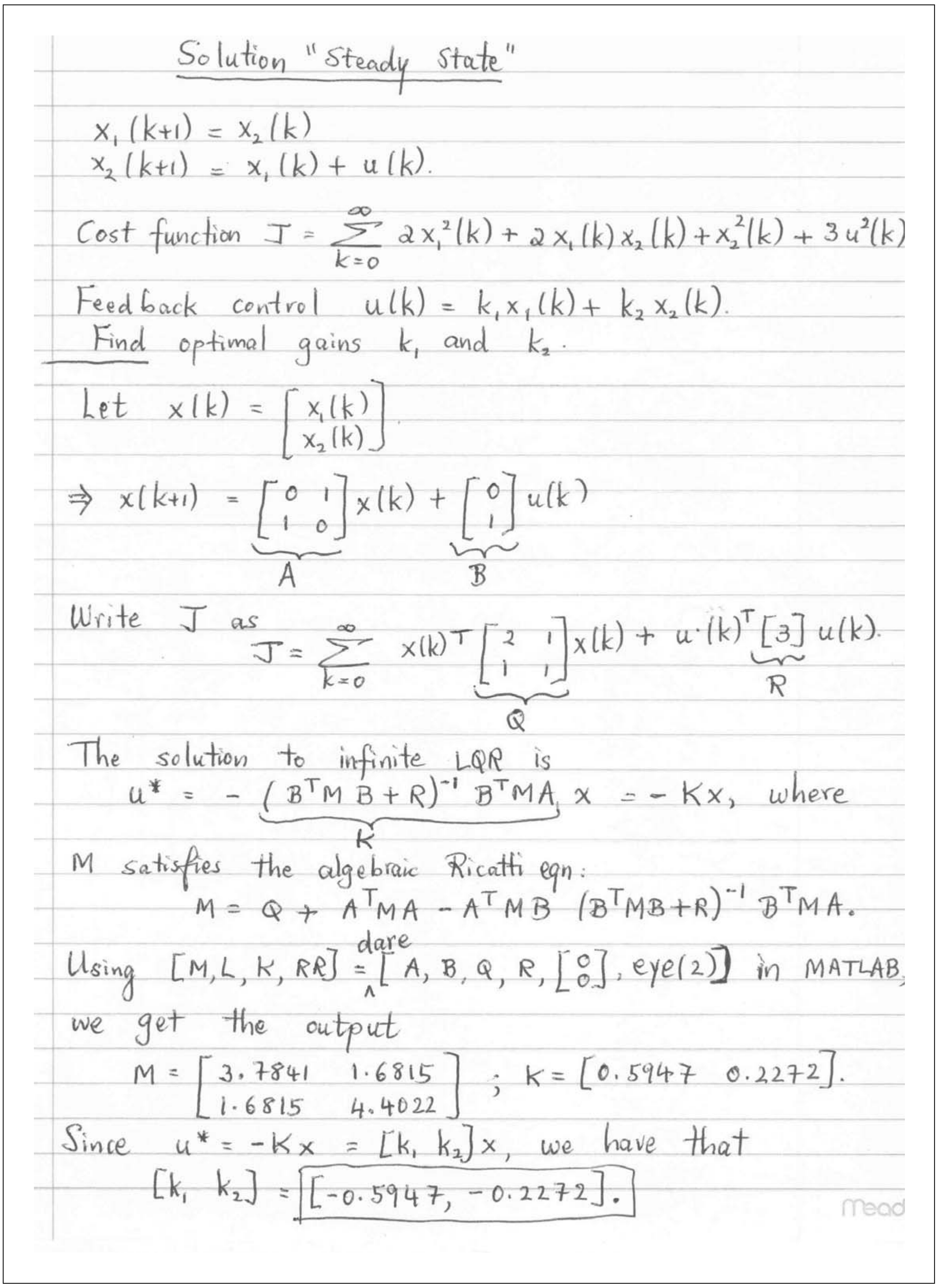
$J^* = \sqrt{2}/(1+\sqrt{2})$

Case 4 $\sqrt{2} < M < 2$ The point $u(0) = M$ in figure is left of $M/M-1$. Hence $u^*(0) = M$

$x_1(1) = -1 + M$ Hence $u^*(1) = M \, \text{sgn}(\overbrace{M-1}^{>0}) = M;$

$J^* = M(M-1)$

Case 5: $M \geq 2$ Again $u^*(0) = u^*(1) = M$ but $J^* = M$



Solution "Steady State"

$x_1(k+1) = x_2(k)$
$x_2(k+1) = x_1(k) + u(k).$

Cost function $J = \sum_{k=0}^{\infty} 2x_1^2(k) + 2x_1(k)x_2(k) + x_2^2(k) + 3u^2(k)$

Feedback control $u(k) = k_1 x_1(k) + k_2 x_2(k).$
Find optimal gains k_1 and k_2.

Let $x(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$

$$\Rightarrow x(k+1) = \underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}}_{A} x(k) + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{B} u(k)$$

Write J as

$$J = \sum_{k=0}^{\infty} x(k)^T \underbrace{\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}}_{Q} x(k) + u(k)^T \underbrace{[3]}_{R} u(k).$$

The solution to infinite LQR is

$$u^* = -\underbrace{(B^T M B + R)^{-1} B^T M A}_{K} x = -Kx, \text{ where}$$

M satisfies the algebraic Ricatti eqn:

$$M = Q + A^T M A - A^T M B (B^T M B + R)^{-1} B^T M A.$$

Using $[M, L, K, RR] = \text{dare}[A, B, Q, R, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \text{eye}(2)]$ in MATLAB, we get the output

$$M = \begin{bmatrix} 3.7841 & 1.6815 \\ 1.6815 & 4.4022 \end{bmatrix} ; \quad K = [0.5947 \quad 0.2272].$$

Since $u^* = -Kx = [k_1 \ k_2]x$, we have that

$$[k_1 \ k_2] = \boxed{[-0.5947, -0.2272].}$$

Mead

4.8 HW special problem

abstract

`k-means++` cluster analysis was used to partition the population area such that the center of each partition minimizes the within the partition sum of distance squares of each point in the partition to the center of the partition. The number of customers that would visit our stores located at the center of the partitions was then determined. The number of partitions was increased and the calculation repeated on the larger set by trying all of the different combinations of allocating the stores in the new and larger set of partitions. The largest score was selected. Matlab's `kmeans` implementation in the `Statistics and Machine Learning Toolbox` was used to find the set of partitions and their centroids. kmeans++ clustering is known to be computationally NP-hard problem[5]. In addition, the time complexity to analyze each set of partitions is $\mathcal{O}\left(N\binom{p}{n}\right)$ where p is the number partitions and N is the size of the population. This number quickly becomes very large therefore the implementation limits the number of partitions p to no more than 15. A number of small test cases with known optimal store locations were constructed and the algorithm was verified to be correct by direct observations. Locations of competitor stores do not affect the decision to where to locate our stores. Competitor stores locations affects the number of customers our stores will attract, but not the optimal locations of our stores.

4.8.1 Introduction

The problem is the following: We want to locate n stores in an area of given population distribution where there already exists m number of competitor stores. We are given the locations of the competitor stores. We want to find the optimal locations of our n stores such that we attract the largest number of customers by being close to as many as possible. We are given the locations (coordinates) of the population.

4.8.2 Analysis of the problem

The first important observation found is that the locations of the competitor stores did not affect the decision where the location of our stores should be. This at first seemed counter intuitive. But the optimal solution is to put our stores at the center of the most populated partitions even if the competitor store happened to also be in the same exact location.

The idea is that it is better to split large number of customers with the competition, than locate a store to attract all customers but in a less populated area. This was verified using small test cases (not shown here due to space limitation).

This is where cluster analysis using the kmeans++ algorithm was used. Cluster analysis is known algorithm that partitions population into number of clusters or partitions such that each cluster has the property that its centroid has minimum within-cluster sum of squares of the distance to each point in the cluster. The following is the formal definition of kmeans++ clustering algorithm taken from `https://en.wikipedia.org/wiki/K-means_clustering`

Given a set of observations $(x_1, x_1, \dots, x_n)$, where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into $k \leq n$ sets $S = \{S_1, S_2, \dots, S_k\}$ so as to minimize the within-cluster sum of squares (WCSS) (sum of distance functions of each point in the cluster to the K center). In other words, its objective is to find:

$$\underset{S}{\arg\min} \sum_{i=1}^{k} \sum_{x \in S_i} \left\| x - \mu_i \right\|^2$$

where μ_i is the mean of points in S_i

The main difficulty was in deciding on the number of partitions needed. Should we try smaller number than the number of stores, and therefore put more than one store in

[5] non-deterministic polynomial-time hard

same location? Using smaller number of clusters than the number of stores was rejected, since it leads to no improvement in the score (Putting two stores in same location means other areas are not served since we have limited number of stores). Or should we try more partitions than the number of our stores, and then try all the combinations possible between these partitions to find one which gives the larger score? This the approach taken in this algorithm. It was found that by increasing the number of partitions than the number of stores, and trying all possible combinations $\binom{p}{n}$, where p is the number of partitions, a set could be found which has higher than if we used the same number of partitions as the number of stores. The problem with this method is that it has $\mathcal{O}\binom{p}{n}$ time complexity. This quickly becomes large and not practical when $p > 15$. In the test cases used however, there was no case found where p had to be more than two or three larger than n. The implementing limits the number of partitions to 15.

When a score is found which is smaller than the previous score, the search stops as this means the maximum was reached. This was determined by number of trials where it was found that once the score become smaller than before, making more partitions did not make it go up again. There is no proof of this, but this was only based on trials and observations. Therefore, the search stops when a score starts to decrease.

The implementation described here is essentially an illustration of the use of cluster analysis, as provided by Matlab, in order to solve the grocery stores location problem. The appendix contains the source code written to solve this problem.

Before describing the algorithm below, we show an example using the early test send to us to illustrate how this method works. This used $500,000$ population with 5 competitor stores (the black dots) in the plots that follows and $n = 4$ (the green dots).

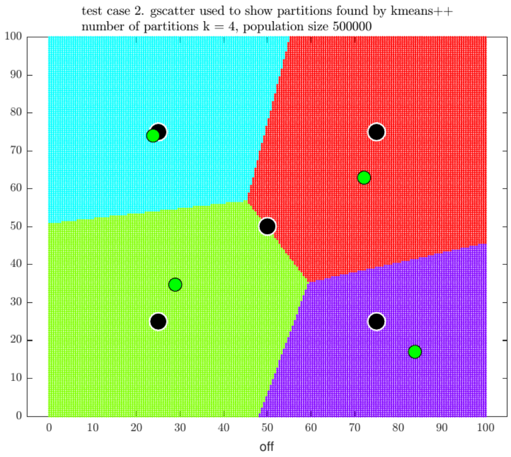


Figure 4.77: Partitions found by kmeans++ with centroid as green dots and competitor sores as black dots

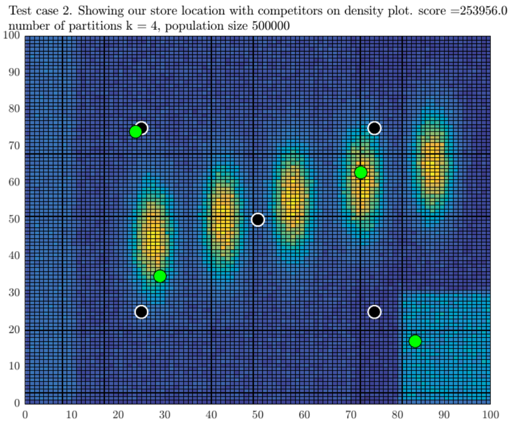


Figure 4.78: Density of population with corresponding store locations found

The partitions were now increased to 5 and $\binom{5}{4}$ different combinations were scored to find the 4 best store locations out of these. This resulted in the following result

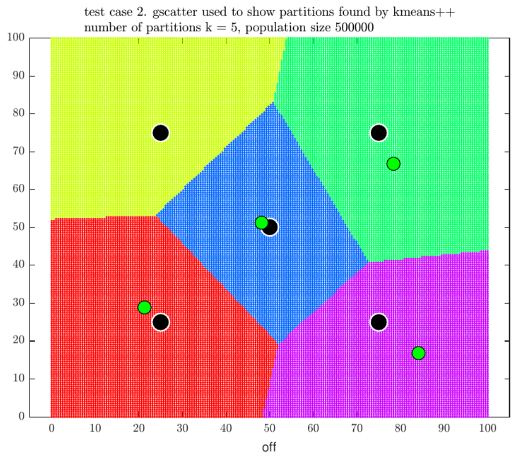


Figure 4.79: Partitions found by kmeans++ with centroid as green dots and competitor sores as black dots

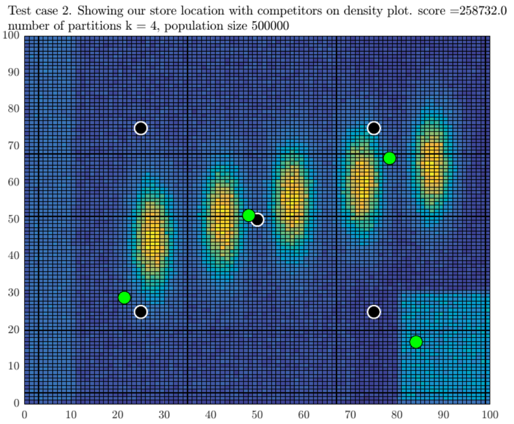


Figure 4.80: Density of population with corresponding store locations found

When trying 6 partitions, the score was decreased, so the search stopped. The program then printed the final result

```
J*=[ 258732.0000] = [%51.75]

 x           y
21.356  28.929
78.378  66.732
48.212  51.197
84.078  16.864
```

4.8.3 Algorithm description

This is a description of the algorithm which uses the kmeans++ cluster analysis function `kmeans()` as part of the Matlab `Statistics and Machine Learning Toolbox` toolbox, which is included in the student version. This is not a description of the kmeans++ algorithm itself, since that is well described and documented in many places such as in references [3,4]. This is a description of the algorithm using kmeans to solve the grocery location problem.

Algorithm 1: Cluster analysis using Kmeans++ for determining optimal store locations

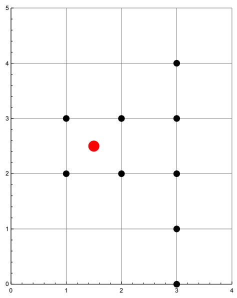
Input: n, X, Y where n is the number of stores to allocate, X is population coordinates, and Y is competitor store location coordinates
Output: list of coordinates to locate our n stores at, and J^* which is size of population our stores will attract when placed at these locations

```
currentNumberOfPartitions ⟵ $n$
bestScore ⟵ 0
keepSearching ⟵ true
bestLocations ⟵ {}
while keepSearching do
    $C$ ⟵ kmeans(currentNumberOfPartitions,X) /* $C$ now contains the centroid of partitions found by kmeans++
        cluster analysis algorithm using Matlab toolbox */
    partitionSets ⟵ combnk(1:size(C,1),n) /* Find all possible combinations of partitions. Warning, this is $\binom{k}{n}$
        which will quickly grow. In practice, it was found we do not need $k$ greater than $n+4$ to find a
        maximum. $n$ is limited to 10. */
    partitionScore ⟵ 0
    winningCombination ⟵ {}
    foreach $e \in$ partitionSets do
        score ⟵ 0
        foreach $x_i \in X$ do
            $d_1$ ⟵ shortest distance of $x_i$ to any of the centroid of the partition $e$
            $d_2$ ⟵ shortest distance of $x_i$ to any of competitor stores in $Y$
            if $d_1 \leq d_2$                /* win this customer or split it. Else competitor is closer */
             then
                if $d_1 = d_2$ then
                    score ⟵ score + $\frac{1}{2}$
                else
                    score ⟵ score + 1
                end
            end
        end
        if score ≥ partitionScore then
            partitionScore ⟵ score
            winningCombination ⟵ $e$
        end
    end
    if partitionScore ≥ bestScore then
        /* score did not go down, keep searching. Increase number of population partitions by one and call
            kmeans++ (above) for new partitions */
        bestScore ⟵ partitionScore
        bestLocations ⟵ winningCombination
        if currentNumberOfPartitions = 15/* stop search if $\binom{k>=15ink}{n}$ due to limitation */
         then
            keepSearching ⟵ false
        else
            currentNumberOfPartitions ⟵ currentNumberOfPartitions + 1
        end
    else
        /* when score goes down, it will not improve any more */
        keepSearching ⟵ false
    end
end
return bestScore,bestLocations
```

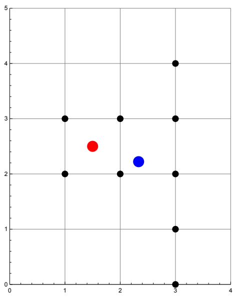
4.8.3.1 Description of test cases used in development

It was very important to check the correctness of the algorithm using small number of test cases to verify it is generating the optimal store locations as it is very hard to determine the optimal solution for any large size problem by hand. The following are some of the test problems used and the result obtained by the implementation, which shows that the optimal locations were found for each case.

test case 1

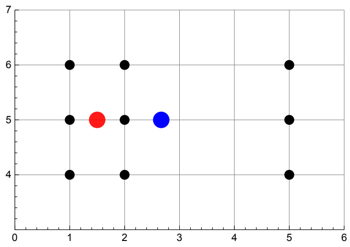


By direct observations, since we have one store only, then we see that by locating it in the center of the population, the score will be 6, which is optimal. The optimal store location found by the program is {2.333, 2.222}

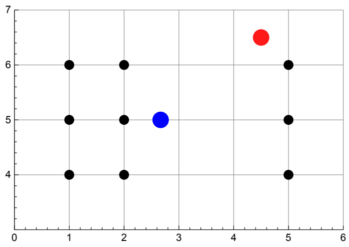


test case 2 This test case shows that the optimal location of our store do not change as the competition store location is changed. Since the optimal location depends on the clustering found and not on the competition location. In the following, the same configuration is used, but one had the competition store is at {1.5, 5} and the other at {4.5, 6.5}. We see by direct counting and observation that the optimal store location is at {2, 5} regardless. The only difference is the number of customers we attract in each case, but not the optimal store location itself. These two plots show this, with the score we obtain given below each configuration.

Clearly when the competitor store is away from the density area, our score will increase. Since the competition also wants to increase its score, then it should also have to locate its store in the same location, which is the kmeans++ optimal location.



optimal {2.667,5} with score 6



optimal {2.667,5} with score 8

Many other test cases where run, using more store locations and they were verified manually that the program result agrees with the finding. It is not possible to verify manually that the result will remain optimal for large population and large number of stores, but these tests cases at least shows that the algorithm works as expected. Now we will show result of large tests cases and the program output generated.

4.8.4 Result applying the algorithm to the supplied input

The following table summarizes the result of running the store location algorithm on the 5 test cases provided.

For illustration, the following four plots show the locations of our stores (the green dots)

Table 4.9: Summary of store location score of each test case

test case	n	m	X (population)	CPU time (minutes)	J^*	percentage
trial/earlier one	4	5	$500,000$	1.42	$258,732$	51.75%
1	9	9	$500,000$	5.49	$371,543$	74.32%
2	10	10	$1,000,000$	3.38	$637,413$	63.74%
3	5	5	$130,000$	1.16	$69,093$	53.15%
4	10	10	$1,000,000$	14.17	$683,899$	68.39%

for the above final four test cases with the location of the competitor stores (black dots) and the final partitions selected.

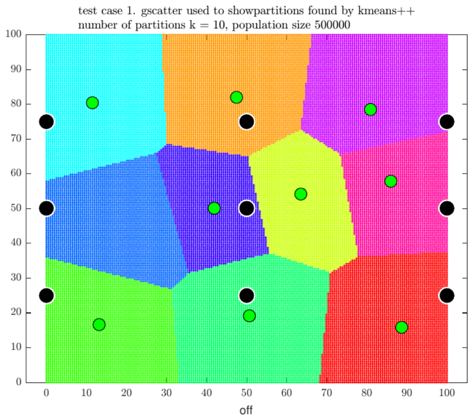


Figure 4.81: Test case 1

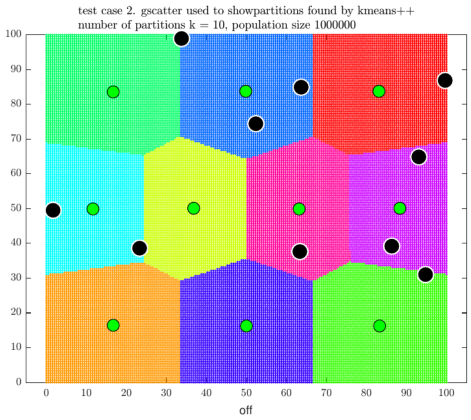


Figure 4.82: Test case 2

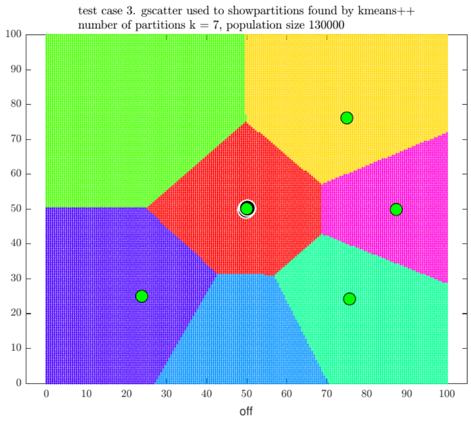


Figure 4.83: Test case 3

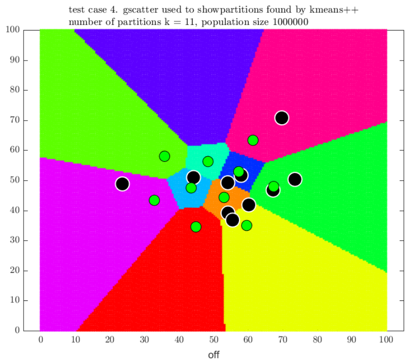


Figure 4.84: Test case 4

4.8.5 Conclusion

`kmeans++` algorithm for cluster analysis appears to be an effective method to use for finding an optimal store locations, but it is only practical for small n as the algorithm used to obtain the partitions is NP-hard. In addition $\binom{p}{n}$ combinations of partitions needs to be searched to select the optimal set.

This implementation shows how `kmeans++` can be used to solve these types of problems. The location of the competitor stores has no influence on where to locate the stores, but it only affects the final possible score. Generating more partitions (using kmeans++) than the number of stores and selecting from them the best set can lead to improved score. It was found in the test cases used that no more than two of three additional partitions than the number of stores was needed to find the a combination of partitions which gave the maximum score. Generating additional partitions made the score go lower. The score used is the number of customers the stores attract out of the overall population. The algorithm was verified to be correct for small number of tests cases (not shown here due to space limitation). More research is needed to investigate how feasible this method can be for solving similar resource allocations problems.

4.8.6 References

1 Matlab cluster analysis toolbox. Mathworks, Natick, MA

2 `https://en.wikipedia.org/wiki/K-means_clustering`

3 Seber, G.A.F. (1984) Multivariate Observations, Wiley, New York.

4.8.7 Appendix

```
function abbasi()
%Special problem. ECE 719, spring 2016
%by Nasser M. Abbasi
%Matlab 2016a
%

clear;      %start with clear env. just in case.
close all;  %is it ok to close all windows?
commandwindow;  %bring command window into focus
cd(fileparts(mfilename('fullpath')));

if license('test','Statistics_Toolbox') ~= 1
   error(['Warning, the needed toolbox does not', ...
       'seem to exist in your Matlab. This program needs',...
       'the Statistics and Machine Learning Toolbox as',...
       'it called kmeans++ cluster analysis\n',...
       'Please use the ver command to check you the toolbox\n']);
end

%this window will close when we are done. Ok to do.
fig = figure('Position',[370 400 400 30],...
  'Name','Optimal store locator. ECE 719. UW-Madison',...
  'NumberTitle','off');
set(fig, 'MenuBar', 'none');
set(fig, 'ToolBar', 'none');
h = uicontrol('Style','text','Position',[4 7 396 15],...
          'BackgroundColor','w',...
          'HorizontalAlignment','left');
drawnow;

DEBUG=true; %set to true to see plots
%change to false before code lockdown as plots slows down time.

OUT(h,'Starting store location program version 1.0.....');
OUT(h,'Checking for mat files.....');

if ~exist('n.mat','file')
    error(['file n.mat does not exist in current folder.',...
           'Please check for lower/upper case and location']);
end
if ~exist('X.mat','file')
    error(['file X.mat does not exist in current folder.',...
          'Please check for lower/upper case and location']);
end
if ~exist('Y.mat','file')
    error(['file n.mat does not exist in current folder.',...
           'Please check for lower/upper case and location']);
end


cd('../official_data/4/');
load('n');
load('X');
load('Y');
OUT(h,'mat files read ok.....');
```

```matlab
cd(fileparts(mfilename('fullpath')));

rng(1); %for reproducability
KEEP_TRYING               = true;  %tells when to stop search
best_score_found_so_far   = 0;
best_locations            = [];
current_number_of_cluster = n;
tstart                    = tic;  %to keep track of CPU time
test_case                 = 4;
MAX_CPU                   = 15; %minutes CPU time limit.
status                    = true;
while KEEP_TRYING
    OUT(h,sprintf(['Best score so far: [%d]. calling kmeans++',...
                  'to make %d partitions...please wait...'],...
      round(best_score_found_so_far),current_number_of_cluster));
    [idx,C]  = make_cluster(X,current_number_of_cluster,...
                                                'sqeuclidean' );
    active_C = combnk(1:size(C,1),n);
    OUT(h,sprintf('created active_C, size is [%d,%d]....',...
            size(active_C,1),size(active_C,2)));

    [status,score,locations]=score_cluster(C,X,Y,active_C,...
                                               tstart,h,MAX_CPU);

    if ~status
        OUT(h,'Allowed CPU time exceeded, stopping the program');
        KEEP_TRYING = false;
    else
        if score>=best_score_found_so_far
            best_score_found_so_far=score;
            best_locations=locations;
            current_number_of_cluster=current_number_of_cluster+1;
            %stop search if size too large, or if number of
            %partitions too large this is due to using k choose m.
            %For k>15 it will need too much RAM.
            if current_number_of_cluster>=size(X,1)...
                                 ||current_number_of_cluster>=15
                KEEP_TRYING = false;
            end
            OUT(h,sprintf('current score %6.2f',...
                                       best_score_found_so_far));
            if DEBUG
               plot_result(test_case,best_locations,X,Y,...
                                      best_score_found_so_far,C);
            end
        else
          OUT(h,sprintf(...
             'Score is %6.2f. Less than last. Terminating..',...
              score));
          KEEP_TRYING = false;
        end
        telapsed = toc(tstart);
        if telapsed>MAX_CPU*60  % CPU limit
            OUT(h,'CPU time exceeded');
            KEEP_TRYING = false;
            status      = false;
        else
            OUT(h,sprintf(...
              'CPU time used to far %6.2f minutes',telapsed/60));
        end
    end
end

```

```
%final result
fprintf('n=%d, X=%d, Y=%d\n',n,size(X,1),size(Y,1));

fprintf('J*=[%6.2f] = [%%%4.2f]\n\n',...
  best_score_found_so_far,best_score_found_so_far/size(X,1)*100);
fprintf('optimal store coordinates\n');
fprintf(' x\t\t y\n');

for i=1:size(best_locations,1)
 fprintf('%3.3f\t%3.3f\n',best_locations(i,1),best_locations(i,2));
end

telapsed = toc(tstart);

if ~status
  fprintf('\nCPU limit reached. Elapsed time is %6.2f minutes\n',...
                                                    telapsed/60);
else
   fprintf('\nElapsed time is %6.2f minutes\n',telapsed/60);
end
if ishandle(fig)
    close(fig);
end
end
%============================
function d = distance_between_2_points(pt1,pt2)
%find distance between 2 points, assuming one can only
%move N-S or E-W, not diagonal.
x1 = pt1(1,1);
y1 = pt1(1,2);
x2 = pt2(1,1);
y2 = pt2(1,2);

d = abs(x1-x2) + abs(y1-y2);
end
%============================================
function best_score_in_cluster = ...
                           find_my_score_in_each_cluster(C,X,Y)
%Takes center of each cluster (C) and customers locations (X)
%and competition store locations (Y) and returns how many
%customers I win in each cluster. Returns an array of number
%of customers we attract from competition in each cluster.

%to store score per cluster
best_score_in_cluster = zeros(size(C,1),1);

for i=1:size(X,1)  %loop of all population to see which we win
    %z1 is competitor, z2 is our store
    [~,z1]   = shortest_distance_to_stores(X(i,:),Y);
    [idx,z2] = shortest_distance_to_stores(X(i,:),C);
    if z2<=z1  %compare with competition to see if we are closer
     if z1==z2
       %oh well, split this customer between us and them
       best_score_in_cluster(idx)=best_score_in_cluster(idx)+0.5;
     else
       %good, we are closer, take this customer.
       best_score_in_cluster(idx)=best_score_in_cluster(idx)+1;
     end
    end
end
end

%================================
function [idx,d] =shortest_distance_to_stores(pt,stores_locations)
%find shortest distance from one customer to a set of stores.
%The stores can be ours or the competition. Returns the shortest
%distance in 'd' and the index of the store who is closest to
%this customer

d = inf;
for i=1:size(stores_locations,1)
    current_distance = distance_between_2_points(pt,...
                                         stores_locations(i,:));
    if current_distance <= d
        d   = current_distance;
        idx = i;
    end
end

end
%==========================================
function [status,best_score,locations]=score_cluster(...
                               C,X,Y,active_C,tstart,h,MAX_CPU)

best_score   = 0;
status       = true;
KEEP_TRYING  = true;

while KEEP_TRYING
  for i=1:size(active_C,1)
    OUT(h,sprintf(['scoring partition %d of %d in score_cluster() ',...
                       'Current best score %d'],...
                          i,size(active_C,1),round(best_score)));

        score = find_my_score_in_each_cluster(...
                                        C(active_C(i,:),:),X,Y);
        score=sum(score);
        if score>best_score
            best_score = score;
            locations  = C(active_C(i,:),:);
        end
        telapsed = toc(tstart);
        if telapsed>MAX_CPU*60
          OUT(h,sprintf('Exceeded CPU time limit in score_cluster'));
          KEEP_TRYING = false;
          status      = false;
        end
    end
    KEEP_TRYING = false;
end
end
%===================================
function [idx,C] = make_cluster(population,how_many,the_option)
%cluster the population. Number of cluster is same as
%number of our stores.  This was found to be optimal by many
%trials and errors. If we use more clusters than number of
%stores, the score actually goes down.
warning('off','all');
[idx,C] = kmeans(population,how_many,'Replicates',5,...
    'MaxIter',50,'Distance',the_option);
warning('on','all');
end

%=====================================================
function plot_result(test_case,store_locations,X,Y,...
                                           overall_best_score,C)
```

```matlab

%figure;
tmp = hist3(X, {0:100 0:100});
n1  = tmp';
n1(size(tmp,1), size(tmp,2)) = 0;
xb = linspace(0,100,101);
yb = xb;

figure;
pcolor(xb,yb,n1);
hold on;
plot(Y(:,1),Y(:,2),'o','MarkerSize',9,...
    'MarkerFaceColor','black',...
    'LineWidth',1,'MarkerEdgeColor','white');
plot(store_locations(:,1),store_locations(:,2),...
    'o','MarkerSize',9,'MarkerFaceColor','green',...
    'MarkerEdgeColor','black');
title( {sprintf(...
         ['Test case $%d$. Showing our store location with',...
          'competitors on density plot. score =%5.1f'],...
    test_case,overall_best_score),...
    sprintf('number of partitions k = $%d$, population size $%d$',...
    size(store_locations,1),size(X,1))},...
        'Fontsize',11,'interpreter','Latex');
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
drawnow;
saveas(gcf, sprintf('../images/1_test_case_%d',test_case), 'pdf');
saveas(gcf, sprintf('../images/1_test_case_%d',test_case), 'png');

figure;
[x1G,x2G] = meshgrid(linspace(0,100,200),linspace(0,100,200));
XGrid = [x1G(:),x2G(:)]; % Defines a fine grid on the plot
warning('off','all');
idx2Region = kmeans(XGrid,size(C,1),'MaxIter',1,'Start',C);
warning('on','all');
cmap = hsv(size(C,1));
gscatter(XGrid(:,1),XGrid(:,2),idx2Region,cmap,[],[],...
                                                 'doLeg','off');
hold on;
plot(Y(:,1),Y(:,2),'o','MarkerSize',12,...
               'MarkerFaceColor','black',...
               'LineWidth',1,'MarkerEdgeColor','white');
plot(store_locations(:,1),store_locations(:,2),...
    'o','MarkerSize',9,'MarkerFaceColor','green',...
    'MarkerEdgeColor','black');
title( {sprintf(['test case $%d$. gscatter used to show',...
                 'partitions found by kmeans++'],test_case),...
   sprintf('number of partitions k = $%d$, population size $%d$',...
   size(C,1),size(X,1))},'Fontsize',11,'interpreter','Latex');
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
drawnow;
saveas(gcf, sprintf('../images/2_test_case_%d',test_case), 'pdf');
saveas(gcf, sprintf('../images/2_test_case_%d',test_case), 'png');
end
%=================================================
function OUT(h,the_string)
fprintf(the_string);
fprintf('\n');

%if ishandle(h)
%   set(h,'String',the_string);
%   drawnow;
%end
```

```
end
```

```
function nma_generate_output(test_case)
%Program to generate output to test special problem with
%Nasser M. Abbasi
%ECE 719, UW Madison

cd(fileparts(mfilename('fullpath')));

switch test_case
    case 1
        X=[1,2;
            1,3;
            2,2;
            2,3;
            3,0;
            3,1;
            3,2;
            3,3;
            3,4];
        Y=[1.5,2.5];
        n=1;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

  case 105
        X=[1,2;
            1,3;
            2,2;
            2,3;
            3,0;
            3,1;
            3,2;
            3,3;
            3,4];
        Y=[1.5,2.5];
        n=2;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

%        fprintf('best score is %3.3f\n',...
%            find_my_score_in_each_cluster([2.5,2.5],X,Y))
    case 2
        X=[1,4;
            1,5;
            1,6;
            2,4;
            2,5;
            2,6;
            5,6;
            5,5;
            5,4];
        Y=[4,6.5];
        n=1;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

%        fprintf('best score is %3.3f\n',...
%            find_my_score_in_each_cluster([2,5],X,Y))

    case 3
        X=[1,4;
            1,5;
            1,6;
            2,4;
            2,5;
            2,6;
            5,6;
            5,5;
            5,4];
        Y=[1.5,5];
        n=1;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

%        fprintf('best score is %3.3f\n',...
%            find_my_score_in_each_cluster([2,5],X,Y))

    case 4
        X=[1,1;
            1,2;
            2,1;
            2,2;
            4,3;
            4,4;
            5,3;
            5,4];
        Y=[3,2.5];
        n=1;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

%        fprintf('best score is %3.3f\n',...
%            find_my_score_in_each_cluster([2,5],X,Y))

    case 5
        X=[1,1;
            1,2;
            1,3;
            2,1;
            2,2;
            2,3;
            3,2;
            4,2;
            4,3;
            4,4;
            4,5;
            5,1;
            5,2;
            5,3;
            5,4;
            5,5;
            6,3;
            6,4;
            6,5];
        Y=[5,4];
        n=1;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

%        fprintf('best score is %3.3f\n',...
%            find_my_score_in_each_cluster([2,5],X,Y))

    case 6
        X=[ 1,1;
            2,1;
            3,1;
            1,2;
            2,2;
            3,2;
            1,3;
            2,3;
            3,3];
        Y=[2,2];
        n=2;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

%        fprintf('best score is %3.3f\n',...
%            find_my_score_in_each_cluster([2,5],X,Y))

    case 7
        rng default; % For reproducibility
        N=10000;
        X=[ 30 + 2*randn(N,1),30 + 8*randn(N,1);
            40 + 2*randn(N,1),40 + 10*randn(N,1);
            25 + 2*randn(N,1), 50 + 4*randn(N,1);
            20 + 2*randn(N,1),30 + 4*randn(N,1);
            50 + 2*randn(N,1),50 + 4*randn(N,1)];

        n=9;  %this gives 50%, since competition is allready optimal
        Y=[41.6552   35.4282;
            24.5046    33.8534;
            30.5928   30.0431];
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');


    case 8
        rng default; % For reproducibility
        N=1000;
        X=[30+randn(N,1),30+randn(N,1);
            40+randn(N,1),40+randn(N,1);
            25+randn(N,1),50+randn(N,1);
            20+randn(N,1),30+randn(N,1);
            50+randn(N,1),50+randn(N,1)];

        n=4;  %this gives 50%, since competition is allready optimal
        Y=[41.6552   35.4282;
            24.5046    33.8534;
            30.5928   30.0431];
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

    case 9
        rng default; % For reproducibility
        N=1000;
        X=[30+randn(N,1),30+randn(N,1);
            40+randn(N,1),40+randn(N,1);
            25+randn(N,1),50+randn(N,1);
            20+randn(N,1),30+randn(N,1);
            50+randn(N,1),50+randn(N,1)];

        n=7;  %this gives 50%, since competition is allready optimal
        Y=[41.6552   35.4282;
            24.5046    33.8534;
            30.5928   30.0431];
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

case 10
        rng default; % For reproducibility
        N=100000;
        X=[30+randn(N,1),30+randn(N,1);
            40+randn(N,1),40+randn(N,1);
            25+randn(N,1),50+randn(N,1);
            20+randn(N,1),30+randn(N,1);
            50+randn(N,1),50+randn(N,1)];

        n=10;  %this gives 50%, since competition is allready optimal
        Y=[41.6552   35.4282;
            24.5046    33.8534;
            30.5928   30.0431;
            40.5928   30.0431;
            70.5928   30.0431];
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');


end
end
```

Chapter 5

study notes

Local contents

5.1 Some HOWTO questions

This in place to keep some study notes, and other items to remember while taking this hard course.

5.1.1 How to show that sum of two convex functions is also convex function?

Let $G(u) = g(u) + f(u)$ where we know g, f are two convex functions. We need to show $G\left(u^{\lambda}\right)$ is also convex. Then, pick point $u^{\lambda} \in U$ therefore

But the set U is convex (it must be, these are convex functions, so their domain is convex by definition). Pick a point $u^{\lambda} = (1-\lambda)u^1 + \lambda u^2$ where $\lambda \in [0,1]$ and $u^1, u^2 \in U$. Hence we can write

$$\begin{aligned} G\left(u^{\lambda}\right) &= g\left(u^{\lambda}\right) + f\left(u^{\lambda}\right) \\ G\left((1-\lambda)u^1 + \lambda u^2\right) &= g\left((1-\lambda)u^1 + \lambda u^2\right) + f\left((1-\lambda)u^1 + \lambda u^2\right) \end{aligned}$$

But $g\left((1-\lambda)u^1 + \lambda u^2\right) \leq (1-\lambda)g\left(u^1\right) + \lambda g\left(u^2\right)$ and the same for f. Then the above reduces to

$$\begin{aligned} G\left((1-\lambda)u^1 + \lambda u^2\right) &\leq (1-\lambda)g\left(u^1\right) + \lambda g\left(u^2\right) + (1-\lambda)f\left(u^1\right) + fg\left(u^2\right) \\ &= (1-\lambda)\left(g\left(u^1\right) + f\left(u^1\right)\right) + \lambda\left(g\left(u^2\right) + f\left(u^2\right)\right) \end{aligned}$$

But $G(u) = g(u) + f(u)$, then the RHS above becomes

$$G\left((1-\lambda)u^1 + \lambda u^2\right) \leq (1-\lambda)G\left(u^1\right) + \lambda G\left(u^2\right)$$

Therefore G is a convex function.

5.1.2 What is convex Hull?

Smallest set that contains all the sets inside it, such that it is also convex. (put a closed convex "container" around everything)

5.1.3 Is convex full same as Polytope?

No. Polytope is region which has straight edges (flat sides) and also be convex. But `http://mathworld.wolfram.com/Polytope.html` says "The word polytope is used to mean a number of related, but slightly different mathematical objects. A convex polytope may be defined as the convex hull of a finite set of points" And `https://en.wikipedia.org/wiki/Polytope` says "In elementary geometry, a polytope is a geometric object with flat sides, and may exist in any general number of dimensions n as an n-dimensional polytope or n-polytope"

5.1.4 What is difference between polytope and polyhedron?

`https://en.wikipedia.org/wiki/Polytope` says "In elementary geometry, a polyhedron (plural polyhedra or polyhedrons) is a solid in three dimensions with flat polygonal faces, straight edges and sharp corners or vertices."

Polyhedron can be convex or not. But polyhedron can be open? While polytope not. Need to check.

5.2 Some things to remember

1. Principle of optimality, by Bellman: "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision" Proof is by contradiction. See page 54, optimal control theory by Donald kirk. Simplest proof I've seen. An important case is when the performance index J is quadratic as with LQR. We only looked at case where is no coupling term in the LQR in this course. $J = \min_u \sum_{k=0}^{\infty} x^T Qx + u^T Ru$. This is solved for steady state by solving Riccati equation. For discrete case, use Matlab dare() function. See Introduction to Dynamic Programming: International Series in Modern Applied Mathematics and Computer Science, Volume 1 (Pergamon International Library ... Technology, Engineering & Social Studies)

2. Remember difference between state variables, and decision variable. There can be more than one state variable in the problem, but the number of decisions to make at each state is different. see problem 1, HW 7 for example. The fire stations allocation problem. In that problem, we had one state variable, which is the number of stations available. There more state variables there are, the harder it will be to solve by hand.

3. $$\lim_{\lambda \to 0} \frac{J(\boldsymbol{u} + \lambda \boldsymbol{d}) - J(\boldsymbol{u})}{\lambda} = [\nabla J(\boldsymbol{u})]^T \cdot \boldsymbol{d}$$

   Remember, $\nabla J(\boldsymbol{u})$ is column vector. $\nabla J(\boldsymbol{u}) = \begin{bmatrix} \frac{\partial J(u)}{\partial u_1} \\ \vdots \\ \frac{\partial J(u)}{\partial u_n} \end{bmatrix}$. This vector is the direction along which function $J(\boldsymbol{u})$ will increase the most, among all other directions, at the point it is being evaluated at.

4. For polytope, this is useful trick.

   $$\begin{aligned} \boldsymbol{u} &= \sum_{i=1}^{m} \lambda_i v^i \\ \|\boldsymbol{u}\| &= \left\| \sum_{i=1}^{m} \lambda_i v^i \right\| \\ &\leq \sum_{i=1}^{m} \|\lambda_i v^i\| \end{aligned}$$

   The last step was done using triangle inequality.

5. Definition of continuity: If $u^k \to u^*$ then $J\left(u^k\right) \to J(u^*)$. We write $\lim_{k\to\infty} J\left(u^k\right) = J(u^*)$. This is for all u^k sequences. See real analysis handout. If $u^k \to u^*$ then this is the same as $\lim_{k\to\infty} \left\|u^k - u^*\right\| = 0$

6. closed sets is one which include all its limits points. (includes it boundaries). Use [0,1] for closed. Use (0,1) for open set. A set can be both open and closed at same time (isn't math fun?, wish life was this flexible).

7. Intersection of closed sets is also closed set. If sets are convex, then the intersection is convex. But the union of convex sets is not convex. (example, union of 2 circles).

8. B-W, tells us that a sequence u^k that do not converge, as long as it is in a compact set, it will contain at least one subsequence in it, $u^{k,i}$ which does converge to u^*. So in a compact set, we can always find at least one subsequence that converges to u^* even inside non-converging sequences.

9. If a set is not compact, then not all is lost. Assume set is closed but unbounded. Hence not compact. What we do, it set some R large enough, and consider the set of all elements $\|u\| \leq R$. Then the new set is compact.

10. $J(u) = au^2 + bu + c$ is coercive for $a > 0$. Note, the function $J(u)$ to be coercive, has

to blow up in all directions. For example, e^u is not coercive. If A is positive definite matrix and $\boldsymbol{b} \in \Re^n$ and $c \in \Re$, then $J(u) = \boldsymbol{u}^T A\boldsymbol{u} + \boldsymbol{b}^T\boldsymbol{u} + c$ is coercive function. To establish this, convert to scalar. Use $\lambda_{\min} \|\boldsymbol{u}\|^2 \leq \boldsymbol{u}^T A\boldsymbol{u}$ and use $\boldsymbol{b}^T\boldsymbol{u} \leq \|\boldsymbol{b}\| \|\boldsymbol{u}\|$, then $J(u) \leq \lambda_{\min} \|\boldsymbol{u}\|^2 + \|\boldsymbol{b}\| \|\boldsymbol{u}\| + c$. Since P.D. matrix, then $\lambda_{\min} > 0$. Hence this is the same as $J(u) = au^2 + bu + c$ for $a > 0$. So coercive.

11. If in $J(u) = \boldsymbol{u}^T A\boldsymbol{u}+\boldsymbol{b}^T\boldsymbol{u}+c$ the matrix A is not symmetric., write as $J(u) = \frac{1}{2}\boldsymbol{u}^T \left(A^T + A\right)\boldsymbol{u}+ \boldsymbol{b}^T\boldsymbol{u} + c$. Now it expressions becomes symmetric.

12. $\sum_{i,j} x_i x_j = \left(\sum_i x_i\right)^2$

13. If $\bar{\alpha} = \frac{1}{n}\sum_i \alpha_i$ then $\sum_i (\alpha_i - \bar{\alpha})^2 \geq 0$. Used to show Hessian is P.D. for given $J(u)$. See HW 2, last problem.

14. $x^T Ax = \sum_{ij} A_{ij} x_i x_j$

15. To find a basic solution x_B which is not feasible, just find basic solution with at least one entry negative. Since this violates the constraints (we also use $x \geq 0$) for feasibility, then x_B solves $Ax = b$ but not feasible. i.e. $\begin{bmatrix} I & B \end{bmatrix}\begin{bmatrix} x_B \\ 0 \end{bmatrix} = b$ with some elements in x_B negative. For basic solution to also be feasible, all its entries have to be positive. (verify).

16. Solution to $Ax = b$ are all of the form $x_p + x_h$ where x_h is solution to $Ax = 0$ and x_p is a particular solution to $Ax = b$. This is similar to when we talk about solution to ODE. We look for homogeneous solution to the ODE (when the RHS is zero) and add to it a particular solution to original ODE with the rhs not zero, and add them to obtain the general solution.

17. difference between Newton method and conjugate gradient, is that CG works well from a distance, since it does not need the Hessian. CG will converge in N steps if $J(u)$ was quadratic function. Newton will converge in one step for quadratic, but it works well only if close to the optimal since it uses the Hessian as per above.

18. CG has superlinear convergence. This does not apply to steepest descent.

19. difference between steepest descent and conjugate direction is this: In SD, we use $\nabla J(u^k)$ as the direction to move at each step. i.e we use
$$u^{k+1} = u^k - h\frac{\nabla J(u^k)}{\|\nabla J(u^k)\|}$$
Where h above is either fixed step or optimal. But In CD we use v^k which is the mutual conjugate vector to all previous v^i. See my table of summary for this below as they can get confusing to know the difference.

20. To use Dynamic programming, the problem should have optimal substructure, and also should have an overlapping sub-problems. Sometimes hard to see or check for this.

21. Steepest descent with optimal step size has quadratic convergence property.

22. For symmetric Q, then $\frac{\partial\left(x^T Q x\right)}{\partial x} = 2Qx$

5.3 Example using conjugate directions

This example is solved in number of ways. Given quadratic function $J(x_1, x_2) = \frac{1}{2}x^T Ax + bx^T$ where $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$. To find x^*, which minimizes $J(x)$. Let $A = \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}$ and $b = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

5.3.1 First method, Direct calculus

$$
\begin{aligned}
\nabla J(x) &= 0 \\
Ax + b &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} 4x_1 + 2x_2 - 1 \\ 2x_1 + 2x_2 + 1 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{aligned}
$$

Solving gives

$$
x^* = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -\frac{3}{2} \end{bmatrix}
$$

5.3.2 Second method, basic Conjugate direction

Since A is of size $n = 2$, then this will converge in 2 steps using conjugate directions. let $x^0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. Let first direction be

$$
v^0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}
$$

Then

$$
h_0 = \frac{-\left(v^0\right)^T \nabla J\left(x^0\right)}{\left(v^0\right)^T A v^0} = \frac{-\left(v^0\right)^T \left(Ax^0 + b\right)}{\left(v^0\right)^T A v^0} = \frac{-\begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix}}{\begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix}} = \frac{1}{4}
$$

Hence

$$
\begin{aligned}
x^1 &= x^0 + h_0 v^0 \\
&= \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \frac{1}{4}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} \\ 0 \end{bmatrix}
\end{aligned}
$$

Second step. We need to find v^1. Using conjugate mutual property of A, we solve for v^1 using

$$
\begin{aligned}
\left(v^0\right)^T A v^1 &= 0 \\
\begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} &= 0 \\
4v_1 + 2v_2 &= 0
\end{aligned}
$$

Let $v_1 = 1$ then $v_2 = -2$ and hence

$$
v^1 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}
$$

Now we find the next optimal step

$$h_1 = \frac{-\left(v^1\right)^T \nabla J\left(x^1\right)}{\left(v^1\right)^T A v^1} = \frac{-\left(v^1\right)^T \left(Ax^1 + b\right)}{\left(v^1\right)^T A v^1} = \frac{-\begin{bmatrix}1 & -2\end{bmatrix}\left(\begin{bmatrix}4 & 2\\ 2 & 2\end{bmatrix}\begin{bmatrix}\frac{1}{4}\\ 0\end{bmatrix} + \begin{bmatrix}-1\\ 1\end{bmatrix}\right)}{\begin{bmatrix}1 & -2\end{bmatrix}\begin{bmatrix}4 & 2\\ 2 & 2\end{bmatrix}\begin{bmatrix}1\\ -2\end{bmatrix}} = \frac{3}{4}$$

Hence

$$\begin{aligned} x^2 &= x^1 + h_1 v^1 \\ &= \begin{bmatrix}\frac{1}{4}\\ 0\end{bmatrix} + \frac{3}{4}\begin{bmatrix}1\\ -2\end{bmatrix} \\ &= \begin{bmatrix}1\\ -\frac{3}{2}\end{bmatrix} \end{aligned}$$

Which is x^* that we found in first method. Using $n = 2$ steps as expected. In implementation, we will have to check we converged by looking at $\nabla J\left(x^2\right)$ which will be

$$\begin{aligned} \nabla J\left(x^2\right) &= Ax^2 + b \\ &= \begin{bmatrix}4 & 2\\ 2 & 2\end{bmatrix}\begin{bmatrix}1\\ -\frac{3}{2}\end{bmatrix} + \begin{bmatrix}-1\\ 1\end{bmatrix} \\ &= \begin{bmatrix}0\\ 0\end{bmatrix} \end{aligned}$$

As expected.

5.3.3 Third method. Conjugate gradient

The difference here is that we find v^i on the fly after each step. Unlike the conjugate direction method, where v^i are all pre-computed. Let $v^0 = \nabla\left(J\left(x^0\right)\right) = \begin{bmatrix}-1\\ 1\end{bmatrix}$. In this method, we always pick $v^0 = \nabla\left(J\left(x^0\right)\right)$, where x^0 is the starting guess vector. First step

$$h_0 = \frac{-\left(v^0\right)^T \nabla J\left(x^0\right)}{\left(v^0\right)^T A v^0} = \frac{-\left(v^0\right)^T \left(Ax^0 + b\right)}{\left(v^0\right)^T A v^0} = \frac{-\begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}-1\\ 1\end{bmatrix}}{\begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}4 & 2\\ 2 & 2\end{bmatrix}\begin{bmatrix}-1\\ 1\end{bmatrix}} = \frac{-2}{2} = -1$$

Hence

$$\begin{aligned} x^1 &= x^0 + h_0 v^0 \\ &= \begin{bmatrix}0\\ 0\end{bmatrix} - 1\begin{bmatrix}-1\\ 1\end{bmatrix} = \begin{bmatrix}1\\ -1\end{bmatrix} \end{aligned}$$

Now we find the mutual conjugate v^1 as follows

$$\begin{aligned} \beta_0 &= \frac{\left(\nabla J\left(x^1\right)\right)^T A v^0}{\left(v^0\right)^T A v^0} = \frac{\left(Ax^1 + b\right)\left(Av^0\right)}{\left(v^0\right)^T A v^0} = \frac{\left(\begin{bmatrix}4 & 2\\ 2 & 2\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix} + \begin{bmatrix}-1\\ 1\end{bmatrix}\right)^T\left(\begin{bmatrix}4 & 2\\ 2 & 2\end{bmatrix}\begin{bmatrix}-1\\ 1\end{bmatrix}\right)}{\begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}4 & 2\\ 2 & 2\end{bmatrix}\begin{bmatrix}-1\\ 1\end{bmatrix}} \\ &= \frac{\begin{bmatrix}1\\ 1\end{bmatrix}^T\begin{bmatrix}-2\\ 0\end{bmatrix}}{2} = \frac{-2}{2} = -1 \end{aligned}$$

Hence

$$\begin{aligned} v^1 &= -\nabla J\left(x^1\right) + \beta_0 v^0 \\ &= -\left(\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) - (1)\begin{bmatrix} -1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ -2 \end{bmatrix} \end{aligned}$$

Now that we found v^1, we repeat the process.

$$h_1 = \frac{-\left(v^1\right)^T \nabla J\left(x^1\right)}{\left(v^1\right)^T A v^1} = \frac{-\left(v^1\right)^T \left(Ax^1 + b\right)}{\left(v^1\right)^T A v^1} = \frac{-\begin{bmatrix} 0 & -2 \end{bmatrix}\left(\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right)}{\begin{bmatrix} 0 & -2 \end{bmatrix}\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 0 \\ -2 \end{bmatrix}} = \frac{2}{8} = \frac{1}{4}$$

Hence

$$\begin{aligned} x^2 &= x^1 + h_1 v^1 \\ &= \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \left(\frac{1}{4}\right)\begin{bmatrix} 0 \\ -2 \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ -\frac{3}{2} \end{bmatrix} \end{aligned}$$

Which is the same as with conjugate direction method. Converged in 2 steps also.

5.3.4 Fourth method. Conjugate gradient using Fletcher-Reeves

In this method

$$\beta_k = \frac{\nabla J\left(u^{k+1}\right)^T \nabla J\left(u^{k+1}\right)}{\nabla J\left(u^k\right)^T \nabla J\left(u^k\right)} = \frac{\left\|\nabla J\left(u^{k+1}\right)\right\|^2}{\left\|\nabla J\left(u^k\right)\right\|^2}$$

We also start here with $v^0 = \nabla J\left(u^0\right) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ in this example.

$$h_0 = \frac{-\left(v^0\right)^T \nabla J\left(x^0\right)}{\left(v^0\right)^T A v^0} = \frac{-\left(v^0\right)^T \left(Ax^0 + b\right)}{\left(v^0\right)^T A v^0} = \frac{-\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix}}{\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix}} = \frac{-2}{2} = -1$$

Hence

$$\begin{aligned} x^1 &= x^0 + h_0 v^0 \\ &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} - 1\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \end{aligned}$$

Now find the mutual conjugate v^1 as follows, using Fletcher-Reeves formula

$$\beta_0 = \frac{\left\|\nabla J\left(u^1\right)\right\|^2}{\left\|\nabla J\left(u^0\right)\right\|^2} = \frac{\left\|Ax^1 + b\right\|^2}{\left\|Ax^0 + b\right\|^2} = \frac{\left\|\left(\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right)\right\|^2}{\left\|\left(\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right)\right\|^2} = \frac{\left\|\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\|^2}{\left\|\begin{bmatrix} -1 \\ 1 \end{bmatrix}\right\|^2} = \frac{\left(\sqrt{2}\right)^2}{\left(\sqrt{2}\right)^2} = 1$$

$$\begin{aligned} v^1 &= -\nabla J\left(x^1\right) + \beta_0 v^0 \\ &= -\left(\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) + (1)\begin{bmatrix} -1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} -2 \\ 0 \end{bmatrix} \end{aligned}$$

Now that we found v^1, we repeat the process.

$$h_1 = \frac{-\left(v^1\right)^T \nabla J\left(x^1\right)}{\left(v^1\right)^T A v^1} = \frac{-\left(v^1\right)^T \left(Ax^1 + b\right)}{\left(v^1\right)^T A v^1} = \frac{-\begin{bmatrix} -2 & 0 \end{bmatrix}\left(\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right)}{\begin{bmatrix} -2 & 0 \end{bmatrix}\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 0 \\ -2 \end{bmatrix}} = \frac{2}{8} = \frac{1}{4}$$

Hence

$$\begin{aligned} x^2 &= x^1 + h_1 v^1 \\ &= \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \left(\frac{1}{4}\right)\begin{bmatrix} 0 \\ -2 \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ -\frac{3}{2} \end{bmatrix} \end{aligned}$$

Which is the same as with conjugate direction method. It converges in 2 steps also.

5.3.5 Fifth method. Conjugate gradient using Polak-Ribiere

In this method

$$\beta_k = \frac{\nabla J\left(u^{k+1}\right)^T \left(\nabla J\left(u^{k+1}\right) - \nabla J\left(u^k\right)\right)}{\nabla J\left(u^k\right)^T \nabla J\left(u^k\right)}$$

We also start here with $v^0 = \nabla J\left(u^0\right) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ in this example.

$$h_0 = \frac{-\left(v^0\right)^T \nabla J\left(x^0\right)}{\left(v^0\right)^T A v^0} = \frac{-\left(v^0\right)^T \left(Ax^0 + b\right)}{\left(v^0\right)^T A v^0} = \frac{-\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix}}{\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix}} = \frac{-2}{2} = -1$$

Hence

$$\begin{aligned} x^1 &= x^0 + h_0 v^0 \\ &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} - 1\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \end{aligned}$$

Now we find the mutual conjugate v^1 direction as follows, using Polak-Ribiere formula

$$\beta_0 = \frac{\nabla J\left(u^1\right)^T \left(\nabla J\left(u^1\right) - \nabla J\left(u^0\right)\right)}{\nabla J\left(u^0\right)^T \nabla J\left(u^0\right)}$$

But

$$\begin{aligned} \nabla J\left(u^1\right) &= \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \nabla J\left(u^0\right) &= \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \end{aligned}$$

Hence

$$\beta_0 = \frac{\begin{bmatrix}1 & 1\end{bmatrix}\left(\begin{bmatrix}1\\1\end{bmatrix} - \begin{bmatrix}-1\\1\end{bmatrix}\right)}{\begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}-1\\1\end{bmatrix}} = \frac{2}{2} = 1$$

Hence

$$\begin{aligned}
v^1 &= -\nabla J\left(x^1\right) + \beta_0 v^0\\
&= -\left(\begin{bmatrix}4 & 2\\2 & 2\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix} + \begin{bmatrix}-1\\1\end{bmatrix}\right) + (1)\begin{bmatrix}-1\\1\end{bmatrix}\\
&= \begin{bmatrix}-2\\0\end{bmatrix}
\end{aligned}$$

Now that we found v^1, we repeat the process.

$$h_1 = \frac{-\left(v^1\right)^T \nabla J\left(x^1\right)}{\left(v^1\right)^T A v^1} = \frac{-\left(v^1\right)^T\left(Ax^1 + b\right)}{\left(v^1\right)^T A v^1} = \frac{-\begin{bmatrix}-2 & 0\end{bmatrix}\left(\begin{bmatrix}4 & 2\\2 & 2\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix} + \begin{bmatrix}-1\\1\end{bmatrix}\right)}{\begin{bmatrix}-2 & 0\end{bmatrix}\begin{bmatrix}4 & 2\\2 & 2\end{bmatrix}\begin{bmatrix}0\\-2\end{bmatrix}} = \frac{2}{8} = \frac{1}{4}$$

Hence

$$\begin{aligned}
x^2 &= x^1 + h_1 v^1\\
&= \begin{bmatrix}1\\-1\end{bmatrix} + \left(\frac{1}{4}\right)\begin{bmatrix}0\\-2\end{bmatrix}\\
&= \begin{bmatrix}1\\-\frac{3}{2}\end{bmatrix}
\end{aligned}$$

Which is the same as with conjugate direction method. Converges in 2 steps also as expected

5.4 collection of definitions

Basic solution for LP This is solution $\vec{x}$ which has non zero entries that correspond to linearly independent column in A. Where the constraints are $Ax = b$.

feasible solution for LP This is solution $\vec{x}$ which is in the feasible region. The region that satisfy the constraints. Feasible solution do not have to be basic.

basic and feasible solution for LP This is solution $\vec{x}$ which is both feasible and basic. Once we get to one of these, then simplex algorithm will jump from one basic feasible to the next, while reducing the $J(u)$ objective function until optimal is found.

Basic but not feasible solution is there one? Need example.

Newton Raphson method Iteration is

$$u^{k+1} = u^k - \frac{\nabla J(u^k)}{\nabla^2 J(u^k)}$$

where $\nabla^2 J(u^k)$ is the hessian. This is a A matrix in the quadratic expression

$$J(u) = \frac{1}{2}u^T A u + b^T u + c$$

Of course we can't divide by matrix, this is the inverse of the Hessian. So the above is

$$u^{k+1} = u^k - \left[\nabla^2 J(u^k)\right]^{-1} \nabla J(u^k)$$

See handout Newton for example with $J(u)$ given and how to use this method to iterate to u^*. If $J(u)$ was quadratic, this will converge in one step.

Quadratic expression An expression is quadratic if it can be written as

$$\sum_i \sum_j a_{ij} u_i u_j + \sum_i b_i u_i + c$$

For example, $x_1^2 + 9x_1x_2 + 14x_2^2$ becomes

$$\begin{aligned} x_1^2 + 9x_1x_2 + 14x_2^2 &= a_{11}x_1x_1 + a_{21}x_2x_1 + a_{12}x_1x_2 + a_{22}x_2x_2 + b_1x_1 + b_2x_2 + c \\ &= a_{11}x_1^2 + a_{21}x_2x_1 + a_{12}x_1x_2 + a_{22}x_2^2 + b_1x_1 + b_2x_2 + c \end{aligned}$$

comparing both sides, we see that by setting $a_{11} = 1, a_{21} = \frac{9}{2}, a_{21} = \frac{9}{2}, a_{22} = 14$ and by setting $b_1 = 0, b_2 = 0$ and $c = 0$ we can write it in that form. Hence it is quadratic and

$$A = \begin{pmatrix} 1 & \frac{9}{2} \\ \frac{9}{2} & 14 \end{pmatrix}, b = \begin{pmatrix} 0 & 0 \end{pmatrix}$$

Therefore

$$\begin{aligned} x_1^2 + 9x_1x_2 + 14x_2^2 &= x^T A x + b^T x + c \\ &= \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 1 & \frac{9}{2} \\ \frac{9}{2} & 14 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + 0 \end{aligned}$$

Since we are able to write $x_1^2 + 9x_1x_2 + 14x_2^2 = x^T A x + b^T x + c$ it is quadratic. Notice that the A matrix is always symmetric.

superlinear convergence A sequence $\{u^k\}$ in $\Re^n$ is said to converge superlinearly to u^* if the following holds. Given any $\theta \in (0,1]$ then

$$\lim_{k\to\infty} \frac{\|u^k - u^*\|}{\theta^k} \to 0$$

Example is $u^k = e^{-k^2}$ Since $u^* = 0$ then $\frac{e^{-k^2}}{\theta^k} \to 0$ no matter what $\theta \in (0,1]$ is. Remember, it has to go to zero for any θ

Quadratic convergence theorem Given quadratic

$$J(u) = \frac{1}{2}u^T A u + b^T u + c$$

And given N set of mutually conjugate vectors (with respect to A) $\{v^0, v^2, \dots, v^{N-1}\}$ then the conjugate direction algorithm converges to the optimal $u^* = -A^{-1}b$ in N steps of less. Proof in lecture 3/1/2016 (long)

A-conjugate vectors There are mutually conjugate vectors with respect to A. The directions $\{v^0, v^1, \dots, v^{N-1}\}$ are said to be mutually conjugate with respect to A if

$$(v^i)^T A v^j = 0$$

For all $i \neq j$

5.5 Summary of iterative search algorithms

5.5.1 steepest descent

5.5.1.1 steepest descent, any objective function $J(x)$

The input is $x(0)$ the initial starting point and $J(x)$ itself.

```
1. init      $x^0 = x(0),\ k = 0$
2. $g^k$     $= \nabla J(x^k)$
3. $\alpha_k$   $= \min_\alpha J\left(x^k - \alpha \frac{g^k}{\|g^k\|}\right)$ (line search)
4. $x^{k+1}$   $= x^k - \alpha_k \frac{g^k}{\|g^k\|}$
5. $k$       $= k + 1$
6. goto 2
```

5.5.1.2 steepest descent, Quadratic objective function $J(x)$

If the objective function $J(x)$ is quadratic $J(x) = x^T A x - b^T x + c$ then there is no need to do the line search.

The input is $x(0)$ the initial starting point and A, b. The algorithm becomes

```
1. Init      $x^0 = x(0),\ k = 0$
2. $g^k$     $= \nabla J(x^k) = Ax^k - b$
3. $\alpha_k$   $= \frac{[g^k]^T g^k}{[g^k]^T A g^k}$
4. $x^{k+1}$   $= x^k - \alpha_k g^k$
5. $k$       $= k + 1$
6. goto 2
```

5.5.2 conjugate direction, Quadratic function $J(x)$

For quadratic $J(x) = x^T A x - b^T x + c$ the conjugate direction algorithm is as follows.

Input $x(0)$ starting point, and A, b and set of n mutually conjugate vectors $\{v^0, v^1, \dots, v^{n-1}\}$ with respect to A, where n is the size of A. In other words, $(v^i)^T A v^j = 0$ for $i \neq j$.

These v^i vectors have to be generated before starting the algorithm. With the conjugate gradient (below), these A-conjugate vectors are generated on the fly inside the algorithm as it iterates. This is the main difference between conjugate direction and conjugate gradient.

```
1. init      $u^0 = x(0),\ k = 0$
2. $g^k$     $= \nabla J(x^k) = Ax^k - b$
3. $\alpha_k$   $= \frac{[g^k]^T v^k}{[g^k]^T A v^k}$
4. $x^{k+1}$   $= x^k - \alpha_k v^k$
5. $k$       $= k + 1$
6. goto 2
```

We see the difference between the above and the steepest descent before it, is in line 3,4. Where now v^k replaces g^k in two places.

5.5.3 conjugate gradient, Quadratic function $J(x)$

Conjugate direction required finding set of v vectors before starting the algorithm. This algorithm generates these vectors as it runs.

Input $x(0)$ starting point, and A, b.

```
1. Init      $u^0 = x(0),\ k = 0,\ g^0 = \nabla J(x^0) = Ax^0 - b,\ v^0 = -g^0$
2. $\alpha_k$      $= \frac{[g^k]^T v^k}{[g^k]^T A\, v^k}$
4. $x^{k+1}$    $= x^k + \alpha_k v^k$
5. $g^{k+1}$    $= \nabla J(x^{k+1}) = Ax^{k+1} - b$
6. $\beta$       $= \frac{[g^{k+1}]^T A v^k}{[v^k]^T A v^k}$
7. $v^{k+1}$    $= -g^{k+1} + \beta v^k$
8. $k$         $= k + 1$
9. goto 2
```

5.5.4 conjugate gradient, None quadratic function $J(x)$, Hestenses-Stiefel

If we do not have quadratic function, then we can not use A, b to generate β. The above algorithm becomes using Hestenses-Stiefel

Input $x(0)$ starting point.

```
1. Init      $u^0 = x(0),\ k = 0,\ g^0 = \nabla J(x^0),\ v^0 = -g^0$
2. $\alpha_k$      $= \min_\alpha J\left(x^k + \alpha v^k\right)$ (line search)
3. $x^{k+1}$    $= x^k + \alpha_k v^k$
4. $g^{k+1}$    $= \nabla J(x^{k+1})$
5. $\beta$       $= \frac{[g^{k+1}]^T [g^{k+1} - g^k]}{[v^k]^T [g^{k+1} - g^k]}$
6. $v^{k+1}$    $= -g^{k+1} + \beta v^k$
7. $k$         $= k + 1$
8. goto 2
```

5.5.5 conjugate gradient, None quadratic function $J(x)$, Polak-Ribiere

If we do not have quadratic function, then we can not use A, b to generate β. The conjugate gradient algorithm becomes using Polak-Ribiere as follows

Input $x(0)$ starting point.

```
1. Init      $u^0 = x(0)$, $k = 0$, $g^0 = \nabla J(x^0)$, $v^0 = -g^0$
2. $\alpha_k$      $= \min_\alpha J\left(x^k + \alpha v^k\right)$ (line search)
3. $x^{k+1}$   $= x^k + \alpha_k v^k$
4. $g^{k+1}$   $= \nabla J(x^{k+1})$
5. $\beta$       $= \frac{\left[g^{k+1}\right]^T \left[g^{k+1} - g^k\right]}{\left[g^k\right]^T g^k}$
6. $v^{k+1}$   $= -g^{k+1} + \beta v^k$
7. $k$       $= k + 1$
8. goto 2
```

5.5.6 conjugate gradient, None quadratic function $J(x)$, Fletcher-Reeves

If we do not have quadratic function, then we can not use A, b to generate β. The conjugate gradient algorithm becomes using Fletcher-Reeves as follows

Input $x(0)$ starting point.

```
1. Init      $u^0 = x(0)$, $k = 0$, $g^0 = \nabla J(x^0)$, $v^0 = -g^0$
2. $\alpha_k$      $= \min_\alpha J\left(x^k + \alpha v^k\right)$ (line search)
3. $x^{k+1}$   $= x^k + \alpha_k v^k$
4. $g^{k+1}$   $= \nabla J(x^{k+1})$
5. $\beta$       $= \frac{\left[g^{k+1}\right]^T g^{k+1}}{\left[g^k\right]^T g^k}$
6. $v^{k+1}$   $= -g^{k+1} + \beta v^k$
7. $k$       $= k + 1$
8. goto 2
```

Index