Lecture Notes on

Optimization Methods

Compiled and Curated for

CSE 481: Optimization Methods, Spring 2019

Instructor: C. V. Jawahar

IIIT Hyderabad

Ver 0.1 (Updated on Jan. 1, 2019)

Contents

Chapter 0

Introduction to Optimization Methods

What is the scope of this course?
Examples of optimization problems.
How will this course be run?
How will the performance be evaluated?
What should I aim to learn by end of this course?
This chapter gives you answers to some of these.

0.1 A Course on Optimization Methods

We see a variety of optimization problems in our day to day life. We solve many of them very comfortably. (not sure whether we solve all of them 'optimally' !!). Some time, we also struggle with such problems. Even formally stating the problem can be very challenging in many situations.

We had studied many optimization problems in the high school days. We also have seen optimization in different areas of computer science (and of course in the wider engineering). There are many important aspects for the optimizarion that demand a formal study of this class of problems and the associated solution schemes. This course focuses on a set of fundamental aspects related to optimization methods. (We will not be able to cover all important aspects in a first level course like this. Please read classical text books, monographs or research papers for advancing your knowledge.) This is purely an introductory course.

There are many important aspects of optimization methods that we are interested in. They include:

- What is the nature of a specific optimization problem? How to characterize the problem?
- How do we formally state a problem as an optimization problem (say as a Linear Programming problem (LP))? (Do not get confused with the word programming with your favorite python programming!!. Both may look very different on surface.)
- How do we solve an optimization problem (say for a hard problem)? What are some of the popular algorithms?
- Do we have an efficient solution to the problem of interest (say a polynomial time algorithm)?
- If we do not have an efficient solution, can we come up with an efficient but approximate solution?
- When we solve an optimization problem, will we obtain the "best" solution always? (do we get the global optima or local optima)?
- What are the applications of optimization methods in other related disciplines (such as machine learning)?
- What are some of the numerical linear algebra related tools that are needed for solving optimization problems? (eg. how to solve a system of equations efficiently?)

This course assumes background in (i) mathematics (specially linear algebra) and (ii) algorithms (graph algorithms, computational complexity), both at an introductory level.

Though the major focus of this course is on optimization methods, we demonstrate the methods on two specific domains (i) approximate algorithms and (ii) machine learning. Minimal understanding of the terminologies in these spaces will be useful; but not very critical. Some of these are summarized in Chapter 1.

0.2 Why OM?

Many computational problems are optimization problems. When you want to design a minimal spanning tree (MST) for a graph, you are fining the "best" tree out of many possible ones. How do we find the best ones? Indeed, some one had already told you the algorithm to compute this. However, when you have a new problem in hand, how do we design an algorithm? You might have seen many specific tools in a course on design and analysis of algorithms. If you want to pick only one tool, then it could be LP or IP, that will be widely applicable.

Modern machine learning is dominated by optimization. The entire "training" process of a machine learning algorithm is often an optimization with some additional tricks. Deep learning methods may be optimizing an objective function of Millions or Billions variables. Even after optimizing for days or weeks, the solution that we obtain is not guaranteed to be the best (or optimal)!!. Even if the models/solutions obtained through the optimization process is useful and "very good", as an optimization person, you should appreciate that

this leaves tremendous scope for designing far superior solution to many practically important problems, if we ever can design a better optimization scheme.

0.3 What should I look for?

This course will give emphasis on (i) How to formulate the problem at hand as an optimization problem. (plain english/task to a mathematical definition of the problem). (ii) How to identify the nature of the problem (is it a linear one? it it a nonlinear one? is it a non-convex one?) (iii) What are the popular algorithms (may be a smaller set of the popular ones) to solve the optimization problem exactly or some times approximately. (iv) How optimization techniques gets used in machine learning, design of algorithms etc.

Implementatin of many of these optimization problems are numerically tricky. Though we will see some numerical and computational procedures, the objective of this course is NOT to train you in implementing the solution to the optimization problem on a computer. Whereever programming is required, student will be encouraged and guided to use some of the standard libraries/packages. Minimal familiarity with computer programming is expected in this regard.

0.4 Classes of Optimization Problems

There are many interesting dichotomies in this space. Some of them worth noticing:

1. Linear Vs Nonlinear
2. Convex vs NonConvex
3. Discrete Vs Continuos
4. Constrained vs Unconstrained

The names may make these distinctions reasonably obvious. You can keep in mind these words as and when we progress. However, the nature of the problem, solution scheme, guarantees etc. could be very different across the categories. This is why it becomes important for us to know where our problem lies in practice.

Question Contrast and elaborate these classification schemes in detail at the end this course. Give examples of problems and also pick classical or practical problems in each of these categories.

By end of this course, you may aim to develop skills that can help you to place a problem in an appropriate class.

0.5 Spring 2019: Couse on "Optimization Methods"

0.5.1 Course Plan

Each chapter in this note series, correspond to approximately one lecture of 1.5 Hrs. The assumption is that you should learn content worth 25 to 30 lectures by end of this course. This lecture notes and the Internet resources will help you to know these further. It is important that the student puts effort him/herself and go some what beyond the lectures. Content worth 20 to 25 lectures will be discussed in the lectures. It is not expected that you come prepared for the lectures (at this stage) by reading these notes in advance. But it is expected that you read it after the lecture. (Notes may not be descriptive or complete enough for you to read and follow smoothly at this stage).

0.5.2 Evaluation/Grading

I wish a course like this gets evaluated purely based on class works, homeworks, personalized evaluations. May be we are not fully ready to get rid of the examination at this stage. We will continue to have mid and final examinations. There is no plan to make these exams tricky or super challenging.

Regular Homeworks The main evaluation is based on regular homeworks. You are expected to solve 50 homework problems "at your convinience". i.e., whenever you think you have time, you can "ask for" a question and you will be given a question immediately. (this implies that you do not have to take special permissions for family functions or festivals.). To make the load (and more importantly learning) uniform, you can solve a max of one question a day. However, you will have to submit your answer within the next 24Hrs. We assume that the average time required for solving a question to be 1 to 2 Hrs for a typical student, provided that the student is familiar with the content. Indeed, there will be situations where you are unable to solve/submit due to personal reasons, or limitations. Student may make mistakes too. Therefore, you will have access to 60 questions and the best 50 will be used for the grading. We typically expect you solve 2-3 questions between sucessive lectures. If you move faster that that, you will get questions from future lectures and it becomes your responsibility to learn yourself.

We wish you collaborate openly. Once you collaborate, do acknwoledge. Your points could also be divided accordingly. The person who helps (including TAs) should be given due credit.

Questions that you solve may be different from that your friends solve. It is a serious offtense if (i) you take the question out of the system (question could be watermarked with visible and/or invisible watermarks) inlcuding photography/screenshots (ii) if you disucss the solutions electronically (electronic groups, mailing lists). (iii) take help of people without acknwledging.

Grading

- Two mid semester exams: 25%
- One Final exam: 20%
- Homeworks: 40%
- Assignments (Max 3) 15%
- Course participation (discussions in class+course portal): 5%

There could be a maximum of ±5% change from this plan. It is expected that the class participation score will be zero for most students. However, it can be positive and negative for a smaller fraction of the students.

0.6 Text Books and References

There are many popular text books and references. A list is available on the course page.

0.6.1 Books (B)

1. M T Heath, "Scientific Computing", TMH (Most of First six chapters)
2. C H Papadimitriou and K Steiglitz, "Combinatorial Optimization: Algorithms and Complexity" (Most of First seven chapters), Dover
3. S. Boyd and L Vandenberghe, "Convex Optimization", Cambridge University Press (Online Copy available at: http://www.stanford.edu/ boyd/cvxbook/)

4. L Vandenberghe, Lecture Notes for Applied Numerical Computing, (Online available at: http://www.ee.ucla.edu denbe/103/reader.pdf)

5. D Bertisimas and J N Tsitsiklis, "Introduction to Linear Optimization", Athena Scientific http://personal.vu.nl/

6. J Matousek and B. Gartner, "Understanding and Using Linear Programming", Springer, 2007 (http://blogs.epfl.ch/extrema/documents/Maison

0.6.2 NOTES/BLOGS (A)

1. TOADD

0.6.3 Papers (P)

1. TOADD

0.7 Notes and Disclaimers

- **Notations** There is no guarantee that the notations are fully consistent. Though it is attempted at many places.
- **Disclaimer** This is from the draft lecture notes for CSE481 at IIIT Hyderabad in the past. Do not circulate these notes beyond the registered students. Please do report bugs/errors/omissions on the course web page. This is only a draft version. Also the objective of posting these notes is to initiate discussions within the class, but outside the class room. Also do suggest better explanations of the concepts or better examples. This will help to improve the notes further. This is built over the lectures/notes/scribes of the previous offerings.
- **Enhancements** The notes are expected to be enhanced and more complete by the end of the semester. Whevener there is a new version, you can see on the course page.
- **Academic Honesty** This is very important. If you take help of your students, or TAs, please acknowledge them well. Attempt to use social media, electronic forums or other means to do unethical collaboration may lead to academic actions.

0.8 Version History

This is a lecture notes based on the notes from the previous offerings. This is still evolving. Please see this section to know where we are, and how it changed over versions.

- **Ver 0.1** [Jan 1 2019] Some structural changes and cosmetic changes to suite the plans for the Spring 2019. Available to the students who have enrolled to roughly appreciate the course content.
- **Ver 0.2** Coming soon .. (EDA Jan 15, 2019)

Chapter 1

Background

This is the time to revise what we had learned in the past. Many of the terminologies could be familiar to you; but it is worth reading and refreshing. If you are too far from these terminologies, you may find it hard to go through the rest of the material.

1.1 Background on Matrices

This course assumes some amount of familiarity of mathematics. Specially, linear algebra. Basic understanding of geometry (points, lines, planes) is expected. Also basics of calculs (differentiation) is required. Here are some definitions/terms that you should have heard already:

- **Vectors**
- **Matrices**
- **Types of Matrices** (i) square (ii) triangular
- **Determinant of a matrix**
- **Rank of a matrix**
- **Eigen Values and Eigen Vectors**
- **Norms** $L0, L1, L2, Lp, L_\infty$ norms
- **System of Equations** System of linear equations represented as $\mathbf{Ax} = \mathbf{b}$. Where $\mathbf{A}$ is a matrix and $\mathbf{b}$ is a vector.
- **Lines, Planes, Hyper Planes and Half Spaces** $ax_1 + bx_2 = c$ is a line in 2D. $ax_1 + bx_2 \leq c$ is a line plus one side of the line (half space) in 2D. Similarly $ax_1 + bx_2 \geq c$. In 3D this is equivalent to planes as $a_1x_1 + a_2x_2 + a_3x_3 = b$ and in d dimension, it is $\mathbf{a}^T\mathbf{x} = b$.
- What does it mean by solving $\mathbf{Ax} = \mathbf{b}$ and Finding Feasible points for $\mathbf{Ax} \leq \mathbf{b}$

1.2 Background on Graphs

This course assumes some amount of familiarity with basics of Graphcs (eg. what you learn from a first course on Discrete Mathematice or Data Structures).

You are expected to be familar with:

- **Graphs and Trees**
- **Nodes/Vertices and Edges**
- **Sample Problem: Shortest Path Problem**
- **Sample Problem: Minimal Spanning Tree**

1.3 Hard Problems and Approximate Algorithms

- **Big O** Computational complexity. Big Oh Notation.
- **Complexity** Why is it very important.
- **P and NP** Class of problems.
- **Sample Problem: Knapsack problem**
- **Sample Problem: TSP Problem**

1.4 Optimization Problems in Machine Learning and Signal Processing

- **Least Square Problem**
- **Regression**
- **Binary Classification**
- **Multi Class Classification**
- **Support Vector Machines and Deep Learning**

Reading

This lecture notes, in no ways, remove the need to read fundamental text books, references and learn more.
You will find excellent references for the above at many places including the courses that you have done and also the text books that you used. The best is to use the text book that you used for your mathematics, data structures and algorithm courses.

Chapter 2

Linear and Integer Programming

Let us a start with a simple, but popular class of problems.

2.1 Introduction to Linear Programming(LP)

Linear Function A function $f(x_1, x_2, \ldots, x_n)$ of $x_1, \ldots x_n$ is a linear function if and only if for some set of constants $c_1, c_2, \ldots, c_n$,

$$f(x_1, x_2, \ldots, x_n) = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

Linear Inequalities For any linear function $f(x_1, x_2, \ldots, x_n)$ and any number b, the inequalities such as

$$f(x_1, x_2, \ldots, x_n) \leq b$$

$$f(x_1, x_2, \ldots, x_n) \geq b$$

are linear inequalities.

2.1.1 Linear Programming (LP)

A linear programming problem may be defined as the problem of maximizing or minimizing a linear function subject to linear constraints. The constraints may be equalities or inequalities.

To discuss a bit more in detail, a Linear Programming Problem is an optimization problem for which

1. We attempt to maximize (or minimize) a linear function (or objective function) of the decision variables. Objective function (z) is the criterion for selecting the "best" values of the decision variables. Decision variables (x_i) are the variables of interest here.
2. We are often interested in the decision variables at the optima (x_i^*) and the value of the objective (z^*). Not always both!!.
3. The values of the decision variables must satisfy a set of constraints. Each of these constraints can be equalities or inequalities. Constraints are like limitations on the resource availability. There can be more than one constraints.
4. Sign restriction could be there on each variable. i.e., x_i $x_i \geq 0$, $x_i \leq 0$ or x_i is unrestricted.

In Linear programming, the decision variables can take any real value.

2.1.2 Integer Programming (IP)

In Integer programming, the decision variables can take only integer value. This may seem to be a simple change from LP. But this makes solving a general IP to be very hard.

2.2 Formal Introduction to LP and IP

Linear programs are problems that can be expressed in a canonical form:

$$maximize \quad \mathbf{c^T x}$$

subject to

$$\mathbf{Ax} \leq \mathbf{b}$$

$$\mathbf{x} \geq \mathbf{0}; \mathbf{x} \in \mathbf{R}^n$$

where $\mathbf{x}$ represents the vector of n variables (the decision variables of our interest), $\mathbf{c}$ and $\mathbf{b}$ are vectors of (known) coefficients, $\mathbf{A}$ is a (known) $m \times n$ matrix of coefficients, and $(.)^T$ is the matrix transpose. Note that the coefficients from the m constraints are now arranged into a matrix form in $\mathbf{A}$.

The expression to be maximized or minimized is called the objective function. The inequalities $\mathbf{Ax} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$ are the constraints which specify a convex polytope over which the objective function is to be optimised.

For Linear Program (LP), x_is are real numbers i.e., $\mathbf{x} \in \mathrm{R}^n$

For Integer Program (IP), x_is are integers, i.e., $\mathbf{x} \in \mathrm{Z}^n$

We may also have mixed integer programs where some variables are real and some are integers.

Though the LP and IP may seem to be very similar, they can be very difficult in practice. An LP can be solved "easily", while solving an IP can be "hard". We shall see these more in detail as we move forward.

2.2.1 Related Tricks to Remember

You may not be able to get a problem that can be written in the above form easily.

Given a problem, we often rewrite the problem into a standard form.

Formulations of LP can be transformed from one form to another easily. i.e., maximization of objective function to minimization. Some simple rules that guide the transformations are:

1. $max \iff min$ is same as $\mathbf{c^T x} \iff -\mathbf{c^T x}$
2. Absolute Values $|x_i| \leq 3 \iff x_i \leq 3$ AND $x_i \geq -3$
3. Equality $x_i = 3 \iff x_i \leq 3$ and $x_i \geq 3$

2.2.2 Related Terms

1. **Solution:** A solution of a LP program is a setting or state of the decision variables.
2. **Feasible Solution:** A solution that satisfies all the constraints.
3. **Feasible Region:** Set of all feasible solutions. A different way of defining is as a region enclosed by all the inequalities involved. Feasible Region is also referred to as search space, solution space or feasible set. An optimization method searches for the best solution from the feasible region.
4. **Optimal Solution:** Optimal solution is the feasible solution with the largest objective function value for a maximization problem or a feasible solution with the smallest objective function value for the minimization problem.

2.3 Numerical Example

Let us now start with a simple numerical problem.

$$maximize \quad 3x_1 + 2x_2$$

subject to

$$2x_1 + x_2 \leq 6$$

$$7x_1 + 8x_2 \leq 28$$

$$x_1 \geq 0 \ ; \ x_2 \geq 0 \text{ and both real}$$

In this problem there are two unknowns, and four constraints. There are several important terms to understand before solving any LP problem. They are given below.

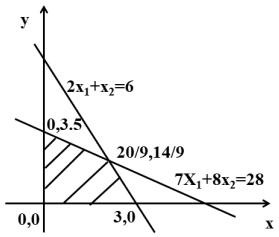


Figure 2.1: Constraints and the Feasible Region.

Objective Function The function to be maximized (or minimized) is called the objective function. Here, the objective function is $3x_1 + 2x_2$ and the objective needs to be maximized.

Question: For the above problem what is m? what is n? 2 and 2. Can you write **A**, **b** and **c**?

Constraints A constraint involves an inequality or equality in some linear function of the variables. The two constraints, $x_1 \geq 0$ and $x_2 \geq 0$, are special. These are called **non-negativity constraints** and are often found in linear programming problems. The other constraints are then called the **main constraints**. Here, the main constraints are (i) $2x_1 + x_2 \leq 6$ and (ii) $7x_1 + 8x_2 \leq 28$

Feasible Region A feasible region or solution space is the set of all possible points of an optimization problem that satisfy the problem's constraints. The feasible region for the example problem mentioned above is given in the figure.

Feasible Solution A feasible solution to a linear program is a solution that satisfies all the constraints which effectively lies within the feasible region. The point $(1, 1)$ is a feasible solution in our example.

Optimal Solution An optimal solution to a linear program is a feasible solution with the largest objective function value (for a maximization problem). A linear program may have multiple optimal solutions, but only one optimal solution value.

Note:

1. Since every inequality forms a half plane (which is a convex set), the set of feasible solutions to an LP (feasible region) forms a (possibly unbounded) convex set.
2. Optimal solution lies in any face/edge of the convex set. Eventually, every linear program has an **extreme point** (corner point) that is an optimal solution.

Optimization Method Do we have to search at all the vertices? Do we have to search at all the points inside the shaded region also?

2.4 Graphical Method of Solving LP

For the example problem, the objective is to maximize $3x_1 + 2x_2$. From the figure, the corner points are (0,3.5), (0,0), (3,0) and (20/9,14/9). Their corresponding values are 7, 0, 9 and 9.77 respectively. Since Max value is 9.77 the optimal point is (20/9,14/9)

Let us look at a procedure to find this maxima. Consider a line $3x_1 + 2x_2 = k$. Let us draw this line for various various values of k. When $k = 0$, this passes through origin, now when we increase k, this line moves away from origin (in a certain direction). (If k is very large, this line does not intersect our polygon of interest.) Let us increase k slowly from zero. And at certain point, the last line that intersect/touch the polygon is obtained. This is the one of our interest. In this case, this is when $k = 9.77$ and the line passes through $(20/9, 14/9)$.

Let us now summarise our graphical way of solving LPs.

Given a LP problem i.e., an objective function and a set of constraints, we proceed as follows:

1. Construct a graph and plot the constraint inequalities.
2. Determine the valid side of each constraint line. This can be done by substituting the origin in the inequality. If origin satisfies the inequality, then all the points on the origin side of the line are feasible (valid), and all the points on the other side of the line are infeasible (invalid). The reverse is true of the origin doesn't satisfy the inequality.
3. Identify the feasible solution region. This is the area of the graph that is valid for all the constraints. Choosing any point in the region results in a valid solution.
4. Plot the objective function line and use it to move in the direction of improvement that is in the direction of the greater value if the objective is to maximize the objective function and in the direction of the lowest value if the objective is to minimize.
5. Optimal Solution aways occurs at the corners. So algebraically calculate the coordinates of all corners of the feasible region and use the objective function line to find the most optimal corner.
6. Now use the coordinates of the optimal corner (optimal solution) to get the objective function value.

2.5 Some cases of special interest

Every linear program either

1. is infeasible,
2. is unbounded,
3. has a unique optimal solution value

Question: Does (3) mean that every LP has only one optima?

Infeasible Solution A linear program is infeasible if it has no feasible solutions, i.e. the feasible region is empty. Here is an example for Infeasible Solution

Problem

$$maximize \quad z = x_1 + x_2$$

subject to

$$x_1 - 2x_2 \geq 6$$
$$2x_1 + x_2 \leq 4$$
$$x_1, x_2 \geq 0$$

Unbounded Feasible Region A linear program is infeasible if it has no feasible solutions, i.e. the feasible region is empty. Here is an example for Unbounded Region

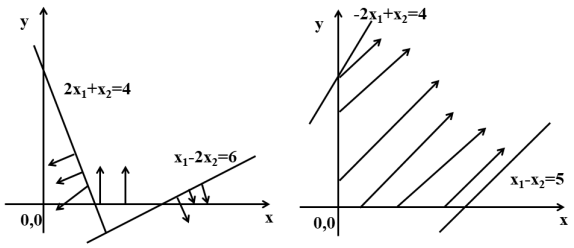


Figure 2.2: Infeasible Solution and Unbounded Solution

Problem:

$$maximize \quad z = x_1 + x_2$$

subject to

$$x_1 - x_2 \leq 5$$

$$-2x_1 + x_2 \leq 4$$

$$x_1, x_2 \geq 0$$

2.5.1 Excercise

Solve the following problem graphically.

$$Minimize \quad z = 5x_1 + 2x_2$$

Subject to:

$$6x_1 + x_2 \geq 6$$

$$4x_1 + 3x_2 \geq 12$$

$$x_1 + 2x_2 \geq 4$$

$$x_i \geq 0 \text{ and real}$$

Graph Method: Notes

1. Draw the lines corresponding to each constraints in a graph and shade the half planes according to the inequalities.
2. If there is no intersection of the shaded regions then there is no solution.
3. If there is a bounded intersection region, find the coordinates of the corners by solving the systems of intersecting equations.
4. Find which coordinate gives the optimum value by applying them over the optimization function.

Note: Is this how we plan to solve all the problems as we move forward? No never. However it is important to understand the geometric view point of LP, the constraints and the objectives.

Question: Can an IP also be solved like this? No that is a larger story for the next few lectures.

Question: Solve the LPs that we had seen with a numerical solver on your computer (say Simplex) in your favourite library. Do compare your answers with that.

2.6 LP/IP Formulations

One of our objective is to learn how to formulate a problem as LP or IP in the initial part of this course. We will take many examples in the next few lectures. This is an important skill to have. It is expected that you practice formulating problems on your own.

2.7 Pattern Classification Problem

Let us now consider an example problem i.e., pattern classification. This is a fundamental problem in machine learning offern referred to as supervised classification problem. If you are given examples of "apples" and "not-apples", a machine learning algorithm is asked to lean a **classifier** that separates the apples fron others. Let us assume that every item (sample/object) of interest is characterized by two variables x_i and y_i (say highit and weight).

Problem Statement Given that they are two disjoint patterns. One pattern is a positive pattern consisting of all coordinates $\{(x_i^+, y_i^+)\}$ Another pattern is a negative pattern consisting of all coordinates $\{(x_i^-, y_i^-)\}$. Assume there are n^+ positive points and n^- negative examples. We have to find a line such that it classifies the patterns (or the 2D points) in the best possible way i.e the line must be in between and must be as far as possible from every point on both the patterns classes (i.e., positives and negatives).

Note that in practice this classifier need not be a line. It can be of any arbitrary shape. However, many practical problems still use linear classifiers.

There are actually many lines as possible solution to this problem. A good classifier also wants the points to be as away as possible from the separating line/plane. Can we maximize this separation? Let us formulate this problem as LP.

LP Formulation We would like to maximize the separation δ of each sample (to be precise the nearest sample) from the line.

We take a variable δ and maximize it while making sure that that the distance of the nearest point in both the classes (i.e., positive class and negative class) is at least δ (or) the distance of the line to each and every point in the data set is at least δ i.e.,

- The distance from any one of the positive pattern points to the line at least δ.
- The distance from the line to any of the negative pattern points is at least δ.
- The line is defined as $y_i = ax_i + b$, and positive samples are on one side and negative samples on the other side.

Formulation of the problem is as follows:

$$Maximize \quad \delta$$

Subject to:

$$y_i^+ \geq ax_i^+ + b + \delta \quad i = 1, \ldots, n^+$$

$$y_i^- \leq ax_i^- + b - \delta \quad i = 1, \ldots, n^-$$

If we solve this problem, we obtain a line that separates the points from the line with at least the distance δ.

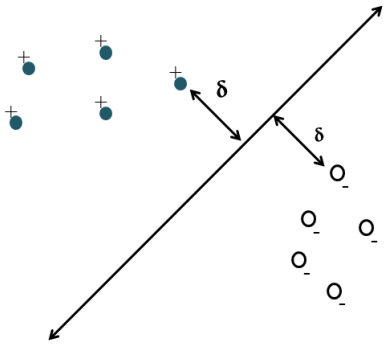


Figure 2.3: Line which separates the patterns with distance of at least δ

Comments:

1. We have formulated the LP for simple Euclidean distance measured along y. Not the orthogonal distance from the line. The problem becomes difficult if orthogonal distance is to be considered. (Note that the figure does not show the problem setting correctly.)
2. We have assumed that positive pattern is above the line and negative pattern is below the line. It could be the reverse also. i.e., negative above and positive above. **Question:** Does it matter to us really?

Reading

Chapter 1 and 2 of [B6] are easy to read. If you want more formal introduction read Chapter 1 of [B5]

Chapter 3

LP and IP Formulations

- We had seen the basic definitions of the LP and IP in the last lecture.
- We also know how to solve an LP graphically (not all LPs!!) on paper.

One of our objective is to learn how to formulate a problem as LP or IP in the initial part of this course.

3.1 Formulations

In this lecture, we see more examples on how to formulate LP and IP. Though each example may be one of its own form, do appreciate the utility and also the spectrum of problems that getts mapped to LP and IP. Try to pickup the skill of formulating.

3.2 Problem: Line Fitting

Given a set of points (x_i, y_i), where $i = 1, 2, 3, \ldots, n$, find the "best" line to fit these points. i.e., we want to find a straight line of the form $y = ax + b$ that best describes the points or data set. This is a classical problem with interest in many areas.

Needless to say, we may not be able to find a line that pass through all the points in most cases. Our objective is then to find the best line. Best in what sense? That defines (and change the nature of the problem) the problem.

Assume our objective is to find a line such that it is "close" to all the points, i.e., sum of the distances of all the points to the line is minimum. Objective function is then:

$$minimize \quad \sum_{i=1}^{n} |y_i - (ax_i + b)|$$

(Note: here we have used L_1 Norm because it helps to get deviation or error of the line from the point, while L_2 Norm magnifies the error i.e. $(y_i - (ax_i + b))^2$.)

Let error $e_i = |y_i - (ax_i + b)|$. Then we write the problem as:

$$minimize \sum_{i=1}^{n} e_i$$

subject to:

$$|y_i - (ax_i + b)| \leq e_i \quad i = 1, 2, \ldots, n$$

If we eliminate norm, and write as our familiar LP, we obtain:

$$minimize \quad \sum_{i=1}^{n} e_i$$

subject to:

$$(y_i - (ax_i + b)) \leq e_i$$

$$-(y_i - (ax_i + b)) \leq e_i \quad i = 1, 2, \ldots, n$$

We have a total of $N + 1$ variables $(e_1, e_2, ...e_n, a, b)$ and $2n$ constraints.

3.2.1 Variations

There are many interesting variations of this problem, where the error is defined as L0, L1 or L2 norm. (We may see some of them later on in this course.) The objectives in these cases become

$$E_0 = \sum_i \| y_i - (ax_i \dot{+} b) \|_0$$

$$E_1 = \sum_i \| y_i - (ax_i \dot{+} b) \|_1$$

$$E_2 = \sum_i \parallel y_i - (ax_i \dot{+} b) \parallel_2$$

They have different interpretations

Minimization of E_0 leads to line passes through as many points as possible.

Minimization of E_1 leads to least absolute derivation we saw above.

Minimization of E_2 is the ordinary least squares.

Question What about L_∞ norm?

3.3 Minimizing Norms

We had seen already the how the interpretation of the problem changes with a change in the norm. Let us now consider couple of examples.

3.3.1 Example 1

Consider the problems of

$$Minimize \ \|\mathbf{Ax} - \mathbf{b}\|_1$$

subject to

$$\|\mathbf{x}\|_\infty \leq 1$$

(With our notations, it should be obvious that $\mathbf{A}$ is a matrix and $\mathbf{b}$ as well as $\mathbf{x}$ are vectors. We are given $\mathbf{A}$ and $\mathbf{b}$. We optimize over $\mathbf{x}$. Indeed over non-negativity constraints and real vector constraints are not explicitly written here.)

The above problem can be formulated as an LP. Let us assume that $\mathbf{A}$ is a matrix of dimension $m \times n$, $\mathbf{x}$ is a vector of dimension $n \times 1$ and $\mathbf{b}$ is a vector of dimension $m \times 1$ We know that $\|x\|_\infty$ norm measure the maximum of the absolute values of the elements in a vector.

$$\|\mathbf{x}\|_\infty = max(|x_1|, |x_2|, |x_3|, \ldots, |x_n|)$$

Since the constraint $\|x\|_\infty \leq 1$ the maximum element of the vector is less than equal to 1 which means that every element of the vector is less than or equal to 1.

$$-1 \leq x_j \leq 1 \, , j = 1, \ldots, n$$

We know that $||\mathbf{x}||_1$ norm measure the sum of the absolute values of the elements in a vector.

$$\|\mathbf{x}\|_1 = |x_1| + |x_2| + \ldots + |x_n|.$$

thus

$$\|\mathbf{Ax} - \mathbf{b}\|_1 = |A_{11}x_1 + A_{12}x_2 + \ldots + A_{1n}x_n - b_1| + |A_{21}x_1 + A_{22}x_2 + \ldots + A_{2n}x_n - b_2| \ldots + |A_{m1}x_1 + A_{m2}x_2 + \ldots + A_{mn}x_n - b_m$$

As we have to minimize every element of $\|\mathbf{Ax} - \mathbf{b}\|_1$, Here we take a vector $\mathbf{y}$ of dimension $\mathbf{m \times 1}$ where each element of the vector **Ax-b** ranges from $-y_i$ to y_i. We now mimimize the summation of all elements of the $\mathbf{y}$ vector. Thus the final objective function can be written as follows.

$$Minimize \ \sum_{i=1}^{m} y_i$$

Subject to:

$$-y_i \leq \sum_{j=1}^{n}(a_{ij} \cdot x_j) - b_i \leq y_i \, , i = 1, \ldots, m$$

,

$$-1 \leq x_j \leq 1 \, , j = 1, \ldots, n$$

Question: How many constraints are there here in our standard form? Do write them as $\mathbf{Ax} \leq \mathbf{b}$.

3.3.2 Example 2

Let us now consider another problem:

Minimize $\|\mathbf{x}\|_\mathbf{1}$ subject to constraint $\|\mathbf{Ax} - \mathbf{b}\|_\infty \leq 1$

This problem can also be formulated as an LP.

We know that $||\mathbf{x}||_\infty$ norm measure the maximum of the absolute values of the elements in a vector.

$$\|\mathbf{x}\|_\infty = max(|x_1|, \ldots, |x_n|)$$

Since the constraint is $\|\mathbf{Ax} - \mathbf{b}\|_\infty \leq 1$, the maximum element of the vector is less than equal to 1 which means that every element of the vector is less than or equal to 1.

$$-1 \leq \sum_{j=1}^{n}(a_{ij} \cdot x_j) - b_i \leq 1 \, , i = 1 \ldots, m$$

We know that $\|\mathbf{x}\|_1$ norm measure the sum of the absolute values of the elements in a vector.

$$\|\mathbf{x}\|_1 = |x_1| + |x_2| + \ldots + |x_n|.$$

As we have to minimize every element of $\|\mathbf{x}\|_\mathbf{1}$, Here we take a vector $\mathbf{y}$ of dimension $\mathbf{n} \times \mathbf{1}$ where each element of the vector $\mathbf{x}$ ranges from $-y_i$ to y_i.

The LP problem can be formulated as follows

$$Minimize \quad \sum_{j=1}^{n} y_j$$

Subject to:

$$-1 \leq \sum_{j=1}^{n} a_{ij} x_j - b_i \leq 1 \, , i = 1 \ldots, m$$

$$-y_j \leq x_j \leq y_j \, , j = 1 \ldots, n$$

3.4 Example Problem: Cutting Paper Rolls

Let us now consider an engineering problem. Paper rolls come in (gets manufactored) 3m (300 cm) width. A roll is really long. Length of the roll does not matter, since the customers are also looking for only rolls.

There is an on order to serve:

(i) 97 rolls of width 135 cm (ii) 610 rolls of width 108 cm (iii) 395 rolls of width 93 cm (iv) 211 rolls of width 42 cm

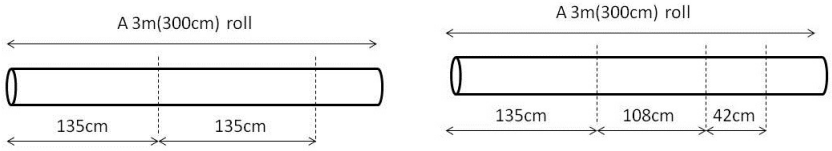


Figure 3.1: First and second possible ways of cutting the role

What is the smallest number of 3m roll sto be cut?

The trivial way of meeting the order is to get $97 + 610 + 395 + 211 = 1313$ rolls. If we cut all these rolls and take rolls of the appropriate widths, we may be wasting a lot. Our interest is reducing the number of rolls to be cut.

We can cut these paper rolls in 12 different ways:

- P1 : 2×135
- P2 : 135+108+42
- P3 : 135+93+42
- P4 : 135+3*42
- P5 : 2×108+2*42
- P6 : 108+2*93
- P7 : 108+93+2×42
- P8 : 108+4*42
- P9 : 3×93
- P10 : 2×93+2×42
- P11 : 93+4×42
- P12 : 7×42

Note that all these are less than 300cm.

Formulation Let x_i be the number of times ith possiblity is used in cutting a 3m roll. Then problem can be formulated as follows:

$$Minimize \sum_j x_j$$

Subject to:

$$2x_1 + x_2 + x_3 + x_4 \geq 97$$

$$x_2 + x_5 + x_6 + x_7 + x_8 \geq 610$$

$$x_3 + 2x_6 + x_7 + 3x_9 + 2x_{10} + x_{11} \geq 395$$

$$x_2 + x_3 + 3x_4 + 2x_5 + 2x_7 + 4x_8 + 2x_{10} + 4x_{11} + 2x_{12} \geq 211$$

with non-negativity constraints.

Upon Solving the above problem as LP, we get $x_1 = 48.5, x_5 = 206.25, x_6 = 197.5$

Therefore, the final solution is $x_1 + x_5 + x_6 = 452.5$ which is a non-integer solution. To convert above solution into an integer solution we round up (why rounding up?) the values of x_1, x_5, x_6.

Rounding up each of the above values we get $x_1 = 49, x_5 = 207, x_6 = 198$ and $x_1 + x_5 + x_6 = 454$. Therefore, the number of rolls to be cut is 454. However, is this minimum?

The optimal solution to the problem is 453 which is obtained by $x_1 = 49, x_5 = 207, x_6 = 196, x_9 = 1$ and $x_1 + x_5 + x_6 + x_9 = 453$.

Thus rounding of the result of an LP solution to and integer solution may or not be the optimal solution to the IP problem but it will be nearer to it.

What went wrong here? Nothing. This was not an LP. This was an IP. Solving by forgetting the integer constraint does not guarantee the optimal solution.

3.5 Example Problem: MaxFlow

Finding maximum flow in a graph is a classical problem of theoretical and practical interest. The max flow passing from source node to destination node in a network equals the minimum capacity which when removed from the network results in a situation where there is no flow from source to destination. (In any network, the value of max flow equals of min-cut. That is, if there exists a max flow 'f', there exists a cut whose capacity equals the value of f.)

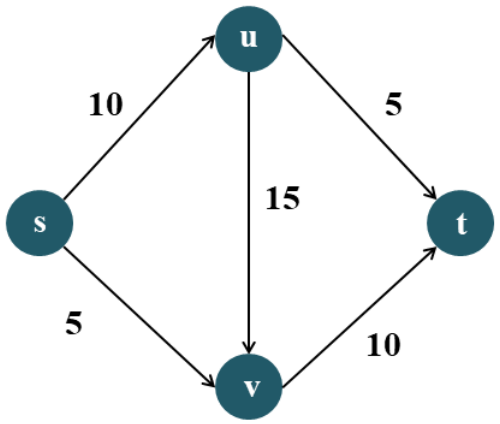


Figure 3.2: A flow network.

For the above graph, let the variables $f_{su}, f_{sv} f_{uv}, f_{ut}$ and f_{vt} be the maximum flows along the respective edges. We also know that the flow can not be more than the edge capacity. Also since there is no storage in nodes, at every nodes, incoming and outgoing flows should match.

LP problem would be as follows,

$$Maximize \quad f_{su} + f_{sv}$$

Subject to

$$f_{su} = f_{uv} + f_{ut}$$

$$f_{sv} + f_{uv} = f_{vt}$$
$$0 \leq f_{su} \leq 10$$
$$0 \leq f_{sv} \leq 5$$
$$0 \leq f_{uv} \leq 15$$
$$0 \leq f_{ut} \leq 5$$
$$0 \leq f_{vt} \leq 10$$

with non-negativity and real constraints.

Question: Write the c, A and b for the graph in Figure (b).

Discussions In any network, the value of max flow equals of min-cut. That is, if there exists a max flow 'f', there exists a cut whose capacity equals the value of f. Can we also formulate the min-cut problem as an LP/IP? Which one? How?

Are there some relationships between these problems? Very often problems come in "pairs". We call them as dual problems. More about duality in one of the later lectures.

Excercise Consider the LP

$$Minimize \;\; c_1x_1 + c_2x_2 + c_3x_3$$

subject to

$$x_1 + x_2 \geq 1$$
$$x_1 + 2x_2 \leq 3$$
$$x_1 \geq 0; x_2 \geq 0; x_3 \geq 0$$

Give the optimal value and the optimal set for the following values of c (i) $c = (-1, 0, 1)$ (ii) $c = (0, 1, 0)$ (iii) $c = (0, 0, -1)$

Excercise Formulate the following problem as LP

$$Minimize \|Ax - b\|_1 + \|x\|_\infty$$

Reading

Chapter 2 of [B6] has a large list of problems.

Chapter 4

Solving IP using Branch and Bound

- We had seen many LP and IP formulations in the past. We will see some more as we move forward.
- We had seen how LP can be solved using graphical method.
- Here we will see how an IP can be solved.

4.1 How to solve an IP?

The advantage of IP is that they are a very expressive language to formulate optimization problems, and they can capture in a natural and direct way a large number of combinatorial optimization problems. The disadvantage of IP is that finding optimal solutions for IPs is NP-Hard, in general. (There are, in fact, lucky IP problems that can be solved easily. They are topics of interest for a future lecture.)

Let us now see how to solve some of the IP problems. We are interested in two methods at this stage.

- Branch and Bound
- LP Relaxation

If we relax the integer constraints from the IP and assume that the variables are real, we get an LP. We can then solve LP. However, how are the solutions related? Note that the optima of the IP is inferior (or equal) to the optima of LP. (for a maximization problem ($IP^* \leq LP^*$ and for a minimization problem ($IP^* \geq LP^*$). LP relaxation is a technique of relatxing the IP as an LP and solving it efficiently. We will come back to this later.

We 'assume" that we know how to solve LP. We use the LP solver as a black box/sub-routine here. Note that we know how to solve LP using graphical method for simpler problems and use simplex solvers for large LP on a computer.

4.2 Branch and Bound for IP

This is a classical paradigm that can be used for solving many other hard problems. (refer to your text books on Algorithms for details.)

Basic idea is the following: We divide a large problem into multiple smaller ones. (This is the "branch" part.) The bounding part is done by estimating how good a solution we can get for each smaller problems (to do this, we may have to divide the problem further) The optimal value from a subproblem will tell you whether there is a need to further divide it or not. That is the "bound" part. i.e., we do not have to divide until we get trivial/small problems.

We use the linear programming relaxation to estimate the bound on the optimal solution of the integer programming.

4.2.1 General Branch and Bound Algorithm

Consider the problem statement:

Maximize $\mathbf{c^T x}$ subject to $\mathbf{Ax} \leq \mathbf{b}$

1. Initialize the list of problme (or constraints) as $L = \{\mathbf{Ax} \leq \mathbf{b}\}$. The set of problems to be solved is represented by the constraints here, since the objective does not change.
2. Initialize $\mathbf{x}^- = \emptyset$, $l = -\infty$. Here l is the presently seen best solution in the algorithm and $\mathbf{x}^-$ is the presently seen best optimal value in the algorithm.
3. while $L \neq \emptyset$
   (a) Pick subproblem *maximize* $\mathbf{c^T x}$, $\mathbf{A'x} \leq \mathbf{b'}$ and solve the LP. Also delete the subproblem constraints from L.
   (b) Let $\mathbf{x}^*$ be the optimum solution when you solve the above LP problem.
   (c) If $\mathbf{x}^* \in Z^n$ and $\mathbf{c^T x}^* > l$. Then set $\mathbf{x}^- = \mathbf{x}^*$and $l = \mathbf{c^T x}^*$.

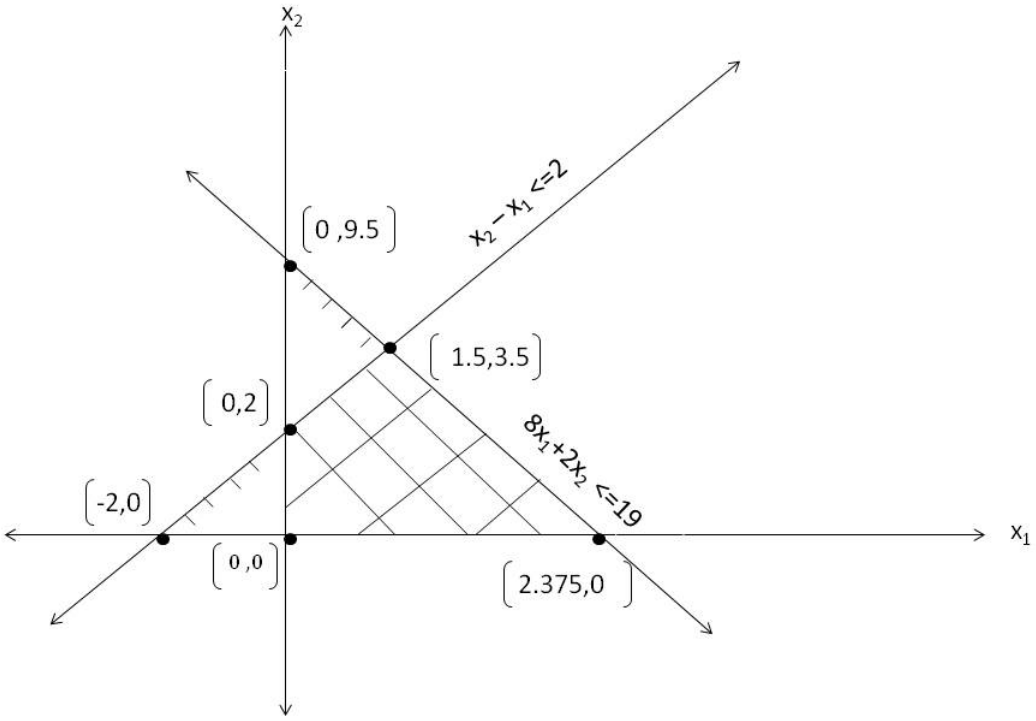


Figure 4.1: Feasible region of the main problem

(d) If $\mathbf{x}^* \notin Z^n$ and $\mathbf{c^T x}^* > l$ then
add two subproblems into the list L for $x_j^* \notin$ Z

- Max $\mathbf{c^T x}$ such that $\mathbf{A'x} \leq \mathbf{b'}$ and $x_j \leq \lfloor x_j^* \rfloor$ and add to L.
- Max $\mathbf{c^T x}$ such that $\mathbf{A'x} \leq \mathbf{b'}$ and $x_j \geq \lceil x_j^* \rceil$ and add to L.

Quesion: Refer steps 3(c) and 3(d): Can't $\mathbf{c^T x} < l$?. What should we do in such cases?

4.3 Numerical Examples of Branch and Bound

Let us now solve some IPs numerically.

4.3.1 Example 1

Given the following problem.Obtain an Integer Solution.
Maximize $x_1 + x_2$
subject to :
$x_2 - x_1 \leq 2$
$8x_2 + 2x_1 \leq 19$
$x_1, x_2 \geq 0 \quad x_1 \in Z \quad x_2 \in Z$

Solution:

- We plot the graph by taking two axis $x_1(x_2{=}0)$ and $x_2(x_1{=}0)$.
- Intersection of the lines x_2 - $x_1{=}$ 2 and $8x_2{+}$ $2x_1{=}$ 19 with the x_1 and x_2 axis give the points (-2,0),(0,2) ,(0,9.5),(2.375,0)
- Intersection of the lines x_2 - $x_1{=}$ 2 and $8x_2{+}$ $2x_1{=}$ 19 with each other gives (1.5,3.5).
- The feasible region points are (0,0) ,(0,2) ,(2.375,0),(1.5,3.5).
- Out of these points the point which gives the highest value of the objective function $x_1{+}$ x_2 is (1.5,3.5) and the value is 5.
- However, as (1.5,3.5) is not an integer solution.We branch on x_1 and divide the problem into two subproblems in which one subproblem whill have the added constraint $x_1 \leq \llcorner 1.5 \lrcorner = 1$and other will have the added constraint $x_1 \geq \ulcorner 1.5 \urcorner = 2$.

Subproblem 1 :

Maximize $x_1{+}$ x_2
subject to:
x_2 - $x_1{\leq}$ 2
$8x_2{+}$ $2x_1{\leq}$ 19
$x_1{\leq}$ 1
$x_1, x_2 \geq 0 \quad x_1 \in Z \; x_2 \in Z$

Solution:

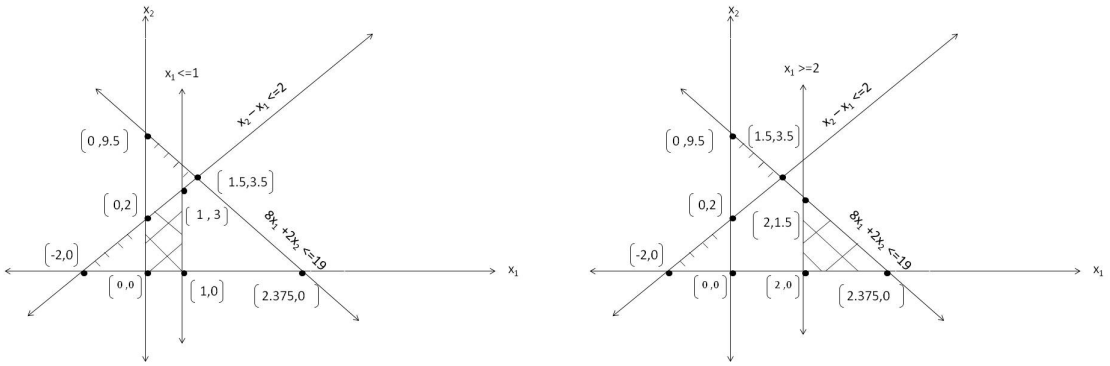


Figure 4.2: (a) Feasible region of the sub problem with the added constraint $x_1{\leq}1$. (b) Feasible region of the sub problem with the added constraint $x_1{\geq}2$

- We plot the graph by taking two axis $x_1(x_2{=}0)$ and $x_2(x_1{=}0)$.
- Intersection of the lines x_2 - $x_1{=}$ 2 and $x_1{=}$ 1 with the x_1 and x_2 axis give the points (-2,0),(0,2) ,(2,0),(1,0)
- Intersection of the lines x_2 - $x_1{=}$ 2 and $x_1{=}$ 1 with each other gives (1,3).
- The feasible region points are (0,0) ,(0,2) ,(1,0),(1,3).
- Out of these points the point which gives the highest value of the objective function $x_1{+}$ x_2 is (1,3) and the value is 4. This is an integer solution.
- As we have got an integer solution. There is no need to branch this further.

While we need to explore the other subproblem further.

Subproblem 2 :

Maximize $x_1 + x_2$
subject to:
x_2 - $x_1 \leq 2$
$8x_2 + 2x_1 \leq 19$
$x_1 \geq 2$
$x_1, x_2 \geq 0 \quad x_1 \in Z; x_2 \in Z$

Solution:

- We plot the graph by taking two axis $x_1(x_2=0)$ and $x_2(x_1=0)$.
- Intersection of the lines $x_1 = 2$ and $8x_2 + 2x_1 = 19$ with the x and y axis give the points (2,0) ,(0,9.5),(2.375,0)
- Intersection of the lines $x_1 = 2$ and $8x_2 + 2x_1 = 19$ with each other gives (2,1.5).
- The feasible region vertices are (2,0),(2.375,0),(2,1.5).
- Out of these points the point which gives the highest value of the objective function $x_1 + x_2$ is (2,1.5) and the value is 3.5 which is less than above integer value 4 obtained. Hence there is no need to branch further as we cannot get a solution which is more than 3.5. Hence we stop here.

From above two subproblems we see that the maximum integer solution obtained is (1,3) and the optima z^* is 4.

4.3.2 Example 2

Obtain an Integer solution to the following problem:
Maximize $2x_1 + x_2$
such that
$x_1 + x_2 \leq 8$
$2x_1 \leq 9$
$x_1, x_2 \geq 0 \quad x_1 \in Z;\ x_2 \in Z$

Solution:

- We plot the graph by taking two axis $x_1(x_2=0)$ and $x_2(x_1=0)$.
- Intersection of the lines $2x_1 = 9$ and $x_1 + x_2 = 8$ with the x_1 and x_2 axis give the points (4.5,0),(0,8) ,(8,0)
- Intersection of the lines $2x_1 = 9$ and $x_1 + x_2 = 8$ with each other gives (4.5,3.5).
- The vertices of the feasible region are (0,0) ,(4.5,0),(0,8) ,(4.5,3.5).
- Out of these points the point which gives the highest value of the objective function $x_1 + x_2$ is (4.5,3.5) and the value is 12.5.
- However, as (4.5,3.5) is not an integer solution.We branch on x_1 and divide the problem into two subproblems in which one subproblem whill have the added constraint $x_1 \leq \lfloor 4.5 \rfloor = 4$ and other will have the added constraint $x_1 \geq \lceil 4.5 \rceil = 5$.

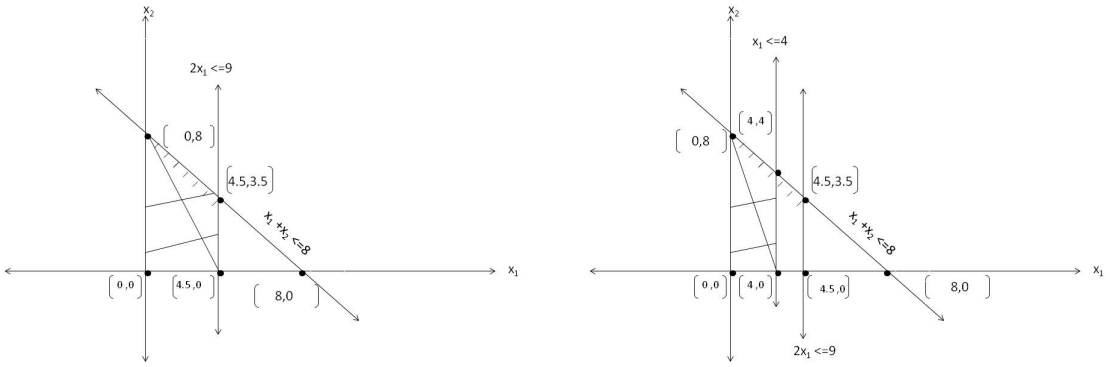


Figure 4.3: (a) Feasible region of the main problem (b) Feasible region of the sub problem with the added constraint $x_1 \leq 4$

Subproblem 1 :

Maximize $2x_1 + x_2$
subject to
$x_1 + x_2 \leq 8$
$2x_1 \leq 9$
$x_1 \geq 5$
$x_1, x_2 \geq 0 \quad x_1 \in Z \quad x_2 \in Z$

Solution:

There is no feasible region for above system of equations and hence there is no solution and no need to branch further.

Subproblem 2 :

Maximize $2x_1 + x_2$
subject to
$x_1 + x_2 \leq 8$
$2x_1 \leq 9$
$x_1 \leq 4$
$x_1, x_2 \geq 0 \quad x_1 \in Z \quad x_2 \in Z$

Solution:

- We plot the graph by taking two axis $x_1(x_2{=}0)$ and $x_2(x_1{=}0)$.
- Intersection of the lines $2x_1{=}\ 9$ and $x_1 \leq 4$ and $x_1 + x_2{=}8$ with the x_1 and x_2 axis give the points (4.5,0),(0,8) ,(8,0),(4,0)
- Intersection of the lines $2x_1{=}\ 9$ and $x_1 + x_2{=}8$ with each other gives (4.5,3.5).
- Intersection of the lines $x_1{=}\ 4$ and $x_1 + x_2{=}8$ with each other gives (4,4).
- The feasible region points are (0,0) ,(4,0),(0,8) ,(4,4).
- Out of these points the point which gives the highest value of the objective function $x_1{+}\ x_2$ is (4,4) and the value is 12.

- As we have got an Integer solution (4,4) which gives a value of 12. There is no need to branch further.So ans is (4,4).

4.3.3 Example 3

Let us use branch and bound algorithm to :

Maximize $5x_1 + 8x_2$
Such that
$x_1 + x_2 \leq 6$
$5x_1 + 9x_2 \leq 45$
$x_1, x_2 \geq 0 \quad x_1 \in Z \quad x_2 \in Z$

Solution:

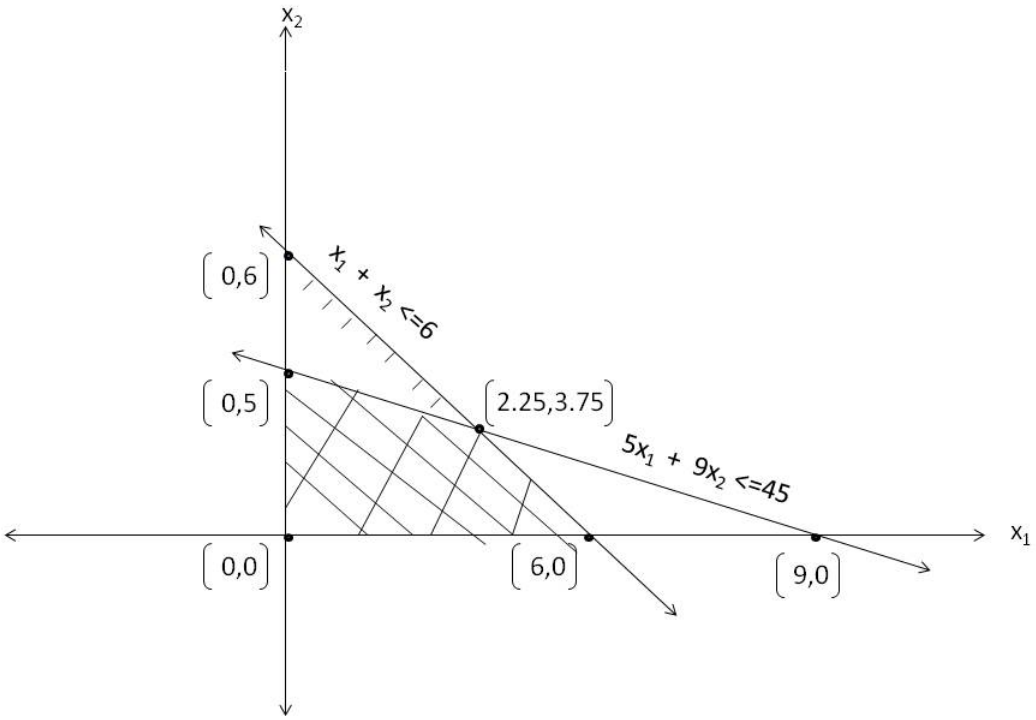


Figure 4.4: Feasible region of the main problem

- We plot the graph by taking two axis $x_1(x_2{=}0)$ and $x_2(x_1{=}0)$.
- Intersection of the lines $x_1 = x_2 = 6$ and $5x_1 + 9x_2 = 45$ with the x_1 and x_2 axis give the points (0,6) ,(0,5),(6,0),(9,0)
- Intersection of the lines $x_1 + x_2 = 6$ and $5x_1 + 9x_2 = 45$ with each other gives (2.25,3.75).
- The feasible region points are (0,0) ,(0,5) ,(6,0),(2.25,3.75).
- Out of these points the point which gives the highest value of the objective function $5x_1 + 8x_2$ is (2.25,3.75) and the value is 41.25 other values are 30 by (6,0) and 40 by (0,5) .

- But as (2.25,3.75) is not an integer solution.We branch on x_1 and divide the problem into two subproblems in which one subproblem whill have the added constraint $x_1 \leq \llcorner 2.2.5 \lrcorner = 2$ and other will have the added constraint $x_1 \geq \ulcorner 2.25 \urcorner = 3$.

Subproblem 1:

Maximize $5x_1 + 8x_2$
Subject to
$x_1 + x_2 \leq 6$
$5x_1 + 9x_2 \leq 45$
$x_1 \leq 2$
$x_1, x_2 \geq 0 \quad x_1 \in Z \quad x_2 \in Z$

Solution:

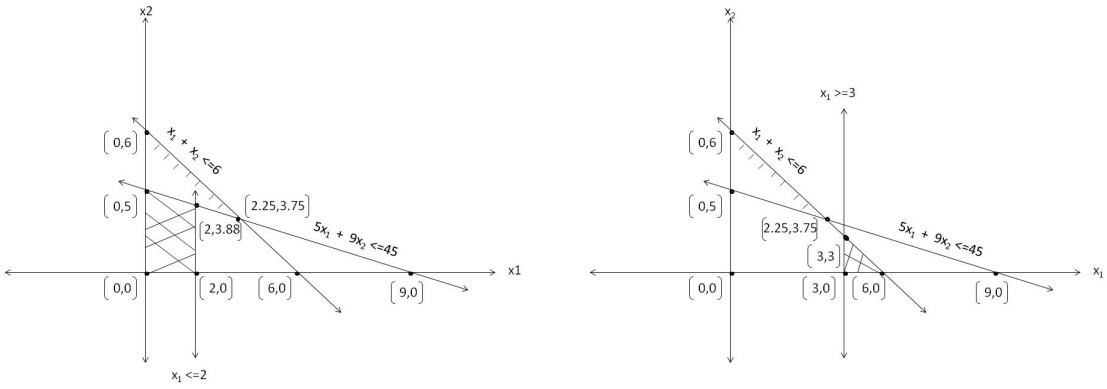


Figure 4.5: Feasible region of the sub problem with the added constraint $x_1 \leq 2$ (b) Feasible region of the sub problem with the added constraint $x_1 \geq 3$

- We plot the graph by taking two axis $x_1(x_2=0)$ and $x_2(x_1=0)$.
- Intersection of the lines $x_1 = 2$ and $5x_1 + 9x_2 = 45$ with the x_1 and x_2 axis give the points (0,5),(2,0),(9,0)
- Intersection of the lines $x_1 = 2$ and $5x_1 + 9x_2 = 45$ with each other gives (2,3.88).
- The feasible region points are (0,0) ,(0,5) ,(2,0),(2,3.88).
- Out of these points the point which gives the highest value of the objective function $5x_1 + 8x_2$ is (2,3.88) and the value is 41.1 other values are 10 by (2,0) and 40 by (0,5) .
- Let us see subproblem 2 now.

Subproblem 2:

Max $5x_1 + 8x_2$
such that
$x_1 + x_2 \leq 6$
$5x_1 + 9x_2 \leq 45$
$x_1 \geq 3$
$x_1, x_2 \geq 0 \quad x_1 \in Z \quad x_2 \in Z$

Solution:

- We plot the graph by taking two axis $x_1(x_2{=}0)$ and $x_2(x_1{=}0)$.
- Intersection of the lines $x_1{=}$ 3 and $x_1 + x_2{\leq}$ 6 with the x_1 and x_2 axis give the points (6,0),(3,0),(0,6)
- Intersection of the lines $x_1{=}$ 3 and $x_1 + x_2{\leq}$ 6 with each other gives (3,3).
- The feasible region points are (6,0) ,(3,0) ,(3,3).
- Out of these points the point which gives the highest value of the objective function $5x_1{+}$ $8x_2$ is (3,3) and the value is 39 .
- As We have already have an Integer solution of 40 from (0,5). We dont need to branch this problem further.

  We will now branch further into subproblem 1 We branch on x_2 and divide the subproblem 1 into two subproblems in which one subproblem whill have the added constraint $x_2 \leq \llcorner 3.88 \lrcorner = 3$and other will have the added constraint $x_2 \geq \ulcorner 3.88 \urcorner = 4$.

Subproblem 1(a) :

Maximize $5x_1 + 8x_2$
siubject to:
$x_1 + x_2{\leq} 6$
$5x_1{+}$ $9x_2{\leq}$ 45
$x_1{\leq}$ 2
$x_2{\leq}$ 3
$x_1, x_2 \geq 0 \quad x_1 \in$Z $\;\; x_2 \in$Z

Solution:

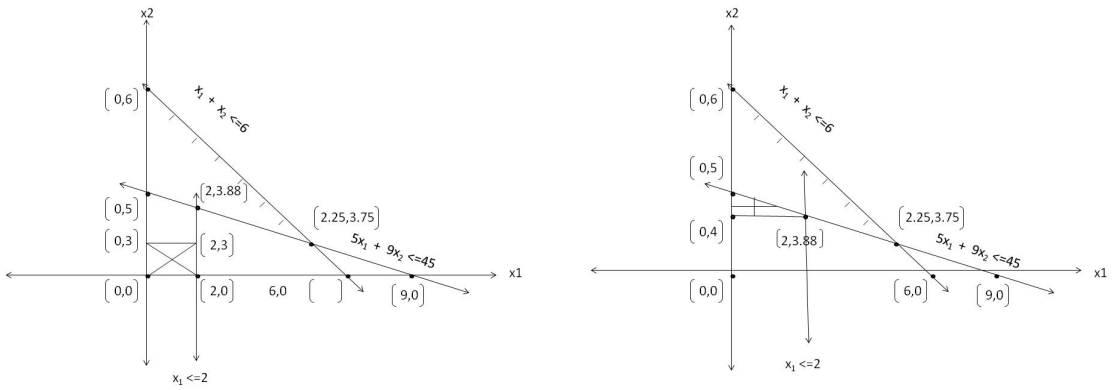


Figure 4.6: Feasible region of the sub problem with the added constraint $x_2{\leq}3$ (b) Feasible region of the sub problem with the added constraint $x_2{\geq}4$

- We plot the graph by taking two axis $x_1(x_2{=}0)$ and $x_2(x_1{=}0)$.
- Intersection of the lines $x_1{=}$ 2 and $x_2{=}$ 3 and $5x_1{+}$ $9x_2{=}$ 45 with the x_1 and x_2 axis give the points (0,5),(2,0),(9,0)
- Intersection of the lines $x_1 =$ 2 and $x_2{=}$ 3 with each other gives (2,3).
- The feasible region points are (0,0) ,(0,3) ,(2,0),(2,3).
- Out of these points the point which gives the highest value of the objective function $5x_1{+}$ $8x_2$ is (2,3) and the value is 34 other values are 10 by (2,0) and 24 by (0,3) .
- As this is less than before seen feasible solution.There is no need to branch this problem further.

Subproblem 1(b) :

Maximize $5x_1 + 8x_2$
Subject to
$x_1 + x_2 \leq 6$
$5x_1 + 9x_2 \leq 45$
$x_1 \leq 2$
$x_2 \geq 4$
$x_1, x_2 \geq 0 \quad x_1 \in Z \quad x_2 \in Z$

Solution:

- We plot the graph by taking two axis $x_1(x_2=0)$ and $x_2(x_1=0)$.
- Intersection of the lines $x_1 = 2$ and $x_2 = 4$ and $5x_1 + 9x_2 = 45$ with the x_1 and x_2 axis give the points (0,4),(2,0),(9,0)(0,5)
- Intersection of the lines $x_2 = 4$ and $5x_1 + 9x_2 = 45$ with each other gives (1.8,4).
- The feasible region points are (0,4) ,(0,5) ,(1.8,4).
- Out of these points the point which gives the highest value of the objective function $5x_1 + 8x_2$ is (1.8,4) and the value is 41.
- But as (1.8,4) is not an integer solution.We branch on x_1 and divide the problem into two subproblems in which one subproblem whill have the added constraint $x_1 \leq \llcorner 1.8 \lrcorner = 1$and other will have the added constraint $x_1 \geq \ulcorner 1.8 \urcorner = 2$.

Subproblem 1(b)(i) :

Maximize $5x_1 + 8x_2$
Subject to
$x_1 + x_2 \leq 6$
$5x_1 + 9x_2 \leq 45$
$x_1 \leq 2$
$x_1 \geq 2$
$x_2 \geq 4$
$x_1, x_2 \geq 0 \quad x_1 \epsilon Z \quad x_2 \epsilon Z$

Solution:

- There will be only one point (2,3.8) which is not an IP solution. There os no need to branch further.

Subproblem 1(b)(ii) :

Max $5x_1 + 8x_2$
such that
$x_1 + x_2 \leq 6$
$5x_1 + 9x_2 \leq 45$
$x_1 \leq 2$
$x_1 \leq 1$
$x_2 \geq 4$
$x_1, x_2 \geq 0 \quad x_1 \in Z \quad x_2 \in Z$

Solution:

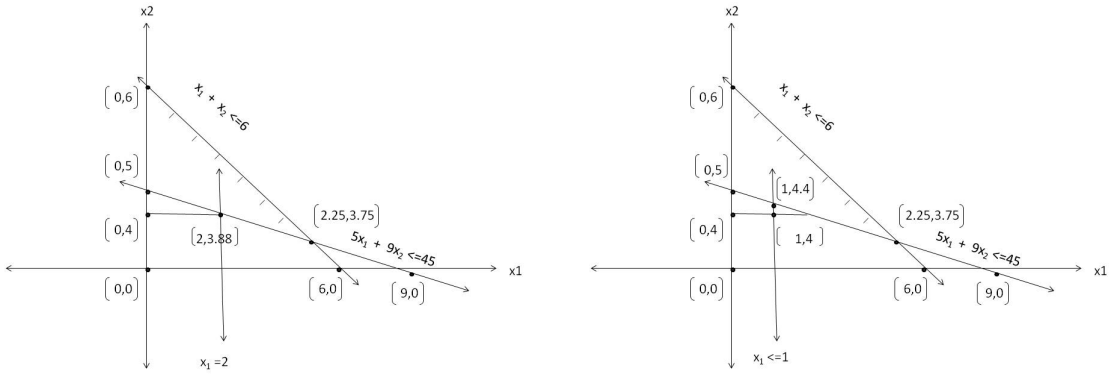


Figure 4.7: (a) Feasible region of the sub problem with the added constraint $x_1 \geq 2$.Only one point is in the region (b) Feasible region of the sub problem with the added constraint $x_1 \leq 1$.

- The feasible points in this region are (1,4) , (0,4) ,(0,5) and (1,4.44) out of which highest is (1,4.44) and the value is 40.552. But Since we already have a solution of 40 .We can stop the branching here.

  From Above two subproblems we see that the maximum integer solution obtained is (0,5) and the solution is 40.

Chapter 5

LP Relaxation

- We had seen some LP and IP formulations in the past. We will see some more as we move forward.
- We know how to solve toy problems on pen and paper.
- LP relaxation is a powerful way to solve IPs.
- Beyond obtaining a solution, this line of thinking leads to many insights about the problem, properties of the problem, guarantees of an approximate solution etc.

5.1 LP Relaxation

If we are interested in designing a polynomial time algorithm(exact or approximate) for a combinatorial optimiation problem, formulating the combinatorial optimization problem as an IP is useful as a first step in the following methodology(the discussion assumes that we are working with minimization problem):

1. Formulate the combinatorial optimiation problem as an IP.
2. Derive a LP from the IP by removing the constraint that the variables have to take integer value. The resulting LP is called a "relaxation" of the original problem. Note that in the LP we are minimizing the same objective function over a larger set of solutions, so opt(LP) $\leq$opt(IP). In other words, z^*_{IP} is inferior to z^*_{LP}
3. Solve the LP optimally using an efficient algorithm for linear programming.

- If the optimal LP solution has integer values, then it is a solution for the IP of cost opt(LP)=opt(IP). We have now found an optimal solution for the IP and hence an optimal solution for our combinatorial optimization problem.
- If the optimal LP solution x* has fractional values, but we have a rounding procedure that transforms x* into an integral solution x' such that cost(x')$\leq$c . cost(x*) for some constant c, then we are able to find a solution to the IP of cost$\leq$c . opt(LP) $\leq$c . opt(IP), and so we have a c-approimate algorithm for our combinatorial optimization problem.

5.2 Bipartite Matching

Definition

When there are an equal number of nodes on each side of a bipartite graph, a perfect matching is an assignment of nodes on the left to nodes on the right, in such a way that

1. each node is connected by an edge to the node it is assigned to, and
2. no two nodes on the left are assigned to the same node on the right

Formulation

Let G=(V,E) be a graph where V represents the set of vertices and E represents the set of edges. As G is bipartite, let V be divided into two disjoint sets X and Y such that $\mid X \mid = \mid Y \mid$and edges are from X$\rightarrow$Y only. A perfect matching M$\subset$E is such that each vertex in X as well as Y appears only once in M.

Our problem of interest is to find the maximum weight matching. i.e., sum of weights on the edges in the match M is maximum., i.e., Max $\sum_{e \in M} w_e$

Note: This many similar problems can be seen as "selection" of a set of objects (like edges of a graph). It is common to introduce a binary variable $\in \{0, 1\}$ that defines whether the object is selected or not.

The IP formulation of the problem is as follows:

Let $x_e \in \{0, 1\}$ to say whether edge is in M or not.

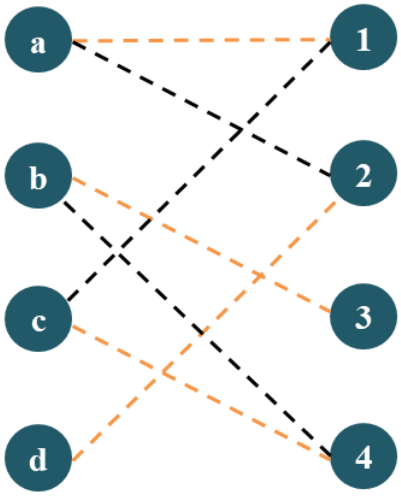


Figure 5.1: Bipartite Matching

$$Maximize \sum_{e \in E} w_e \cdot x_e$$

subject to;

$$\sum_{e \in E; v \in e} x_e = 1 \text{ for all v} \in V$$

$$x_e \in \{0, 1\} \quad \forall e \in E$$

This is an IP. Let us relax the $x_e \in \{0, 1\}$ to $x_e \in [0, 1]$ to obtain an LP by relaxing the IP.

$$Maximize \sum_{e \in E} w_e \cdot x_e$$

subject to;

$$\sum_{e \in E; v \in e} x_e = 1 \text{ for all v} \in V$$

$$x_e \in [0, 1] \quad \forall e \in E$$

Now, after relaxation, we obtain an IP as follows: Maximize $\Sigma x_e w_e$

such that $\Sigma x_e = 1 \; \forall \; v \in V$

$0 \leq x_e \leq 1 \; \forall e \in E$

While solving LP, we conclude/observe:

1. Infeasibility in LP implies infeasible IP.
2. LP provides an upper bound on the IP. (note: maximization problelm).
3. If LP itself gives an integer solution, a rare good problem.

Unfortunately, this LP need not give an integer solution. However, the situation is not bad.

Rounding

We make an argument below that helps us in designing a "loss-less" rounding scheme.

Assume one of the edge (a_1, b_1) has $0 < x_e < 1$. Then there exists another edge (b_1, a_2) which is also $0 < x_e < 1$. We can then get a chain $a_1 b_1\ a_2\ b_2$. . . . . a_1 of unsatisfactory edges such that all have $0 \leq x_e \leq 1$.

Note that number of edges will be even e_1. . . . . e_t .

Consider now modifying the x_e in this chain (cycle) such that

$$y_e = \begin{cases} x_e^* - \epsilon & \text{for } e \in \{e_1, e_3, \ldots, e_{t-1}\} \\ x_e^* + \epsilon & \text{for } e \in \{e_2, e_4, \ldots, e_t\} \\ x_e^* & \text{otherwise} \end{cases}$$

Easy to see that $\sum y_e = 1$ and y_e is a valid / feasible solution for small ϵ.

Let us now look at the cost of matching.

$$W(Y) = \sum_e w_e y_e = W(X^*) + \epsilon \sum (-1)^i w_{ei} = w(X^*) + \epsilon \triangle$$

Since X^* is optimal, $\triangle = 0$ (otherwise we would have found an ϵ such that $W(Y) > W(X^*)$.) If $\triangle$ is negative, then we would have taken a negative ϵ.

What does it mean? This change in x_e is not going to change the matching cost. This helps is finding a rounding scheme that does not change the cost.

1. Find largest ϵ such that y is still feasible
2. Then y will have lesser non - integer values. Repeat for many chains until you set an integer solution.
3. We can repeat this until we get all binary values.

Result If LPR has a feasible solution, then it has one integer optimal solution. We can obtain by an appropriate rounding.

5.3 Minimum Vertex Cover

Let us now consider another problem that of finding a minimal vertex set that cover all the edges.

Definition Formally, a vertex-cover of an undirected graph $G = (V, E)$ is a subset V' of V such that if edge (u, v) is an edge of G then either u in V' or v in V' (or both). The set V' is said to cover the edges of G.
A minimum vertex cover is a vertex cover of smallest possible size. Finding it is NP-Hard.

IP formulation Here we need to select vertices. Let us define a new binary variable x_v to denote whether a vertex is selected or not.

$$\begin{array}{ll} \text{Minimize} & \sum_{v \in V} x_v \\ \text{Such that} & x_u + x_v \geq 1 \ \forall (u, v) \in E \\ & x_u \in 0, 1 \end{array} \tag{5.1}$$

Indeed, we can now relax the integral constraints and create an LP.

LP Relaxation

$$\begin{array}{ll} \text{Minimize} & \sum_{v \in V} x_v \\ \text{Such that} & x_u + x_v \geq 1 \ \forall (u,v) \in E \\ & 0 \leq x_u \leq 1 \end{array} \tag{5.2}$$

Solution using LPR Solve the LPR using an appropriate LP solver to obtain the optimal solution. Let it be x^*. This need not be an integral solution.

Now we create the following vertex cover S_{LP} from x^*.

$$S_{LP} = \{v \in V | x_v \geq \frac{1}{2}\} \tag{5.3}$$

Analyzing the above equation carefully we will see that it is a vertex cover. (Why?.

Now we want to check how good/bad is this cover from the minimum.

Let, y be the optimal solution by IP and let S_{OPT} be the corresponding vertex cover. We have,

$$|S_{LP}| \geq |S_{OPT}| \tag{5.4}$$

Why? Remember that the IP optima is inferior to the LP optima. We can also conclude the following from LP-IP relationships. Now we want to see exactly how bad is our solution.

$$\sum_{v \in V} x_v^* \leq \sum_{v \in V} y_v \tag{5.5}$$

Now we have:

$$|S_{LP}| = \sum_{v \in S_{LP}} 1 \leq \sum_{v \in S_{LP}} 2.x_v^* \leq \sum_{v \in V} 2.x_v^* \leq 2 \sum_{v \in V} y_v \leq 2|S_{OPT}| \tag{5.6}$$

So, we can conclude that :

$$|S_{OPT}| \leq |S_{LP}| \leq 2|S_{OPT}| \tag{5.7}$$

5.4 Facilty Location problem

A company wants to set up factories at some of the i locations so that it can supply materials to all its customers in j places. The cost of setting up the factory at location i is f_i. Let $x_i \in \{0, 1\}$ indicate whether factory is set up or not. Therefore the cots of setting up the factories is $C_f = \sum_i f_i x_i$.

Now the goods need to be transferred from the factories (that are set up) to the customers. Let $c(i, j)$ be the cost of transportation from factory i to customer j. Let $y_{ij} = \{0, 1\}$ indicate whether customer j is assigned to factory i. Total transportation cost is $C_r = \sum_{ij} y_{ij} c(i, j)$. Final problem is now:

$$minimize \sum_i f_i x_i + \sum_{ij} c(i,j) y_{ij}$$

subject to

$$\sum_i y_{ij} \geq 1 \ \ \forall j$$

$$x_i \geq y_{ij}$$
$$x_i \in \{0,1\} \quad y_{ij} \in \{0,1\}$$

The first constraint says that each customer should be assigned to at least one facility. The second says that is a customer is assigned to a facility, then that facility must be open. This is an IP. We relax the last constraint as $x_i \in [0,1]$ and $y_{ij} \in [0,1]$. This leads to an LP.

Question: Suggest an appropriate rounding scheme for the above problem. Also can you come up with a bound on the approximation errors?

Hint:

`http://pages.cs.wisc.edu/~shuchi/courses/787-F09/scribe-notes/lec10.pdf`

5.5 Maximum Independent Set

Let us look at another vertex selection problem.

Definition Independent set : Let there be a graph (G,V). $s \in V$, such that no two vertices in s are connected by an edge in G is called an independent set. Our goal is to maximize the number of elements in s.

IP formulation

$$\begin{array}{ll} \text{Maximize} & \sum_{v \in V} x_v \\ \text{Such that} & x_u + x_v \leq 1 \; where \; (u,v) \in E \\ & x_u \in 0,1 \end{array} \tag{5.8}$$

LP Relaxation

$$\begin{array}{ll} \text{Maximize} & \sum_{v \in V} x_v \\ \text{Such that} & x_u + s_v \leq 1 \; where \; (u,v) \in E \\ & 0 \leq x_u \leq 1 \end{array} \tag{5.9}$$

Analysis We, can see the if $LP = \frac{1}{2}$, then all the above constraints are satisfied. Also the objective is then $\frac{|V|}{2}$. Therefore, the optimal value is $\frac{|V|}{2}$ or larger.

So, the LP has a feasible solution. Now it can be clearly seen that.

$$LP^* \geq \frac{|V|}{2} \tag{5.10}$$

Consider a fully connected graph. The maximal independent set is a single node. However the LP^* is $\frac{n}{2}$. This does not allow us to come up with a guarantee on the solution.

As, we can see unlike the the minimum vertex cover problem, there are no good bounds here. So, LP relaxation tells nothing about Maximum Independent Set.

5.6 Reading

Strongly urge to read chapter 3 of [B6].

Chapter 6

More on IP Formulations

- We had seen seen how many of the classical problems (eg. graph algorithms) get formulated as an IP.
- We had see how to solve IP by (i) branch and bound (ii) LP relaxation.
- We had also seen how approximate algorithms can be designed using LP relaxation.
- Here, we will see how many problems will become a natural IP problem.

6.1 BIP and MIP Formulations

We had seen how many problems get formulated as LP, and some others as IP. We now see two special class of problems.

- BIP: Binary integer programming problems. In this class the decision variables are binary i.e., $x_i \in \{0, 1\}$
- MIP: Mixed integer programming problems. In this class, some of the decision variables are real and some others integer.

In binary problems, each variable can only take on the value of 0 or 1. This may represent the selection or rejection of an option, the turning on or off of switches, a yes/no answer, or many other situations.

A mixed integer programming (MIP) problem results when some of the variables in your model are real valued (can take on fractional values) and some of the variables are integer valued. The model is therefore "mixed". When the objective function and constraints are all linear in form, then it is a mixed integer linear program (MILP). In common parlance, MIP is often taken to mean MILP, though mixed integer nonlinear programs (MINLP) also occur, and are much harder to solve.

6.1.1 Example Formulations

In the next section, we will see a variety of problems getting modelled as IP. Mostly BIP.

6.2 Function of K Discrete Variables

Consider a problem where $x_1 + x_2 + x_3$ will have to be 5 or 10 or 20 depending on something else (say what mode of transportation or what is the cost of raw material). We encode this as

$$x_1 + x_2 + x_3 = 5y_1 + 10y_2 + 20y_3$$

with y_i as binary variables and only one of the y_i can be 1 at a time. This last constraint is added as.

$$y_1 + y_2 + y_3 = 1$$

6.3 Either-OR constraints

This class of constraints arise when we need to enforce only one of the constraints. Assume we had a constraint of the form $|x_1| \geq 3$. This results two disconnected sets (non-convex region/set.). This only means that either $x_1 \geq 3$ or $x_1 \leq -3$. If we convert both as $\leq$,

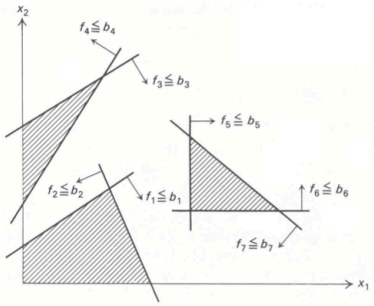
$$x_1 \leq -3$$

$$-x_1 \leq -3$$

Both can not be true at the same time. How do we do this? The idea is simple. When a constraint is getting violated, we make it true "trivially". Let L be a large quantity. (for the sake of argument, it can be the maximum value your integer/real variable can store. But in practice, it is a reasonable value to manage numerically.) We can write this constraints as either

$$x_1 \leq -3$$

$$-x_1 \leq -3 + L$$



or

$$x_1 \leq -3 + L$$
$$-x_1 \leq -3$$

Note that the addition of the large quantity makes the constraint trivially true. In the first set, only the first constraint is active. While in the second, only the second constraint is active. However, we do not know which constraint needs to be made true. For this purpose, we add an additional integer variable y. i.e.,

$$x_1 \leq -3 + y \cdot L$$
$$-x_1 \leq -3 + (1 - y) \cdot L$$
$$y \in \{0, 1\}$$

6.4 K out of N constraints

An extension of the same idea to K out of N constraints to be true is as follows.

$$f_1(\mathbf{x}) \leq b_1 + y_1 L$$
$$f_2(\mathbf{x}) \leq b_2 + y_2 L$$
$$\cdots\cdots\cdots$$
$$f_N(\mathbf{x}) \leq b_N + y_N L$$
$$\sum_i y_i = N - K$$

This final constraint can be interpreted as follows. Since we want K constraints to hold out of N, there must be $N - K$ constraints that do not hold. So this constraint insures that $N - K$ of the binary variables take the value 1 so that associated M values are turned on, thereby eliminating the constraint.

6.5 Modelling Compond Alternatives

Consider the problem of allowing three disjoint regions in the constraints as shown in the figure.

This can be done using the ideas discussed above.
Region 1:

$$f_1(x_1, x_2) - Ly_1 \leq b_1$$
$$f_2(x_1, x_2) - Ly_1 \leq b_2$$

Regioon 2:

$$f_3(x_1, x_2) - Ly_2 \leq b_3$$

$$f_4(x_1, x_2) - Ly_2 \leq b_4$$

Region 3:

$$f_5(x_1, x_2) - Ly_3 \leq b_5$$

$$f_6(x_1, x_2) - Ly_3 \leq b_6$$

$$f_7(x_1, x_2) - Ly_3 \leq b_7$$

with additional constraints such as

$$y_1 + y_2 + y_3 \leq 2$$

$$x_1 \geq 0, x_2 \geq 0$$

$$y_i \in \{0, 1\}$$

6.5.1 Problems with a fixed cost and variable cost

Most of the problems that we had seen are with a variable cost (a linear function is the objective.) If x_i increase, our objective $c_i x_i$ also increases.

Consider a class of problems, where there is a fixed cost for a variable. For example, when you sign up an agreement, you pay an initial amount and then you pay a fixed cost for the regular use of x_i (say the bandwidth). The initial cost may be associated with the setting up of a specific hardware unit or cabling so that an internet connectivity can be provided.

Therefore the objective is 0 if $x_1 = 0$ and $K + c_1 x_1$ if $x_1 > 0$. Here K is the fixed charge.

This can be modelled as

$$\text{Minimize} Ky + c_1 x_1 + \text{rest of the terms}$$

subject to:

$$x_1 - Ly \leq 0$$

$$\text{other constraints}$$

$$y \in \{0, 1\}$$

6.6 Modelling Piecewise Linear Cost

Consider the cost of production of certain material. This can be $c_1 x$ until the production reaches 5 units. Then the cost could be different say $c_2 x$ until the production reaches 12. And then $c_3 x$ until 20. etc. Note that when the production is 7, cost is $5c_1 + 2c_2$. Here the objective is piecewise linear.

Let d_1, d_2 and d_3 are the amount of production in the three cost ranges. When d_2 is nonzero, d_1 is 5. When d_3 is nonzero, d_1 is 5 and d_2 is 7.

$$x = d_1 + d_2 + d_3$$

Cost is $c_1 d_1 + c_2 d_2 + c_3 d_3$. By adding two additional binary variables, the constraints are:

$$5w_1 \leq d_1 \leq 5$$

$$7w_2 \leq d_2 \leq 7w_1$$

$$0 \leq d_3 \leq 8w_2$$

When $w_1 = 1$, d_1 is at the upper bound i.e., 5. When $w_2 = 1$, d_2 is at the upper bound i.e., 7.

6.7 Solving BIP using Bala's Algorithm

We had seen how branch and bound can be used to solve IP. Let us see another algorithm specially designed to solve BIPs — Bala's additive algorithm. It requires the problem to be in a standard form.

- The objective function has the form. Minimize $Z = \sum_i c_i x_i$ with x_i as binary variables.
- The m constraints are of the form $\sum a_{ij} x_j \geq b_i$ for $i = 1, \ldots m$
- c_is are non-negative. Also $0 \leq c_1 \leq c_2 \leq \ldots \leq c_n$

This may seem to be restrictive at first look. But not that bad. For example, negative coefficients are converted by changing x_i to $1 - x_i$.

The objective function is minimization. And c_is are positive. Also $x_i \in \{0, 1\}$. Therefore if all $x_i = 0$ is feasible, it will be the minima.

If there are N variables, we need to evaluate 2^N possible configuration of the $\mathbf{x}$. Bala's algorithm uses as depth first search approach.

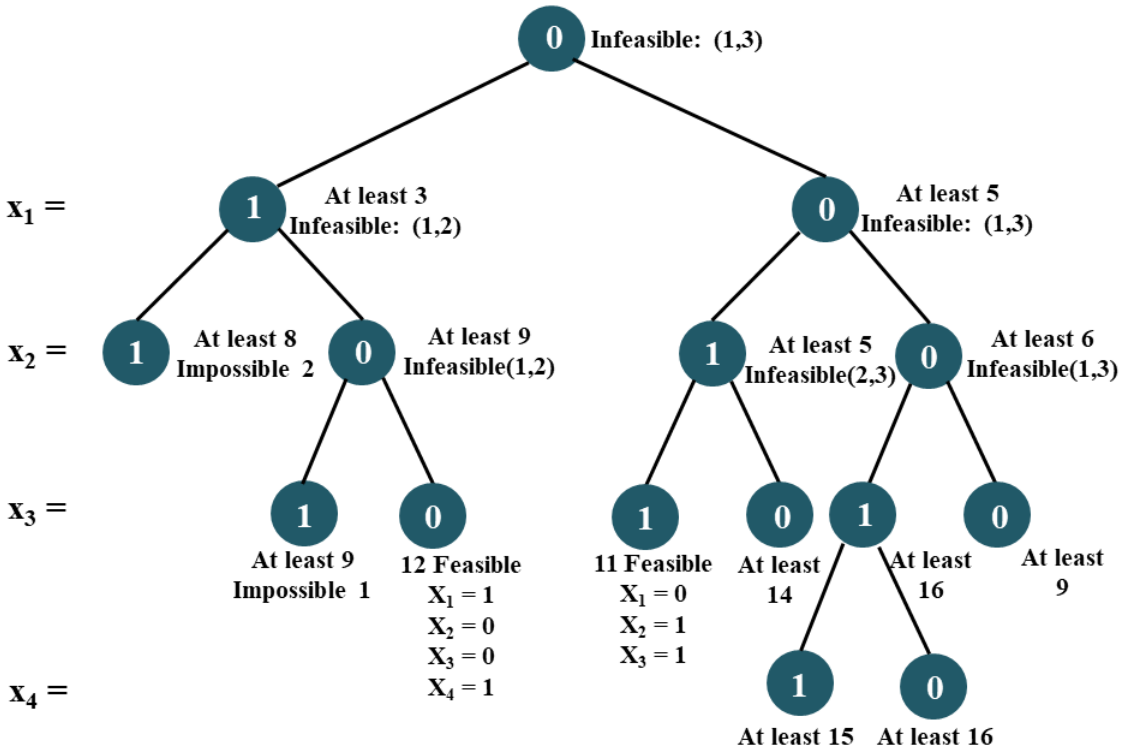
Bala's algorithm starts expanding tree with x_1 and then move to x_2 etc. At each step, it evaluates the objective (or come with a bound) and checks whether any of the constraints are violated or not. When we find a feasible point, we stop expanding further that tree. Also the bound that we obtain for certain nodes will tell us whether to expand a node further or no.

6.7.1 Example Problem

$$\text{Minimize } z = 3x_1 + 5x_2 + 6x_3 + 9x_4 + 10x_5 + 10x_6$$

subject to:

$$-2x_1 + 6x_2 - 3x_3 + 4x_4 + x_5 - 2x_6 \geq 2$$
$$-5x_1 - 3x_2 + x_3 + 3x_4 - 2x_5 + x_6 \geq -2$$
$$5x_1 - x_2 + 4x_3 - 2x_4 + 2x_5 - x_6 \geq 3$$
$$x_i \in \{0, 1\}$$


0
Infeasible: (1,3)
x1 =
1
At least 3
Infeasible: (1,2)
0
At least 5
Infeasible: (1,3)
x2 =
1
At least 8
Impossible 2
0
At least 9
Infeasible(1,2)
1
At least 5
Infeasible(2,3)
0
At least 6
Infeasible(1,3)
x3 =
1
At least 9
Impossible 1
0
12 Feasible
X1 = 1
X2 = 0
X3 = 0
X4 = 1
1
11 Feasible
X1 = 0
X2 = 1
X3 = 1
0
At least
14
1
At least
16
0
At least
9
x4 =
1
At least 15
0
At least 16

Chapter 7

More on LP Relaxation

- We had seen seen how many of the classical problems (eg. graph algorithms) get formulated as an IP.
- We had see how to solve IP by (i) branch and bound (ii) LP relaxation. (ii) Bala's Algorithm
- We had also seen how approximate algorithms can be designed using LP relaxation. We have here one more example.

7.1 Scheduling for Unrelated Parallel Machines

Scheduling is the allocation of shared resources over time to competing activities. It has been the subject of a signiïficant amount of literature in the operations research field. Emphasis has been on investigating machine scheduling problems where jobs represent activities and machines represent resources. Each machine can process at most one job at a time.

In this lecture we consider the MAKESPAN SCHEDULING problem. We are interested in minimizing the time required to complete all the jobs. That is, the time for the last machine to complete the last job should be minimized. There are n jobs, indexed by the set $J = \{1, ..., n\}$ and m machines for scheduling, indexed by the set $M = \{1, ..., m\}$. Also, given, d_{ij} is the time job j takes to run on machine i. Then $t_i = \sum_{j \in J} x_{ij} d_{ij}$ is the *completion time* of machine i, where x_{ij} is 1 when job j is assigned to machine i, otherwise 0. The task is to assign jobs to machines so that the $t = \max_i(t_i)$, i.e., the maximum job completion time, also called the *makespan* of the schedule, is minimized. Note that, the order in which the jobs are processed on a particular machine does not matter.

There are many variants of the problem. Some of them are :

1. Minimum makespan scheduling on identical machines: $d_{ij} = d_j \ \ \forall i$.
2. Minimum makespan scheduling on unrelated machines: each machine can take different time to do a job. This is the problem of interest to us.
3. Non splittable jobs: $x_{ij} \in \{0, 1\}$ and splittable jobs: $x_{ij} \in [0, 1]$.
4. Jobs have precedence constraints.

Minimum makespan scheduling for spilltable jobs can be solved exactly in polynomial time (Problem is in **P** and can be formulated as LP). Here we discuss only minimum makespan scheduling on unrelated machines for non-splittable jobs (Problem is in **NP-hard**). For identical machines, you can use references.

7.2 Minimum makespan scheduling on unrelated machines

Formulating the problem as Integer Program

$$\begin{aligned} \textbf{minimize} \quad & t \\ \textbf{subject to} \quad & \\ & \sum_{i=1}^{m} x_{ij} = 1 \ \ \forall j \in J \\ & \sum_{j=1}^{n} x_{ij} d_{ij} \leq t \ \ \forall i \in M \\ & x_{ij} \in \{0, 1\} \end{aligned}$$

- Number of variables: $nm + 1$
- Number of constraints: $n + m$

Some Observations on LP . Let us make some observations on LP. We can come back to these in detail at a later stage.

- A problem of the form Maximize $c^T x$ subject to $\{Ax \leq b; x \geq 0\}$ can be converted to Maximize $c^T x$ subject to $\{Ax = b; x \geq 0\}$ by adding slack variables. Positive slack variables are added to each inequality and coverted them to equality.

- Consider a system of equations $Ax = b$ where A is $m \times n$. If $m \leq n$, there could be multiple solutions. If x_1 is a solution and x_2 is a solution $\alpha x_1 + (1 - \alpha)x_2$ is also a valid solution.
- Let B be a $m \times m$ submatrix of A by selecting m columns. Let the x vector with the selected columns be x_B. Then $Bx_B = b$. Assume we solve this. x_B may be a vector with no non-zero elements. Cosnider the original vector with elements from x_B and all other elements x_i equal to zero. $x_i = 0$ if $i \notin B$
- This vector is a basic feasible solution. (Vertex of the convex polygon of our interest). At least one of the basic feasible solutions is optimal.
- In x, at least $n - m$ variables/elements are zero.

Note: We know the importance of solving $Ax = b$. This will be the focus on the next few lectures.

7.2.1 LP-relaxation algorithm for 2 machines

Using LP Relaxation

- There are two machines, m_1 and m_2.
- There are $2n + 1$ variables and $n + 2$ constraints.
- How many zeros are there in $\{x_{ij} | 1 \leq j \leq n, i = 1 \text{ or } 2\}$?
  Answer: $(2n + 1) - (n + 2) = n - 1$ zeros and, $n + 2$ non-zeros.

We know $t > 0$, therefore number of non-zeros in $\{x_{ij} | 1 \leq j \leq n, i = 1 \text{ or } 2\}$ is $n + 1$. This means only one job need to be set splittable across 2 machines such that $x_{1j} + x_{2j} = 1$. Let s be the job that gets split. We assign s to m_1 if $x_{1s} > x_{2s}$ otherwise, assign to m_2 (Figure 7.2 shows an example). If T^* be IP optimal makespan and T_{approx} be LP relaxation makespan, then $T_{approx} \leq 2T^*$.

Theorem 7.2.1. LP relaxation used here is 2-*approximation* for makespan scheduling for 2 machines.

PROOF. Let T^* be IP optimal makespan, T_{approx} be LP relaxation makespan, T_s be makespan of s and $T^{'}$ be optimal makespan before assigning s.

$$T_{approx} \leq T^{'} + T_s \leq T^* + T^*$$
$$\text{Since, } T^{'} < T^* \text{ and } T_s < T^*$$
$$T_{approx} \leq 2T^*$$

□

7.2.2 LP-relaxation algorithm for minimum makespan scheduling on unrelated machines

Here we consider makespan scheduling on unrelated machines for non-splittable jobs, which means that job j takes time d_{ij} if scheduled on machine i. Firstly, we define an IP to solve the problem, The algorithm is based on a suitable LP-formulation and a procedure for rounding the LP. The LP formulation for minimum makespan scheduling on unrelated machines can be written as

$$\begin{aligned}
&\textbf{minimize} && t \\
&\textbf{subject to} && \\
& && \sum_{i=1}^{m} x_{ij} = 1 \ \ \forall j \\
& && \sum_{j=1}^{n} x_{ij} d_{ij} \leq t \ \ \forall i \\
& && x_{ij} \in \{0, 1\}
\end{aligned}$$

If we relax the constraints $x_{ij} \in \{0, 1\}$, it turns out that this formulation has unbounded integrality gap. The main cause of the problem is an unfair advantage of the LP-relaxation.

Example Suppose we have only one job, which has a processing time of m on each of the m machines. Clearly, the minimum makespan is m. However, the optimal solution to the linear relaxation is to schedule the job to the extent of $1/m$ on each machine, thereby leading to an objective function value of 1, and giving an integrality gap of m.

If $d_{ij} > t$ for an arbitrary t, then we must have $x_{ij} = 0$ in any feasible integer solution due to $\sum_{j=1}^{n} x_{ij} d_{ij} \leq t \ \ \forall i$ constraint. But we might have fractional $x_{ij} > 0$ in feasible fractional solutions for LP relaxed problem. However, we can not formulate the statement "if $d_{ij} > t$ then $x_{ij} = 0$" in terms of linear constraints. The question arises here is therefore, how to correctly choose t?

Parametric Pruning

We will make use of a technique called parametric pruning to overcome this difficulty. We "guess" the parameter t which is a lower bound for the actual makespan T^*. One way to obtain suitable value is to do binary search on t. Note that since we already know t we don't need to check whether $d_{ij} > t$ or not. Therefore, we are now able to enforce constraints $x_{ij} = 0$ for all machine-job pairs (i, j) for which $d_{ij} > t$. We now define a family LP(T) of linear programs, one for each value of the parameter t. LP(T) uses only those variables x_{ij} for which $(i, j) \in S_t$, where $S_t = \{(i, j) : d_{ij} < t\}$, and asks if there is a feasible solution. Remember that, we've relaxed constraints on the variables x_{ij} to $x_{ij} \geq 0$. With t fixed, we define a family LP(T) of linear programs

$$
\begin{aligned}
&\textbf{minimize} \quad 0 && (LP(t)) \\
&\textbf{subject to} \\
&\qquad \sum_{i=1}^{m} x_{ij} = 1 \ \ \forall j \\
&\qquad \sum_{j=1}^{n} x_{ij} d_{ij} \leq t \ \ \forall i \\
&\qquad x_{ij} \geq 0 \qquad \forall (i, j) \in S_t
\end{aligned}
$$

Let T be the minimum value(LP optima) for which $LP(t)$ has a feasible solution obtained using binary search. Let T^* be IP optima makespan. Then certainly, $T^* \geq T$. That is, the actual makespan is bounded below by T. But we still don't have IP feasible solution! Our solution is obtained by rounding an extreme point solution of $LP(T)$. We'll later see that makespan thus obtained from rounding is actually atmost $2.T^*$.

LP-rounding

Clearly, an extreme point solution to $LP(t)$ has atmost $n + m$ many non-zero variables. Also, it is easy to prove that any extreme point solution to $LP(t)$ must set atleast $n - m$ many jobs integrally, i.e., $x_{ij} \in \{0, 1\}$.

The LP-rounding algorithm is based on several interesting properties of extreme point solutions of $LP(T)$. For any extreme point solution x for $LP(T)$ define a bipartite graph $G = (M \cup J, E)$ such that $(i, j) \in E$ if and only if $x_{ij} > 0$. Let $F \subset J$ be the set of fractionally set jobs in x. Let H be the subgraph of G induced by the vertex set $M \cup F$. Clearly, $(i, j) \in E(H)$ if $0 < x_{ij} < 1$. A matching H is called a perfect matching if it matches every job $j \in F$.

Each job that is integrally set in x has degree 1 and exactly one edge incident at it in G (Figure 1(a)). Remove these jobs together with their incident edges from G. The resulting graph is clearly H (Figure 1(b)). In H, each job has a degree of at least two. So, all leaves in H must be machines. Keep matching a leaf with the job it is incident to and remove them both from the graph (Figure 1(c)). At each stage all leaves must be machines. In the end we will be left with even cycles Match alternating edges of each cycle. This gives a perfect matching P (Figure 1(d)). Refer to Figure 2 for an example.

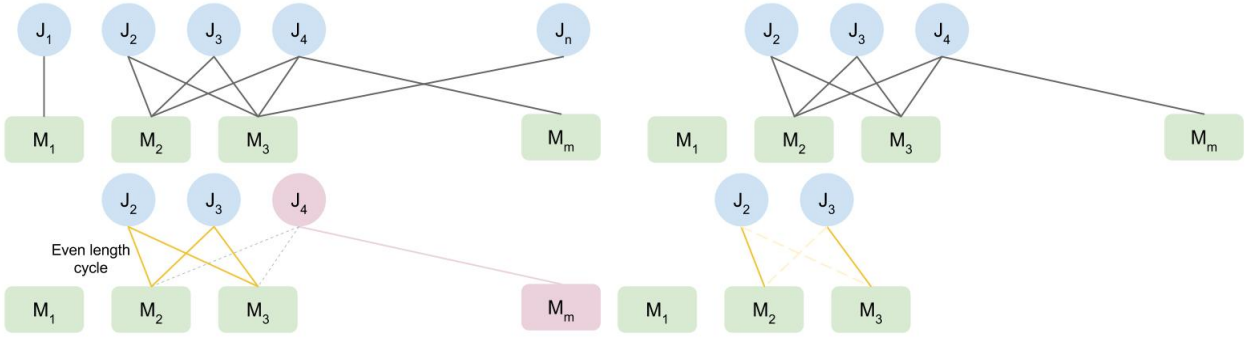


Figure 7.1: Steps in LP relaxation for minimum makespan scheduling on unrelated machines. M is the set of machines, J is set of jobs and edge (J_i, M_k) means job J_i has been scheduled on machine M_k. All nodes in J with degree 1 has been integrally set and all the nodes with degree atleast two has been fractionally set. (a) $G = (M \cup J, E)$. Jobs J_1 and J_n are integrally set. (b) $H = (M \cup F, E')$ contains only fractionally set jobs. (c) Assign a machine to the job that has edge to leaf machine node and remove them from the graph (assign M_n to J_4, remove them and all edges incident on J_4). (d) Match alternate edge of each cycle(M_2 to J_2 and M_3 to J_3).

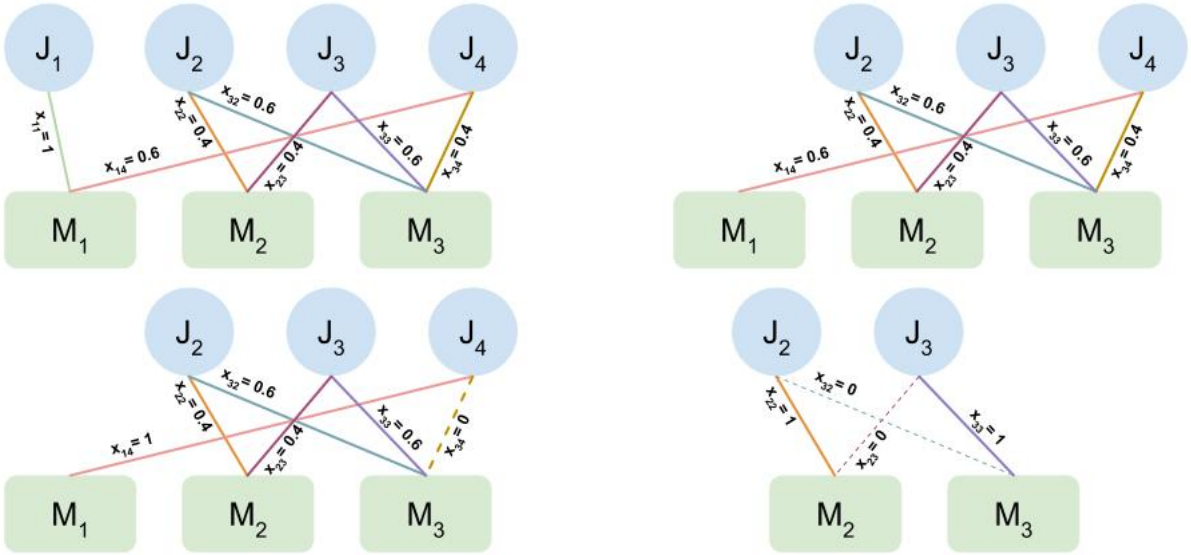


Figure 7.2: An example showing steps in LP relaxation for minimum makespan scheduling on unrelated machines. M is the set of machines, J is set of jobs and edge (J_i, M_k) means job J_i has been scheduled on machine M_k. (a) $G = (M \cup J, E)$. J_1 is integrally set. (b) $H = (M \cup F, E')$ contains only fractionally set jobs obtained by removing J_1 and edge x_{11} from G. (c) Assign a machine to the job that has edge to leaf machine node and remove them from the graph (assign M_1 to J_4, remove them and all edges incident on J_4). (d) Match alternate edge of each cycle(M_2 to J_2 and M_3 to J_3).

Theorem 7.2.2. LP relaxation used here is 2-*approximation* for minimum makespan scheduling on unrelated machines.

Proof. Let T be LP optimal value and T^* be IP optimal value. Then clearly, $T \leq T^*$ since we chose T such that $LP(T)$ has a solution. The extreme point solution x to $LP(T)$ has a fractional makespan of atmost T. Therefore, each integrally set jobs also has an integral makespan of atmost T. Each edge (i, j) of graph H satisfies $d_{ij} \leq T$. The perfect matching found in H schedules atmost one extra job on each machine. Hence, the total makespan is atmost $2.T \leq 2.T^*$. □

References: V. Vazirani, Approximate Algorithm, Chapter 17, Chapter 10

Chapter 8

Solving Ax = b

We had seen many optimization problems, formulating them as LP and IP, and solving with techniques like graphical methods, branch and bound, and also LP relaxation.
In practice, we solve these on digital computers. Let us now understand how these can be solved.
A basic block of all the optimization scheme is numerically solving equations of the form $\mathbf{Ax} = \mathbf{b}$ and $f(\mathbf{x}) = 0$.
In this chapter, we start with revising our understanding of how to solve $\mathbf{Ax} = \mathbf{b}$

8.1 Introduction

In this lecture, let us consider solving a system of linear equations of the form

$$\mathbf{Ax} = \mathbf{b} \tag{8.1}$$

. Let us make the following assumptions on the structures of these matrices:

- A is a square, non-singular matrix
- b is not a zero vector.

Since A is non-singular, the solution is $x = A^{-1}b$. Since, $b \neq 0$ the trivial solution $x = 0$ does not exist.

However, computing A^{-1} is a costly operation (i.e., it requires many numerical operations or precisely FLOPS). Also this has numerical computing and stability issues in many situations (eg. if one of the eigen value is very small or condition number is poor). If the structure of $\mathbf{A}$ is known, and it has some special properties, we can use this knowledge to solve $\mathbf{Ax} = \mathbf{b}$ much more efficiently. This lecture looks at solving the problem when $\mathbf{A}$ is an identity matrix, a permutation matrix, a triangular matrix and a positive definite matrix. As the constraints on the matrix A eases, the complexity of solving it increases. More general matrices are the topics of the next lecture. Also the situations when $\mathbf{A}$ is not a square matrix is the topic of one of the next lectures.

In order to compute the complexity of these operations, we are not interested in the big-O complexity. Instead, we try to calculate the number of *flops* or *floating point operations* needed for each method. We define a flop as one addition, subtraction, multiplication or division of two floating-point numbers. To evaluate the complexity of an algorithm, we count the total number of flops, express it as a function (usually a polynomial) of the dimensions of the matrices and vectors involved, and simplify the expression by ignoring all terms except the leading (i.e., highest order or dominant) terms.

Let us consider different structures of $\mathbf{A}$ so that $\mathbf{Ax} = \mathbf{b}$ can be solved efficiently.

8.2 Simple (near trivial) Situations

8.2.1 A is an Identity matrix

If A is *Identity* I, then $A^{-1} = A = I$. Therefore,

$$x = A^{-1}b \implies x = b$$

This does not require any computation. Therefore, **Flop Count** $= 0$. A trivial problem to solve.

8.2.2 A is a Permutation Matrix

A **Permutation matrix** P is a square binary matrix that has exactly one entry of 1 in each row and each column and 0s elsewhere. Each such matrix represents a specific permutation of n elements and, when used to multiply another matrix, can produce that permutation in the rows or columns of the other matrix.

Example

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

is a permutation matrix. On multiplying this matrix with $\begin{bmatrix} 2 & 3 & 4 \end{bmatrix}^T$, we get $\begin{bmatrix} 2 & 4 & 3 \end{bmatrix}^T$. Or the elements are rearranged.

Result: Inverse of a permutation matrix is its transpose i.e. $P^{-1} = P^T$. Prove this. Hence, the value of x is nothing but a permutation of the matrix b. In this case also, **Flop Count** $= 0$.

8.2.3 A is a Diagonal Matrix

A diagonal matrix is a matrix in which the entries outside the main diagonal are all zero. The diagonal entries themselves may or may not be zero. Thus, the matrix A $= (a_{i,j})$ with n rows and n columns is diagonal if

$$a_{ij} = 0 \text{ if } i \neq j \ \forall i, j \in \{1, 2, ..., n\}$$

For example, the following matrix is diagonal:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Here, **A** is non singular implies that none of the diagonal elements are zero. Therefore, the equation $Ax = b$ can be written as a set of equations

$$a_{11}x_1 = b_1; a_{22}x_2 = b_2; \ldots; a_{nn}x_n = b_n$$

Hence, we can directly compute $x_1, x_2, \ldots, x_n$ as $x_i = \frac{b_i}{a_{ii}}$. There are n multiplications(division) to find the solution, and therefore **Flop Count** $= n$

8.3 A is a Triangular Matrix

- A matrix A is *lower triangular* if the elements above the diagonal are zero, i.e., $a_{ij} = 0$ for $i < j$. The matrix

$$P = \begin{bmatrix} -2 & 0 & 0 & 0 \\ 3 & -6 & 0 & 0 \\ 1 & 3 & 5 & 0 \\ 4 & 4 & 2 & 7 \end{bmatrix} \tag{8.2}$$

  is an example of a 4×4 lower triangular matrix.
- A matrix A is *upper triangular* if the elements below the diagonal are zero: $a_{ij} = 0$ for $i > j$.
- A matrix is *diagonal* if the off-diagonal elements are zero: $a_{ij} = 0$ for $i \neq j$. A diagonal matrix is both upper triangular and lower triangular.
- Nonsingularity of the triangular matrices implies nonzero diagonal elements.
- What is the determinant of the triangular matrix?

8.3.1 Forward Substitution

Suppose **A** is a lower triangular matrix of order n with nonzero diagonal elements. Consider a system of equations **Ax** = **b**:

$$\begin{bmatrix} a_{11} & 0 & \ldots & 0 \\ a_{21} & a_{22} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{8.3}$$

We solve for **x** as:

$$\begin{aligned} x_1 &= \frac{b_1}{a_{11}} \\ x_2 &= \frac{b_2 - a_{21}x_1}{a_{22}} \\ x_3 &= \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}} \\ &\vdots \\ x_n &= \frac{b_n - a_{n1}x_1 - a_{n2}x_2 - \ldots - a_{n,n1}x_{n1}}{a_{nn}}. \end{aligned}$$

Flops Count $= 1 + 3 + 5 + \ldots + (2n - 1) = n^2$, since x_1 can be computed with one operation, x_2 can be computed with three operations and so on.

Recursive Formulation: If A is lower triangular, it can be represented as $\begin{bmatrix} a_{11} & \mathbf{0} \\ A_{21} & A_{22} \end{bmatrix}$ where

- a_{11} is 1x1
- **0** is 1x(n-1)
- A_{21} is (n-1) x 1
- A_{22} is a *lower triangular* matrix of size (n-1) x (n-1)

Now, the forward substitution algorithm can be written recursively using this representation.

$$\begin{bmatrix} a_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ B_2 \end{bmatrix} \tag{8.4}$$

Algorithm: Forward Substitution

1. $x_1 = \frac{b_1}{a_{11}}$
2. Solve $A_{22}X_2 = (B_2 - A_{21}x_1)$ by **Forward Substitution**

Similarly and upper triangular matrix can be represented as

$$\begin{bmatrix} A_{11} & A_{12} \\ \mathbf{0} & a_{nn} \end{bmatrix} \begin{bmatrix} X_1 \\ x_n \end{bmatrix} = \begin{bmatrix} B_1 \\ b_n \end{bmatrix} \tag{8.5}$$

and solved recursively with a backward substitution algorithm as:

Algorithm: Backward Substitution

1. $x_n = \frac{b_n}{a_{nn}}$
2. Solve $A_{11}X_1 = (B_1 - A_{12}x_n)$ by **Backward Substitution**

Flop Count $= n^2$

8.4 Cholesky Decomposition of Positive Definite (PD) Matrix

8.4.1 PD Matrix

A matrix is said to be positive definite if it satisfies

1. The matrix is Symmetric
   - A symmetric matrix is a square matrix that is equal to its transpose. i.e., $A^T = A$
2. $x^T Ax > 0 \ \forall x$, non zero.

A matrix is postive semi definite (PSD) if the second constraint is relaxed as $x^T Ax \geq 0 \ \forall x$

Example 1. The identity matrix I $= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ is positive definite because, for every real vector z $= \begin{bmatrix} a \\ b \end{bmatrix}$, $z^T Iz = z^T z = \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = a^2 + b^2$, which is positive.

8.4.2 Cholesky Decomposition

Every positive definite matrix A can be factored as

$$A = LL^T \tag{8.6}$$

where L is lower triangular with positive diagonal elements. This is called the *Cholesky factorization* of A. If n = 1, i.e A is a scalar, then the Cholesky factor of A is just the square root of A.

Example 2. An example of a 3×3 Cholesky factorization is

$$\begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} = \begin{bmatrix} 5 & 0 & 0 \\ 3 & 3 & 0 \\ -1 & 1 & 3 \end{bmatrix} \times \begin{bmatrix} 5 & 3 & -1 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{bmatrix} \tag{8.7}$$

Example 3. An example of a 3x3 Cholesky factorization is

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 70 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 4 & 0 \\ 3 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

FLOPS The Cholesky factorization takes $\frac{1}{3}n^3$ flops.

8.5 Algorithm for Cholesky Factorization

Now, let A = $\begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix}$, L = $\begin{bmatrix} l_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}$ and $A = LL^T$

Question: What are the dimensions of these matrices?.

We can thus write

$$\begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} l_{11} & L_{21}^T \\ 0^T & L_{22} \end{bmatrix}$$

Now, we can form the following set of equations from above

- $a_{11} = l_{11} \cdot l_{11} \implies l_{11} = \sqrt{a_{11}}$
- $A_{21} = L_{21} \cdot l_{11} \implies L_{21} = \frac{A_{21}}{l_{11}}$
- $A_{22} = L_{21}L_{21}^T + L_{22}L_{22}^T \implies (A_{22} - L_{21}L_{21}^T) = L_{22}L_{22}^T$

Algorithm: Cholesky

1. Calculate $l_{11} = \sqrt{a_{11}}$
2. Calculate $L_{21} = \frac{A_{21}}{l_{11}}$
3. Use **Cholesky** to compute L_{22} as $(A_{22} - L_{21}L_{21}^T) = L_{22}L_{22}^T$

The cost of this algorithm is $(1/3)n^3$ flops.

Question: Verify this.

Let us look at an example for computing Cholesky factorization.

Example 4. Consider A = $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 70 \end{bmatrix}$. This can be factorized as $\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & 0 \end{bmatrix}$

- $l_{11} = \sqrt{a_{11}} = 1$
- $L_{21} = \frac{A_{21}}{l_{11}} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$. Now, L = $\begin{bmatrix} 1 & 0 & 0 \\ 2 & l_{22} & 0 \\ 3 & l_{32} & l_{33}. \end{bmatrix}$
- We have to do Cholesky factorization of A_{22} - $L_{21}{L_{21}}^T$ = $\begin{bmatrix} 16 & 20 \\ 20 & 61 \end{bmatrix} = \begin{bmatrix} l_{22} & 0 \\ l_{32} & l_{33}. \end{bmatrix} . \begin{bmatrix} l_{22} & l_{32} \\ 0 & l_{23} \end{bmatrix}$
- $l_{22} = \sqrt{a_{22}} = 4$
- $l_{23} = \frac{20}{4} = 5$

- The matrix is now $\begin{bmatrix} 1 & 0 & 0 \\ 2 & 4 & 0 \\ 3 & 5 & l_{33} \end{bmatrix}$
- We have to factorize 61 - $5 \cdot 5 = 36$. This gives l_{33} as 6
- The final answer is
$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 70 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 4 & 0 \\ 3 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$

8.6 Solving linear equations by Cholesky factorization

Now we see how cholesky factorization is useful.

Solve $AX = b$ where A is PD matrix of order $n \times n$. $LL^T x = b$

1. *Cholesky factorization:* factor A as LL^T ($(1/3)n^3$ flops).
2. *Forward substitution:* Solve $Lw = b$ (n^2 flops).
3. *Back substitution:* Solve $L^T x = w$ (n^2 flops).

Total cost is $(1/3)n^3 + 2n^2$ or roughly $(1/3)n^3$.

8.7 Finding Inverse using Cholesky factorization

The inverse of a positive definite matrix can be computed by solving

$$Ax = I \text{ or } AX = [e_1, e_2, \ldots, e_n] \tag{8.8}$$

where e_i is a vector with 1 only at the ith place and zero every where else. Also X is a matrix where we are intersted in solving column by column as $Ax_i = e_i$. We do this with cholesky decomposition. However, only one Cholesky factorization of A is required, while we need to solve the linear system of equations $Ax_i = e_i$ n times. This needs n forward and backward substitutions to solve the triangular systems (step 2 and 3). The cost of computing the inverse using this method is $(1/3)n^3 + n \cdot 2n^2 = (7/3)n^3$.

8.8 Additional Example Problems

8.8.1 Example 1

Compute Cholesky factorization of
A = $\begin{bmatrix} 4 & 6 & 2 & -6 \\ 6 & 34 & 3 & -9 \\ 2 & 3 & 2 & -1 \\ -6 & -9 & -1 & 38 \end{bmatrix}$

Sol: Cholesky factorization of $A = LL^T$

$$\begin{bmatrix} 4 & 6 & 2 & -6 \\ 6 & 34 & 3 & -9 \\ 2 & 3 & 2 & -1 \\ -6 & -9 & 1 & 38 \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{43} & l_{43} & l_{44} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} & l_{41} \\ 0 & l_{22} & l_{32} & l_{42} \\ 0 & 0 & l_{33} & l_{43} \\ 0 & 0 & 0 & l_{44} \end{bmatrix} \tag{8.1}$$

- Determine l_{11} and L_{21} where $L_{21} = \begin{bmatrix} l_{21} \\ l_{31} \\ l_{41} \end{bmatrix}$

$$l_{11} = \sqrt{a_{11}}$$
$$l_{11} = 2$$
$$L_{21} = \frac{1}{l_{11}} \begin{bmatrix} 6 \\ 2 \\ -6 \end{bmatrix}$$
$$L_{21} = \begin{bmatrix} 3 \\ 1 \\ -3 \end{bmatrix}$$

- To compute L_{22} i.e. $A_{22} - L_{21}L_{21}^T = L_{22}L_{22}^T$

$$\begin{bmatrix} 34 & 3 & -9 \\ 3 & 2 & -1 \\ -9 & 1 & 38 \end{bmatrix} - \begin{bmatrix} 3 \\ 1 \\ -3 \end{bmatrix} \begin{bmatrix} 3 & 1 & -3 \end{bmatrix} = \begin{bmatrix} l_{22} & 0 & 0 \\ l_{32} & l_{33} & 0 \\ l_{43} & l_{43} & l_{44} \end{bmatrix} \begin{bmatrix} l_{22} & l_{32} & l_{42} \\ 0 & l_{33} & l_{43} \\ 0 & 0 & l_{44} \end{bmatrix}$$

$$\begin{bmatrix} 25 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 2 & 29 \end{bmatrix} = \begin{bmatrix} l_{22} & 0 & 0 \\ l_{32} & l_{33} & 0 \\ l_{43} & l_{43} & l_{44} \end{bmatrix} \begin{bmatrix} l_{22} & l_{32} & l_{42} \\ 0 & l_{33} & l_{43} \\ 0 & 0 & l_{44} \end{bmatrix}$$

$$l_{22} = 25$$
$$l_{22} = 5$$
$$l_{32} = 0$$
$$l_{42} = 0$$

- We are left with

$$\begin{bmatrix} 1 & 2 \\ 2 & 29 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} l_{33} & 0 \\ l_{43} & l_{44} \end{bmatrix} \begin{bmatrix} l_{33} & l_{43} \\ 0 & l_{44} \end{bmatrix} \tag{8.2}$$

$$l_{33}^2 = 1$$
$$l_{33} = 1$$
$$l_43 = 2/l_{33}$$
$$l_{43} = 2$$

To solve for l_{44} we have $29 - \begin{bmatrix} 2 \end{bmatrix} \begin{bmatrix} 2 \end{bmatrix} = l - 44^2$

$$l_{44} = 5$$

Putting all the values in the first equation

$$\begin{bmatrix} 4 & 6 & 2 & -6 \\ 6 & 34 & 3 & -9 \\ 2 & 3 & 2 & -1 \\ -6 & -9 & 1 & 38 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 3 & 5 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ -3 & 0 & 2 & 5 \end{bmatrix} \begin{bmatrix} 2 & 3 & 1 & -3 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

8.8.2 Example 2

You are given a Cholesky factorization of A = LL^T of positive definite A of order n.

1. What is the Cholesky factor of the (n+1)x(n+1) matrix B = $\begin{bmatrix} A & u \\ u^{\mathrm{T}} & 1 \end{bmatrix}$ where B is positive semidefinite?

   - $\begin{bmatrix} A & u \\ u^{\mathrm{T}} & 1 \end{bmatrix} = \begin{bmatrix} L_{11} & \mathbf{0} \\ L_{21} & l_{\mathrm{n+1,n+1}} \end{bmatrix} \begin{bmatrix} {L_{11}}^{\mathrm{T}} & {L_{21}}^{\mathrm{T}} \\ \mathbf{0} & l_{\mathrm{n+1,n+1}} \end{bmatrix}$
   - A = $\mathrm{L_{11}{L_{11}}^{T}}$
   - $\therefore$ $\mathrm{L_{11}}$ = L
   - u = $\mathrm{L_{11}{L_{21}}^{T}} \rightarrow {\mathrm{L_{21}}^{\mathrm{T}}} = \mathrm{{L_{11}}^{-1}u}$
   - $1 = \mathrm{L_{21}{L_{21}}^{T}} + {\mathrm{l_{n+1,n+1}}}^2 \rightarrow {\mathrm{l_{n+1,n+1}}}^2 = 1 - \mathrm{L_{21}{L_{21}}^{T}} = 1 - u^T L^{-\mathrm{T}} L^{-1} u = 1 - u^T A^{-1} u$
   - Thus, the factors of B are $\begin{bmatrix} L & \mathbf{0} \\ u^T L^{-\mathrm{T}} & \sqrt{1 - u^T A^{-1} u} \end{bmatrix} \begin{bmatrix} L^T & L^{-1}u \\ \mathbf{0} & \sqrt{1 - u^T A^{-1} u} \end{bmatrix}$

2. What is the cost of computing Cholesky of B if factors of A is given?

   - n^2 flops
   - We are given L, so we only need to compute $\mathrm{L_{21}}$ and $\mathrm{L_{22}}$.
   - We can compute $\mathrm{L^{-1}u}$ by solving the set of equations :x = u which takes n^2 flops because L is lower triangular.
   - Computing $\sqrt{1 - u^T A^{-1} u} = \sqrt{1 - x^T x}$ takes roughly 2n flops

3. Suppose $||L^{-1}|| \geq 1$. Show that B is positive definite for all u such that $||u|| \geq 1$

   - Given a matrix M = $\begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$, M is positive definite if A and $C - B^T A^{-1} B$ are positive definite. (Refer to [**?**] for the proof).
   - So in our case, A and $1 - u^T A^{\mathrm{T}} u$ should be positive definite. We already know that A is positive definite.
   - From the first part of this solution, u = $\mathrm{L{L_{21}}^{T}}$, A = $\mathrm{LL^{T}}$

$$1 - u^T A^{-1} u \tag{8.3}$$
$$= 1 - (L{L_{21}}^T)^T (LL^T)^{-1} L{L_{21}}^T \tag{8.4}$$
$$= 1 - L_{21} L^T L^{-\mathrm{T}} L^{-1} L^T {L_{21}}^T \tag{8.5}$$
$$= 1 - L_{21}{L_{21}}^T \tag{8.6}$$
$$= {l_{\mathrm{n+1,n+1}}}^2 \tag{8.7}$$

   which is a positive scalar and hence is positive definite.

   - Hence B is positive definite.

8.8.3 Example 3

Solve efficiently LX + $\mathrm{XL^{T}}$ = B given L(lower triangular matrix) and B. $L_{\mathrm{ii}} + L_{\mathrm{jj}} \neq 0 \forall i, j$.
Analyze the complexity.

- Let us write the equation using the recursive form.
  $\begin{bmatrix} l_{11} & \mathbf{0} \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} x_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} + \begin{bmatrix} x_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} l_{11} & {L_{21}}^T \\ \mathbf{0}^T & {L_{22}}^T \end{bmatrix} = \begin{bmatrix} b_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$
- Simplifying the L.H.S and equating to R.H.S, we get the following four equations:

$$2 l_{11} x_{11} = b_{11} \tag{8.8}$$
$$l_{11} X_{12} + x_{11} {L_{21}}^T + X_{12} {L_{22}}^T = B_{12} \tag{8.9}$$
$$L_{21} x_{11} + L_{22} X_{21} + X_{21} l_{11} = B_{21} \tag{8.10}$$
$$L_{21} X_{12} + L_{22} X_{22} + X_{21} {L_{21}}^T + X_{22} {L_{22}}^T = B_{22} \tag{8.11}$$

- Equation (8.8) can be solved in 1 flop
- For equation (8.9), the unknown is X_{12}. To simplify the L.H.S (multiplications and additions of matrices), number of flops $= 2(n-1)^2 + 3(n-1)$
- After equating the L.H.S to the R.H.S, we get a system of linear equations of the form

$$f(x_{12}, x_{13}, \ldots, x_{1n}) = b_{11} \tag{8.12}$$

$$f(x_{13}, x_{14}, \ldots, x_{1n}) = b_{12} \tag{8.13}$$

$$\vdots$$

$$f(x_{1n}) = b_{1n} \tag{8.14}$$

  This upper triangular system of equations can be solved using backward substitution in $(n-1)^2$ flops.
- Total number of flops for equation (8.9) $= 3(n-1)^2 + 3(n-1) \approx 3n^2$
- Similarly, equation (8.10) can be solved in $3n^2$ flops.
- For equation (8.11), the unknown is X_{22}. The equation can be rewritten as

$$L_{22}X_{22} + X_{22}{L_{22}}^T = B_{22} - L_{21}X_{12} - X_{21}{L_{21}}^T \tag{8.15}$$

- The R.H.S can be computed in $5(n-1)^2$. The equation is of the form $LX + XL^T = B$ which can be computed recursively.
- Each recursive step takes on its own $1 + 3n^2 + 3n^2 + 5n^2 \approx 11(n-1)^2$. Total number of flops is $\sum_{i=1}^{n} 11n^2 = \frac{11n(n+1)(2n+1)}{6} = \frac{11}{6}(2n^3 + 3n^2 + n)$. Considering only the leading term for flop calculation, total number of flops $= \frac{11}{3}n^3$

8.8.4 Exercise

Given A_{nxn} is a P.D matrix and $B = \begin{bmatrix} A & -A \\ -A & \beta A \end{bmatrix}$ is a P.D matrix. Find the range of values of β .

Chapter 9

Matrix Decompositions: LU, QR and SVD

Now that we know how to solve $\mathbf{Ax} = \mathbf{b}$, let us look at a class of matrix factorization schemes that can make the solution (i) fast and/or (ii) numerically more stable and accurate.

9.1 Review and Summary

Matrix decomposition or Matrix Factorization is the method of trasnforming a given matrix to a product of canonical matrices. Matrix decompositions are usually carried out for making the problem computationally convenient to solve and simple to analyze. For example, matrix inversion, solving linear systems and least squares fitting problem can be infeasible to solve optimally in an explicit manner. Thus, converting them into a set of several easier tasks such as solving a diagonal or triangular system helps speed up the process. It also helps in identifying the underlying structure of the matrices involved. In the previous class, we took a look at the Cholesky Factorization method. More formally, it can be stated as follows:

Solve $Ax = b$ where A is a non-singular $n \times n$ matrix.

Obviously, $x = A^{-1}b$ is not computationally preferable.
If A were a Permutation matrix it would take 0 flops to solve this problem. Similarly, if it were Identity it would require 0 flops, if it were Diagonal, n flops and lastly if it were Positive Definite, it would take $\frac{1}{3}n^3 + 2n^2$. Cholesky factorization involves decomposing A as $A = LL^T$ and it takes only $\frac{1}{3}n^3$ flops.

In today's lecture, we take a look at two more methods of factorization namely, **LU**, **QR** and **SVD**.

To solve system of linear equations, in which each equation is in the form $a^T x = c$, where a is a $n - vector$ (A vector of size n) of the form $\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$ such that $a_i \in R \ \forall i = 1 \dots n$ and x is a variable *n-vector* of the form $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ Therefore resultant equation is $a_1x_1 + a_2x_2 + \ldots + a_nx_n = c$

The set of linear equation is represented in the form $Ax = b$:

$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$ where A: $R^{m \times n}$ (A matrix of order $m \times n$, all elements belongs to Real numbers), x is a $n - vector$ and b is a $m - vector$.

Indeed many of our discussions on real matrices, are directly applicable for matrices of complex numbers as such or with minimal changes. However, that is not attempted here.

Flops for various matrices

Flops= Total number of floating point operations or flops required to perform numerical algorithm.

Let us assume that $m = n$ (A is a square matrix) and also non-singular (Inverse of A, A^{-1} exists).

Following table shows flops required by the different types of matrices to solve linear equations $Ax = b$

Matrix type	flops
Identity Matrix	0
Permutation Matrix	0
Diagonal Matrix	n
Upper Triangular Matrix	n^2
Lower Triangular Matrix	n^2
Positive Definite Matrix	$\frac{1}{3}n^2 + 2n^2$

9.2 LU Factorization

9.2.1 Definition

Factorize or decompose a square non-singular matrix into product of two matrices L, U with the help of permutation matrix, if necessary.

$$A = PLU$$

where A is any non singular matrix, P is Permutation Matrix, L is Lower triangular Matrix, U is an Upper triangular matrix

Most of the times P is an Identity matrix. For a matrix, A multiple $LU - decompostions$ are possible. To get unique decomposition, principle diagonals of either L or U must be one. From now onwards in our discussion we assume that $P = I$ and $L_{ii} = 1$, $\forall i$. The standard algorithm for computing $LU - decompostion$ is called Gaussian elimination.

Cost The cost is $\frac{2}{3}n^3$ flops.

9.2.2 LU Factorization

An example of an LU factorization is

$$\begin{bmatrix} 0 & 5 & 5 \\ 2 & 9 & 0 \\ 6 & 8 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 0 & 15/19 & 1 \end{bmatrix} \begin{bmatrix} 6 & 8 & 8 \\ 0 & 19/3 & -8/3 \\ 0 & 0 & 135/19 \end{bmatrix} \tag{9.1}$$

The algorithm for computing the LU factorization is called *Gaussian elimination* and it takes $\frac{2}{3}n^3$ flops.

The Gaussian elimination method in its pure form is unstable at times and the permutation matrix **P**, is used to control this inability. It does this by permutating the order of the rows of the matrix being operated upon. Such operations are called *pivoting*. However, it is possible to factor several non-singular matrices as LU without pivoting. In the following subsection, we first describe the method and then provide an algorithm for the same.

9.3 Computing the LU factorization

Let us consider the simple case where $P = I$ such that

$$A = LU \tag{9.2}$$

The above equation can be partitioned and rewritten as

$$\begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} \tag{9.3}$$

$$\begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} l_{11}u_{11} & l_{11}U_{12} \\ u_{11}L_{21} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix} \tag{9.4}$$

However, $l_{11} = 1$ as L is a unit lower triangular matrix. Therefore, equating both sides and performing appropriate substitutions gives us:

$$u_{11} = a_{11}, \qquad U_{12} = A_{12}, \qquad L_{21} = \frac{1}{a_{11}}A_{21} \tag{9.5}$$

$$L_{22}U_{22} = A_{22} - \frac{1}{a_{11}}A_{21}A_{12} \tag{9.6}$$

L_{22} and U_{22} can be calculated recursively by performing a LU factorization of dimension $(n-1)$ on the matrix obtained on the right hand side of equation (5). This process continues till we arrive at a 1×1 matrix. The algorithm can be summarized as follows :

9.3.1 Computational procedure

Given a $n \times n$ nonsingular matrix A

1. Calculate the first row of U : $u_{11} = a_{11}$ and $U_{12} = A_{12}$
2. Calculate the first column of L : $l_{11} = 1$; $L_{12} = (1/a_{11})A_{21}$.
3. Recursively calculate the LU factorization of the $(n-1) \times (n-1)$ matrix $L_{22}U_{22} = A_{22} - \frac{1}{a_{11}}A_{21}A_{12}$

9.3.2 Example 1

: $A = \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix}$

$$A = \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} 0 & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

This is not possible since $L_{21} = \frac{1}{0}(1)$. Suppose $P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

$$PA = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

Now, $L_{21} = \frac{1}{1}(0) = 0$ and $U_{12} = -1$

Therefore,

$$PA = \begin{bmatrix} 1 & 0 \\ 0 & L_{22} \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & U_{22} \end{bmatrix}$$

and,

$$L_{22}U_{22} = A_{22} - 0(-1) = 1$$

which implies that $L_{22} = U_{22} = 1$.

Therefore,

$$PA = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

Finally,

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

9.3.3 Example 2

$$A = \begin{bmatrix} 6 & 3 & 1 \\ 2 & 4 & 3 \\ 9 & 5 & 2 \end{bmatrix}$$

Factoring,

$$A = \begin{bmatrix} 6 & 3 & 1 \\ 2 & 4 & 3 \\ 9 & 5 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

$$\begin{bmatrix} 6 & 3 & 1 \\ 2 & 4 & 3 \\ 9 & 5 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 3/2 & l_{32} & 1 \end{bmatrix} \begin{bmatrix} 6 & 3 & 1 \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

$$L_{22}U_{22} = \begin{bmatrix} 1 & 0 \\ l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{22} & u_{23} \\ 0 & u_{33} \end{bmatrix} = \begin{bmatrix} 4 & 3 \\ 5 & 2 \end{bmatrix} - (1/6) \begin{bmatrix} 2 \\ 9 \end{bmatrix} \begin{bmatrix} 3 & 1 \end{bmatrix}$$

$$L_{22}U_{22} = \begin{bmatrix} 1 & 0 \\ l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{22} & u_{23} \\ 0 & u_{33} \end{bmatrix} = \begin{bmatrix} 3 & 8/3 \\ 1/2 & 1/2 \end{bmatrix}$$

$$L_{22}U_{22} = \begin{bmatrix} 1 & 0 \\ 1/6 & 1 \end{bmatrix} \begin{bmatrix} 3 & 8/3 \\ 0 & u_{33} \end{bmatrix}$$

$$u_{33} = 1/2 - (1/3)(1/2)(8/3) = 1/18$$

$$A = \begin{bmatrix} 6 & 3 & 1 \\ 2 & 4 & 3 \\ 9 & 5 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 3/2 & 1/6 & 1 \end{bmatrix} \begin{bmatrix} 6 & 3 & 1 \\ 0 & 3 & 8/3 \\ 0 & 0 & 1/18 \end{bmatrix}$$

9.4 Solving linear equations by LU Factorization

The use of LU factorization is a standard way of solving linear equations with a general nonsingular coefficient matrix A. Given a set of linear equations $Ax = b$, the following algorithm is used to find the solution :

1. *LU factorization*: Factor A as $A = PLU$ — $(2/3)n^3$ flops
2. *Permutation*: Calculate $w = P^T b$ by reordering the rows of b — 0 flops
3. *Forward Substitution*: Solve $Lz = w$ for z — n^2 flops
4. *Backward Substitution*: Solve $Ux = w$ — n^2 flops

The total cost is $(2/3)n^2 + 2n^2$ or simply $(2/3)n^2$ flops.

9.5 Computing the Inverse using LU

In order to compute the inverse A^{-1}, we can solve the equation

$$AX = I$$

That is to say, we can solve the system of n equations, $Ax_i = e_i$ where x_i is the ith column of A^{-1} and e_i is the ith unit vector. The cost of this computation would be $\frac{2}{3}n^3 + n(2n^2) = \frac{8}{3}n^3$ flops(one LU factorization and n forward and n backward substitutions).

9.6 Solution of $Ax = b$ with a direct inverse

$Ax = b$

$x = A^{-1}b$

Step1: Find Inverse of A, A^{-1}

Total cost: $\frac{8}{3}n^3$ flops

Step2: Matrix multiplication between A^{-1} and b, $A^{-1}b$

It requires $n^2 + n(n-1)$ flops.

Therefore total $\frac{8}{3}n^3 + n^2 + n(n-1)$ flops required.

Note: A solution based on inverse is not attractive.

9.6.1 Example 3

Solve

$$\begin{bmatrix} 6 & 3 & 1 \\ 2 & 4 & 3 \\ 9 & 5 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 18 \\ 17 \\ 29 \end{bmatrix}$$

We shall use the factorization performed in Example 2. As $P = I$, the permutation step does not affect the outcome. We first solve,

$$\begin{bmatrix} 1 & 1 & 0 \\ 1/3 & 1 & 0 \\ 3/2 & 1/6 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 18 \\ 17 \\ 29 \end{bmatrix}$$

which gives the solution $(18, 11, 1/6)$ by forward substitution. Finally, we solve

$$\begin{bmatrix} 6 & 3 & 1 \\ 0 & 3 & 8/3 \\ 0 & 0 & 1/18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 18 \\ 11 \\ 1/6 \end{bmatrix}$$

which gives the solution $x = (2, 1, 3)$ by backward substitution.

9.6.2 Eample Problem

Solve the following problem with 3 parts.

1. For what values of a_1, a_2, ..., a_n is the $n \times n$ matrix

$$A = \begin{bmatrix} a_1 & 1 & 0 & \dots & 0 & 0 \\ a_2 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & 0 & 0 & \dots & 1 & 0 \\ a_{n-1} & 0 & 0 & \dots & 0 & 1 \\ a_n & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

   non-singular?
2. Assume A is non-singular, how many floating point operations do you need to solve $Ax = b$
3. Assume A is non-singular, what is the inverse of A^{-1}(In other words, express the elements of A^{-1} in terms of a_1, a_2, ..., a_n)

Solution: **1.** A is non-singular if and only if $Ax = 0$ implies $x = 0$

$Ax = 0$ means

$x_2 = -a_1 x_1$

$x_3 = -a_2 x_1$

$\vdots$

$x_n = -a_{n-1}x_1$

$a_n x_1 = 0$

If $a_n \neq 0$ then from the last equation, $x_1 = 0$, because of that remaining elements of x also becomes zero, $x_2 = x_3 = \ldots = x_n = 0$ i.e., $x = 0$, so A is non-singular.

If $a_n = 0$, and $x_1 = 1$ then, $x_2 = -a_1$, $x_3 = -a_2$, etc. and obtain a non zero x with $Ax = 0$. So if $a_n = 0$ then resultant Matrix becomes singular

$\forall a_n \neq 0$ A is non-singular. a_1, a_2, ..., a_{n-1} can take any value.

2. If we put the last equation first we obtain

$a_n x_1 = b_n$

$a_1 x_1 + x_2 = b_1$

$a_2 x_1 + x_3 = b_2$

$\vdots$

$a_{n-1}x_1 + x_n = b_{n-1}$

We can solve these equations by forward substitution.

$x_1 = \frac{b_n}{a_n}$ (1flop)

$x_2 = b_1 - a_1 x_1$(from now onwards 2 flops in each equation)

$x_3 = b_2 - a_2 x_1$

$\vdots$

$x_n = b_{n-1} - a_{n-1}x_1$

thus it takes $2n - 1$flops.

3.We can find A^{-1} by solving $AX = I$ column by column.

$$\begin{bmatrix} a_1 & 1 & 0 & \ldots & 0 & 0 \\ a_2 & 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & 0 & 0 & \ldots & 1 & 0 \\ a_{n-1} & 0 & 0 & \ldots & 0 & 1 \\ a_n & 0 & 0 & \ldots & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 & \vdots & x_{n-1} & x_n \end{bmatrix} = \begin{bmatrix} e_1 & e_2 & \ldots & e_{n-1} & e_n \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} 0 & 0 & 0 & \ldots & 0 & \frac{1}{a_n} \\ 1 & 0 & 0 & \ldots & 0 & \frac{-a_1}{a_n} \\ 0 & 1 & 0 & \ldots & 0 & \frac{-a_2}{a_n} \\ 0 & 0 & 1 & \ldots & 0 & \frac{-a_3}{a_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & \frac{-a_{n-1}}{a_n} \end{bmatrix}$$

9.7 QR Factorization

Factorizations have a wide number of applications. However, some factorizations like Cholesky are not always appropriate or efficient enough due to certain inherent restrictions. For example, the Least Squares problem can be solved faster by QR factorization which we cover in this section. (more in the next lecture). Also we need methods to factorize non square matrices.

A left-invertible $m \times n$ matrix can be factored as

where Q is an $m \times n$ orthogonal matrix and R is an $n \times n$ upper triangular matrix with positive diagonal elements. This is called the QR *factorization* of A.
Just to recap, an $m \times n$ orthogonal matrix has the property that:

1. $Q^TQ = I$, when $m > n$
2. $QQ^T = I$, when $m < n$
3. $Q^TQ = QQ^T = I$, when $m = n$

An example of a QR factorization is

$$(1/5)\begin{bmatrix}3 & -6 & 26\\ 4 & -8 & -7\\ 0 & 4 & 4\\ 0 & -3 & -3\end{bmatrix} = \begin{bmatrix}3/5 & 0 & 4/5\\ 4/5 & 0 & -3/5\\ 0 & 4/5 & 0\\ 0 & -3/5 & 0\end{bmatrix}\begin{bmatrix}1 & -2 & 2\\ 0 & 1 & 1\\ 0 & 0 & 5\end{bmatrix} \tag{9.7}$$

9.8 QR Factorization: The Method

We now describe the method and the algorithm to compute the QR factorization and its cost is $2mn^2$. As an exercise for further understanding, one can attempt to factorize the matrix on the left hand side of the above equation and arrive at the matrices on the right hand side.

The first step is to factorize the A, Q and R matrices in the following manner,

$$A = \begin{bmatrix}a_1 & A_2\end{bmatrix}, \qquad Q = \begin{bmatrix}q_1 & Q_2\end{bmatrix}, \qquad R = \begin{bmatrix}r_{11} & R_{12}\\ 0 & R_{22}\end{bmatrix}$$

where each of the elements follow the standard block notation.
Since Q is orthogonal, we know $Q^TQ = \begin{bmatrix}q_1^T\\ Q_2^T\end{bmatrix}\begin{bmatrix}q_1 & Q_2\end{bmatrix} = \begin{bmatrix}q_1^Tq_1 & q_1^TQ_2\\ Q_2^Tq_1 & Q_2^TQ_2\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & I\end{bmatrix}$

Therefore,

$$q_1^Tq_1 = 1, \qquad q_1^TQ_2 = 0\ , \qquad Q_2^TQ_2 = I$$

We also know, that $r_{11} > 0$ and R_{22} is an upper triangular matrix with positive diagonals.

Combining the above, we get,

$$\begin{bmatrix}a_1 & A_2\end{bmatrix} = \begin{bmatrix}q_1 & Q_2\end{bmatrix}\begin{bmatrix}r_{11} & R_{12}\\ 0 & R_{22}\end{bmatrix} = \begin{bmatrix}q_1r_{11} & q_1R_{12} + Q_2R_{22}\end{bmatrix} \tag{9.8}$$

Comparing the left and the right matrices we can conclude that $a_1 = q_1r_{11}$. But since q_1 is unit norm, $r_{11} = ||a_1||$ and $q_1 = \frac{a_1}{r_{11}}$. And, $A_2 = q_1R_{12} + Q_2R_{22}$.
To simplify this, notice that we can premultiply both sides by q_1^T thereby obtaining,

$$q_1^TA_2 = q_1^Tq_1R_{12} + q_1^TQ_2R_{22}$$

Since, $q_1^Tq_1 = 1$ and $q_1^TQ_2 = 0$,

$$R_{12} = q_1^TA_2 \tag{9.9}$$

$\therefore \begin{bmatrix}A_2 - q_1R_{12}\end{bmatrix} = Q_{12}R22$

9.8.1 Algorithm: QR FACTORIZATION

The algorithm to compute the QR factors of the matrix can be concisely written as follows:

Given an $m \times n$ matrix A, it can be factored as A=QR

1. Compute the preliminary values of the first rows/columns: $r_{11} = ||a_1||$, $q_1 = \frac{a_1}{r_{11}}$ and $R_{12} = q_1^T A_2$
2. Computer QR factor of $A_2 - q_1 R_{12}$ as $Q_{12}R22$

Computational cost: $2mn^2 \frac{2}{3}n^3 flops, or \frac{4}{3}n^3$ for square matrices.

9.8.2 Example

Find $QR - factorization$ of $\begin{bmatrix} 2 & 8 & 13 \\ 4 & 7 & -7 \\ 4 & -2 & -13 \end{bmatrix}$. Show steps.

Solution:

Recursive Algorithm:

1. $r_{11} = \|a_1\|$
2. $q_1 = \frac{a_1}{\|a_1\|}$
3. $R_{12} = {q_1}^T A_2$
4. Compute $Q_2 R_{22}$by $QR - factorization$ of $A_2 - q_1 R_{12}$

Step1:

$r_{11} = \|a_1\| = \sqrt{4 + 16 + 16} = 6$; $q_1 = \frac{a_1}{\|a_1\|} = \frac{1}{6}\begin{bmatrix} 2 \\ 4 \\ 4 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{bmatrix}$; $R_{12} = {q_1}^T A_2 = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \end{bmatrix}\begin{bmatrix} 8 & 13 \\ 7 & -7 \\ -2 & -13 \end{bmatrix} = \begin{bmatrix} 6 & -9 \end{bmatrix}$;

$A_2 - q_1 R_{12} = \begin{bmatrix} 8 & 13 \\ 7 & -7 \\ -2 & -13 \end{bmatrix} - \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{bmatrix}\begin{bmatrix} 6 & -9 \end{bmatrix} \Rightarrow \begin{bmatrix} 8 & 13 \\ 7 & -7 \\ -2 & -13 \end{bmatrix} - \begin{bmatrix} 2 & -3 \\ 4 & -6 \\ 4 & -6 \end{bmatrix} \Rightarrow \begin{bmatrix} 6 & 16 \\ 3 & -1 \\ -6 & -7 \end{bmatrix}$

$A = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{bmatrix}\begin{bmatrix} 6 & 6 & -9 \\ 0 & & \\ 0 & & \end{bmatrix}$

Step2:

$r_{11} = \|a_1\| = \sqrt{36 + 9 + 36} = 9$; $q_1 = \frac{a_1}{\|a_1\|} = \frac{1}{9}\begin{bmatrix} 6 \\ 3 \\ -6 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ \frac{1}{3} \\ \frac{-2}{3} \end{bmatrix}$; $R_{12} = {q_1}^T A_2 = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} & \frac{-2}{3} \end{bmatrix}\begin{bmatrix} 16 \\ -1 \\ -7 \end{bmatrix} = \begin{bmatrix} 15 \end{bmatrix}$;

$A_2 - q_1 R_{12} = \begin{bmatrix} 16 \\ -1 \\ -7 \end{bmatrix} - \begin{bmatrix} \frac{2}{3} \\ \frac{1}{3} \\ \frac{-2}{3} \end{bmatrix}\begin{bmatrix} 15 \end{bmatrix} \Rightarrow \begin{bmatrix} 16 \\ -1 \\ -7 \end{bmatrix} - \begin{bmatrix} 10 \\ 5 \\ -10 \end{bmatrix} \Rightarrow \begin{bmatrix} 6 \\ -6 \\ -3 \end{bmatrix}$

$A = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} \\ \frac{2}{3} & \frac{-2}{3} \end{bmatrix}\begin{bmatrix} 6 & 6 & -9 \\ 0 & 9 & 15 \\ 0 & 0 & \end{bmatrix}$

Step3:

$r_{11} = \|a_1\| = \sqrt{36 + 36 + 9} = 9$; $q_1 = \frac{a_1}{\|a_1\|} = \frac{1}{9}\begin{bmatrix} 6 \\ -6 \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ \frac{-2}{3} \\ \frac{1}{3} \end{bmatrix}$; $R_{12} = {q_1}^T A_2 = \begin{bmatrix} \frac{2}{3} & \frac{-2}{3} & \frac{1}{3} \end{bmatrix} 0 = 0$;

$A_2 - q_1 R_{12} = 0$

$$A = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & \frac{-2}{3} \\ \frac{2}{3} & \frac{-2}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 6 & 6 & -9 \\ 0 & 9 & 15 \\ 0 & 0 & 9 \end{bmatrix}$$

9.9 Applications of QR

Let us consider the two typical applications (when A is square)

Solution to $Ax = b$

$$Ax = b$$

$$QRx = b$$

$$Rx = Q^T b$$

This requires a matrix vector product followed by solving a triangular system of equations.

Inverse of A We solve the system of equations $AX = I$ or we solve $Ax_i = e_i$.

Question: Find the computational complexity in both the cases.

9.10 Factorization using SVD

Singular value decomposition is a very popular factorization scheme with many applications. The singular value decomposition (SVD) is a factorization of a real or complex matrix. It has many useful applications in signal processing, statistics and optimization.

Formally, the singular value decomposition of an $m \times n$ real or complex matrix M is a factorization of the form $M = UDV^T$,

- U is $m \times n$, $U^T U = I_n$, D is diagonal $n \times n$ and V is $V^T V = VV^T = I_n$.

(Alternatively one may also look at U as an $m \times m$ real or complex unitary matrix, D is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V^T (the conjugate transpose of V, or simply the transpose of V if V is real) is a $n \times n$ real or complex unitary matrix.)

The diagonal entries D_{ii} of D are known as the singular values of M. The m columns of U and the n columns of V are called the left-singular vectors and right-singular vectors of M, respectively.

Note that $U^T U = I$ and $V^T V = VV^T = I$

Computational cost of SVD is $2mn^2 + 11n^3$.

The singular value decomposition and the eigen decomposition are closely related. Namely:

- The left-singular vectors of M are eigenvectors of MM^T.
- The right-singular vectors of M are eigenvectors of $M^T M$.
- The non-zero singular values of M (found on the diagonal entries of D) are the square roots of the non-zero eigenvalues of both $M^T M$ and MM^T.

Applications that employ the SVD include computing the pseudoinverse, least squares fitting of data, matrix approximation, and determining the rank, range and null space of a matrix.

$$M = \sum_i D_i u_i v_i^T$$

9.11 Computing Inverse using SVD

Let us assume $A = UDV^T$

$$A^{-1} = VD^{-1}U^T$$

D^{-1} is easy to calculate since it is diagonal in n flops.

If A is singular, one can find the approximation by discarding the zero elements. $D_i^{-1} = \frac{1}{D_i}$ is $D_i > t$ and zero otherwise.

One can solve $Ax = b$ with the help of this inverse.

Consider a set of homogeneous equations $Ax = 0$. Any vector x in the null space of A is a solution. Hence any column of V whose corresponding singular value is zero is a solution

9.12 Lease Square Minimization of $Ax = b$

9.13 Solving Homogeneous System of Equations $Ax = 0$

9.14 Additional Examples

9.14.1 Example

Demonstrate that cholesky factorization can be done in $\frac{1}{3}n^3$operations

Sol: Let A is $n \times n - matrix$ which Positive Definite Matrix.

Then generalized cholesky factorization can be done as follows

$$\begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & O \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} l_{11} & {L_{21}}^T \\ O & {L_{22}}^T \end{bmatrix}$$

$a_{11} = {l_{11}}^2 \Rightarrow l_{11} = \sqrt{a_{11}}$

$A_{21} = L_{21}l_{11} \Rightarrow L_{21} = \frac{A_{21}}{l_{11}}$

$A_{22} = L_{21}{L_{21}}^T + L_{22}{L_{22}}^T$

$A_{22} - L_{21}{L_{21}}^T = L_{22}{L_{22}}^T$

Recursive Algortihm:

Calculate the first column of L: $l_{11} = \sqrt{a_{11}}$ and $L_{21} = \frac{A_{21}}{l_{11}}$

Compute Cholesky factorization $A_{22} - L_{21}{L_{21}}^T = L_{22}{L_{22}}^T$

Time or number of operations (flops) required to complete algorithm for n as order

$T(n) = 1 + (n-1) + 2(n-1)^2 + T(n-1)$ (By Master's theorem)

$T(n) = aT(n-1) + O(n^2)$ $\Rightarrow$it requires $O(n^3)$flops

Explanation:

1. calculation of l_{11}takes 1 flop in each recursion
2. calculation of L_{21} takes $(n-1)$ flops if current matrix order is n. So upto now each recursion takes n flops
3. calculation of $L_{22}{L_{22}}^T$ i.e., $A_{22} - L_{21}{L_{21}}^T$ can be done in $2(n-1)^2$flops, if current matrix order is n. So each recursion takes $n + 2(n-1)^2$flops approximately.

So total flops required for $LL^T - factorization$ or Cholesky factorization of $n \times n - matrix$ is

$[n + (n-1) + \ldots 1] + 2(n-1)^2 + 2(n-2)^2 + \ldots + 1$

$\Rightarrow [1 + 2 + \ldots + n] + 2[1 + 4 + \ldots + (n-1)^2]$ (Sum of n natural numbers and 2*(Sum of squares of $(n-1)$ natural numbers))

$\Rightarrow [\frac{n(n+1)}{2}] + 2[\frac{n(n-1)(2n-1))}{6}] \approxeq \frac{1}{3}n^3$

9.14.2 Example

Suggest an efficient algorithm to compute $Z = (I + A^{-1} + A^{-2} + A^{-3})b$. Analyze the complexity/flops

Solution:

$Z = (I + A^{-1} + A^{-2} + A^{-3})b$

$\Rightarrow Ib + A^{-1}b + A^{-2}b + A^{-3}b$

$\Rightarrow$Let $w = A^{-1}b$ and ;$x = A^{-2}b$;$y = A^{-3}b$

$\Rightarrow Ib + A^{-1}b + A^{-1}(A^{-1}b) + A^{-1}(A^{-2}b)$

$\Rightarrow Ib + A^{-1}b + A^{-1}w + A^{-1}x$

Consider$w = A^{-1}b$

$\Rightarrow Aw = b$

$\Rightarrow$Let $A = LU$ ($LU - decompostion$ requires $\frac{2}{3}n^3$ flops.)—-(1)

$\Rightarrow Aw = LUw = b$

$\Rightarrow$Let $Uw = v$ substitute in previous in previous equation

$\Rightarrow Lv = b$ (Solve v by forward substitution. n^2flops required)

$\Rightarrow Uw = v$ (Solve w by backward substitution. n^2flops required)

Total $\frac{2}{3}n^3 + 2n^2$ flops are required to solve linear equation:$Aw = b$.

Conider $x = A^{-1}w$

Already we found $LU - decompostion$ of A from (1)

$\Rightarrow Ax = LUx = w$

$\Rightarrow$Let $Ux = p$ substitute in previous in previous equation

$\Rightarrow Lp = b$ (Solve p by forward substitution. n^2flops required)

$\Rightarrow Ux = p$ (Solve x by backward substitution. n^2flops required)

Total $2n^2$ flops are required to solve linear equation:$Ax = w$.

Conider $y = A^{-1}x$

Already we found $LU - decompostion$ of A from (1)

$\Rightarrow Ay = LUy = x$

$\Rightarrow$Let $Uy = p$ substitute in previous in previous equation

$\Rightarrow Lp = b$ (Solve p by forward substitution. n^2flops required)

$\Rightarrow Uy = p$ (Solve y by backward substitution. n^2flops required)

Total $2n^2$ flops are required to solve linear equation:$Ay = x$.

Addition of 4 $n \times 1 - matrices$ take $3n$ flops.

Therefore total $\frac{2}{3}n^3 + 6n^2 + 3n$ flops required in efficient manner.

9.14.3 Exercise

Consider the set of linear equations

$$(D + uv^T)x = b$$

where u, v, and b are given n-vectors, and D is a given diagonal matrix. The diagonal elements of D are nonzero and $u^T D^{-1} v \neq -1$.

1. What is the cost of solving these equations using the following method?
   (i) First calculate $A = (D + uv^T)$
   (ii) Then solve Ax = b using the standard LU method
2. Compute the inverse of the above A matrix using suitably efficient algorithm and determine the cost.

9.14.4 Excercise

Calculate the LU factorization without pivoting of the matrix $A = \begin{bmatrix} -3 & 2 & 0 & 3 \\ 6 & -6 & 0 & -12 \\ -3 & 6 & -1 & 16 \\ 12 & -14 & -2 & -15 \end{bmatrix}$. Provide all the steps during calculation.

Chapter 10

Optimization Problems: Least Square and Least Norm

A class of problems in real life falls into least square and least norm. Some of them have nice structure that leads to efficient solutions. Let us look at this class of problems here.

10.1 Introduction

In the last two lectues, we had seen the problem of solving $Ax = b$ when A is non-singular and square. What if $m \neq n$? There are two cases that we discuss today: (i) $m > n$ and (ii) $m < n$.

- Consider the case when $m > n$, i.e., the number of equations is more than number of variables/unknowns. Linear equations with $m > n$ are called over-determined equations. They may not have a solution that satisfy all the equations. However, we explore the "most approximate" (or optimal) solution leading to an optimization problem. This is a problem with many practical interest, since the lack of a consistent solution to the entire set of equations may be due to practical issues like 'noise'. They are often formulated as least square error (LSE) minimization problem or popularly known as least square problem. Least squares (LS) problems are optimization problems in which the objective (error) function is expressed as a sum of squares. They have a natural relationship to model fitting problems, and the solutions may be computed analytically using the tools of linear algebra.
- When $m < n$ the situation is very different. There may be too many solutions that satisfy the too few equations that we have. In such situations our interest is in finding the solution that has some special character, for example, the one with minimum norm.

We first derive the closed form expressions for these two (for L2 norm) and then discuss how computationally efficient and accurrate solutions can be designed.

10.2 Least Square Problem

The least squares problem is defined as the problem of finding a vector x, that minimizes $\|Ax - b\|_2^2$. This problem gets it name from the following equation that define the error (or residual) for the i th equation as:

$$\forall i = 1, 2..., m, \qquad r_i(x) = a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n - b_i.$$

Note that $r_i(x)$ is the ith component of $Ax - b$. Often r_i is called the residual or error and will have some physical significance. Let us come back to our familiar matrix form of the equations.

$$Ax = b$$

Our problem is now to minimize $\|r\|$ such that $Ax + r = b$ or minimize $\|Ax - b\|$

Let us first understand the objective function

$$\|Ax - b\| = [Ax - b]^T[Ax - b] = \ xA^TAx + b^Tb - 2x^TA^Tb$$

This has a quadratic term in x, linear term in x and a constant. (like your familair $ax^2 + bx + c$)

To minimize this objective (function of x), let us do a partial differentiation of this w.r.t. xi and equate to zero. This leads to:

(Note: If you do not know how to differentiate functions of matrices/vectors, read: Tom Minka's technical report on "Old and New Matrix Algebra Useful for Statistics".)

$$\begin{aligned} 2A^TAx - 2A^Tb &= 0 \\ \text{or } A^TAx &= A^Tb \end{aligned} \tag{10.1}$$

By multiplyinf $(A^TA)^{-1}$ on both sides,

$$x = (A^TA)^{-1}A^Tb \tag{10.2}$$

10.3 Efficient Computation of LS solution

Solution by using cholesky Reaggarnging the solution gives:

$$A^TAx = A^Tb \tag{10.3}$$

Note that A^TA is PD and we can use cholesky to solve this.

Question: Prove that A^TA is PD.

Our solution is to form an equation $Cx = d$ with $C = A^TAx$ and $d = A^Tb$. The steps can be summarized as:

1.Compute C=A^TAx and $d = A^Tb$
2.Compute C=LL^T using Cholesky factorization
3.Solve Lw=d forward substitution
4.Solve L^Tx=w backward substitution

Complexity for calculating step 1 to compute C we need mn^2 flops. Note that C is symmetric and we need to compute only half the elements approximately. And to compute d we need $2mn$.
Complexity for calculating step 2 is 1/3 n^3.
Complextity for calculating step 3 is n^2.
Complexity for calculating step 4 is n^2.

Therefore, the total cost will be mn^2+$2mn$+1/3 n^3+n^2+ n^2 which is taken as 1/3 n^3+mn^2

Solution using QR decomposition Given matrix A is written in the form $A = QR$ then

$$\begin{aligned} A^TAx &= A^Tb \\ (QR)^T(QR)x &= (QR)^Tb \\ R^TQ^TQRx &= R^TQ^Tb \\ R^TRx &= R^TQ^Tb \\ Rx &= Q^Tb \end{aligned}$$

Steps for solving least square problems using QR

1.Factorize A as QR
2.Compute w=Q^Tb
3.Solve Rx=w backward substitution

Complexity for calculating step 1 is $2mn^2$.
Complextity for calculating step 2 is $2mn$.
Complexity for calculating step 3 is n^2.

Therefore, the total cost will be $2mn^2$+$2mn$+n^2 which is approximated as $2mn^2$

Solution using SVD Let us recollect SVD.

Given matrix A is written in the form $A = UDV^T$
U is $m \times n$ orhtogonal matrix U^TU=I
D is $n \times n$ Diagonal matirix
V is $n \times n$ orthogonal so V^TV=VV^T=I

$$\begin{aligned} (A^TA)x &= A^Tb \\ [(UDV^T)^T(UDV^T)]x &= ((UDV^T)^Tb \\ [VDU^TUDV^T]x &= (UDV^T)^Tb \\ [VD^2V^T]x &= (VDU^T)b \\ DV^Tx &= U^Tb \end{aligned}$$

We solve this as:

$$\begin{aligned} Dw &= p \\ V^T x &= w \\ x &= Vw \end{aligned}$$

Steps for solving least square problems using SVD

1.Factorizw A=UDV^T using SVD.
2.Compute p=$U^T b$
3.Solve Dw=p
4.Find x=Vw

10.4 Least Norms Problems

Linear equations with $m < n$ are called under-determined. This under-determined system has infinitely many solutions. However, we are seeking a solution x^* such that $||x^*||$ is minimal (or the norm is minimum).

$$\begin{aligned} &\text{Minimize} ||x|| \\ &\text{subject to } Ax = b \end{aligned}$$

is unique and is given by $\hat{x} = A^T(AA^T)^{-1}b$

Verify First we verify and show that this is the solution. We show below how this can be derived using Langrangians below.

1. First we check if $\hat{x}$ satisfies $A\hat{x} = b$

   $$\hat{x} = A^T(AA^T)^{-1}b$$

   substitute $\hat{x}$ in $A\hat{x} = b$

   $$\begin{aligned} &= AA^T(AA^T)^{-1}b \\ &= AA^TA^{-T}A^{-1}b \\ &= AA^{-1}b \\ &= b \end{aligned}$$

   This means that this is one of the possible solutions.
2. Now we show that any other solution of the equation has a norm greater than $\hat{x}$
   suppose x satisfies $Ax = b$

   $$\begin{aligned} ||x|| &= ||\hat{x} + (x - \hat{x})||^2 \\ &= ||\hat{x}||^2 + ||(x - \hat{x})||^2 + 2\hat{x}^T(x - \hat{x}) \end{aligned}$$

   The third term is zero since

   $$\begin{aligned} \hat{x}^T(x - \hat{x}) &= (A^T(AA^T)^{-T}b)^T(x - \hat{x}) \\ &= b^T(AA^T)^{-1}A(x - \hat{x}) \\ &= 0 \end{aligned}$$

   Since $Ax = A\hat{x} = 0$.
   Thus we have:

   $$\begin{aligned} ||x||^2 &= ||(x - \hat{x})||^2 + ||\hat{x}||^2 \\ &= ||\hat{x}||^2 + \text{positive term} \end{aligned}$$

   This implies that $||\hat{x}||^2$ is the minimum.

Derivation

$$Minimize x^T x$$

subject to

$$Ax = b$$

Combining the constraints, Langrangian is

$$L(x, \lambda) = x^T x + \lambda^T [Ax - b]$$

Differntiating with repect to x and λ and equating to zero:

$$2x + A^T \lambda = 0$$

$$Ax - b = 0$$

or

$$x = -\frac{A^T \lambda}{2}$$

Substituting

$$\frac{A(-A^T \lambda)}{2} - b = 0$$

$$\lambda = -2(AA^T)^{-1} b$$

Substituting

$$x = A^T (AA^T)^{-1} b$$

10.5 Efficient Solutions to Least Norm Pronlems

Solution using cholesky If A is $m \times n$ and right invertibe matrix

---

1.Compute C=AA^T
2.Decompose C as C=LL^T
3.Solve Lw=b Forward substitution
4.Solve $L^T z$=w Backward substitution
5.Find $x = A^T z$

Computing step 1 will take nm^2 flops.
Step 2 will take 1/3 m^3.
Step 3 will take m^2.
Step 4 will take m^2 and step 5 will take 2mn.
So the total complexity is given as $1/3m^3 + nm^2$

Solution using QR

$$A^T = QR \tag{10.4}$$

$Q^T Q$=I and R is upper right triangualr matrix with positive diagonal

$$\|\hat{x}\| = A^T (AA^T)^{-1} b \tag{10.5}$$

substitute A^T=QR in above equation

$$= QR(R^T Q^T QR)^{-1} b$$
$$= QR(R^{-1} R^{-T}) b$$
$$= QR^{-T} b$$

First we find $\hat{x}$ by computing $R^{-T} b$ by solving $R^T z = b$ and then multiply this by Q.

Steps to solve least norm problems using QR

1. Compute QR factorixation A^T=QR
2. Solve $R^T z$=b forward substitution
3. Solve x=Qz

for computing step 1 we need $2nm^2$
step 2 we need m^2
step 3 we need 2mn. The total complexity is $2nm^2$

Solution using SVD Remember that SVD factorizes A as UDV^T.

Given matrix A is written in the form A=UDV^T
U is mxn orhtogonal matrix U^TU=I
D is nxn Diagonal matirix
V is nxn orthogonal so V^TV=VV^T=I

$$\hat{x} = A^T(AA^T)^{-1}b$$

We can factorize A^T using SVD and substitute in the above equation. This leads to:

$$\begin{aligned} DU^T x &= Vb \\ Dw &= p \\ U^T x &= w \\ x &= Uw \end{aligned}$$

Steps for solving least square problems using SVD

1.Factorizw $A^T = UDV^T$
2.Compute p=Vb
3.Solve Dw=p
4.Find x=Uw

Cost may be higher. But the stability is superior.

10.6 Basis Pursuit

Consider now another problem

$$\begin{aligned} &\text{Minimize } \|x\|_0 \\ &\text{Subject to: } Ax = b \text{ and } \|x\|_\infty \leq R \end{aligned}$$

Note that $\|x\|_0 = \#\{i|x_i \neq 0\}$

If $\|x\|_0 \leq\leq m$, x is sparse. This is NP-Hard.

Let us first rewrite the problem as MIP.

$$\begin{aligned} &\text{Minimize } 1^T z \\ &\text{Subject to: } Ax = b; |x_i| \leq Rz_i; \text{ and } z_i \in \{0,1\} \end{aligned}$$

Assume we relax z_i from $\{0,1\}$ to $[0,1]$.

$$\begin{aligned} &\text{Minimize } 1^T z \\ &\text{Subject to: } Ax = b; |x_i| \leq Rz_i; \text{ and } 0 \leq z_i \leq 1 \end{aligned}$$

Observing that $z_i = \frac{x_i}{R}$ at the optimum, the problem is equivalent to:

$$\text{Minimize } \frac{\|x\|_1}{R}$$
$$\text{Subject to: } Ax = b;$$

This is similar to replacing L0 norm by L1 norm. Assume we use L1 norm instead of L0 norm.

$$\text{Minimize } \|x\|_1$$

$$\text{Subject to: } Ax = b \text{ and } x \in R^n$$

Question: Can the L1 norm based solution be the same as that of L0? or how bad this can be? Let us wait for some lectures to know the answer.

Equivalent LP

$$\text{Minimize } u_1 + u_2 + \ldots u_n$$
$$\text{Subject to: } Ax = b$$
$$-u \leq x \leq u$$
$$x, u \in R^n \text{ and } u \geq 0$$

10.7 Additional Examples

10.7.1 Example 1

minimize $\|Ax - b\|^2 + \|x\|^2$
solution

$$\begin{aligned}
\text{let } \ y &= \|Ax - b\|^2 + \|x\|^2 \\
&= [Ax - b]^T [Ax - b] + x^T x \\
&= [x^T A^T - b][Ax - b] + x^T x \\
&= x^T A^T Ax - x^T A^T b - b^T Ax + b^T b + x^T x \\
&= x^T A^T Ax + x^T x - x^T A^T b - b^T Ax + b^T b \\
&= x^T A^T Ax + x^T x - 2x^T A^T b + b^T b
\end{aligned}$$

differentiating with respect to x on boths sides

$$\begin{aligned}
&= 2A^T Ax + 2x - 2A^T b \\
2A^T b &= 2A^T Ax + 2x \\
A^T b &= (A^T A + I)x
\end{aligned}$$

solving above problem using cholesky

1.compute B=$(A^T A + I)$
2.Choeskly of B as LL^T
3.$LL^T x = A^T b$
4.$Lw = A^T b$
5.$L^T x = w$

---

solving above problem using QR - complexity

1.compute B=$(A^TA + I$
2. Bx=A^Tb
3.$B^T = QR$
4.$R^Tz = b$
5.x=Qz -

10.7.2 Example 2

Suggest an efficient algorithm to minimize
$\|Ax - b_1\|^2 + \|Ax - b_2\|^2$ given matrix A is mxn and b_1 and b_2 are two m vectors
Solution:

$$y = [Ax - b_1]^T[Ax - b_1] + [Ax - b_2]^T[Ax - b_2]$$
$$y = [x^TA^T - b_1^T][Ax - b_1] + [x^TA^T - b_2^T][Ax - b_2]$$
$$y = x^TA^TAx - x^TA^Tb_1 - b_1^TAx + b_1^Tb + x^TA^TAx - x^TA^Tb_2 - b_2^TAx + b_2^Tb_2$$

differentiating with respect to x on both sides

$$4A^TAx - 2A^Tb_1 - 2A^Tb_2 = 0$$
$$2A^TAx = A^Tb_1 + A^Tb_2$$

substitute A=QR-2mn^2

$$2(QR)^T(QR)x = (QR)^Tb_1 + (QR)^Tb_1$$
$$2R^TQ^TQRx = R^TQ^Tb_1 + R^TQ^Tb_2$$
$$R^TRx = \frac{R^TQ^T(b_1 + b_2)}{2}$$
$$Rx = \frac{Q^T(b_1 + b_2)}{2}$$

solving above problem using cholesky

1.let w=$\frac{Q^T(b_1+b_2)}{2}$ -$2mn + m$
2.Rx=w -n^2

so total complexity is $2mn^2 + 2mn + m + n^2$

So total complexity is $2mn^2 + 2mn + m + n^2$

10.7.3 Excercise

$$min(\|x - x_0\|^2) \tag{10.6}$$

Such that

$$Ax = b; (m < n) \tag{10.7}$$

TODO Notes (i) Fix flops for SVD (ii) double check costs.

Chapter 11

Constrained Optimization: Lagrange Multipliers and KKT Conditions

This chapter introduces two fundamental concepts in optimization: (i) Lagrange Multipliers and (ii) KKT Conditions, in the context of constrained optimization.

11.1 Introduction

Constrained Optimization vs Unconstrained Optimization

11.2 Lagrange Multipliers

11.3 KKT Conditions

11.4 Problems

Chapter 12

Eigen Value Problems in Optimization

A class of optimization problems lead to solutions that are eigen vectors of certain matrices. Let us study some such problems here.

12.1 Eigen Values and Eigen Vectors

In linear algebra, an eigenvector or characteristic vector of a square matrix is a vector that does not change its direction under the associated linear transformation. That is, if

$$Ax = \lambda x$$

then x is the eigen vector of A and λ is the correposining eigen value. Note that an $n \times n$ matrix A may have at max n eigen vectors and eigen values. Eigen values can also be zero.

Geometrically, an eigenvector corresponding to a real, nonzero eigenvalue points in a direction that is stretched by the transformation and the eigenvalue is the factor by which it is stretched.

Another intuitive explanation of eigen vectors is as follows. We know that when a vector gets multipled by a matrix (a linear transformation), the vector changes its direction. However, certain vectors do not change their direction. They are called eigen vectors. i.e., $Ax = \lambda x$. There is only a scale change characterized by λ on multiplication. Such vectors x are the eigen vectors and the corresponding λ, the scale factor is the eigen value. If A is an identity matrix, every vector is an eigen vector, and all of them have eigen value of $\lambda = 1$.

12.1.1 Basics and Properties

1. When A is squared, the eigenvectors stay the same and eigenvalues get squared.
   Prove this.
2. The product of eigen values is determinant.
   Prove this.
3. Sum of eigen values is Trace
   Prove this.
4. Symmetric matrcies have real eigen values
   Prove/Verify this.

12.1.2 Numerical Computation

To compute the eigen values and eigen vactors, we start with $Ax = \lambda x$. or $(A - \lambda I)x = 0$. If this system has to have a non trivial solution, the determinant of $A - \lambda I$ should be zero.

$$|A - \lambda I| = 0$$

Solving the above for λ gives the eigen values. Substituting this in

$$Ax = \lambda x$$

yield x.

Example: Compute the eigen values and eigen vectors of the following matrix.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

12.1.3 Numerical Algorithms

Note done for S-2016

12.2 Applications in Optimization

A number of problems lead to a formulation of the form $Minimize$ or $Maximize\ x^TAx$ with an additional constraint that $\|x\|^2 = 1$. Often we need the constraing to avoid the trivial cases.

If we create an unconstrained optimization with the help of a lagrangain.

$$Minimize x^TAx - \lambda(x^x - 1)$$

Differentiating with respect to x and equating to zero leads to

$$Ax = \lambda x$$

Or the optima is an eigen vector or the A.

If we substitute this in the objective function,

$$x^TAx = \lambda x^Tx = \lambda$$

If the problem is maximization problem, then we will pick the eigen vector corresponding to the largest eigen value. If the problem is minimization, we will pick the eigen vector corresponding to the smallest eigen value.

Another class of functions lead to a problem of the form:

$$Ax = Bx$$

12.3 Optimization: Application in Line Fitting

In the previous lectures, we had seen the problem of line fitting (as LP, as LSE). However, we always wondered why don't we minimize the orthogonal distance.

Let us assume that we are given N points in 2D i.e., (x_i, y_i). We are interested in finding the equation of a line $ax + by + c = 0$. that minimize the sum or orthohonal disctances. We know that this line passes through the origin.

Let us assume that the points are mean centered. i.e., mean is zero. Now the line equation can be written as $ax + by = 0$. With no loss in generality we can assume that the vector $u = [a, b]^T$ is normalized such that the norm is unity.

Assume that the data is arranged in a data matrix M as $N \times 2$. We are interested in finding

$$Minimize \|Mu\|^2 \text{ such that} \|u\| = 1$$

Let us take an SVD of M as UDV^T.

$$\|Mu\|^2 = [Mu]^T[Mu] = u^TM^TMu = u^TVDU^TUDV^Tu = u^TVD^2V^Tu$$

Since V is an orthogonal matrix, multiplication by V is not to change the length of u. Let us assume $v = Vu$. Then the problem is now.

$$Minimize v^TD^2v \text{ such that} \|v\| = 1$$

This is nothing but the square of the largest singular value.

12.3.1 Relationshp to SVD

Discussed along with the SVD. When A is not a square matrix, eigen vectors of A^TA and AA^T are related to the U and V in the SVD.

12.4 Application in solving $Ax = 0$

Consider the problem of $Minimize\ ||Ax||$ where A is a $m \times n$ matrix with $m > n$. This immediately lead to a minimization problem of the form x^TA^TAx with the unit norm constraint on x. This leads to the eigen vectors of A^TA.

However which eigen vector to pick? one corresponding to the smallest or largest?

12.5 Optimization: Application in PCA

Problem: Given a set of samples $\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_N}$, each with M features $x_i^1, \ldots x_i^M$, Find a new feature representation (such as the jth feature is a linear combination of the original features) as:

$$y_i^1 = \alpha_1^1 x_i^1 + \alpha_2^1 x_i^2 + \ldots + \alpha_M^1 x_i^M$$

$$y_i^2 = \alpha_1^2 x_i^1 + \alpha_2^2 x_i^2 + \ldots + \alpha_M^2 x_i^M$$

$$\ldots\ldots$$

$$y_i^k = \alpha_1^2 x_i^k + \alpha_2^k x_i^2 + \ldots + \alpha_M^k x_i^M$$

$$\mathbf{Y_i} = \mathbf{A}\mathbf{X_i}$$

Where $\mathbf{Y_i}$ is a $k \times 1$ vector, $\mathbf{A}$ is a $K \times M$ matrix and $\mathbf{X_i}$ is a $M \times 1$ vector (usually $k < M$)

Let the original vector $\mathbf{x_j}$ is projected into a new dimension $\mathbf{u_i}$ (basis vector) as $v_{ij} = \mathbf{x_j} \cdot \mathbf{u_i}$.

It is easy to observe that the mean after the projection is same as the projection of the original mean.

$$\bar{\mathbf{x}} \cdot \mathbf{u_i} = \frac{\sum_j \mathbf{x}_j}{N} \cdot \mathbf{u_i} = \frac{1}{N} \sum_j \mathbf{x_j} \cdot \mathbf{u_i} = \frac{1}{N} \sum_j v_j$$

$$\bar{v_i} = \bar{\mathbf{x}} \cdot \mathbf{u_i}$$

or else

$$\bar{v} = \bar{\mathbf{x}} \cdot [\mathbf{u_1}, \ldots, \mathbf{u_k}] = \mathbf{U}\bar{\mathbf{x}}$$

Objective

Let the objective is to maximize the variance after projection. (Or only minimal information is lost.) Let us first find the "best dimension" ($\mathbf{u}$) in this regard i.e.,

$$Max_{\mathbf{u}} \ var(\mathbf{v}) = Max_{\mathbf{u}} \ \sum_i ||\mathbf{v_i} - \bar{\mathbf{v}}||^2$$

$$Max_{\mathbf{u}} \sum_i ||\mathbf{v_i} - \bar{\mathbf{v}}||^2 = \sum_i ||(\mathbf{x_i} - \bar{\mathbf{x}}) \cdot \mathbf{u}||^2$$

$$Max_u \mathbf{u^T} \sum_i [\mathbf{x_i} - \bar{\mathbf{x}}][\mathbf{x_i} - \bar{\mathbf{x}}]^T \mathbf{u} = Max_u, \mathbf{u^T}\Sigma\mathbf{u}$$

An unconstrained optimization of the same could give extremely large $\mathbf{u}$. Therefore, we introduce a constraint $\mathbf{u^T u} = 1$ or the problem is

$$Maximize \ \mathbf{u^T}\Sigma\mathbf{u} - \lambda(\mathbf{u^T u} - 1)$$

$$Maximize \ \mathbf{u^T}\Sigma\mathbf{u} - \lambda(\mathbf{u^T u} - 1)$$

Differentiating with respect to $\mathbf{u}$ and equating to zero.

$$\Sigma\mathbf{u} = \lambda\mathbf{u}$$

Or $\mathbf{u}$ will have to be eigen vector of Σ.

It can be now easily seen that the basis vectors which preserve maximum variance are the sorted (in decreasing order) eigen vectors.

(The largest λ maximizes the objective)

12.6 Optimization: Graph Cuts and Clustering

Let us now look at another interesting application of eigen vectors. Consider a graph $G = (V, E)$.

We are interested in partitioning the graph into two subsets of vertices A and B. Or, we want to find a cut, which is the sum of edges that we need to cut.

This has many applications in clustering. Consider that we are given N points $\{x_1, \ldots, x_N\}$ and we are interested in clustering these points into two clusters. Assume A is an affinity (or some similarity matrix) where A_{ij} is the similarity of x_i and x_j. This is also in a way related to the weight matrix W of the graph. If one needs more intuituition, one can define weight w_{ij} as $e^{\frac{d(x_i,x_j)}{\sigma}}$.

Let w_{ij} be the weight of the edges. We need a cut that partition the vertices into two. It cuts a set of edges.

$$Cut(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

However, this is not very useful in many cases. More useful measure is

$$NCut(A, B) = cut(A, B)(\frac{1}{Vol(A)} + \frac{1}{Vol(B)}$$

with

$$vol(A) = \sum_{i \in A, j \in V} w_{ij}$$

$$vol(A) = \sum_{i \in A} d_i$$

this normalization helps to remove the easy case of cutting an outlier point as the preferred point.

Let $D(i, i) = \sum_j w_{ij}$.

Consider a vector x such that x_i is 1 if i th vertex is in A and -1 if it is in B.

$$Ncut(A, B) = \frac{\sum_{x_i>0; x_j<0} -w_{ij}x_ix_j}{\sum_{x_i>0} d_i} + \frac{\sum_{x_i<0; x_j>0} -w_{ij}x_ix_j}{\sum_{x_i<0} d_i}$$

where x is an N dimensional vector such that $x_i = 1$ is $i \in A$ and $x_i = -1$ is i is in B. And $d_i = \sum_j w_{ij}$

Rewriting in terms of W and D, this is

$$NCut(x) = \frac{(1 + x)^T(D - W)(1 + x)}{k1^T D1} + \frac{(1 - x)^T(D - W)(1 - x)}{(1 - k)1^T D1}$$

Let $y = \frac{1}{2}((1 + x) - b(1 - x))$

$$min_x NCut(x) = \min_y \frac{y^T(D - W)y}{y^T Dy}$$

with the condition that $y^T D1 = 0$.

Minimize the equation using the standard tricks:

$$D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}z = \lambda z$$

where $z = D^{\frac{1}{2}}y$

using Rayleigh quotient, the second smallest eigen vector turns out to be the real valued solution and the solution to the normalized cut problem

12.7 Optimization: Generalized Eigen Value Problem

A very related problem is that of generalized eigen value problem involving two matrices.

$$Ax = \lambda Bx$$

When $B = I$, this problem reduced to a popular equation we had seen in Equation (1).

If we multiply both side by either of B^{-1} or A^{-1}, then we can get the popular simple eigen value problem. However the problem is that the matrices like $A^{-1}B$ need not be symmetric.

A standard trick is to find sqrt of the B (either as $B^{\frac{1}{2}}$) or using cholesky LL^T. i.e., $Ax = \lambda LL^T x$ or $L^{-1}Ax = \lambda L^T x$ or

$$L^{-1}AL^{-T}[L^T x] = \lambda L^T x$$

or

$$A'y = \lambda y$$

where $A' = L^{-1}AL^{-T}$ and $y = L^T x$

12.8 Optimization: Spectral Graph Theory

Note done for S-2016

Chapter 13

Introduction to simplex method

One of the most popular technique for solving an LP is using Simplex Algorithm. Let us understand the theory of simplex.

13.1 Introduction

In the previous lectures, we have studied several formulations of problems as LP and some of the applications of LP. We had also seen how some of the *toy* LPs gets solved on paper. We now describe an efficient way to solve the linear programs. This method is popular and is widely used in many real life problems. This is the well known simplex method. The graphical method to solve linear programs becomes tedious and computationally intensive for large linear programs (when the numer of variables increases). As the number of variables increases, every new constraint adds an exponential number of vertices to be evaluated (*see below*). The graphical method basically requires enumerating and evaluating the objective on all the possible vertices or *extreme points* of the feasible region described by the constraints of the linear program. This makes the algorithm unsuitable for practical use in large linear programs. The SIMPLEX method describes a procedure to solve these linear programs efficiently. In this lecture, we start with a brief conceptual introduction to the simplex method, and then proceed to describe more formal details of the algorithm. In the next couple of lectures, we will see how simplex method can be understood/derived, and a tablau based solution can be used to solve it on paper.

We had argued in the past that the optima is an extreme point of the convex ploygon (or feasible region) formed by the constraints. Based on this, let us state a naive version (Ver 0) of the simplex algorithm. This is a simplified and intuitive description of the simplex algorithm.

1. Form the feasible region, and start with any one of the extreme points.
2. Repeat:
    - Move to one of the neighbouring extreme points which has a better objective value.
    - If no such point exists terminate algorithm with the current extreme point as the optima

There are many important questions to be answered at this stage, both theoretical and practical. For example, will this algorithm converge? Will this lead to a global optima? How to move from one extreme point to another?

13.1.1 Remark on LP as a Convex Optimization

The procedure start with *any one* of the extreme points. Then it move in *one specific direction* that improves the ojective. *Those who are familiar with the gradient descent style optimization may argue that we should pick the direction of greatest change in the cost. That is fine. But there are more important aspects to look into.* However, if the objective function is convex (which is so in our case), it does not really matter. Eventually we will reach the same extreme point (local as well as global optima). Given that linear programming problem is a convex optimization problem, the solution found would be optimal. The above algorithm keeps finding a better solution till it reaches a local minima (i.e none of its neighbours have a better solution). Therefor by the property of convex optimization problems, this local optima is also the global optima that we are looking for.

The above algorithm aims at finding the best solution by exploiting the advantages of the convex optimization. A linear program is a convex optimization problem. A convex optimization problem optimizes a convex function over a convex region/set.

A set S (eg. $\subset \mathbb{R}^n$) is **convex** if, for any $\mathbf{x}_1, \mathbf{x}_2 \in S$, and for $\theta \in [0, 1]$,

$$\mathbf{x_3} = \theta \mathbf{x}_1 + (1 - \theta)\mathbf{x}_2$$

is also in S. This simply implies that the line joining $\mathbf{x_1}$ and $\mathbf{x_2}$ is also within the set.

Since our method search over a set of extreme points, let us see how many extreme points are present in an LP?. In fact, there are too many.

The feasible region for a linear programming problem, given by, $\{x|\mathbf{Ax} \leq b$ where, A is an $m \times n$ matrix, c $\in \mathbb{R}^n$ and b $\in \mathbb{R}^m$, $m < n$, is represented by a polyhedron. A polyhedron has a finite number of extreme points or vertices which are bounded by

$$^mC_n = \frac{m!}{n!(m-n)!}$$

which represents m equations and choosing n variables if $m > n$. As m and n increase, the value of ^mC_n increases. Hence, for a general L.P. problem, the number of vertices can be very large. Let us not think of finding all of them and evaluating the objective on all of them.

13.1.2 Remark on Computational Complexity

1. Simplex is a generalization of triangles. You may see many related aspects of the simplex on

   `https://en.wikipedia.org/wiki/Simplex`

2. TODO: More details on the vertices, faces and complexity.

13.1.3 Historical Notes on Algorithms that solve LP

- Simplex. Invented by George Dantzig in 1947 (Stanford University)
- L. G. Khachian introduced an ellipsoid method (1979) that seemed to overcome some of the simplex method's limitations. Complexity: $O(n^6)$. Disadvantage – runs with the same complexity on all problems
- Narendra K. Karmarkar of AT&T Bell Laboratories proposed in1984 a new very efficient interior-point algorithm. Complexity: $O(n^{3.5})$. In empirical tests it performs competitively with the simplex method.

13.2 Standard Slack Form

The standard Linear Programming (LP) problem consists of a linear objective function, which is subject to a certain number of linear constraints. In general, it is represented as:

$$\begin{array}{rl} \text{Minimize:} & \mathbf{c^T x} \\ \text{such that} & \mathbf{Ax} \leq b;\ x \geq 0 \end{array}$$

where, A is an $m \times n$, matrix, c $\in \mathbb{R}^n$ denotes the coefficients of the objective function and b $\in \mathbb{R}^m$, is part of the constants. We are given A, b and c. We need to find the optimal x.

In order to use the simplex algorithm, we require the linear program to be cast in a specific format known as the standard slack form. Therefore, conversion of the above inequality into an equality is now required. This is done by adding a variable to each of the constraints. Hence, the LP problem becomes

$$\begin{array}{rl} \text{Minimize:} & \mathbf{c^T x} \\ \text{such that} & \mathbf{Ax} = b \\ & \mathbf{x} \geq 0 \end{array}$$

For example, an inequality like

$$2x_1 + 3x_2 \leq 4$$

is converted to an equality by adding the variable x_3. Thus, the inequality changes to

$$2x_1 + 3x_2 + x_3 = 4$$

with $x_3 \geq 0$. The new variable x_3 is also called the slack variable.

The slack form consists of a linear program's constraints expressed as $Ax = b$. That is without the use of any inequality expressions. The slack form of the linear program is expected in the following format :

$$\begin{array}{rl} \text{Minimize:} & c^T x \\ \text{Subject To:} & Ax = b \\ & x \geq 0 \end{array}$$

The conversion of the linear program to the slack form is done by introducing additional slack variables, for each inequality constraint in the original standard form expression. For example, if the original linear program has m constraints with n variables. The converted slack form's constraints become:

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + \mathbf{x_{n+1}} & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + \mathbf{x_{n+2}} & = & b_2 \\ & \ldots & \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n + \mathbf{x_{n+m}} & = & b_m \end{array}$$

Therefore the new slack form constraints $Ax = b$, has an A with dimensions $m \times (m + n)$. Due to the additional m variables introduced. These new variables are known as slack variables. Their values are intended to be set based on the other variables in order to make up for the slack in an inequality, thereby allowing us to express the LP now as an equality. This slack form will be very important for our formulation of the simplex method.

13.3 Simplex as Search over Basic Feasible Solutions (BFS)

We had seen the significance of the extreme points (vertices of the convex polygon defined by $Ax \leq b$) in understanding the solution to the LP. Let us see what is the equivalent when the constraints are in the form of $Ax = b$.

- A **basic solution** to a system of m linear equations in n unknowns ($n \geq m$) is obtained by setting $n - m$ variables to 0 and solving the resulting system to get the values of the other m variables.
- The variables set to 0 are called **nonbasic variables**; the variables obtained by solving the system are called **basic variables**.
- A basic solution is called **feasible** if all its (basic) variables are nonnegative.

We know that the algorithm of enumerating vertices to find the minimum is not efficient and attractive. Instead, an alternative algorithm as below can be used (we also saw the same in the last section).

Simplex Ver 0 The two intuitive steps of the algorithm are:

1. Start from any vertex as a possible solution.
2. If a neighbouring vertex is better, solution is updated.
3. Repeat step2 until there is no better vertex.

13.3.1 Basic Feasible Solution

We now assume that we have the linear program expressed in slack form as $Ax = b$, where A has m constraints and n variables. Here in the slack form, due to introduced variables, we have $m \leq n$ and therefore $Ax = b$ has multiple valid solutions. In such a situation if we have two possible solutions x_1 and x_2. Then all x of the form:

$$x = \alpha x_1 + (1 - \alpha)x_2, \text{where } 0 \leq \alpha \leq 1$$

are solutions to the LP. This will be an important property that we use in the simplex method to jump from one solution to the another (till we reach the optimum). Now for the LP problem, the extreme points or vertices are obtained when the column vectors A_i obtained with corresponding non-zero basic variables are linearly independent. Such an x with a valid solution to the LP is known as a Basic Feasible Solution or BFS. A linear program will have multiple BFS, but if we are able to obtain at least one BFS (the initial BFS), then we can say that there must exist an optimal BFS. With these definitions we now look into the simplex method.

Let B be the set of m basic variables. i.e., $B \subseteq \{1 \dots n\}$ be a subset of the n variables. Let us set $x_i = 0, \forall i \notin B$. We represent B as a set of indices of the variables that are selected. Also we use the notation $B(i)$ to represent the i th element of the set.

Now, assume $\mathbf{A}$ to be a matrix of rank m. We select m columns from $\mathbf{A}$, then

$$\left[\mathbf{A_{B(1)}}, \mathbf{A_{B(2)}}, \cdots \mathbf{A_{B(m)}}\right] \mathbf{x}_B = b \tag{13.1}$$

where $\left[\mathbf{A_{B(1)}}, \mathbf{A_{B(2)}}, \cdots \mathbf{A_{B(m)}}\right]$ is an $m \times m$ matrix, x_B is an $m \times 1$ vector and b is an $m \times 1$ vector. If

$$\bar{\mathbf{B}} = \left[\mathbf{A_{B(1)}}, \mathbf{A_{B(2)}}, \cdots \mathbf{A_{B(m)}}\right]$$

then,

$$\begin{aligned} &\bar{\mathbf{B}}\mathbf{x}_B = b \\ \implies &\mathbf{x}_B = \bar{\mathbf{B}}^{-1}b,\ x_i = 0,\ i \notin B \end{aligned} \tag{13.2}$$

The elements of the vector x can be now taken from x_B and the rest zero.

Now, the feasible region defined by the LP problem, $\mathbf{Ax} = b,\ x_i \geq 0$ is a convex polyhedron. So, x is a vertex, if and only if the subset of the n variables, i.e. the column vectors $\mathbf{A_i}$, of the matrix $\mathbf{A}$, corresponding to non-zero entries of $x(x_i \neq 0)$ are linearly independent. Such an x is called a **Basic Feasible Solution (BFS)**. So, in this case, $B \subseteq \{1 \dots n\}$ is the basic variables. And $\bar{B}$ is a non-singular matrix. For a given linear programming problem, there can be many basic feasible solutions possible. If an optimal solution exists for an L.P., then one of the **BFS** is optimal.

13.4 Simplex Algorithm - Ver 1

Now we can create various $\bar{B}$ matrices and find the respective x. (There are such ${}^{m}C_{n}$ matrices. However, this is not attractive. Our strategy is the following.

Simplex - Ver1

1. Start with a **BFS**.
2. Repeat
    - If a better solution exists, move to that

Its practical implementation is done by finding the matrices $\bar{B}_1 \to \bar{B}_2 \to \bar{B}_3 \to \bar{B}_4 \to \bar{B}_5 \dots$ Where finding a new $\bar{B}$ is done by removing a column and adding a new column. Therefore, the problem of simplex will soon get reduced to finding the departing column (or variable) and finding the entering column (or variable). Often, the first vertex is found easily as $B = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

13.4.1 An Intuitive Version

1. Start with an Initial BFS.
2. Repeat:
    - If we have a better BFS , move to that
    - else stop with the present BFS and report it is as the optimum.

Often the initial BFS is chosen with $\bar{B} = I$. That is by setting all the slack variables to be Basic variables (non-zero), while the remaining originally existing variables are classified as non-basic and are by default set to 0. (This is equivalent to the origin in your convex plogon defined by the inequalities.) This usually (not always) provides us a simple to obtain BFS as $x_{B(i)} = b_i$ as the rest of the variables in the constraint are non-basic and zero.

We design the simplex algorithm such that in each iteration of the algorithm, exactly one of the Non-Basic variables become a basic variable and a basic variable leaves to become a non-basic variable.

Now we proceed to formalize the process by witch we switch between two Basic Feasible Solutions, making sure that it still satisfies the original LP's problem.

13.4.2 Moving across BFSs - Cost difference

Let d be a difference vector, and $\theta \in [0, 1]$ be a scalar. The vector d takes us from one solution to another as $x_{\text{new}} = x + \theta d$.

Let $d_j = 1$, for some $j \notin B$. $\forall i \notin B - \{j\}, d_i = 0$ i.e., only one variable is moving into B (and one moving out).

$$
\begin{aligned}
Ax_{\text{new}} = Ax = b \implies A(x + \theta d) = Ax = b \implies Ad = 0 \\
\therefore \sum_{i=1}^{m} d_{B(i)} A_{B(i)} + A_j = 0 \\
\implies \bar{B} d_B + A_j = 0 \\
\implies d_B = -B^{-1} A_j
\end{aligned}
$$

Let us calculate the cost difference in the objective for a given choice of j.

$\bar{C_j} = C_j - C_B^T B^{-1} A$, here $\bar{C_j}$ is the difference in cost due to j th variable becoming basic. We compute it for all the variables.

$$\bar{C} = [\bar{C_1}\bar{C_2} \dots \bar{C_n}]$$

If $\bar{C} \geq 0$, (i.e., all the elements are positive or objective can not be further minimized), then it means that our current BFS is the optimum. Otherwise we choose one of the j from $\bar{C}$, such that $\bar{C_j} < 0$. We can have some nice heuristics in picking the best j out of many possible ones. Such heuristics may work in many situations, but not always. In any case, we know that the problem is convex!!.

An important question not yet answered is the choice of θ. How do we fix theta? If our theta is small, we move only by a small amount. If we take a large θ, we move out of the feasible region. The trick is finding the largest θ such that we still remain in the feasible region. We show below, how to find the θ such that we go as far as possible without violating the constraints. This leads to a new dimension/variable l that exit from the basic variable set. In short our objective is to find j that enters and l that exits in every iteration.

13.4.3 Simplex Ver 1: A more formal version

1. Start with an initial BFS, (Set B, etc.)
2. Repeat:
   - Calculate $\bar{C}$, if ≥ 0 , then stop repeating.
   - else, select a j from $\bar{C}$, such that $\bar{C_j} < 0$.
   - set $d_B = -B^{-1}A$
   - Find l and θ , such that:
     - θ is continuously increased till,
     - the first l , such that on increasing θ , Basic variable x_l becomes zero. i.e :

       $$x_l + d_l\theta = 0$$

   - l exits B, and j enters B.

     $$B = B - \{l\} + \{j\}$$
3. The current BFS is the optima

13.5 Examples

Example 5. Consider an example to obtain an optimal solution by checking all possible basic feasible solutions.

$$\begin{array}{lrl} \text{Minimize:} & -x_1 - x_2 & \\ \text{such that} & -x_1 + x_2 & \leq 1 \\ & x_1 & \leq 3 \\ & x_2 & \leq 2 \\ & x_1,\ x_2 & \geq 0 \end{array}$$

The inequalities are converted to equalities by adding variables to each of the constraints. This results in

$$\begin{array}{lrcl} \text{Minimize:} & -x_1 - x_2 & & \\ \text{such that} & & & \\ & -x_1 + x_2 + x_3 & = & 1 \\ & x_1 + x_4 & = & 3 \\ & x_2 + x_5 & = & 2 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 & & \end{array}$$

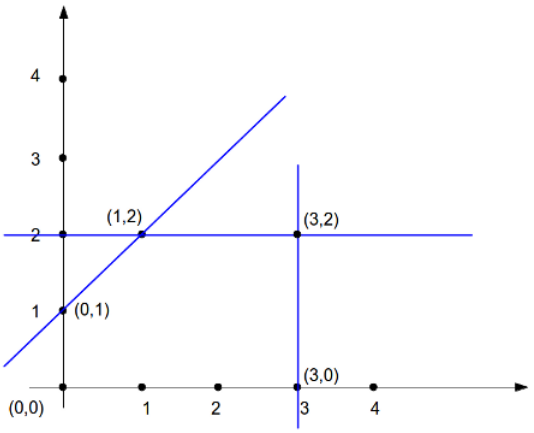


Figure 13.1: The constraints on a 2D plane. One can see the extreme points of the original problem. Also notice the relationship to the Basic Feasible Solutions.

Here,

$$\mathbf{A} = \begin{bmatrix} -1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

Since,

$$\mathbf{A}\mathbf{x} = b \tag{13.3}$$

Therefore,

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

Initially, the basic feasible solution is $\mathbf{x} = [0, 0, 1, 3, 2]$. As the 3^{rd}, 4^{th} and the 5^{th} columns of the matrix $\mathbf{A}$ form an identity matrix ($\mathbf{I}$), $B = \{3, 4, 5\}$. The objective function, $z = 0$.
Rearranging the equations so as to write them in terms of x_3, x_4 and x_5,

$$\begin{aligned} x_3 &= 1 + x_1 - x_2 \\ x_4 &= 3 - x_1 \\ x_5 &= 2 - x_2 \\ \hline z &= -x_1 - x_2 \\ \implies z &= 0 \end{aligned}$$

x_2 can increase upto 1 while keeping $x_1 = 0$. Thus, the equations transform as,

$$\begin{aligned} x_2 &= 1 + x_1 - x_3 \\ x_4 &= 3 - x_1 \\ x_5 &= 1 - x_1 + x_3 \\ \hline z &= -x_1 - x_2 \\ &= -x_1 - (1 + x_1 - x_3) \\ &= -1 - 2x_1 + x_3 \\ \implies z &= -1 - 2(0) + 0 \\ &= -1 \end{aligned}$$

In this case, 2 enters and 3 exits. Therefore, $\mathbf{x} = [0, 1, 0, 3, 1]$, $B = \{2, 4, 5\}$ and $z = -1$.

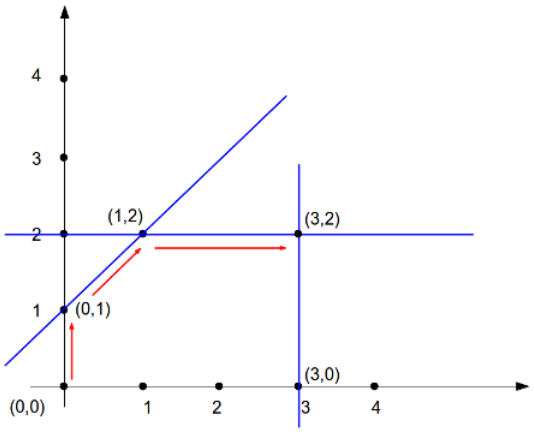


Figure 13.2: Navigation of the vertices of the feasible region.

Figure 13.1 shows the navigation through the basic feasible solutions to reach an optimal solution.

Now, x_1 can be increased to 1. So, the equations transform as,

$$\begin{aligned} x_1 &= 1 + x_3 - x_5 \\ x_2 &= 2 - x_5 \\ x_4 &= 2 - x_3 + x_5 \\ \hline z &= -1 - 2x_1 + x_3 \\ &= -1 - 2(1 + x_3 - x_5) + x_3 \\ &= -3 - x_3 + 2x_5 \\ \implies z &= -3 - 0 + 2(0) \\ &= -3 \end{aligned}$$

In this case, 1 enters and 5 exits. Therefore, $\mathbf{x} = [1, 2, 0, 2, 0]$, $B = \{1, 2, 4\}$ and $z = -3$.

It can be seen that z can be minimized by increasing x_3. So, the equations transform as,

$$\begin{aligned} x_1 &= 3 - x_4 \\ x_2 &= 2 - x_5 \\ x_3 &= 2 - x_4 + x_5 \\ \hline z &= -x_1 - x_2 \\ &= -(3 - x_4) - 2 - x_5 \\ &= -5 + x_4 + x_5 \\ \implies z &= -5 + 0 + 0 \\ &= -5 \end{aligned}$$

In this case, 3 enters and 4 exits. Therefore, $\mathbf{x} = [3, 2, 2, 0, 0]$, $B = \{1, 2, 3\}$ and $z = -5$. z cannot be minimized further. Hence, the final minimized value of z is -5.

Example 6. Find a sequence of **BFS** and arrive at an optimal point.

$$\begin{aligned} \text{Minimize:} \quad & z = x_1 + x_2 \\ \text{such that} \quad & x_1 + 5x_2 \leq 5 \\ & 2x_1 + x_2 \leq 4 \\ & x_1, x_2 \geq 0 \end{aligned}$$

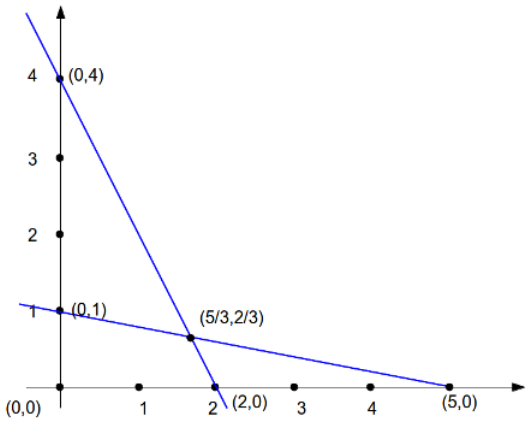


Figure 13.3: Graphical representation.

The graphical representation of the given problem is as follows:

Variables are introduced to convert the inequalities to equalities.

$$\begin{array}{lrrrrl} \text{Minimize:} & & z = x_1 + x_2 & & & \\ \text{such that} & & & & & \\ & x_1 + & 5x_2 + & x_3 & & = 5 \\ & 2x_1 + & x_2 + & & x_4 & = 4 \\ & & x_1, x_2 \geq 0 & & & \end{array}$$

So,

$$\mathbf{A} = \begin{bmatrix} 1 & 5 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

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

Therefore,

$$\begin{bmatrix} 1 & 5 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

$\mathbf{x} = [0, 0, 3, 4]$. If $B = \mathbf{I}$, then $B = \{3, 4\}$. After rearrangement of the equations,

$$\begin{array}{rl} x_3 & = 5 - x_1 - 5x_2 \\ x_4 & = 4 - 2x_1 - x_2 \\ \hline z & = x_1 + x_2 \\ \implies z & = 0 \end{array}$$

Since x_1, x_2 both are positive, increasing them will increase the value of z. So, the optimal solution is reached, with $z = 0$.

13.6 Additional Problems

1. Find a sequence of BFS to arrive at the optimal point for the following Linear program:

$$\begin{aligned} \text{Minimize :} \quad & z = -x_1 - x_2 \\ \text{Subject to :} \quad & \\ & x_1 + 10x_2 \leq 5 \\ & 4x_1 + x_2 \leq 4 \\ & x_1, x_2 \geq 0 \end{aligned}$$

2. Find a sequence of BFS to arrive at the optimal point for the following Linear program:

$$\begin{aligned} \text{Minimize:} \quad & z = 2x_1 + 3x_2 & \\ \text{such that} \quad & 5x_1 + 25x_2 & \leq 40 \\ & x_1 + 3x_2 & \geq 20 \\ & x_1 + x_2 & = 10 \\ & x_1, x_2 & \geq 0 \end{aligned}$$

Chapter 14

More on Simplex

14.1 Introduction

In the last lecture, we have seen a basic version of Simplex. That method involved moving from one extreme point of the feasible polyhedra to another, and finally stopping at the extreme point corresponding to optimal solution, when no such adjacent movement leads to cost minimization. We had seen this in an example. We know how to apply this method, without getting into much of the theory corresponding to it. In this lecture we develop fomal mathematical details that guides the Simplex method, so that we are able to apply this method with more confidence.

The simplex method is based on this fact that if an optimal solution exist then there is basic fesible solution(BFS) that is optimal. It searches for an optimal solution by moving from one basic feasible solution to another, along the edges of the feasible set. We make sure that we move always in a cost reducing direction. Eventually, a special basic feasible solution is reached at which none of the available edges leads to a cost reduction. Such a basic feasible solution is optimal and the algorithm terminates.

To start the simplex problem we need to know at least one basic feasible solution. Simplex often starts with $\bar{B} = I$.

14.2 Basics

Let us consider the following problem

$$\begin{aligned} &\text{Minimize: } \mathbf{c^T x} \\ &\text{such that } \mathbf{Ax} = b \end{aligned}$$

We assume that the dimensions of the matrix A are $m \times n$ where $m < n$. Here, A_i is the i^{th} column of the matrix A. Let P be the corresponding feasible set. Note that the problem is stated in the standard slack form.

The simplex method searches for an optimal solution by navigating from one vertex to another, along the edges of the feasible region such that the cost is reduced. When we reach a basic feasible solution at which there is no neighbour that has a lesser cost, we have reached the minima, and the algorithm terminates.

A **polyhedron** is a set that can be described in the form $\{x \in \mathbb{R}^n \mid Ax \leq b\}$, where $\mathbf{A}$ is $m \times n$ matrix and $\mathbf{b}$ is vector in $\mathbb{R}^n$.

We need to find the vector $\mathbf{x}'$ s.t. $x' \in P$ and $c^T x'$ is minimized. Instead of directly working with inequalities we introduce slack variables and convert the given problem into a standard form. The vector x^* s.t. $\{x^* \in \mathbb{R}^n\}$ is **Basic Feasible Solution (BFS)** if all the constraints are satisfied (i.e., feasible solution), and $n - m$ of the elements are zero (basic solution).

For the rest of lecture we assume our problem has been already converted into the standard form, with A being $m \times n$ matrix representing m equalities in n variables. All the m rows being linearly independent.

We define a set of basic variables as $B \subset \{1, 2, ..., n\}$, s.t.

- $i \in B \implies x_i$ is a basic variable.
- $i \notin B \implies x_i = 0$ & x_i is a nonbasic variable.

As discussed in the last lecture, simplex algorithm needs to find two variables in each iteration — one that enters the Basic variable set (i.e., j) and the other that exits the basic variable set (i.e., l). Here is a summary of the process. (We will go down to the details and remind the notation in the next section.)

Finding l, θ once we know j :

1. Compute d using $d_B = -\bar{B}^{-1} A_j$, $d_j = 1$ and $d_i = 0 \forall i \neq j$ & $i \notin B$
2. As we have $x^{i+1} = x^i + \theta d$, if $d_i \leq 0$, then θ is limited by $x_i \geq 0$
   $\theta = \min_{i \in B}\{-x_i/d_i\}$
3. l is where this minimum happens

14.3 How simplex method works?

Suppose that we are at a point $\mathbf{x} \in$ P and that we want to move away from $\mathbf{x}$, in the direction of a vector $\mathbf{d} \in \mathbb{R}^n$. We should move in a direction that does not take us outside the feasible region. $\mathbf{d} \in \mathbb{R}^n$ is said to be a feasible direction at $\mathbf{x}$, if there exists a positive scalar θ for which

$$\mathbf{x} + \theta\mathbf{d} \in P. \tag{14.1}$$

Let $\mathbf{x}$ be a basic feasible solution to our problem, let *B(1), …, B(m)* be the indices of the basic variables, and let

$$\bar{B} = \left[\mathbf{A_{B(1)}} \cdots \mathbf{A_{B(m)}}\right] \tag{14.2}$$

be the corresponding basis matrix, such that $\mathbf{x}_i = 0$ for non-basic variables ($i \notin \mathbf{B}$). While the basic variables are given by

$$\mathbf{x}_B = \bar{B^{-1}}b \tag{14.3}$$

Simplex Algorithm:

1. Start with a BFS.
2. Find θ and $\mathbf{d}$ such that
$$\mathbf{x}^{i+1} \longleftarrow \mathbf{x}^i + \theta\mathbf{d} \tag{14.4}$$
if the cost is reduced.

Note:

1. Often the first vertex is found easily when $\mathbf{B}$ is the identity matrix.
2. In each new iteration, one new variable enter the basis and one exits.

Consider a new point $\mathbf{y} = \mathbf{x} + \theta\mathbf{d}$. Now, select a non-basic variable x_j ($j \notin \mathbf{B}$) and change its value to a positive value θ. Let the other non-basic variables x_i ($i \notin \mathbf{B}, i \neq j$)be zero. That is, $d_j = 1$ and $d_i = 0$

We know that,

$$\begin{aligned} \mathbf{Ax} = b \text{ and } \mathbf{Ay} &= b \\ \mathbf{A}(\mathbf{x} + \theta\mathbf{d}) &= b \\ \therefore \mathbf{Ad} &= \mathbf{0} \end{aligned} \tag{14.5}$$

When j enters $\mathbf{B}$

$$\begin{aligned} d_j &= 1 \\ \mathbf{B}\mathbf{x}_B &= b \\ \mathbf{x}_B &= \mathbf{B}^{-1}b \end{aligned} \tag{14.6}$$

The l^{th} column goes out and the j^{th} column comes in the basis.

$$\mathbf{x}_B^{new} = \mathbf{B}_{\text{new}}^{-1}b \tag{14.7}$$

Recall now that $d_j = 1$, and that $d_i = 0$ for all other non-basic indices i. Then,

$$\mathbf{Ad} = \mathbf{0} \Longrightarrow \sum_{j=1}^{n} A_j d_j = 0.$$

Since d_j is scalar, we can write

$$\sum_{j=1}^{n} d_j A_j = 0.$$

Splitting the summation into different parts for the basic and non-basic variables, we get

$$\sum_{i \in \mathbf{B}} d_i A_i + d_j A_j + \sum_{\substack{i \notin \mathbf{B} \\ i \neq j}} d_i A_i = 0.$$

We know that $d_j = 1$ and for $i \notin \mathbf{B}$, $d_i = 0$.

$$\therefore \sum_{i \in \mathbf{B}} d_i A_i + 1.A_j = 0$$

$$\sum_{i=1}^{m} d_{B(i)} A_{B(i)} + A_j = 0$$

$$\mathbf{B}\mathbf{d}_B + A_j = 0$$

$$A_j = -\mathbf{B}\mathbf{d}_B$$

$$\mathbf{d}_B = -\mathbf{B}^{-1} A_j \tag{14.8}$$

Next, we need to find θ, i.e., how much to move in the feasible direction. Given that

$$\mathbf{y} = \mathbf{x} + \theta\mathbf{d}$$

$$\mathbf{x}_i^{\text{new}} = \mathbf{x}_i + \theta\mathbf{d}_B \tag{14.9}$$

If $d_i < 0$, then θ is limited by $\mathbf{x}_i \geq 0$.

$$\therefore \theta = \min_{\substack{i \in \mathbf{B} \\ d_i < 0}} \left\{ \frac{-\mathbf{x}_i}{d_i} \right\} \tag{14.10}$$

Let l be the minimizing index.

$$\mathbf{x}_l^{\text{new}} = 0$$

$$\bar{B} = \left[\mathbf{A_{B(1)}} \cdots \mathbf{A_j}\ \mathbf{A_{B(l+1)}} \cdots \mathbf{A_{B(m)}} \right]$$

Change (reduction) in cost due to $d_j = 1$:
Let c_j be the change in cost due to j entering the basis. The rate of cost change along direction $\mathbf{d}$ is given by

$$c^T\mathbf{d} = c_B^T\mathbf{d}_B + c_j \tag{14.11}$$

Using Eq. (18.23), we get

$$c^T\mathbf{d} = c_j - c_B^T\mathbf{B^{-1}}\mathbf{A}_j \tag{14.12}$$

c_j is the cost per unit increase in the variable x_j , and the term $-c_B^T\mathbf{B^{-1}}\mathbf{A}_j$ is the cost of the compensating change in the basic variables brought about by the constraint $\mathbf{A}\mathbf{x} = b$.

$$\bar{c_j} = c_j - c_B^T\mathbf{B^{-1}}\mathbf{A}_j \tag{14.13}$$

$\bar{c_j}$ is the reduction in cost due to j entering $\mathbf{B}$.

$$\bar{c} = [c_1 \cdots c_n] \tag{14.14}$$

If $\bar{c} \geq 0$, the solution is optimal, so we stop. Else, we pick a j such that $\bar{c_j} < 0$.

Now the cost is given by $c^T x$, on changing the BFS x to $y = x + \theta d$, results in change of cost given by $c^T d$

Let $\bar{C}_j$ denote the cost change due to j entering B. Let $\bar{C} = [\bar{C}_1, ..., \bar{C}_n]$ denote the vector of cost change corresponding to each j.

Lemma 14.3.1. If $\bar{C}_j \geq 0\ \forall\ \bar{C}_j \in \bar{C}$, then the current BFS x is optimal.

If the above lemma holds, stop the algorithm and report the solution. Else $\exists j$ s.t. $\bar{C}_j < 0$.
$\bar{C}_j = C^T d$
$= C_B^T d_B + C_j$ (As j is entering B, $d_j = 1$ and $d_i = 0 \forall i \notin B$ & $i \neq j$.)
$= C_j - C_B^T B^{-1} A_j$

14.3.1 Simplex: Steps

Simplex method :

1. Start with X as BFS
2. Compute $\bar{C}$
3. If $\bar{C}_j \geq 0\ \forall\ \bar{C}_j \in \bar{C}$, then stop and report x as solution.
   Else $\exists j$ s.t. $\bar{C}_j < 0$, compute $d_B = -\bar{B}^{-1} A_j$
4. Find l and θ s.t. j enters B and l leaves B.
5. Update $x, B, \mathbb{B}$ and goto step 2.

1. Start with $\mathbf{x}$ as a BFS.
2. Compute $\bar{c}$.
3. If $\bar{c} \geq 0$, then the solution is optimal, terminate.
   Else for $c_j < 0$
   Compute $\mathbf{d}_B = -\mathbf{B}^{-1}A_j$.
4. Find l and θ such that
   $\mathbf{x}_i^{\text{new}} = \mathbf{x}_i + \theta \mathbf{d}_B$
   and l exits $\mathbf{B}$ and j enters $\mathbf{B}$.
5. Update $x_j, \mathbf{B}$ and go to step 2.

14.3.2 Computing $\bar{\mathbf{B}}^{-1}$

$\mathbf{B}$ and $\bar{\mathbf{B}}$ differ only by one column.

$$\mathbf{B} = \left[\mathbf{A_{B(1)}} \cdots \mathbf{A_{B(m)}}\right] \text{ and } \bar{\mathbf{B}} = \left[\mathbf{A_{B(1)}} \cdots \mathbf{A_{B(l-1)}}\ \mathbf{A_j}\ \mathbf{A_{B(l+1)}} \cdots \mathbf{A_{B(m)}}\right]$$

Given $\mathbf{B}$, how can we efficiently compute $\bar{\mathbf{B}}$?
We know

$$\begin{aligned}\mathbf{B}^{-1}\mathbf{B} &= I \\ &= [\mathbf{e_1}\ \mathbf{e_2} \cdots \mathbf{e_m}] \\ &= [\mathbf{e_1} \cdots \mathbf{e_{l-1}}\ \mathbf{u}\ \mathbf{e_{l+1}} \cdots \mathbf{e_m}]\end{aligned}$$

Only the l^{th} column is non-zero on the RHS in the above equation. Here $u = \mathbf{B}^{-1}\mathbf{A}_j$. Now, we need to do elementary row transformations such that we get RHS as $\mathbf{I}$ and $\mathbf{B}^{-1} \longrightarrow \bar{\mathbf{B}}^{-1}$.

Given $\mathbb{B}^{-1}$, can we find $\bar{\mathbb{B}}^{-1}$ efficiently ?

1. We start with $\mathbb{B}^{-1}\mathbb{B} = I = [e_1, ..., e_m]$
2. Replacing $\mathbb{B}$ with $\bar{\mathbb{B}}$, we get $\mathbb{B}^{-1}\bar{\mathbb{B}} = [e_1, ..., u, ..., e_m]$ (As only one column of the 2nd matrix is changed, and as each of the columns of product matrix is determined by the corresponding column of 2nd matrix)
3. Apply row operations to product matrix, changing the column u to e_l, and apply the same row operation to the first matrix. After atmost $m-1$ iterations, this is converted to the form :
   $\bar{\mathbb{B}}^{-1}\bar{\mathbb{B}} = I = [e_1, ..., e_u, ..., e_m]$ which can be used in the next iteration direclty.

14.3.3 Simplex Algorithm

1. Let x be basic feasible solution to the standard form problem, let B(1),..., B(m) be the indices of the basic variables, and let B = $[A_{B(1)} \ldots\ A_{B(m)}]$ be the corresponding basis matrix. In particular, we have $X_i = 0$ for every nonbasic variable, while the vector $X_B = (x_{B(1)}, \ldots, x_{B(m)})$ of basic variables is given by $X_B = B^{-1}$b.
2. Compute the reduced costs $\bar{c}_j = c_j - c_B^T\ B^{-1}A_j$ for all nonbasic indices j . If they are all nonnegative, the current basic feasible solution is optimal, and the algorithm terminates; else, choose some j for which $c_j < 0$.
3. Compute d = $B^{-1}A_j$. If no component of u is negative, we have $\theta^* = \infty$, the optimal cost is $-\infty$, and the algorithm terminates.
4. If any component of d is negative, let $\theta^* = \{i = 1, ..., \overset{min}{m} | u_i < 0\} \frac{-X_{B(i)}}{d_i}$.
5. Let l be such that $\theta^* = \frac{-X_{B(l)}}{d_l}$. Form a new basis by replacing $A_{B(l)}$ by A_j.If Y is the new basic solution then value of the new basic variables are Y=X+θ*d,where $Y_j = \theta^*$and $Y_{B(i)} = X_{B(i)} + \theta^*$ d_i and i≠l.
6. Repeat step 1 with X as Y until algorithm terminates.

14.4 On Computational Complexity

TODO

14.5 Example Problems

Example 7. Consider the linear programming problem

$$\begin{array}{lrcrcrcrcl} \text{minimize} & c_1x_1 & + & c_2x_2 & + & c_3x_3 & + & c_4x_4 & & \\ \text{subject to} & x_1 & + & x_2 & + & x_3 & + & x_4 & = & 2 \\ & 2x_1 & & & + & 3x_3 & + & 4x_4 & = & 2 \end{array}$$

$$x_1, x_2, x_3, x_4 \geq 0.$$

The first two columns of the matrix $\mathbf{A}$ are $\mathbf{A}_1 = (1, 2)$ and $\mathbf{A}_2 = (1, 0)$. Since they are linearly independent, we can choose x_1 and x_2 as our basic variables. The corresponding basis matrix is

$$\mathbf{B} = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}$$

We set $x_3 = x_4 = 0$, and solve for x_1, x_2, to obtain $x_1 = 1$ and $x_2 = 1$. Therefore, we have obtained a non-degenerate basic feasible solution. A basic direction corresponding to an increase in the non-basic variable x_3, is constructed as follows. We have $d_3 = 1$ and $d_4 = 0$. The direction of change of the basic variables is obtained using Eq. (18.23) as follows:

$$\begin{aligned} \mathbf{d}_B &= \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} d_{B(1)} \\ d_{B(2)} \end{bmatrix} \\ &= -\mathbf{B}^{-1} A_3 \\ &= -\begin{bmatrix} 0 & 1/2 \\ 1 & -1/2 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} \\ &= \begin{bmatrix} -3/2 \\ 1/2 \end{bmatrix} \end{aligned}$$

The cost of moving along this basic direction is $\mathbf{c^T d} = -3c_1/2 + c_2/2 + c_3$. This is the same as the reduced cost of the variable x_3. Suppose that $\mathbf{c} = (2, 0, 0, 0)$, in which case, we have $\bar{c_3} = -3$. Since $\bar{c_3}$ is negative, we form the corresponding basic direction, which is $\mathbf{d} = (-3/2, 1/2, 1, 0)$, and consider vectors of the form $\mathbf{x} + \theta\mathbf{d}$, with $\theta \geq 0$.
As θ increases, the only component of $\mathbf{x}$ that decreases is the first one (because $d_1 < 0$). The largest possible value of θ is given by $\theta^* = -(x_1/d_1) = 2/3$.
This takes us to the point $\mathbf{y} = \mathbf{x} + 2\mathbf{d}/3 = (0, 4/3, 2/3, 0)$.
Note that the columns $\mathbf{A}_2$ and $\mathbf{A}_3$ corresponding to the non-zero variables at the new vector $\mathbf{y}$ are (1, 0) and (1, 3), respectively, and are linearly independent.
Therefore, they form a basis and the vector $\mathbf{y}$ is a new basic feasible solution. In particular, the variable x_3 has entered the basis and the variable x_1 has exited the basis.

Example 8. Let x be an element of standard form polyhedra $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$. Prove that a vector $d \in \mathbb{R}^n$ is a feasible direction at x iff $Ad = 0$ and $d_i \geq 0 \ \forall \ i$ s.t. $x_i = 0$.

Let $x, y \in \mathbb{R}^n$ s.t. $y = x + \theta d$ where d_i indicates whether we are moving in that component, in forward or backward direction and θ is the scaling factor as in the speed at which we are moving.
For the forward direction of proof, we assume d to be a feasible direction vector and show the mentioned conditions hold
$Ay = b$
$\implies A(x + \theta d) = b$
$\implies Ax + \theta Ad = b$
$\implies \theta Ad = 0$
$\implies Ad = 0$ (As θ is scaling factor and $\theta \neq 0$
Also $\forall i$ s.t. $x_i = 0$, x_i can't decrease further, so it can only increase or stay at zero level. In formal terms $d_i \geq 0$.This proves the forward direction.

For the converse, we can let $d \in \mathbb{R}^n$ s.t. $Ad = 0$ and $d_i \geq 0 \ \forall \ i \mid x_i = 0$ and will show that d is feasible direction vector.
$Ad = 0$
$\implies \theta Ad = 0$ (multiplying by scalar on both sides)
$\implies Ax + A\theta d = Ax$ (adding same dimension vector on both sides)
$\implies A(x + \theta d) = b$
Let $y = x + \theta d$, as $Ay = b \implies y \in P$. So d is feasible direction vector. This completes the proof.

Example 9. Let $P = \{x \in \mathbb{R}^3 \mid x_1 + x_2 + x_3 = 1, \bar{x} \geq 0\}$ be a polyhedra. Consider a vertex $x = [0, 0, 1]^T$. Find the set of feasible directions at x.

Using the result of the previous proof, we know that d is a feasible direction vector if $Ad = 0$ and $d_i \geq 0 \; \forall \; x_i = 0$. We get,
$Ad = 0 \implies d_1 + d_2 + d_3 = 0$
Here $x = [0, 0, 1]^T$, so $x_1 = x_2 = 0 \implies d_1, d_2 \geq 0$.
So all the feasible direction vector at given x, is given by the polyhedron

$$D = \{d \in \mathbb{R}^3 \mid d_1 + d_2 + d_3 = 0, d_1, d_2 \geq 0\}$$

.

Example 10. Find the sequence at BFS and arrive at optimal point.

minimize $x_1 + x_2$

s.t

$x_1 + x_2 \leq 2$

$2x_1 \leq 2$

$x_1, x_2 \geq 0$

The above problem is converted from inequality to equality form

minimize $x_1 + x_2$

s.t

$x_1 + x_2 + x_3 = 2$

$2x_1 + x_4 \leq 2$

$x_1, x_2, x_3, x_4 \geq 0$

Step 1: This is the initial stage at which $x_1 = 0$ and $x_2 = 0$

$x_3 = 2 - x_1 - x_2$

$x_4 = 2 - 2x_1$

$z = x_1 + x_2$

X = [0, 0, 2, 1],

B = {3, 4}, z = 0

Now both $x_1$and x_2 have positive cofficients so we can't further decrease z since all x_i's are greater than zero and they can't be less than zero.

Chapter 15

Simplex Method Tableaux

One of the most popular technique for solving an LP is using Simplex Algorithm. Let us understand the practice of simplex using Tableaux. A set of well defined steps that helps in solving.

15.1 Simplex

This is an incomplete version. There are inconsistency in the notation and errors still left out. use the text book for careful study.

15.1.1 Simplex: Summary

1. We start with a basis $\bar{B} = [A_{B(1)}, \ldots, A_{B(m)}]$ and associated solution $\mathbf{x}$.
2. Compute the reduced cost $C_j = C_j - C_B^T \bar{B}^{-1} A_j$ for each non basic variable j if they are all positive, current solution is optimal, so exit else choose j such that $C_j > 0$.
3. Compute $u = B^{-1} A_j$ if no component of u is positive, we have $\theta^* = \infty$ and optimal cost $= -\infty$. Exit.
4. If for some computation, u_i is positive then, $\theta^* = \min_{\{i/u_i \geq 0\}}(A_{B(i)}/u_i)$
5. If l is the variable which minimizes then l exits and j enters. For a new basis by replacing $A_{B(i)}$ by A_j.

New solution $y = X + \theta di$ (or) $y_i = \theta^*$ and $y_{B(i)} = X_{B(i)} - \theta^* u_i$; $i \neq l$

Find B^{-1}, Given B^{-1}

$B^{-1}B = I$ or $\rightarrow B^{-1}B = [e_1, e_2 .... e_m]$

The tabloid is of the following format.

$-C_B^T B^{-1} b$	$C^T - C_B B^{-1} A$
$B^{-1} b$	$B^{-1} A$

15.2 Simplex Summary

15.2.1 Intuition and Design of Simplex Method

minimize $c^T x$
subject to $Ax \leq b$
$x \geq 0$
To summarize the intuition, We choose a corner by choosing n planes(assuming the intersection to be feasible). If we remove one plane, then the intersection of n-1 planes gives an edge(To stay in the feasible set we can choose only one direction). If we add another plane to the other n-1 planes such that the new intersection of n planes is in the feasible set then we move to a neighbouring corner. But just moving to the neighbouring corner is not our aim but this idea is used to move from one corner to another along an edge(which reduces the cost).
The main idea of simplex is to move from one corner to another along an edge of the feasible set such that the cost minimizes. A corner is the meeting of n different planes(Each plane is given by an equation). Each corner of the feasible set comes from turning n of n+m inequalites $Ax \leq b$ and $x \geq 0$ into equations and finding the intersection of these n planes.
To understand how simplex works, we introduce positive slack variables $w = Ax - b$ or $Ax - w = b$ which gives us $\begin{bmatrix} A & -I \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} = b$ renaming $\begin{bmatrix} A & -I \end{bmatrix}$ as A and $\begin{bmatrix} x \\ w \end{bmatrix}$ as x.
So we have, minimize $c^T x$, subject to $Ax = b$, $x \geq 0$. The intersection of n planes gives a corner. Suppose one of n intersecting planes is removed. The points that satisfy the remaining n-1 equations form an edge that comes out of the corner. This edge is the intersection of n-1 planes. To stay in the feasible region, only one direction is allowed along each edge which is given by Phase II of the simplex which will be discussed later in this lecture.
In this new setting, we can observe the same thing. A corner is now a new point where n components of new x are zero. These n components of x are free variables of $Ax = b$ set to zero. The remaining m components are basic variables or pivot variables. The basic solution will be a genuine corner if its m nonzero components are positive. We get a corner if x has n zero components. Now suppose we make one of the zero component variable and set one of the m nonzero component to zero. The earlier zero component which is now chosen to be variable can take a value and the system has one degree of freedom. The one zero component which can now take a value is adjusted such that it minimizes the cost $c^T x$. So which corner do we land now? We really wanted to move from one corner to another along an edge. Since two corners are neighbours, m-1 basic variables will remain basic. At the same time, one variable

will move up from zero to become basic. The value of other m-1 basic components will change but remain positive. The choice of edge decides which variable leaves the basis and which enters(and viceversa). The basic variables are computed by solving $Ax = b$. The free components are set to zero.
A corner is degenerate if more than the usual n components of x are zero. The basic set might change without actually moving from the corner.
A way to do the above is Tableau discussed in next subsection.

15.3 The Tableau

A move from one corner to another along an edge is a step of simplex algorithm. Each Simplex step involves decision followed by row operations. In this entering and leaving variable ave to be chosen, and they have to be made to come and go. A way to organize this is to fit A,b,c into a large matrix called tableau:

Tableau is $m+1$ by $m+n+1$ $\qquad T = \begin{bmatrix} A & b \\ c & 0 \end{bmatrix}$

At the start basic variables may be mixed with free variables. Renumbering if necessary, say $x_1, x_2, ..., x_m$ are basic variables. The first m columns of A form a square matrix B(basis matrix for that corner). The last n columns give an m by n matrix N. The cost vector c splits into $[c_B\ c_N]$ and unknowns x into (x_B, x_N).
At the corner, the free variables are $x_N = 0$. So, $Ax = b$ becomes $Bx_B = b$

Tableau at corner $\qquad T = \begin{bmatrix} B & N & b \\ c_B & c_N & 0 \end{bmatrix}$ $x_n = 0$, $x_B = B^{-1}b$, $cost = c_B B^{-1} b$

The basic variables will stand alone when elimination multiplies by B^{-1}

Reduced Tableau $\qquad T' = \begin{bmatrix} I & B^{-1}N & B^{-1}b \\ c_B & c_N & 0 \end{bmatrix}$.

To reach fully reduced row echelon form R = rref(T), substract c_B times the top block row from the bottom row:

Fully Reduced Tableau $\qquad R = \begin{bmatrix} I & B^{-1}N & B^{-1}b \\ 0 & c_N - c_B B^{-1} N & -c_B B^{-1} b \end{bmatrix}$.

Constraints $x_B + B^{-1}Nx_N = B^{-1}b$ $\qquad$ **Corner** $x_B = B^{-1}b$, $x_N = 0$.

The cost $c_B x_B + c_N x_N$ converts to :

Cost $c^T x = (c_N - c_B B^{-1} N) x_N + c_B B^{-1} b$ $\qquad$ **Cost at this corner** $= c_B B^{-1} b$

Every information is there in R. We can decide if the corner is optimal by looking at $r = c_N - c_B B^{-1} N$ in the middle of the bottom row. *If any entry in ris negative, the cost can still be reduced.* We can make rx_N negativeby increasing the component of x_N. That is our next step. *If $r \geq 0$, the best corner has been found. This is the optimality condition.*
Negative component of r corresponds to edges on which cost goes down.
Suppose r_i one of the negative values of r(Possibly the most negative). Then the ith component of x_N is the entering variable which increased from zero to a positive value α at the next corner. As x_i is increased the other component of x may decrease to maintain Ax=b. The x_k which reaches zero first becomes the leaving variable(it changes from basic to free). We have reached the corner when a component of x_B drops to zero. New corner is feasible because we have $x \geq 0$. It is basic because we have n zero components. The ith component of x_N went from zero to α. The kth component of x_B dropped to zero.

At new corner $x_i = \alpha = \min_j \frac{(B^{-1}b)_j}{(B^{-1}u)_j} = \frac{(B^{-1}b)_k}{(B^{-1}u)_k}$

$B^{-1}u$ is the column of $B^{-1}N$ in the reduced tableau R, above the selected negative entry r_i in the bottom row. If $B^{-1}u \leq 0$, the next corner is infinitely far away and the minimal cost is $-\infty$.
We can actually do the tableau method without grouping basic and non-basic column set and just doing row operations as in example section(unlike in the method described).

TODO: the last row in the above should be kept as the first row to be consistent with the tablaues below.

15.4 Example Probems

Example 11. We start with a problem that we have solved in the past.

Min -x_1-x_2
-x_1+x_2+x_3 = 1
x_1+x_4 = 3

$x_2+x_5 = 2$

$C = \begin{bmatrix}-1 & -1 & 0 & 0 & 0\end{bmatrix}$

$A = \begin{bmatrix}-1 & 1 & 1 & 0 & 0\\ 1 & 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0 & 1\end{bmatrix}$

$b = \begin{bmatrix}1\\3\\2\end{bmatrix}$

	x_1	x_2	x_3	x_4	x_5
	0	-1	-1	0	0	0
x_3	1	-1	1	1	0	0
x_4	3	1	0	0	1	0
x_5	2	0	1	0	0	1

$[C^T\text{-}C_B^T B^{-1}A] < 0$ for both x_1 and x_2, we can select any of them as entering variable. Let x_2 be selected.
(A_{B_i}/u_i) is least and u_i is positive for x_2, so it leaves. After performing row operations we get

	x_1	x_2	x_3	x_4	x_5
	1	-2	0	1	0	0
x_2	1	-1	1	1	0	0
x_4	3	1	0	0	1	0
x_5	1	1	0	-1	0	1

$[C^T\text{-}C_B^T B^{-1}A] < 0$ for x_1, so it is the entering variable
(A_{B_i}/u_i) is least and u_i is positive for x_5, so it leaves. After performing row operations we get

	x_1	x_2	x_3	x_4	x_5
	3	0	0	-1	0	2
x_2	2	0	1	0	0	1
x_4	2	0	0	1	1	-1
x_1	1	1	0	-1	0	1

$[C^T\text{-}C_B^T B^{-1}A] < 0$ for x_3, so it is the entering variable
(A_{B_i}/u_i) is least and u_i is positive for x_4, so it leaves. After performing row operations we get

	x_1	x_2	x_3	x_4	x_5
	5	0	0	0	1	1
x_2	2	0	1	0	0	1
x_3	2	0	0	1	1	-1
x_1	3	1	0	0	1	0

$[C^T\text{-}C_B^T B^{-1}A] > 0$ for all the variables so we can end the process.
$(x_1,x_2,x_3,x_4,x_5) = (1,2,2,0,0)$
Objective = -5

Example 12. Min $-10x_1-12x_2-12x_3$
$x_1+2x_2+2x_3 \leq 20$
$2x_1+x_2+2x_3 \leq 20$
$2x_1+2x_2+x_3 \leq 20$
$x_1,x_2,x_3 \geq 0$

Solution:

$x_1+2x_2+2x_3+x_4 = 20$
$2x_1+x_2+2x_3+x_5 = 20$
$2x_1+2x_2+x_3+x_6 = 20$

$C = \begin{bmatrix}-10 & -12 & -12 & 0 & 0 & 0\end{bmatrix}$

$A = \begin{bmatrix}1 & 2 & 2 & 1 & 0 & 0\\ 2 & 1 & 2 & 0 & 1 & 0\\ 2 & 2 & 1 & 0 & 0 & 1\end{bmatrix}$

$b = \begin{bmatrix}20\\20\\20\end{bmatrix}$

		x_1	x_2	x_3	x_4	x_5	x_6
	0	-10	-12	-12	0	0	0
x_4	20	1	2	2	1	0	0
x_5	20	2	1	2	0	1	0
x_6	20	2	2	1	0	0	1

$[C^T\text{-}C_B^T B^{-1} A] < 0$ and has less magnitude for x_1, so it is the entering variable
(A_{B_i}/u_i) is least and u_i is positive for x_5, so it leaves. After performing row operations we get

		x_1	x_2	x_3	x_4	x_5	x_6
	100	0	-7	-2	0	5	0
x_4	10	0	1.5	1	1	-0.5	0
x_1	10	1	0.5	1	0	0.5	0
x_6	0	0	1	-1	0	-1	1

$[C^T\text{-}C_B^T B^{-1} A] < 0$ and has less magnitude for x_3, so it is the entering variable
(A_{B_i}/u_i) is least and u_i is positive for x_4, so it leaves. After performing row operations we get

		x_1	x_2	x_3	x_4	x_5	x_6
	120	0	-4	0	2	4	0
x_3	10	0	1.5	1	1	-0.5	0
x_1	0	1	-1	0	-1	1	0
x_6	10	0	2.5	0	1	-0.5	1

$[C^T\text{-}C_B^T B^{-1} A] < 0$ and has less magnitude for x_2, so it is the entering variable
(A_{B_i}/u_i) is least and u_i is positive for x_2, so it leaves. After performing row operations we get

		x_1	x_2	x_3	x_4	x_5	x_6
	136	0	0	0	3.6	1.6	1.6
x_3	4	0	0	1	0.4	0.4	-0.6
x_1	4	1	0	0	-0.6	0.4	0.4
x_2	4	0	1	0	0.1	-0.6	0.4

$[C^T\text{-}C_B^T B^{-1} A] > 0$ for all the variables so we can end the process.
$(x_1,x_2,x_3,x_4,x_5) = (4,4,4,0,0)$
Objective = -136

Example 13. Max $Z = 5x_1+7x_2$
such that
$2x_1+3x_2 \leq 13$
$3x_1+2x_2 \leq 12$
$x_1,x_2 \geq 0$

Solution:

$2x_1+3x_2+x_3 = 13$
$3x_1+2x_2+x_4 = 12$

		x_1	x_2	x_3	x_4
	0	-5	-7	0	0
x_3	13	2	3	1	0
x_4	12	3	2	0	1

$[C^T\text{-}C_B^T B^{-1}A] < 0$ and has greater magnitude for x_2, so it is the entering variable
(A_{B_i}/u_i) is least and u_i is positive for x_3, so it leaves. After performing row operations we get

		x_1	x_2	x_3	x_4
	91/3	-1/3	0	7/3	0
x_2	13/3	2/3	1	1/3	0
x_4	10/3	5/3	0	-2/3	1

$[C^T\text{-}C_B^T B^{-1}A] < 0$ and has greater magnitude for x_1, so it is the entering variable
(A_{B_i}/u_i) is least and u_i is positive for x_4, so it leaves. After performing row operations we get

		x_1	x_2	x_3	x_4
	31	0	0	11/5	1/5
x_2	5	0	1	3/5	-2/5
x_1	2	1	0	-2/5	3/5

$[C^T\text{-}C_B^T B^{-1}A] > 0$ for all the variables so we can end the process.
$(x_1,x_2,x_3,x_4) = (2,5,0,0)$
Objective = 31

Example 14. Explain how row transformation correctly yields results in zeroth row.

Suppose at the beginning of typical iteration the 0^{th} row is of the form $[0|C^{'}]$-$g^{'}[b|A]$ where $g^{'}=C_B^{'}B^{-1}$, Hence 0^{th} row is equal to $[0|C^{'}]$ + a linear combination of the row [b|A]. Let column j be the pivot column and row l be pivot row. Note that pivot row is of form $h^{'}[b|A]$, where h^1 is l^{th} row of B^{-1}. Hence, after a multiple of pivot row is added to 0^{th} row, that row is again equal to $[0|C^{'}]$ + a (different) linear combination of rows of [b|A] if of form [0|C'] - P'[b|A], for some vector P. Recall that update rule is such that pivot column entry of 0^{th} row becomes zero. $C_{B(l)}$-P'$A_{B(l)}$=C_j-P'A_j=0. Consider row B(l)th column for i not equal to l. The 0^{th} row entry of that column is 0, before that change of basis, since it is reduced cost of basic variable.$B^{-1}A_{B(l)}$ is l^{th} unit vector & i not equal to l, the entry in pivot row for that column is also equal to 0. Hence addming a multiple pivot row to 0^{th}row does not affect the 0^{th} row entry of that column, which is left at zero.

Example 15. Min $-2x_1-x_2$
$x_1-x_2 \leq 2$
$x_1+x_2 \leq 6$
$x_1,x_2 \geq 0$

Solution:

$x_1+x_2+x_3 = 2$
$x_1+x_2+x_4 = 6$

$$C = \begin{bmatrix} -2 & -1 & 0 & 0 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & -1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

b = $\begin{bmatrix}2\\6\end{bmatrix}$

	x_1	x_2	x_3	x_4	
	0	-2	-1	0	0
x_3	2	1	-1	1	0
x_4	6	1	1	0	1

x_4 leaves and x_2 enters

	x_1	x_2	x_3	x_4	
	6	-1	0	0	1
x_3	8	2	0	1	1
x_2	6	1	1	0	1

$x_3$leaves and x_1 enters

	x_1	x_2	x_3	x_4
	10	0	0	0.5	1.5
x_1	8	2	0	1	1
x_2	2	0	1	-0.5	0.5

Further reduction not possible
So Z = -10

Example 16. Min $5x_1+2x_2$
$6x_1+x_2 \geq 6$
$4x_1+3x_2 \geq 12$
$x_1+2x_2 \geq 4$
$x_1,x_2 \geq 0$

Convert into minimization problem

Min Z = $-5x_1-2x_2$
$-6x_1-x_2+x_3 = -6$
$-4x_1-3x_2+x_4 = -12$
$-x_1-2x_2+x_5 = -4$

A = $\begin{bmatrix}-6 & -1 & 1 & 0 & 0\\-4 & -3 & 0 & 1 & 0\\-1 & -2 & 0 & 0 & 1\end{bmatrix}$

b = $\begin{bmatrix}-6\\-12\\-1\end{bmatrix}$

If B=I then, x_1=0, x_2=0 are not satisfied inequalities.

So take B = $\begin{bmatrix}x_1 & x_2 & x_3\end{bmatrix}$

then $-Z = 5\ (\ -4+2x_2-x_5)-2x_2$

$-Z= -20+8x_2-5x_5$
$x_1= 4-2x_2+x_5$
$x_4= -12+4(4-2x_2+x_5)+3x_2$

x_4= 4-5x_2+4x_5
x_3= -6+6x_1+x_2
x_3= 18-11x_2+6x_5

$$B = \begin{bmatrix} -6 & 1 & 0 \\ -4 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} -6 \\ -12 \\ -4 \end{bmatrix}$$

$$C_B = \begin{bmatrix} 0 \\ 8 \\ 0 \\ 0 \\ -5 \end{bmatrix}$$

	-20	0	8	0	0	-5
x_1	4	1	2	0	0	-1
x_3	18	0	11	1	0	-6
x_4	4	0	5	0	1	-4

We see that for x_5 value is negative (-5), so we need to calculate min(x[5]/u[5]) where $u[i] > 0$
But all u(i) are -ve
So optimal sol at x_i=+∞
So min cost -Z_1 =- ∞
Hence max Z= ∞

Example 17. Tableau demonstration:
$minimize x + y$
$x + 2y \geq 6$
$2x + y \geq 6$
$x, y \geq 0$
Introducing slacks, we get, $A = \begin{bmatrix} 1 & 2 & -1 & 0 \\ 2 & 1 & 0 & -1 \end{bmatrix}$ $b = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$ $c = \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix}$

$$\begin{bmatrix} A & b \\ c & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 & -1 & 0 & 6 \\ 2 & 1 & 0 & -1 & 6 \end{bmatrix}$$

1	2	-1	0	6
2	1	0	-1	6
1	1	0	0	0

We start with a feasible point say at P, which is the intersection of x=0 and 2x + y = 6. To relate to theory and be organised we exchange columns 1 and 3 to put to basic variables.

-1	2	1	0	6
0	1	2	-1	6
0	1	1	0	0

Then elimination multiplies the first row by -1 to give a unit pivot, and uses second row to produce zeros in the second column. Now fully reduced form at P is: At first look at r = [-1 1] in the bottom row has a negative entry in

1	0	3	-2	6
0	1	2	-1	6
0	0	-1	1	-6

column 3. So third variable will enter the basis. The current corner P and cost -6 is not optimal. The column above the negative entry $B^{-1}u = (3, 2)$, its ratio with the last column is $\frac{6}{3}$ and $\frac{6}{2}$ Since the first ratio is smaller, the first w, so the first column of basis is pushed out of the basis. We move from corner P to Q. The new tableau exchanges column 1 and 3. Pivoting by elimination gives: The new tableau at Q, has r=[$\frac{1}{3}$ $\frac{1}{3}$] is positive and thus final. The corner x=2, y=2 and z=+4 is optimal.
This can also be done without exchanging rows as shown in next example.

3	0	1	-2	6
2	1	0	-1	6
-1	0	0	1	-6

1	0	$\frac{1}{3}$	$\frac{-2}{3}$	2
0	1	$\frac{-2}{3}$	$\frac{1}{3}$	2
0	0	$\frac{1}{3}$	$\frac{1}{3}$	-4

Example 18. *minimize* $-x_1 - x_2$
$-x_1 + x_2 \leq 1$
$x_1 \leq 3$
$x_2 \leq 2$
$x_1, x_2 \geq 0$

minimize $-x_1 - x_2$
s.t. $-x_1 + x_2 + x_3 = 1$
$x_1 + x_4 = 3$
$x_2 + x_5 = 2$
$x_1, x_2, x_3, x_4, x_5 \geq 0$
We choose x1 to enters as the corresponding r_1 is one of the negative entry for r, x4 leaves as it the the variable

	b	x1	x2	x3	x4	x5
x3	1	-1	1	1	0	0
x4	3	1*	0	0	1	0
x5	2	0	1	0	0	1
z	0	-1	-1	0	0	0

which is equal to α and positive.

	b	x1	x2	x3	x4	x5
x3	4	0	1	1	1	0
x1	3	1	0	0	1	0
x5	2	0	1*	0	0	1
z	3	0	-1	0	1	0

By using similar argument, x2 enters x5 leaves
Answer: z = -5, x1=3, x2=2.

	b	x1	x2	x3	x4	x5
x3	2	0	0	1	1	-1
x1	3	1	0	0	1	0
x2	2	0	1	0	0	1
z	5	0	0	0	1	1

Example 19. Simplex Example solved in two different forms.

minimize $-x_1 - x_2$
such that $-x_1 + x_2 \leq 1$
$x_1 \leq 3$
$x_2 \leq 2$
$x_1, x_2 \geq 0$

min $-x_1 - x_2 + 0x_3 + 0x_4 + 0x_5$
$-x_1 + x_2 + x_3 + 0x_4 + 0x_5 = 1$
$-x_1 + 0x_2 + 0x_3 + x_4 + 0x_5 = 3$

$0x_1 - x_2 + 0x_3 + 0x_4 + x_5 = 2$
$x_1, x_2, x_3, x_4, x_5 \geq 0$

$\min -x_1 - x_2$
$-x_1 + x_2 + x_3 = 1$
$-x_1 + x_4 = 3$
$-x_2 + x_5 = 2$
$x_1, x_2, x_3, x_4, x_5 \geq 0$

Simplex version 1 (Solve above problem using simplex version 1)

1. Initial stage
   $x_3 = 1 + x_1 - x_2$
   $x_4 = 3 - x_1$
   $x_5 = 2 - x_2$
   $z = 0 - x_1 - x_2$
   $BFS = [0, 0, 1, 3, 2], B = \{3, 4, 5\}, z = 0$
2. x_1,x_2 both have negative coefficient any of them can be selected as entering variable let's choose x_2
   x_2 can be increase upto 1 while keeping $x_3 \geq 0$
   x_2 enters x_3 exits
   $x_2 = 1 + x_1 - x_3$
   $x_4 = 3 - x_1$
   $x_5 = 1 - x_1 + x_3$
   $z = -1 - 2x_1 + x_3$
   $x = [0, 1, 0, 3, 1], B = \{2, 4, 5\}, z = -1$
3. x_1 has negative coefficient, x_1 enters
   x_1 can be increase upto 1 while keeping $x_5 \geq 0$
   x_1 enters x_5 exits
   $x_1 = 1 + x_3 - x_5$
   $x_2 = 2 - x_5$
   $x_4 = 2 - x_3 + x_5$
   $z = -3 - x_3 + 2x_5$
   $x = [1, 2, 0, 2, 0], B = \{1, 2, 4\}, z = -3$
4. x_3 has negative coefficient, x_3 enters
   x_3 can be increase upto 2 while keeping $x_4 \geq 0$
   x_3 enters x_4 exits
   $x_3 = 2 - x_4 + x_5$
   $x_1 = 3 - x_4$
   $x_2 = 2 - x_5$
   $z = -5 + x_4 + x_5$
   $x = [3, 2, 3, 0, 0], B = \{1, 2, 3\}, z = -5$

We cannot reduce z any further because all the coefficient of z are positive, $z = -5$ is the optimal solution.

Simplex version 2 (Solve above problem using simplex version 2)

1. Initial stage

		x_1	x_2	x_3	x_4	x_5	
	0	-1	-1	0	0	0	$...a_1$
$x_3 =$	1	-1	1^*	1	0	0	$...a_2$
$x_4 =$	3	1	0	0	1	0	$...a_3$
$x_5 =$	2	0	1	0	0	1	$...a_4$

2. x_2 enters, x_3 exits

			x_1	x_2	x_3	x_4	x_5	
$a_1 + a_2$		1	-2	0	1	0	0	$...b_1$
a_2	$x_2 =$	1	-1	1	1	0	0	$...b_2$
a_3	$x_4 =$	3	1	0	0	1	0	$...b_3$
$a_4 - a_2$	$x_5 =$	1	1^*	0	-1	0	1	$...b_4$

3. x_1 enters, x_5 exits

			x_1	x_2	x_3	x_4	x_5	
$b_1 + 2b_4$		3	0	0	-1	0	2	$...c_1$
$b_2 + b_4$	$x_2 =$	2	0	1	0	0	1	$...c_2$
$b_3 - b_4$	$x_4 =$	2	0	0	1^*	1	-1	$...c_3$
b_4	$x_1 =$	1	1	0	-1	0	1	$...c_4$

4. x_3 enters, x_4 exits

			x_1	x_2	x_3	x_4	x_5	
$c_1 + c_3$		5	0	0	0	1	1	$...d_1$
c_2	$x_2 =$	2	0	1	0	0	1	$...d_2$
c_3	$x_3 =$	2	0	0	1	1	-1	$...d_3$
$c_4 + c_3$	$x_1 =$	3	1	0	0	1	0	$...d_4$

We cannot reduce z any further because all the coefficient of z are positive, $z = -5$ is the optimal solution.

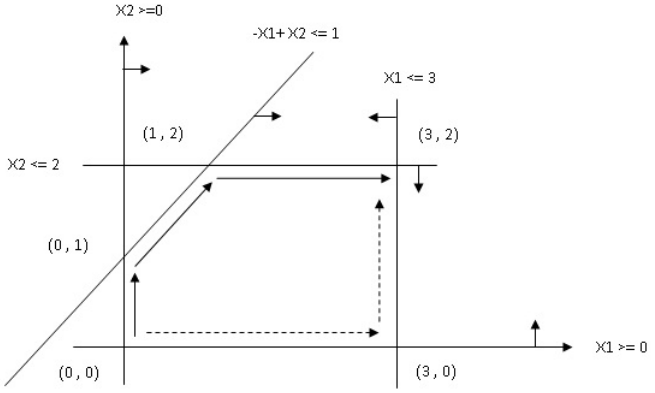


Figure 15.1: Diagrammatic representation of above procedure

Figure 31.1 We started with point (0,0) as initial feasible solution in step 1, after step 2 we reach point (0,1) which has better solution, after step 3 we reach (1,2) and finally to (3,2) after step 4 which is most optimal solution and we cannot go any further. The dotted path is another possible path that might have resulted depending upon selection of entering variable.

Example 20. **Using Simplex method(tableau) to solve:**

minimize $-24x_1+396\ x_2-8x_3-28x_4-10x_5$

s.t

$$\begin{bmatrix} 12 & 4 & 1 & -19 & 7 \\ 6 & -7 & 18 & -1 & -13 \\ 1 & 17 & 3 & 18 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 12 \\ 6 \\ 1 \end{bmatrix}$$

Here, we add artificial variables to these equalities and to the objective by multiplying some M (very large value) so that these artficial variables dont affect the objective. We solve the problem using simplex algorithm as follows,

		x_1	x_2	x_3	x_4	x_5	A_1	A_2	A_3
	0	-24	396	-8	-28	-10	M	M	M
A_1	12	12	4	1	-19	7	1	0	0
A_2	6	6	-7	18	-1	-13	0	1	0
A_3	1	1	17	3	18	-2	0	0	1

Final solution is $(x_1, x_2, x_3, x_4, x_5) = (1, 0, 0, 0, 0)$ and objective is -24.

		x_1	x_2	x_3	x_4	x_5	A_1	A_2	A_3
	24	0	804	64	404	-58	M	M	M+24
A_1	0	0	-200	-35	-235	31	1	0	-12
A_2	0	0	-109	0	18	-1	0	1	-6
x_1	1	1	17	3	18	-2	0	0	1

15.5 Additional problems

Problem. Max Z = -3x_1+ 2x_2 +2x_3

2x_1+ x_2+x_3≤ 2

3x_1+ 4x_2+2x_3≥ 8

x_1,x_2,x_3 ≥ 0

2) Min Z = x_1- 3x_2 +2x_3

2x_1-2x_2+3x_3≤ 7

-2x_1+ 4x_2≤ 12

-4x_1+ 3x_2+8x_3≤ 10

x_1,x_2,x_3 ≥ 0

Problem. min $-2x_1 - 2x_2$
$x_1 - x_2 \leq 2$
$x_1 + x_2 \leq 6$
$x_1, x_2 \geq 0$

Problem. Explain how transformation yields results in zeroth row.

Problem. Solve by Tableu method max $z = 5x_1 + 2x_2$
s.t.
$6x_1 + x_2 \geq 6$
$4x_1 + 3x_2 \geq 12$
$x_1 + 2x_2 \geq 4$

Chapter 16

Dual Problems

16.1 Introduction

Duality means that optimization problems may be viewed from two perspectives, the primal problem or the dual problem. This lead to problems coming in pair — primal and dual.

In many cases, both the primal and dual problems have physical interpretation. Often what is more interesting are the theoretical results (and of course, the practical implications) of the primal dual problems and their optimas.

In this lecture, we start by introducing a set of examples and show how dual problems can be constructed and interpreted.

16.2 A Simple Primal Dual Pair

Consider a linear programming problem:

$$Z = \min 6x_1 + 4x_2 + 2x_3$$

subject to :

$$\begin{aligned} 4x_1 + 2x_2 + x_3 &\geq 5 \\ x_1 + x_2 &\geq 3 \\ x_2 + x_3 &\geq 4 \\ x_i \geq 0, \ \ \forall i &= 1, 2, 3. \end{aligned}$$

Suppose we want to obtain a bound on the optimal value of this LP, we can do it by argueing in the following manner:

- Since all variables are non-negative, $6x_1 + 4x_2 + 2x_3 \geq 4x_1 + 2x_2 + x_3$. Therefore, the value of the LP must be at least 5.
- Also $6x_1 + 4x_2 + 2x_3 \geq (4x_1 + 2x_2 + x_3) + (x_1 + x_2) \geq 5 + 3 = 8$.
- Similarly, $6x_1 + 4x_2 + 2x_3 \geq (4x_1 + 2x_2 + x_3) + 2(x_1 + x_2) \geq 5 + 2 \cdot 3 = 11$.
- And, $6x_1 + 4x_2 + 2x_3 \geq (4x_1 + 2x_2 + x_3) + (x_1 + x_2) + (x_2 + x_3) \geq 5 + 3 + 4 = 12$.

We are now finding a bound that is closer and closer to the optima. Is 12 the optima? How do we know? We can actually determine the best bound we can achieve by setting up a different LP! Note that in all the above, we have been taking linear combination of the constraints to obtain better and better bounds.

Let y_1 be the number of times we take the first constraint, y_2 the number of times we take the second constraint and y_3 the number of times we take the third constraint. Then the lower bound we get is $5y_1 + 3y_2 + 4y_3$, and we need to ensure that this is a lower bound, i.e.

$$6x_1 + 4x_2 + 2x_3 \geq y_1(4x_1 + 2x_2 + x_3) + y_2(x_1 + x_2) + y_3(x_2 + x_3)$$

Since x_is and y_is are non-negative, need additional constraints to guarantee the above inequality. We can do this by ensuring that $4y_1 + y_2 + 0y_3 \leq 6$(since we have $6x_1$ in the objective value, and $4x_1$ in the first constraint, $1x_1$ in the second constraint and $0x_1$ in the third constraint). Similarly $2y_1 + y_2 + y_3 \leq 4$, $y_1 + y_2 \leq 2$. Also, we need to have y_1 , y_2 , $y_3 \geq 0$ (otherwise the inequalities in the constraints change direction, and we would not get a lower bound). We thus obtain the following LP for getting the best bound:

$$\text{maximize } \ 5y_1 + 3y_2 + 4y_3$$

subject to :

$$\begin{aligned} 4y_1 + y_2 &\leq 6 \\ 2y_1 + y_2 + y_3 &\leq 4 \\ y_1 + y_3 &\leq 2 \\ y_i \geq 0, \ \ \forall i &= 1, 2, 3. \end{aligned}$$

Now we have two linear programming problems. They are related. On a close observation, we see that b and c have swapped their roles. A has transposed. Goal is reversed (min to max). Such problems pairs are called the primal and dual problem pairs. In fact, one can construct the primal problem from the dual. You may also see some relationships in the A, b and c across these two problems.

16.3 Primal Dual Probelm Pairs

An optimization problem, called the primal, may be converted to its dual, and the solution to the dual provides a bound to the solution of the primal problem.

In our case, an optimization problem of the general form

$$\begin{aligned} Max\ & c^T x \\ s.t.\ & Ax \leq b \\ & x \geq 0 \end{aligned}$$

is the primal, it's dual can be written as :

$$\begin{aligned} Min\ & b^T y \\ s.t.\ & A^T y \geq c \\ & y \geq 0 \end{aligned}$$

Note: We consider the maximization problem as primal and minimization problem as dual.

16.3.1 Another LP Primal and Dual

Consider another example (this is a maximization problem):

$$\text{Maximize } 2x_1 + 3x_2$$

subject to:

$$4x_1 + 8x_2 \leq 12$$

$$2x_1 + x_2 \leq 3$$

$$3x_1 + 2x_2 \leq 4$$

$$x_1, x_2 \geq 0$$

Its dual is

$$\text{Minimize } 12y_1 + 3y_2 + 4y_3$$

Subject to:

$$4y_1 + 2y_2 + 3y_3 \geq 2$$

$$8y_1 + y_2 + 2y_3 \geq 3$$

$$y_1, y_2, y_3 \geq 0$$

If we solve these two problems (using simplex) (try it out!!), we see the optima of primal is $(\frac{1}{2}, \frac{5}{4})$ and dual is $(\frac{5}{16}, 0, \frac{1}{4})$.

But the objectives of both primal and dual are the same i.e., 4.75.

Is it always so? How are the primal and dual problems generated and related? There are some of the questions that we need to understand.

To make the life simple, we can state that for LP, primal and dual obtimas coincide, (when we have a feasible unbounded solution).

16.4 Primal $\leftrightarrow$ Dual Conversion

Given a problem, one can convert it into its dual using the following table.

PRIMAL	DUAL
$x_1, x_2, ..., x_n$	$y_1, y_2, ..., y_m$
A	A^T
b	c
c	b
$Max\ c^T X$	$Min\ b^T Y$
$\leq$	$y_i \geq 0$
$\geq$	$y_i \leq 0$
$=$	$y_i \in \mathbf{R}$
$x_j \geq 0$	$j^{th} constraint \geq$
$x_j \leq 0$	$j^{th} constraint \leq$
$x_j \in \mathbf{R}$	$j^{th} constraint =$

Problem: Derive the dual problem for a primal problem containing unconstrained variable.

16.5 Numerical Examples

Example 21. Find Dual of the given optimization problem

$$Z = \min x_1 - x_2$$

subject to :

$$2x_1 + 3x_2 - x_3 + x_4 \leq 0$$
$$3x_1 + x_2 + 4x_3 - 2x_4 \geq 3$$
$$-x_1 - x_2 + 2x_3 + x_4 = 6$$
$$x_1 \leq 0, x_2, x_3 \geq 0, x_4 \in \mathbb{R}$$

Using the conversion table, the dual problem can be written as :

$$Z = \max 3y_2 + 6y_3$$

subject to :

$$2y_1 + 3y_2 - y_3 \geq 1$$
$$3y_1 + y_2 - y_3 \leq -1$$
$$-y_1 + 4y_2 + 2y_3 \leq 0$$
$$y_1 - 2y_2 + y_3 = 0$$
$$y_1 \leq 0, y_2 \geq 0, y_3 \in \mathbb{R}$$

Example 22. Given the primal :

$$\begin{aligned} Max \quad & 6x_1 + 14x_2 + 13x_3 \\ s.t. \quad & \frac{1}{2}x_1 + 2x_2 - x_3 \leq 24 \\ & x_1 + 2x_2 + 4x_3 \leq 60 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

The dual can be calculated by referring to the table given above. The dual is found to be :

$$\begin{aligned} Min \quad & 24y_1 + 60y_2 \\ s.t. \quad & \frac{1}{2}y_1 + y_2 \geq 6 \\ & 2y_1 + 2y_2 \geq 14 \\ & -y_1 + 4y_2 \geq 13 \\ & y_1, y_2 \geq 0 \end{aligned}$$

The primal and dual are inter-convertible, i.e., the primal can be obtained by calculating the dual of the dual.

16.6 More Examples

Diet Problem: There are n foods, m nutrients, and a person (the buyer) is required to consume at least b_i units of nutrient i (for $1 \leq i \leq m$). Let a_{ij} denote the amount of nutrient i present in one unit of food j. Let c_i denote the cost of one unit of food item i. One needs to design a diet of minimal cost that supplies at least the required amount of nutrients. This gives the following linear program:

$$minimize \quad c^T x$$

$$subject \quad to$$

$$Ax \geq b$$

$$x \geq 0$$

Now assume that some other person (the seller) has a way of supplying the nutrients directly, not through food. (For example, the nutrients may be vitamins, and the seller may sell vitamin pills.) The seller wants to charge as much as he can for the nutrients, but still have the buyer come to him to buy nutrients. A plausible constraint in this case is that the price of nutrients is such that it is never cheaper to buy a food in order to get the nutrients in it rather than buy the nutrients directly. If y is the vector of nutrient prices, this gives the constraints $A^T y \leq c$. In addition, we have the nonnegativity constrain $y \geq 0$. Under these constraints the seller wants to set the prices of the nutrients in a way that would maximize the sellers profit (assuming that the buyer does indeed buy all his nutrients from the seller). This gives the the following dual LP:

$$maximize \quad b^T y$$

$$subject \quad to$$

$$A^T y \leq c$$

$$y \geq 0$$

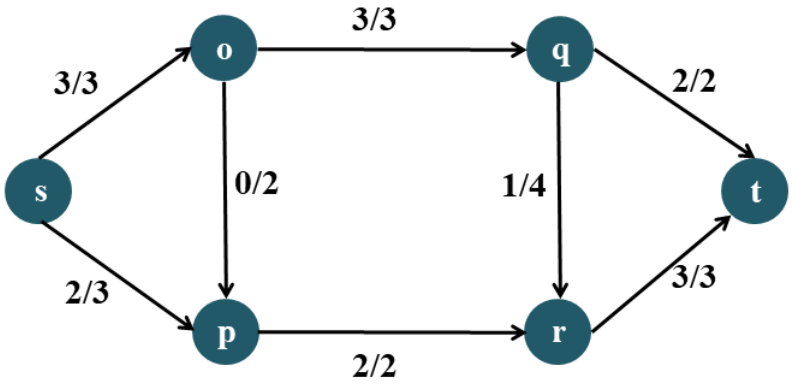
Problem: Suggest another real world problem where the primal and dual has physical interpretations.

16.7 Primal-Dual Pair: Mincut and Maxflow

Now let us take a classical primal-dual pair. The maxflow and mincut problems are a prime example of mathematical duality.

16.7.1 Maxflow

Let $N = (V, E)$ be a network with $s, t \in V$ being the source and the sink of N respectively.



The capacity of an edge is a mapping $c : E \to \mathbf{R}^+$, denoted by c_{uv} or $c(u, v)$. It represents the maximum amount of flow that can pass through an edge.
A flow is a mapping $f : E \to \mathbf{R}^+$, denoted by f_{uv} or $f(u, v)$, subject to the following two constraints:

1. 1. $f_{uv} \leq c_{uv}$, for each $(u, v) \in E$(capacity constraint: the flow of an edge cannot exceed its capacity)

2. 2. $\sum_{u:(u,v)\in E} f_{uv} = \sum_{u:(v,u)\in E} f_{vu}$, for each $v \in V \setminus \{s, t\}$(conservation of flows: the sum of the flows entering a node must equal the sum of the flows exiting a node, except for the source and the sink nodes)

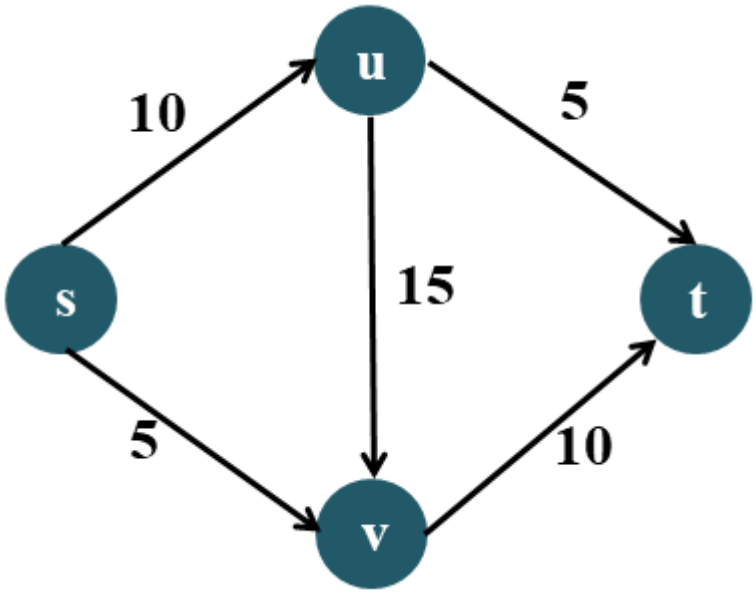
The value of flow is defined by $|f| = \sum_{v:(s,v)\in E} f_{sv}$, where s is the source of N. It represents the amount of flow passing from the source to the sink.

The maximum flow problem is to maximize $|f|$, that is, to route as much flow as possible from s to t.

16.7.2 Mincut

The minimum cut of a graph is a partition of the vertices of a graph into two disjoint subsets that are joined by at least one edge whose cut set has the smallest number of edges (unweighted case) or smallest sum of weights possible.

Example: Now let us consider a simple graph as in the figure and formulate the maxflow problem as linear programming. (we then want to argue that its dual is mincut.)



The primal maxflow problem is formulated as :

$$
\begin{aligned}
Max \quad & x_{su} + x_{sv} \\
s.t \quad & x_{su} + 0 + 0 + 0 + 0 \leq 10 \\
& 0 + x_{sv} + 0 + 0 + 0 \leq 5 \\
& 0 + 0 + x_{uv} + 0 + 0 \leq 15 \\
& 0 + 0 + 0 + x_{ut} + 0 \leq 5 \\
& 0 + 0 + 0 + 0 + x_{vt} \leq 10 \\
& x_{su} - x_{uv} - x_{ut} = 0 \\
& x_{sv} + x_{uv} - x_{vt} = 0
\end{aligned}
$$

$$
\mathrm{A} = \begin{array}{c} \begin{matrix} x_{su} & x_{sv} & x_{uv} & x_{ut} & x_{vt} \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & -1 & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 \end{bmatrix} \end{array}, \ \mathrm{b} = \begin{array}{c} \begin{bmatrix} 10 \\ 5 \\ 15 \\ 5 \\ 10 \\ 0 \\ 0 \end{bmatrix} \\ \begin{matrix} y_{su} \\ y_{sv} \\ y_{uv} \\ y_{ut} \\ y_{vt} \\ u_u \\ u_v \end{matrix} \end{array}
$$

and $\mathrm{c} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \end{bmatrix}^T$

Dual problem The corresponding dual, is:

$$\begin{aligned} Min \quad & 10y_{su} + 5y_{sv} + 15y_{uv} + 5y_{ut} + 10y_{vt} \\ s.t. \quad & y_{su} + u_u \geq 1 \\ & y_{sv} + u_v \geq 1 \\ & y_{uv} - u_u + u_v \geq 0 \\ & y_{ut} - u_u \geq 0 \\ & y_{vt} - u_v \geq 0 \\ & y_i \geq 0 \\ & u_i \in \mathbf{R} \end{aligned}$$

This problems represents the min-cut. Let us assume u_u is a variable that is 1 when if u is in the cut with set S and 0 otherwise. Similarly u_v is a variable for vertex v.

Also $y = 1$ imply the ones to be cut and $y = 0$ are not cut.

- The first constraint now states that if u is not in S, then su should be added to the cut.
- The third constraint imply that if u is with S and v is not, (i.e., $-u_u + u_v = -1$). Then uv should be in the cut.

Though the dual LP does not insist that y_i or u_u to be in $\{0, 1\}$, we loosely argue that the problem nature insists that these variables take only integer values. (see more details in one of the next lectures.)

problems There are many other classical examples of primal-dual pairs in the 'algorithms'. List 3 different pairs.

Chapter 17

More on Duality

17.1 Introduction

In the previous lecture, we saw a set of examples of primal and dual problems. The beuty of the structure of problems in the problem space is worth noticing and appreciating. Duality theory allows lot more than this. For example, it can help you to verify whether a solution is optimal. It can also help you to design approximate algorithms.

We first see some important results in this space in this lecture. Later on, we also see how the primal and dual problem structure can be used for designing approximate algorithms (in one of the next lectures).

17.1.1 More examples

TODO: Add more pairs.

17.2 Key Results on Duality

In this section, we discuss three importants results/theorems from the duality theory.

17.2.1 Farkas' Lemma

Farkas's lemma is a result in mathematics stating that a vector is either in a given convex cone or that there exists a hyperplane separating the vector from the cone. There are no other possibilities.

17.2.2 Weak Duality

Theorem 17.2.1. Let

$$P = \max\{c^T x | Ax \leq b, x \geq 0, x \in \mathbb{R}^n\}$$
$$D = \min\{b^T y | A^T y \geq c, y \geq 0, y \in \mathbb{R}^m\}$$

If x is a feasible solution of P and y is a feasible solution of D then

$$c^T x \leq b^T y$$

Proof.

$$c^T x = x^T c \leq x^T (A^T y) \leq (Ax)^T y \leq b^T y$$

□

- It always holds for convex and nonconvex problems. See the derivation above for the general case using lagrangians.
- It can be used to find nontrivial lower bounds for difficult problems.
- Note that this relationship is for the feasible points and not the optimal points alone.

17.2.3 Strong Duality

Strong duality holds if

$$c^T x^* = b^T y^*$$

Theorem 17.2.2. (Strong duality) If a linear programming problem has an optimal solution, so does its dual, and the respective optimal costs are equal.

i.e., $x^* = y^*$. If x^* and y^* are the optimal solutions of primal and dual problems.

- It does not hold in general

- It (usually) holds for convex problems. (Q: Isn't it always?)
- The conditions that guarantee strong duality in convex problems are called constraint qualifications.

Result 1:

From the weak duality, we can see that, if P is unbounded, then D should be infeasible. Similarly If dual is unbounded, primal should be infeasible.

Result 2:

Let x and y be feasible solutions to the primal and the dual, respectively. And suppose that $b^T y = c^T x$. Then, x and y are optimal solutions to the primal and the dual, respectively. This allows to verify the optimality.

17.2.4 Duality Result for LP

If P and D are Primal-Dual pairs of a LP problem then one of the four cases occur:

1. Both are infeasible.
2. P is unbounded and D is infeasible.
3. D is unbounded and P is infeasible.
4. Both are feasible. There exists an optimal solution x^* and y^* for P and D such that $c^T x^* = b^T y^*$. As LP is a convex optimization problem, strong duality hold and hence when the problem is feasible, the optimal values of primal and dual are same.

In this case, (2) and (3) come from Result 1. (4) come from strong duality. (1) can be proven easily with $A = 0$, $b < 0$ and $c > 0$.

17.2.5 Duality Result for IP

IP allows only weak duality. This can be easily proved as below.

Let $\bar{x}$ and $\bar{y}$ be the optima of the primal and dual integer programs. Let x^* and y^* are the optima of the relaxed primal and dual linear programs. Then:

$$
\begin{aligned}
c^T \bar{x} &= max(c^T x | Ax \leq b, x \geq 0, x \in \mathbf{Z}^n) && (17.1)\\
&\leq max(c^T x | Ax \leq b, x \geq 0, x \in \mathbf{R}^n) && (17.2)\\
&= c^T x^* && (17.3)\\
&= b^T y^* && (17.4)\\
&= min(b^T y | A^T y \geq c, y \geq 0, y \in \mathbf{R}^n) && (17.5)\\
&\leq min(b^T y | A^T y \geq c, y \geq 0, y \in \mathbf{Z}^n) && (17.6)\\
&= b^T \bar{y} && (17.7)\\
& && (17.8)
\end{aligned}
$$

Or

$$c^T \bar{x} \leq b^T \bar{y}$$

17.2.6 Duality Gap

Duality gap is the difference between the primal and dual solutions. If x^* is the optimal primal value and y^* is the optimal dual value then

Duality gap = $y^* - x^*$

For weak duality : Duality Gap > 0 and for strong duality, Duality Gap $= 0$

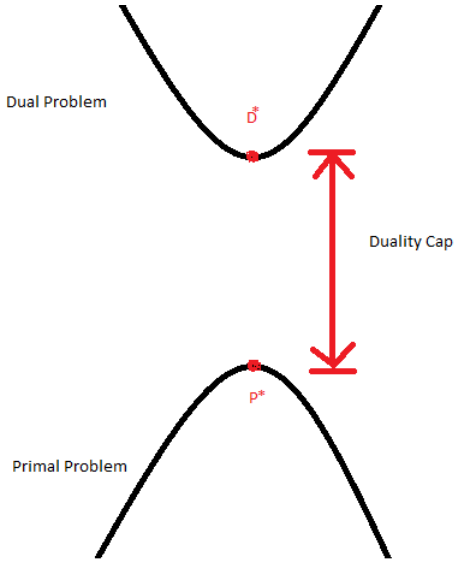


Figure 17.1: Visualization of the duality gap

Their difference is called the duality gap. For convex optimization problems, the duality gap is zero under a constraint qualification condition. Thus, a solution to the dual problem provides a bound on the value of the solution to the primal problem; when the problem is convex and satisfies a constraint qualification, then the value of an optimal solution of the primal problem is given by the dual problem.

17.3 Proof of Strong Duality

Now let us formally argue that $c^T x^* = b^T y^*$ or the strong duality.

Proof is left as an excercise.

17.3.1 Geometric Proof

17.3.2 Proof based on Farkas's Lemma

17.4 Examples

Example 23. Let A be a symmetric square matrix. Consider LP

$$\min c^T x$$

$$Ax \geq c$$

$$x \geq 0$$

Prove that if x^* satisfies $Ax^* = c$ and $x^* \geq 0$ then x^* is an optimal solution.

Forming the dual of the given problem :

$$\max c^T y$$

$$A^T y \leq c$$

$$y \geq 0$$

Given A to be a symmetric matrix, $A^T = A$.
Hence Dual problem becomes :

$$\max c^T y$$
$$Ay \leq c$$
$$y \geq 0$$

Now, given $Ax^* = c$. Hence, x^* satisfies both dual and primal problem constraints and gives
$c^T x^*$ as the objective value for primal problem.
$c^T x^*$ as the objective value for dual problem.
As the given problem is LP, a convex problem, strong duality must hold. Therefore, the dual and primal problems must meet at the optimal solution. Hence x^* is the optimal solution.

17.5 Additional Problems

Problem. Consider the primal problem

$$\min c^T x$$
$$Ax \geq b$$
$$x \geq 0$$

Form the dual problem and convert it into an equivalent minimization problem. Derive a set of conditions on the matrix A and the vectors b, c, under which the dual is identical to the primal, and construct an example in which these conditions are satisfied.

Problem. Consider the LP

$$\begin{aligned} Min \quad & 2x_1 + x_2 \\ s.t. \quad & x_1 + x_2 \leq 6 \\ & x_1 + 3x_2 \leq 3 \\ & x_1, x_2 \geq 0 \end{aligned}$$

(a) plot the feasible region and solve the problem graphically.
(b) Find the dual and solve it graphically.
(c) Verify that primal and dual optimal solutions satisfy the Strong Duality Theorem

Problem. Rock, Paper, and Scissors is a game in which two players simultaneously reveal no fingers (rock), one finger (paper), or two fingers (scissors). The payoff to player 1 for the game is governed by the following table:

Player1/Player2	Rock	Paper	Scissors
Rock	0	-1	1
Paper	1	0	-1
Scissors	-1	1	0

Table 17.1: input data

Payoff to player 1 (or minus payoff to player 2)

Note that the game is void if both players select the same alternative: otherwise, rock breaks scissors and wins, scissors cut paper and wins, and paper covers rock and wins. Use the linear-programming formulation of this game and linear-programming duality theory, to show that both players' optimal strategy is to choose each alternative with probability 1/3 . [**?**]

Problem. Why is it that, if the primal has unique optimal solution x^* , there is a sufficiently small amount by which c can be altered without changing the optimal solution? [**?**]

Chapter 18

More on Duality

18.1 Review of Important Results

P	D
$Max\ c^T x$	$Min\ b^T y$
$Ax \leq b$	$A^T y \geq c$
$x \geq 0$	$y \geq 0$

Examples

1. Numerial Problem:
   max $2x_1 + 3x_2$
   such that $4x_1 + 8x_2 \leq 12$
   $2x_1 + x_2 \leq 3$
   $3x_1 + 2x_2 \leq 4$
   $x_1, x_2 \geq 0$

   The dual of above is
   min $12y_1 + 3y_2 + 4y_3$
   such that $4y_1 + 2y_2 + 3y_3 \geq 2$
   $8y_1 + y_2 + 2y_3 \geq 3$
   $y_1, y_2, y_3 \geq 0$
2. Graph Problem: $Maxflow \leftrightarrow Mincut$
   In this problem we had seen that the maxflow is an LP, while mincut is an IP (selection of edges). How can they be duals?

18.1.1 Some Results

- Weak Duality Theorem (WDT)
$$x^T c \leq b^T y$$
- When both P and D are feasible, $c^T x^* = b^T y^*$ where x^*, y^* represents the optimal solution of primal and dual respectively
- Duality Theorem for LP: If P and D are primal and dual pair for LP, then one of the four cases occur:
  1. Both are infeasible
  2. P in inbounded and D is infeasible
  3. D is inbound and P is infeasible
  4. Both are feasible and there exists solution x and y to P and D such that $c^T x = b^T y$
- IP duals are weak duals
$$P = max(c^T x | Ax \leq b, x \geq 0, x \in Z^n)$$
$$D = min(b^T y | A^T y \geq c, y \geq 0, y \in Z^n)$$
  For any feasible $\bar{x}$ and $\bar{y}$
$$c^T \bar{x} \leq b^T \bar{y}$$

Example: Discuss the four cases of Duality theorem for LP Starting from the duality results of LP, i.e.,
1.Both primal and dual are infeasible
2.Primal is infeasible and dual is unbounded
3.Dual is infeasible and primal is unbounded
4.Both primal and dual are feasible
Example of case 1:
$maximize \quad 2x_1 - x_2$
such that
$$x_1 - x_2 \leq 1 \tag{18.1}$$

$$-x_1 + x_2 \leq -3 \tag{18.2}$$

$$x_1, x_2 \geq 0 \tag{18.3}$$

The dual of for the above primal is
minimize $y_1 - 3y_2$
such that

$$y_1 - y_2 \geq 2 \tag{18.4}$$

$$-y_1 + y_2 \geq -1 \tag{18.5}$$

$$y_1, y_2 \geq 0 \tag{18.6}$$

Both primal and dual are infeasible
Example of case 2:
maximize $2x_2 + x_3$

$$x_1 - x_2 \leq 5 \tag{18.7}$$

$$-2x_1 + x_2 \leq 3 \tag{18.8}$$

$$x_2 - 2x_3 \leq 5 \tag{18.9}$$

minimize $5y_1 + 3y_2 + 5y_3$

$$y_1 - 2y_2 \geq 0 \tag{18.10}$$

$$-y_1 + y_2 + y_3 \geq 2 \tag{18.11}$$

$$-2y_3 \geq 1 \tag{18.12}$$

Here primal is unbounded and dual is infeasible
Example of case 3:
minimize $5x_1 + 3x_2 + 5x_3$

$$x_1 - 2x_2 \geq 0 \tag{18.13}$$

$$-x_1 + x_2 + x_3 \geq 2 \tag{18.14}$$

$$-2x_3 \geq 1 \tag{18.15}$$

$x_1, x_2, x_3 \geq 0$
maximize $2y_2 + y_3$

$$y_1 - y_2 \leq 5 \tag{18.16}$$

$$-2y_1 + y_2 \leq 3 \tag{18.17}$$

$$y_2 - 2y_3 \leq 5 \tag{18.18}$$

$y_1, y_2, y_3 \geq 0$
Here dual is unbounded and primal is infeasible
Example of Case 4:
maximize $2x_1 + x_2$

$$x_1 + x_2 \leq 6 \tag{18.19}$$

$$x_1 + 2x_2 \leq 8 \tag{18.20}$$

minimize $6y_1 + 8y_2$

$$y_1 + y_2 \geq 2 \tag{18.21}$$

$$y_1 + 2y_2 \geq 1 \tag{18.22}$$

Problem: For the following problem construct the dual problem and verify the strong duality (i.e, primal optimum =dual optimum)
Maximize $5x + 4y + 3z$
subject to
$x + y + z \leq 30$
$2x + y + 3z \leq 60$
$3x + 2y + 4z \leq 84$

18.2 Duality from Lagrandian Perspective

18.2.1 Lagrangean and Lagrange Multiplier

Lagrange multiplier is often used in calculus to minimize a function subject to (equality) constraints. For example, in order to solve the below problem,

$$\begin{aligned} min \quad & x^2 + y^2 \\ subject\ to \quad & x + y = 1, \end{aligned} \tag{18.23}$$

we introduce Lagrange multiplier λ and form the Lagrangean $L(x, y, \lambda)$ defined by

$$L(x, y, \lambda) = x^2 + y^2 + \lambda(1 - x - y) \tag{18.24}$$

By keeping λ fixed, we minimize Lagarangean over all x and y subject to no constraints,

$$\frac{\partial L}{\partial x} = 0, \quad \frac{\partial L}{\partial y} = 0 \tag{18.25}$$

optimal solution to this unconstrained problem is $x = y = \lambda/2$ and depends on λ. The constraint $x + y = 1$, (or $\frac{\partial \mathcal{L}}{\partial \lambda}$) gives an additional relation. i.e., $\lambda = 1$ and optimal solution is $x = y = 1/2$.

Basic Idea

- We violate hard constraints in our original constrianed problem (˜18.23) and associate a Lagrange Multiplier or price λ with the amount $(1 - x - y)$ by which it is violated. This gives us an unconstrined problem.
- When price λ is properly chosen, optimal solution to the constrined problem is also optimal solution for unconstrained problem.
- Under specific value of λ, optimal cost is unlatered by the presence or absence of hard constraints.

18.2.2 Duality and Lagrange Multiplier

Consider an optimization problem (it could be nonconvex)

$$p^* = \min_x f_0(x); \quad \text{subject to} \quad f_i(x) \leq 0 \ \ \forall i$$

We define a Lagrangian by combining the constraints with the objective as:

$$\mathcal{L}(x, \lambda) = f_0(x) + \sum_i \lambda_i f_i(x)$$

The variables λ_i are the lagranigain multipliers. We observe that for every feasible x and $\lambda \geq 0$, $f_0(x)$ is bounded below by $\mathcal{L}(x, \lambda)$. i.e., $f_0(x) \geq \mathcal{L}(x, \lambda)$. Or $f_0(x) = \max_{\lambda \geq 0} \mathcal{L}(x, \lambda)$.

Let us define the primal problem now as

$$p^* = \min_x \max_{\lambda \geq 0} \mathcal{L}(x, \lambda)$$

Here we have used the fact that $\max_{\lambda \geq 0} \lambda^T f$ is 0 if $f \leq 0$ and $+\infty$ otherwise. (Note that $\lambda^T f$ is the shortcut for $\sum_i \lambda_i f_i(x)$.)

Now we define a fual function (to be precise lagrangian dual function), $g(\lambda) = \min_x \mathcal{L}(x, \lambda)$. Note that we have the following relationship. (since $f_0(x) \geq \mathcal{L}(x, \lambda)$)

$$f_0(x) \geq g(\lambda)$$

or the dual optima is

$$d^* = \max_{\lambda \geq 0} \min_x \mathcal{L}(x, \lambda)$$

Minmax inequality Another important relation to keep in mind at this stage is the minmax inequality. This says that for any two variables $x \in \mathcal{X}$ and $y \in \mathcal{Y}$,

$$\max_{y \in \mathcal{Y}} \min_{x \in \mathcal{X}} \phi(x, y) \leq \min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} \phi(x, y)$$

To show this,

$$\min_{x' \in \mathcal{X}} \phi(x', y) \leq \max_{y' \in \mathcal{Y}} \phi(x, y')$$

Start from this equation and take $\min_x$ on RHS and $\max_y$ on LHS. This leads to the the general inequality above. Use of this will also lead to weak duality in general case.

18.2.3 Special Case of Linear Programming

Conside the LP in standard inequality form

$$p^* = \max_x \ c^T x : \ \ Ax \leq b$$

where $A \in R^{m \times n}$ and $b \in R^m$

Langranean functions is

$$\mathcal{L}(x, \lambda) = c^T x + \lambda^T (b - Ax)$$
$$\mathcal{L}(x, \lambda) = b^T \lambda + (c^T - \lambda^T A)x$$

We can see that this function is $\geq c^T x$ when $\lambda \geq 0$. Let us define the dual function $g(\lambda)$ as

$$g(\lambda) = \max_x \ g(x, \lambda) \geq p^*$$

We are interested in the best (tightest) upper bound on $g(\lambda)$.

$$d^* = \min_{\lambda \geq 0} g(\lambda)$$

$$d^* = \min_{\lambda \geq 0} \max_x \ \mathcal{L}(x, \lambda)$$
$$d^* = \min_{\lambda \geq 0} \max_x \ c^T x + \lambda^T (b - Ax)$$
$$d^* = \min_{\lambda \geq 0} \max_x \ b^T \lambda + (c^T - \lambda^T A)x$$
$$d^* = \min_{\lambda \geq 0} (b^T \lambda + \max_x \ (c^T - \lambda^T A)x)$$

If $c^T - \lambda^T A$ has nonzero entries, then the maximum over all x is ∞ and is a useless upper bound. We should consider only the case when $A^T \lambda = c$. This leads to the dual problem as

$$d^* = min b^T \lambda \text{ s. t. } A^T \lambda = c; \text{ and } \lambda \geq 0$$

18.3 When does LP yield an integral solution?

We had seen in the past that many problems that gets formulated as IP have efficient solutions. It is important to ask why certain problems yield integer solutions with ease? To answer, we need to know a class of matrices first.

Definition 1. A matrix is totally unimodular(TU) if all its square sub matrices (by removing certain rows and columns) have determinant -1,0 or 1.

This immediately imply that:

- All elements $\in \{-1, 0, 1\}$.
- If A is totally unimodular, then $A^{'} = [\text{AI}]$ is also totally unimodular.

Problem: If A is TU, will A^T also be TU?

Theorem 18.3.1. If A is TU & b is integral, then LP gives integral solution

Proof.

$$max c^T x | Ax \leq b, b \in Z^m, x \geq 0$$

To solve this, we had seen the process of adding slack variables and creating the equality.

$$A^{'} x = b$$

$$x_B = A_B^{'-1} b$$

$$= \frac{1}{det(A_B^{'})} [\ \ ]$$

Since A_B will have determinant as -1 or 1 (and not zero) as well as the cofactors are also integers, x_B has only integer values.

□

18.4 Closer Look at Matching and Cover

In this section, we will have a closer look at another primal dual pair.

$Ex: \ Maximal \ Match \ \leftrightarrow \ Min \ Vertex \ Cover$

We will study this pair when the graph is BPG and for a general case.

Cases:

1. For BPG, $G = (X \cup Y, E)$ (Bipartite graph), A is TU.
2. For a general graph, A is not TU.

Maximum matching:In a graph G, finding the maximum number of edges without any common vertices is maximum matching problem. Minimum vertex cover set: In a graph G, selecting the minimum number of vertices so that atleast one vertex of every edge in graph is present in the selected vertices.

Maximum Matching Formulation:

Let G(V,E) be the graph and $|V| = n$ and $|E| = m$

$$Maximize \sum_{j=1}^{m} X_j \tag{18.26}$$

such that

$$\sum_{k:(j,k)\in E} X_k \leq 1 \quad \forall j \in V \tag{18.27}$$

$$X_i \in \{0, 1\} \tag{18.28}$$

Minimum Vertex Cover formulation:

$$Minimize \sum_{v \in V} X_j \tag{18.29}$$

such that

$$x_u + x_v \geq 1 \quad for \ \ every \{u, v\} \in E \tag{18.30}$$

$$x_u \in \{0,1\} \tag{18.31}$$

Minimum vertex cover and max matching have primal dual relationship.

Consode r a graph with n vertices and m edges $v_1, v_2 ... v_n \quad and \quad e_1, e_2 ... e_m$. Let us define an Incidence matrix A such that:

$$a_{ij} = \begin{cases} 1 & \text{if } x_i \in e_j \\ 0 & \text{else} \end{cases}$$

A is a $n \times m$ matrix. Now we can define these two problems as:

IP1 (Maximum Matching)

$$max \sum_{j=1}^{m} x_j$$
$$Ax \leq 1$$
$$x \geq 0$$
$$x \in Z^m$$

IP2(Minimum Vertex Cover)

$$min \sum_{i=1}^{n} y_i$$
$$A^T y \geq 1$$
$$y \geq 0$$
$$y \in Z^n$$

Our objective is to show that A is TU for BPG.

18.4.1 For BPG, incidence matrix A is TU

Proved by Induction: Q is a l × l sub matrix of A

1. Q is $1 \times 1 \implies$ Q is TU, $|Q| \in \{0, 1, -1\}$
2. Assume $(l-1) \times (l-1)$ matrix has elements $\epsilon\{0, 1, -1\}$, then Q of l×l has det$\epsilon\{0, 1, -1\}$
   Consider the cases where we have a column which has (i) no 1s (ii) one 1 and (ii) two 1s. (note that a column will not have more than two 1s)
   (a) all are zeros, $|Q_{l\times l}| = 0$
   (b) there is one 1,$|Q_{l\times l}| = 1 \times |Q_{(l-1)\times(l-1)}| \quad \in \{-1, 0, 1\}$
   (c) there are two ones $|Q_{l\times l}| = 0$
   (Rearrange rows such that all X are in the first half and Y are in the second half, These rows are linearly dependent. Therefore , $determinant = 0$)

This argues that for BPG, we can always show that A is TU and this will yield integral solutions.

We now see two numerical examples

18.5 Numerical Example

Example 1: Bipartite Graph Consider the BPG as shown below:

For the primal and dual, the A matrix is:

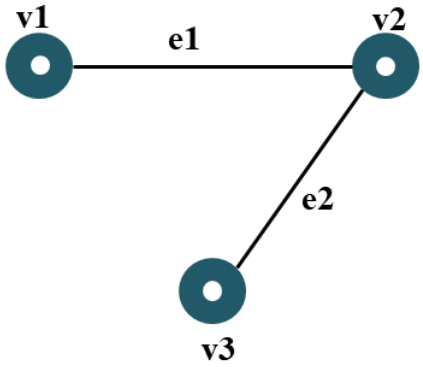


Figure 18.1: BPG

$A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$

See that this matrix is TU.

For the primal, $max 1^T x$ such that $Ax \leq 1$, lead to the optinal x as:

$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$LP^* = IP^* = 1$

Similarly for the dual,

$min\ 1^T y$ such that $A^T y \geq c$ yield the optimal y as $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ $LP^* = IP^* = 1$

$[objective : c^T x = b^T y = 1(same)]$

Case 2: Non-Bipartite Graph Now consider a non-BPG with three vertices.

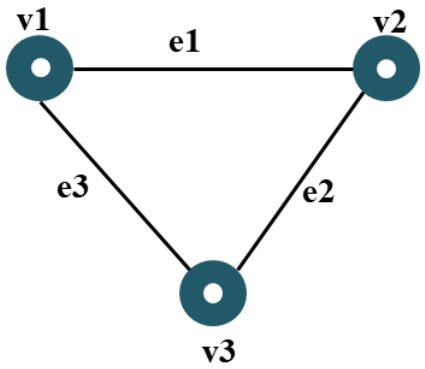


Figure 18.2: Ex Non-BPG

The primal and dual problems are:

$$Primal: \ max\ x_1 + x_2 + x_3$$

$$x_1 + x_2 \leq 1$$
$$x_2 + x_3 \leq 1$$
$$x_3 + x_1 \leq 1$$
$$Dual: \ min \ y_1 + y_2 + y_3$$
$$y_1 + y_2 \geq 1$$
$$y_2 + y_3 \geq 1$$
$$y_3 + y_1 \geq 1$$

on solving primal:

$$LP^* = \frac{3}{2}$$

while the integer optimal is:

$$IP^* = 1$$

on solving dual:

$$LP^* = \frac{3}{2}$$

while the integer optimal is:

$$IP^* = 2$$

This imply that the LP did not have an integral optima. (Why? because A was not TU)

Let us write A properly:

$$\text{Now, A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$|A| = 2$$

$$\Rightarrow \text{A is not a TU}$$

18.6 Additional Problems

Problem: Prove that the matrix A we had for the mincut is TU.

Chapter 19

Primal Dual Methods

19.1 Review and Summary

Duality Theorem

This Theorem states that the problems Primal(P) and Dual(D) are intimately related.We can think their relationship in following table:

P D	unbounded	has solution	not feasible
unbounded	no	no	possible
has solution	no	same values	no
not feasible	possible	no	possible

19.2 Complementary Slackness

Relationship between the primal and dual that is known as complementary slackness. We know that the number of variables in the dual is equal to the number of constraints in the primal and the number of constraints in the dual is equal to the number of variables in the primal.It means that variables in one problem are complementary to constraints in the other. constraint having slack means it is not binding.For an inequality constraint, the constraint has slack if the slack variable is +ve. Complementary slackness means that a relationship between slackness in primal constraints and slackness of it's dual constraints. The following are primal and dual complmentary slackness conditions.

Primal Complementary (PCS)

Conditions for PCS:
$1 \leq j \leq n$ either $X_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j \implies X^T(A^T Y - c) = 0$

Dual Complementary (DCS)

Conditions for DCS:
$1 \leq i \leq m$ either $Y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i \implies Y^T(AX - b) = 0$
The above complementary slackness conditions guarantee that the values of the primal and dual are the same.

19.2.1 Basic Derivation/Proof of Complmentary Slackness

We know about primal and dual methods. Consider Following Primal-Dual Pairs.

$$P = Max c^T X \mid [AX \leq b], [X \geq 0] \tag{19.1}$$

$$D = Min b^T Y \mid [A^T Y \geq c], [Y \geq 0] \tag{19.2}$$

Suppose $X*$ is optimal point for primal problem and Y^* is optimal for Dual.
Further from equations (1) and (2)

$$c^T X* \leq (A^T Y*)^T X* \leq (Y*)^T AX* \leq Y*^T b \tag{19.3}$$

From Strong Duality :

$$c^T X* = b^T Y* \tag{19.4}$$

from equations (3) and (4)

$$c^T X* = (A^T Y*)^T X* \quad \text{and} \quad Y*^T AX* = Y*^T b \tag{19.5}$$

We can rewrite above equation:

$$Y^T[AX - b] = 0 \tag{19.6}$$

$$X^T[A^T Y - c] = 0 \tag{19.7}$$

From eqn (6) either $Y = 0$ or $AX = b$. Similar for eqn (7) either $X = 0$ or $A^T Y = c$.
We concluded that either i^{th} variable is 0 or i^{th} constraint is tight.

19.2.2 Using Complementary Slackness to Solve Duals/Primal

Ensure PCS but not DCS and $1 \leq j \leq n$ either $y_i = 0$ or $\frac{b_j}{\beta} \leq \sum_{j=1}^{n} a_{ij} x_j \leq b_j$ where $\beta \geq 1$. PCS is strictly obeyed and DCS is relax. or vica-verssa.Help in Deciding Solution which is approximate.

Iterative Method for solution above.
$X_i \rightarrow X_{i+1}$
$X_{i+1} \rightarrow X_i + Y$
We have to find Y to calculate X_{i+1}.

19.2.3 Complementary Slackness:Ver2

Complementary slackness tells us that when primal LP sets some variable to non-zero, then it's for some "good reason".

Primal Complementary Slack (PCS) condition

$1 \leq j \leq n$ either $x_j = 0$ or

$\sum_{i=1}^{n} a_{ij} y_i = c_i$, $X^T[A^T Y - c] = 0$

Dual Complementary Slack (DCS) condition

$1 \leq i \leq m$ either $y_i = 0$ or

$\sum_{j=1}^{m} a_{ij} x_j = b_j$, $Y^T[AX - b] = 0$

Note : While solving a problem by P-D method, ensure PCS strictly and DCS relaxed.

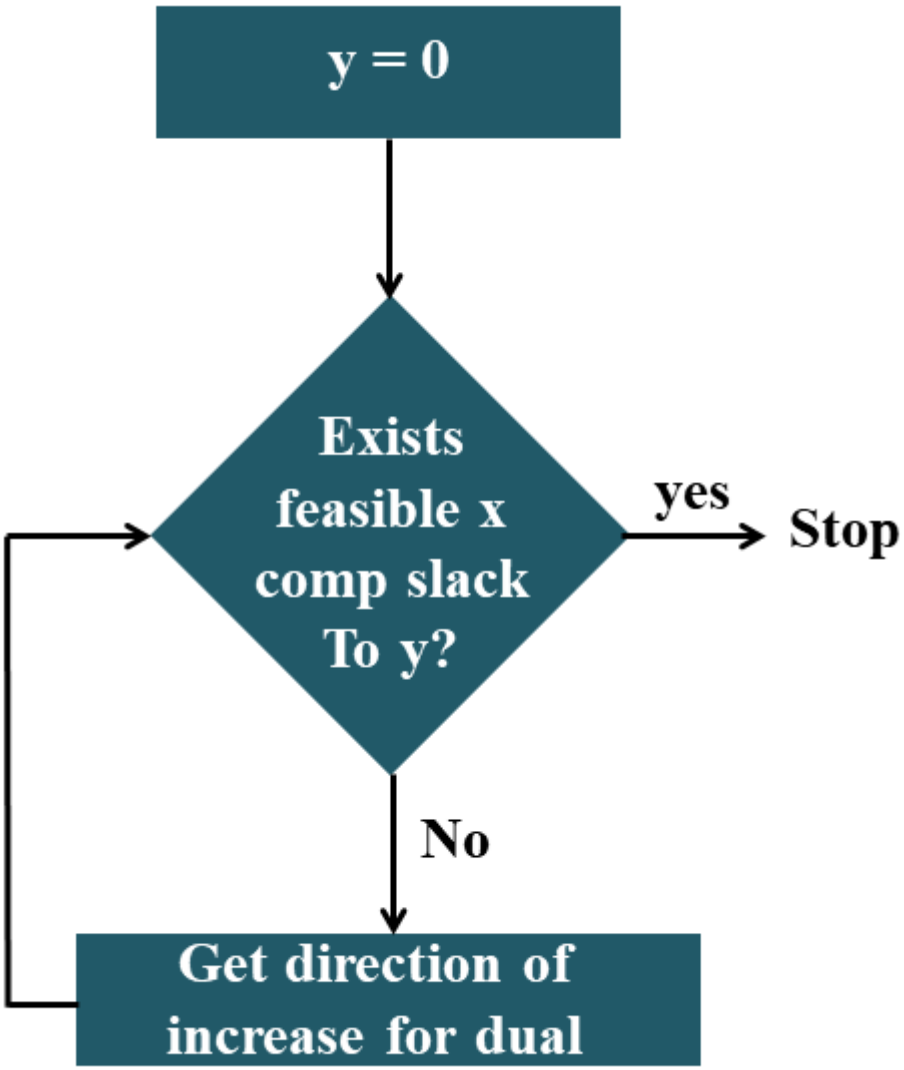
Relaxed DCS can be written as -

$1 \leq i \leq m$ either $y_i = 0$ or $b_j \leq \sum_{j=1}^{m} a_{ij} x_j \leq \beta b_j$

here β is kept for relaxing the relation. $\beta = 1$ when it is tight.

How to solve problems

If the conditions are obeyed for feasible solutions x, y, then solutions are optimal. For solving any problem start with a variable X and change it with condition, until its feasible value is found ensuring it should satisfy slack constraints.



$X^i \to X^{i+1} \Rightarrow X^{i+1} \leftarrow X^i + Y$

19.3 Introduction to Primal Dual Method

The primal-dual method is a standard tool for designing algorithms for combinatorial optimization problems. In this lecture, we focus on showing how to modify the primal-dual method to provide good approximation algorithms for a wide variety of NP-hard problems.
The primal-dual method was originally proposed by Dantzig, Ford and Fulkerson as another means of solving linear programs. However, now it is more widely used for devising algorithms for problems in combinatorial optimization. The main feature of primal-dual method is that it allows weighted optimization problem to be reduced to a purely combinatorial unweighted problem. It also leads to efficient polytime algorithms for solving NP-hard problems.
The following figure shows the general framework of the primal-dual method:

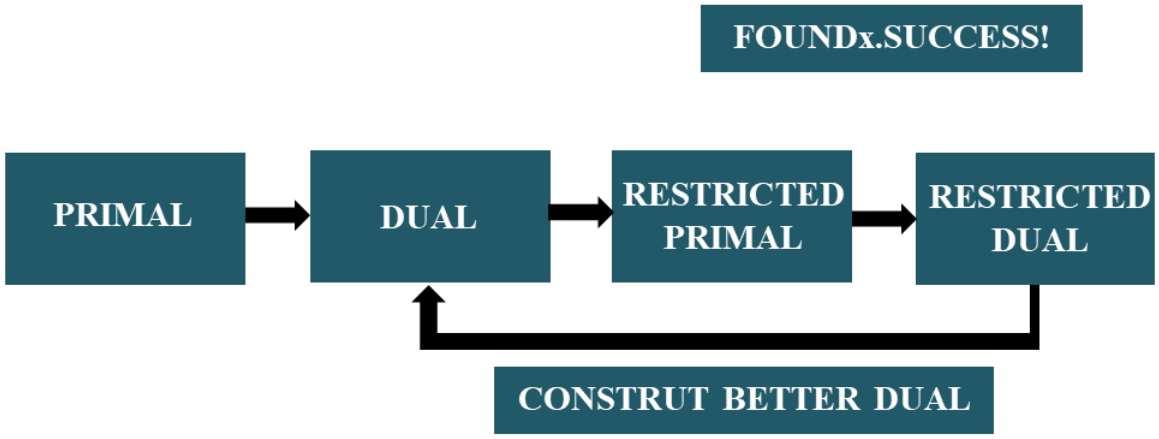


Figure 19.1: General framework of primal-dual method

19.3.1 Overview of the primal-dual method

Consider the following primal program called P:

$$min \sum_{j=1}^{n} c_j x_j$$
$$s.t : \sum_{j=1}^{n} a_{ij} x_j \geq b_i$$
$$x_j \geq 0$$
$$i = 1, 2, 3, 4....m$$
$$j = 1, 2, 3, 4....n$$

Then the dual program D is:

$$max \sum_{i=1}^{m} b_i c_i$$
$$s.t : \sum_{i=1}^{m} a_{ij} y_j \leq c_j$$
$$y_i \geq 0$$
$$i = 1, 2, 3, 4....m$$
$$j = 1, 2, 3, 4....n$$

Recall the complementary slackness conditions:

1) Primal complementary slackness conditions:
   For each $1 \leq j \leq n$: either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$
2) Dual complementary slackness conditions:
   For each $1 \leq i \leq m$: either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i$

Given an optimization problem (NP-hard),we will formulate this problem as an IP and relax it to obtain an LP. Then, we round the optimal solution x^* of the LP to obtain an integral solution. In the primal-dual method, we find a feasible integral solution to the LP (thus to the IP) from scratch (instead of solving LP) using dual D as our guiding.
Specifically, we do either of the following:

1) Ensure the primal conditions and suitably relax the dual conditions:
   For each: $1 \leq i \leq m$ either $y_i = 0$ or $b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \beta b_i$ where $\beta > 1$
2) Ensure the dual conditions and suitably relax the primal conditions:
   For each: $1 \leq j \leq n$ either $x_j = 0$ or $\frac{c_j}{\alpha} \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$ where $\alpha > 1$

If we use the first way, that is, ensure the primal conditions and relax the dual conditions, we have:

Lemma 19.3.1. If x and y are feasible solutions of P and D respectively, satisfying conditions in the first way i.e. primal conditions are ensured and dual conditions are relaxed, then:

$$\sum_{j=1}^{n} c_j x_j \leq \beta \sum_{i=1}^{m} b_i y_i$$

Proof.

$$\sum_{j=1}^{n} c_j x_j = \sum_{j=1}^{n} (\sum_{i=1}^{m} a_{ij} y_i) x_j$$
$$\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} (\sum_{j=1}^{n} a_{ij} x_j) y_i$$
$$\sum_{j=1}^{n} c_j x_j \leq \beta \sum_{i=1}^{m} b_i y_i$$

□

More specifically, let $\alpha = 1$ if the primal conditions are ensured and $\beta = 1$ if the dual conditions are ensured, then we have:

1) Primal complementary slackness conditions:
   Let $\alpha \geq 1$. For each $1 \leq j \leq n$: either $x_j = 0$ or $\frac{c_j}{\alpha} \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$
2) Dual complementary slackness conditions:
   Let $\beta \geq 1$. For each $1 \leq i \leq m$: either $y_i = 0$ or $b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \beta b_i$

Lemma 19.3.2. If x and y are feasible solutions of P and D respectively, satisfying the complementary slackness conditions stated above, then both x and y are $\alpha - \beta$ approximate solutions:

$$\sum_{j=1}^{n} c_j x_j \leq \alpha\beta \sum_{i=1}^{m} b_i y_i$$

Proof.

$$\sum_{j=1}^{n} c_j x_j = \alpha \sum_{j=1}^{n} (\sum_{i=1}^{m} a_{ij} y_i) x_j$$
$$\sum_{j=1}^{n} c_j x_j = \alpha \sum_{i=1}^{m} (\sum_{j=1}^{n} a_{ij} x_j) y_i$$
$$\sum_{j=1}^{n} c_j x_j \leq \alpha\beta \sum_{i=1}^{m} b_i y_i$$

□

19.3.2 Introduction :

Today's tool is an adaption of a fundamental tool in the design of algorithms for linear programming and combinatorial optimization : the primal-dual method.

We know,

Primal P : $\max\{c^T X | AX \leq b, X \geq 0\}$

Dual D : $\min\{b^T Y | A^T Y \geq c, Y \geq 0\}$

from Strong duality theorem, we know that, at optimum,

$c^T X^* = b^T Y^*$- (1)

Substituting c with $A^T Y$ (from D) in LHS of (1), we get

$c^T X^* \leq (A^T Y^*)^T X^* \Rightarrow$*LHS of (1)*

Solving RHS of above more, we get,

$(A^T Y^*)^T X^* = Y^{*T} A X^*$

Substituting AX to b brings $\leq$ inequality to condition from P, which gives,

$Y^{*T} A X^* \leq Y^{*T} b$

finally, we can write above relations substituting in (1), as below,we get fact 1,

Fact 1. $c^T X^* = (A^T Y^*)^T X^*$ and $Y^{*T} A X^* = Y^{*T} A X^* = Y^{*T} b$

Fact 2. (1) $X^{*T}(A^T Y - c) = 0$; that is for all $1 \leq j \leq m$ either $X_j^* = 0$ or $A_j.y^* = c_j$.

(2) $y^*(AX^* - b) = 0$;that is, for all $1 \leq i \leq n$,either $Y_i^* = 0$or $A_i X^* = b_i$.

Proof. follows from derivation of Fact 1. Since there's equality between $b.y^* and\ c.x^*$,all the inequalities are equality, which translates to the fact above.

Fact 3. Let x and y be feasible solutions to (1) and (2) satisfying the following conditions :

(1) For all $1 \leq j \leq m$,either $x_j = 0$ or $A_j.y \geq c_j/\alpha$ and

(2) For all $1 \leq i \leq n$,either $y_i = 0$ or $a_i.x \leq \beta b_i$.

Proof. This is derived from results of Fact 1 and Fact2.

19.3.3 Primal-Dual based approximation algorithm

step 1. Formulate the given problem as an IP. Relax the variable constraints to obtain the primal LP P, then find the dual D.

step 2. Start with a primal infeasible solution x and dual feasible solution y. Usually $x = 0$ and $y = 0$.

step 3. until x is feasible do:

a) Increase the value of y_i in some fashion until dual constraints go tight, i.e. $\sum_{i=1}^{m} a_{ij} y_i = \alpha c_j$ for some j while always maintaining feasibility of y.

b) Select some subset of tight dual constraints and increase the value of primal variables corresponding to these constraints by an integral amount.

step 4. Cost of dual solution is used as a lower bound for primal optimization problem. Note that the approximation guarentee of the algorithm is $\alpha\beta$.

Example In this section we briefly describle how to use pcs to solve Dual problem. Now,take the following example.

$$
\begin{array}{lll}
z_1 & = & max \quad 2x_1 + 4x_2 + 3x_3 + x_4 \\
\text{Constraints:} & & 3x_1 + x_2 + x_3 + 4x_4 \leq 12 \\
& & x_1 - 3x_2 + 2x_3 + 3x_4 \leq 7 \\
& & 2x_1 + x_2 + 3x_3 - x_4 \leq 10 \\
& & x_i \geq 0
\end{array}
$$

We know solution of above: $z_1 = 42; x_1 = 0; x_2 = 10.4; x_3 = 0; x_4 = 0.4$.Now ,we used this information to solve it's dual using complementary slackness. The following is it's dual:

$$
\begin{array}{lll}
z_2 & = & min \quad 12y_1 + 7y_2 + 10y_3 \\
\text{Constraints:} & & 3y_1 + y_2 + 2y_3 \geq 2 \\
& & y_1 + y_2 + 2y_3 \geq 2 \\
& & y_1 - 3y_2 + y_3 \geq 4 \\
& & y_1 + 2y_2 + 3y_3 \geq 3 \\
& & 4y_1 + 3y_2 - y_3 \geq 1 \\
& & y_i \geq 0
\end{array}
$$

In above example x_2 and x_4 are positive. So their corresponding constraints are tight in dual by PCS. Similarly, constraints 1 and 3 are tight but not second constriant. So corresponding varible y_2 equal to 0. then

$$
\begin{aligned}
y_1 + y_3 &= 4 \\
4y_1 - y_3 &= 1
\end{aligned}
$$

After solving above equations we got $y_1 = 1; y_3 = 3$ and $z_2 = 42$. So for this primal and dual problems optimas are same.

Problem Take the following LP:

$$
\begin{array}{ll}
\max & x_1 - x_2 \\
\text{subject to} & -2x_1 + x_2 \leq 2 \\
& x_1 - 2x_2 \leq 2 \\
& x_1 + x_2 \leq 5 \\
& x \geq 0
\end{array}
$$

The dual is:

$$
\begin{array}{ll}
\min & 2y_1 + 2y_2 + 5y_3 \\
\text{subject to} & -2y_1 + y_2 + y_3 \geq 1 \\
& y_1 - 2y_2 + y_3 \geq -1 \\
& y \geq 0
\end{array}
$$

Suppose I claimed that (1,4) solved primal.How could you check this using complementary slackness?

19.4 Example: Numerical Problem

19.5 Shortest Path

19.5.1 Shortest Path Problem:Ver1

Data

1. a digraph $D = (N, A)$ with $|N| = n$ nodes and $|A| = m$ arcs;

2. a source node $s \epsilon N$;

3. a cost function c : $A \to R$.

Problem Statement -

Find all minimum cost (i.e. shortest) paths from s to all nodes N. The problem is also called Shortest Path Tree/Arborescence Problem, because of a property of its solution: the set of all shortest paths forms a spanning arborescence rooted in s.

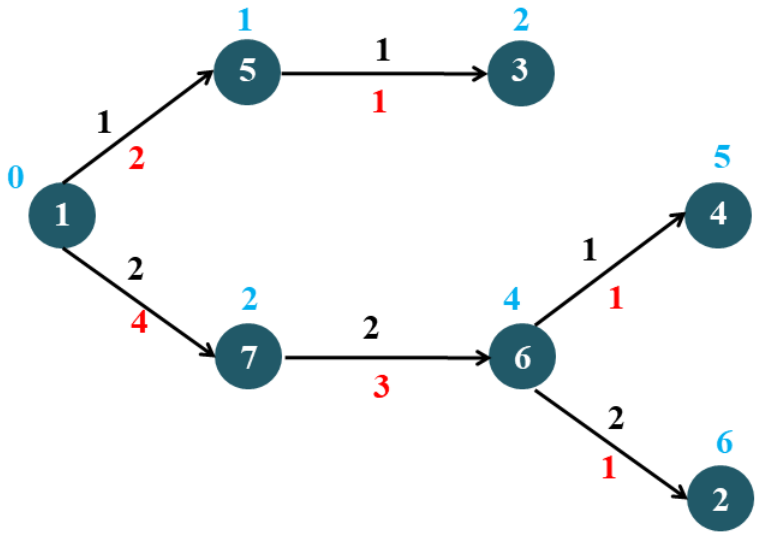


Figura: A shortest paths arborescence (s = 1). Costs are black. Flows are red. Distances are blue.

Primal-Dual pair

P) minimize $\sum_{(i,j)\epsilon A} c_{ij} x_{ij}$

s.t. $\sum_{(j,i)\epsilon \delta_{i-}} x_{ji} - \sum_{(i,j)\epsilon \delta_i^+} x_{ij} = 1 \ \ \forall i \epsilon N \backslash \{s\}$

$\sum_{(j,s)\epsilon \delta_s^-} x_{js} - \sum_{(s,j)\epsilon \delta_s^+} x_{sj} = 1 - n$

$x_{ij} \geq 0 \ \ \forall (i,j) \epsilon A$

D) maximize $\sum_{i \epsilon N \backslash \{s\}} y_i + (1-n) y_s$

s.t. $y_j - y_i \leq c_{ij} \ \ \forall (i,j) \epsilon A$

$y_i \ \epsilon R \ \ \forall i \epsilon N$

Observation 1. If we add a constant α to each y variable, nothing changes. Hence we are allowed to fix one variable:$y_s = 0$

Observation 2. We have m inequality constraints, $n-1$ original y variables and m slack variables. The LP tableau of the dual problem has m rows and $n-1+m$ columns. Hence in each base solution of D there should be m basic variables and $n-1$ non-basic (null) variables. For the complementary slackness theorem, there should be red $n-1$ basic (positive) variables in the primal problem.

Observation 3. We have n equality constraints that are not linearly independent: summing up all the rows we

obtain $0 = 0$. Hence we are allowed to delete one constraint: we delete the flow conservation constraint for s.

Observation 4. We have now $n-1$ equality constraints and m variables. The LP tableau of P has $n-1$ rows and m columns. Hence in each base solution of P there are $n-1$ basic variables and $m-(n-1)$ non-basic variables.

Complementary slackness conditions (CSC)

P') minimize z = $\sum_{(i,j)\epsilon A} c_{ij}x_{ij}$

s.t. $\sum_{(j,i)\epsilon\delta_i^-} x_{ji} - \sum_{(i,j)\epsilon\delta_i^+} x_{ij} = 1 \ \forall i\epsilon N\backslash\{s\}$

$x_{ij} \geq 0 \ \forall (i,j)\epsilon A.$

D') maximize $w = \sum_{i\epsilon N\backslash\{s\}} y_i$

s.t. $y_j - y_i \leq c_{ij} \ \forall (i,j)\epsilon A$

$y_i \epsilon R \ \forall i\epsilon N\backslash\{s\}$

Primal CSCs: $x_{ij}(y_i + c_{ij} - y_j) = 0$

Basic variables in P' correspond to active constraints in D'.

Only arcs (i,j) for which $y_i + c_{ij} = y_j$ can carry flow x_{ij}.

By applying above constraints and any of the graph algorithms Dijkstras or Ford Fulkerson etc. problem is solved.

For example,

The Ford-Fulkerson algorithm (1962)

Data structures:

i) a predecessor label, π_i for each node $i\epsilon N$;

ii) a cost label, y_i for each node $i\epsilon N$.

Algorithm

Step FF1 (initialization):

Set $y_s = 0$ and $y_i = \infty \ \forall i\epsilon N\backslash\{s\}$.

Set $\pi_i = nil \ \forall i\epsilon N$.

Step FF2(Iteration):

Select an arc $(i,j)\epsilon A$ such that $y_j - y_i > c_{ij}$.

If such an arc exists

then Set $y_j = y_i + c_{ij}, \pi_j = i$ and repeat;

else terminate;

Feasiblity

After initialization (Step FF1) we have neither primal feasibility nor dual feasibility.

Primal viewpoint: We have $\pi_i = nil$ for all $i\epsilon N$; hence no flow enters any node.

Dual viewpoint: We have $y_i = \infty$ for all $i\epsilon N\backslash\{s\}$; hence all constraints $y_i - y_s \leq c_{si}$ are violated.

The FF algorithm maintains the CSCs and iteratively enforces primal and dual feasibility.

19.6 Example: MST

19.6.1 MST:Ver1

Solution. Starting with LP formulation of MST, we get,

$\min \sum_e c_e x_e$

s.t. $\sum_{e \, crosses \, \prod} x_e \geq |\prod| - 1 \, \forall \prod$

$x_e \geq 0 \; \forall e \epsilon E$

Here is the dual.

$\max \sum_{\prod} y_{\prod}(|\prod| - 1)$

s.t. $\sum_{e \, crosses \, \prod} y_{\prod} \leq c_e \; \forall e \epsilon E$

$y_{\prod} \geq 0 \, \forall \prod$

In the algorithm, first note that we need to maintain the $y_{\prod}$s. Initially they are all zeroes. At any stage we need to improve the cost function. In general, we can do that by increasing some of the ys and decreasing some. To keep things as simple as possible, we will try and do this by only increasing one of them at a time.

Let us assume that all c_es are strictly positive, so initially, none of the constraints are tight. We wish to increase the cost function. To do so we have to increase some $y_{\prod}$. Which onne? It seems that we gain the most by increasing the one where each vertex is in a separate partition. So, suppose we start to increase this. Ho much can we increase it by? We see that we can increase this upto the weight of the minimum weight edge. At this point the inequalities corresponding to all edges of minimum weight will be tight.

Let us consider the generic step. So suppose that at some stage we have some y. As per our recipe we need to consider only the inequalities which are equalities. So, let F denote the set of edges which are currently tight (the corresponding inequalities are tight.) We need to increase some $y_{\prod}$ such that for each of these edges the sum of the increases in the y that the edge crosses is at most zero. Thsi means we can increase a $y_{\prod}$ such that none of the edges of F cross $\prod$. How do we find such a $\prod$? The most natural is to find the connected components and put each component in one part.

Here then is the algorithm for MSTs :

1. Inititalization: We think of all $y_{\prod}$s to be zero. Note that we cannot explicitly set them.

2. Iterative Step: Let $E^{'}$ denote the set of edges which are tight. Find the connected components of the graph $G^{'} = (V, E')$. Increase $y_{\prod}$till some edge becomes tight, where the parts of $\prod$are the connected components of G'.

3. The previous step terminates when we get one connected component.

Proof of Optimality

We can prove that the above algorithm gives an optimum solution by exhibiting a primal and dual solution of the same cost.

Cost of primal is given by:

$\sum_e c_e$where e is chosen by the algorithm

$= \sum_e \sum_{\prod e \, crosses \, \prod} y_{\prod}$

$= \sum_{\prod} \sum_{ee \, crosses \, \prod} y_{\prod}$

$= \sum_{\prod} y_{\prod} \sum_{e \, crosses \, \prod} 1$

$= \sum_{\prod} y_{\prod} (number \, of \, edges \, which \, crosses \, \prod)$

$= \sum_{\prod} y_{\prod} (|\prod| - 1)$

This is exactly the cost of dual. So we have proved the cost of primal is same as the cost of the dual proving optimality.

MST: Ver2

Here is LP formulation of MST

Min $\sum_{e \in E} c_e x_e$
where $x_e = \begin{bmatrix} 1 & \text{if e is included} \\ 0 & \text{otherwise} \end{bmatrix}$

$$\sum_{e \in E} x_e \geq |V| - 1 \tag{19.8}$$

$$\sum_{\substack{e:e=(u,v) \\ u \in S \\ v \in S\prime}} x_e \geq |S| - 1 \ \ \forall \ \ S \leq V \tag{19.9}$$

$$x_e \geq 0 \ \ \forall \ \ e \in E \tag{19.10}$$

Where equation (8) ensures that we have at least $|V| - 1$ edges in solution. Any MST have exactly $|V| - 1$ edges. Constraint (9) ensures that there is no cycle formation in any subset of S vertex.There can be at most $|S| - 1$ edges b/w vertexes of S in the solution.
Dual of Above:

$$Max(|V| - 1)\alpha + \sum_{S \leq V} (1 - |S|)\beta_s \tag{19.11}$$

$$st \ \ \alpha - \sum_{s,u,v \in S} \beta_s \leq C_e \ \ \forall e = (u,v) \in S \tag{19.12}$$

$$\beta_s \geq 0 \ \ \forall \ \ S \leq V, \alpha \geq 0 \tag{19.13}$$

Algorithm

1. **Initialization:** We think of all $(\alpha - \sum \beta_s)$ to be 0.Note that we cannot explicitly set them.
2. **Iterative Steps:** Let E_p deonte the set of edges which are tight. Find connected components of graph $G_p = (V, E_p)$. Increase $(\alpha - \sum \beta_s)$ till some edge becomes tight where the parts of V are the connected components of G_p.
3. The previous step terminates when we get one connected component.

19.7 Set Cover

Set Cover: Ver 1

Here is Lp-Ip formulation of set cover.

$$min \sum_{s \in S} c_s x_s \tag{19.14}$$

$$st \ \ \sum_{s:u \in S} x_s \geq 1 \ \ \forall u \in V \tag{19.15}$$

$$x_s \geq 0 \ \ \forall \ \ s \in S \tag{19.16}$$

where $x_s = \begin{bmatrix} 1 & \text{if s take} \\ 0 & \text{otherwise} \end{bmatrix}$
where each set $s \in S$
Dual:

$$Max \sum_{y_u} y_u \tag{19.17}$$

$$st \ \ \sum_{u \in S} y_u \leq c_s \forall s \in S \tag{19.18}$$

$$y_u \geq 0 \ \ \forall u \in V \tag{19.19}$$

Approximate Algorithm:
w_i = weight of s_i is taken

1. $y \leftarrow 0$ // start with a feasible solution

2. $J \leftarrow \phi$ // Nothing yet in the cover
3. $I \leftarrow [1, ..., m]$ // Elements yet to be covered
4. **while** $I \neq \phi$ **do**
5. $Pick \;\; i \in I$ // and now try to increase y_i as much as possible
6. $j_i \leftarrow argmin \; [\; w_j | \quad j \in [n], i \in S_j \;]$
7. $y_i \leftarrow w_{j_i}$ // this is the most y_i could be increased to
8. // the dual constraint corresponding to j_i shall become "binding" (becomes equality)
9. **for** each j $\leftarrow$ where $\;\; i \in S_j$ **do**
10. $w_j \leftarrow w_j - w_{j_i}$
11. **end for**
12. $J \leftarrow J \cup \{j_i\}$
13. $I \leftarrow I - S_{j_i}$ // those in $\;S_{j_i}$ are already covered
14. **end while**
15. Return J

19.7.1 Set cover:Ver2

Solution.

The general idea is to work with an LP-relaxation of an NP-hard problem and its dual. Then the algorithm iteratively changes a primal and a dual solution until the relaxed primal-dual complementary slackness conditions are satisfied.

Primal-Dual Schema

Consider the following primal program:

minimize $val(x) = \sum_{j=1}^{n} c_j x_j$,

subject to $\sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad i = 1, ..., m$,

$x_j \geq 0 \quad j = 1, ..., n.$

The dual program is:

maximize $\overline{val(y)} = \sum_{i=1}^{m} b_i y_i$,

subject to $\sum_{i=1}^{m} a_{ij} y_i \leq c_j \quad j = 1, ..., n$,

$y_i \geq 0 \qquad i = 1, ..., m.$

We will move forward using this schema and by ensuring one set of conditions and suitably relaxing the other. We will capture both situations by relaxing both conditions. If primal conditionsare to be ensured, we set $\alpha = 1$below, and if dual conditions are to be ensured, we set $\beta = 1$.

Primal Complementary Slackness Conditions. Let $\alpha \geq 1$. For each $1 \leq j \leq n$:

either $x_j = 0 \, or \, c_j/\alpha \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$.

Dual Complementary Slackness Conditions. Let $\beta \geq 1$.For each $1 \leq i \leq m$:

either $y_i = 0 \, or \, b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \beta b_i$.

Lemma. If x and y are primal and dual feasible solutions respectively satisfying the complementary slackness conditions stated above, then $val(x) \leq \alpha\beta\overline{val(y)}$.

Proof. We calculate directly using the slackness conditions and obtain

$val(x) = \sum_{j=1}^{n} c_j x_j \leq \alpha \sum_{j=1}^{n} (\sum_{i=1}^{m} a_{ij} y_i) x_j$

$= \alpha \sum_{i=1}^{m} (\sum_{j=1}^{n} a_{ij} x_j) \, y_i \leq \alpha\beta \sum_{i=1}^{m} b_i y_i = \overline{val(y)}$

which was claimed.

Procedure to solve set cover problem by this algorithm:

The algorithm starts with a primal infeasible solution and a dual feasible solution; usually these are x = 0 and y=0 initially. It iteratively improves the feasibility of the primal solution and the optimality of the dual solution ensuring that in the end a primal feasible solution is obtained and all conditions stated above, with a suitable choice for α and β, are satisfied. The primal solution is always extended integrally, thus ensuring that the final solution is integral. The improvements to the primal and the dual go hand-in-hand: the current primal solution is used to determine the improvement to the dual, and vice versa. Finally, the cost of the dual solution is used as a lower bound on the optimum value and by above mentioned Lemma, the approximation gaurantee is $\alpha\beta$.

19.8 Example:Weighted Vertex Cover

19.8.1 Ver1

Primal:	$Min \sum_{v \in V} w_v x_v$
Constraints	$x_u + x_v \geq 1 \forall (u,v) \in E$
	$x_v \geq 0$
Dual:	$Max \sum_{e \in E} y_e$
Constraints	$\sum_{v \in e} y_e \leq w_v \forall y_e \geq 0.$

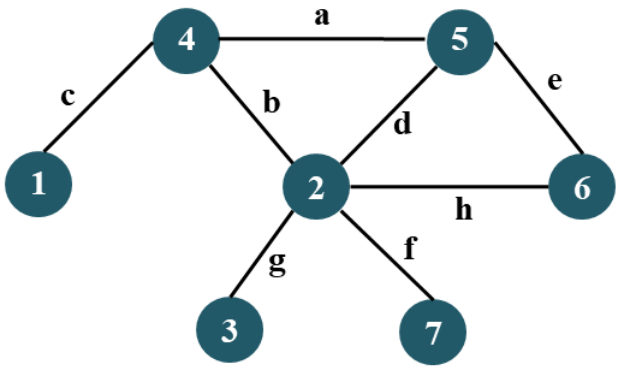


Figure 19.2: Weighted Vertex Cover

Iterative Steps

1. Start with $X = [0 \ldots 0]$ and $Y = [0 \ldots 0]$.It is feasible for D(Dual) but not for P(Primal).Keep Modifing X such that It is feasible for P.
2. Increase y for "a" edge to 4.So that at vertex 4 constrait is tight.$y_a = 4 \implies x_4 = 1$ and freeze a,b and c.(freeze means we have to set value of variables in such a way that all variables are satisfying that constraint .In above example we have to make b=0 and c=0 for $y_a + y_b + y_c = 4$).
3. Increase y for "e" to 1.$y_e = 1 \implies x_5 = 1$ and freeze e and d.
4. $y_g = 2 \implies x_2 = 1$ and freeze g,h and f.

Now solution is satisfying for both P and D. In above case $\beta = 2$.

19.8.2 Iterative Algorithm

1. Start with X_0 Such that D is feasible.

2. Find Z such that D is remains feasible and P is closer to feasible.
3. $X_{i+1} = X_i + Z$.

19.8.3 Summary:

1. PD is exact algorithm for combinational optimized problems.
2. PD for Approximate Algorithm ($1 \leq j \leq n$ either $y_i = 0$ or $\frac{b_j}{\beta} \leq \sum_{j=1}^{n} a_{ij}x_j \leq b_j$ where $\beta \geq 1$.)

PD Method:

1. start with init guess.
2. move to better guess guided by some constraints.
3. when nothing better can be found stop.

19.8.4 Minimum-weighted Vertex Cover:Ver2

Statement - Given weights w_i on $i \epsilon V$, find a min-cost subset S of V, such that at least one endpoint of every edge is in S.

Solution :

Ground Set : V

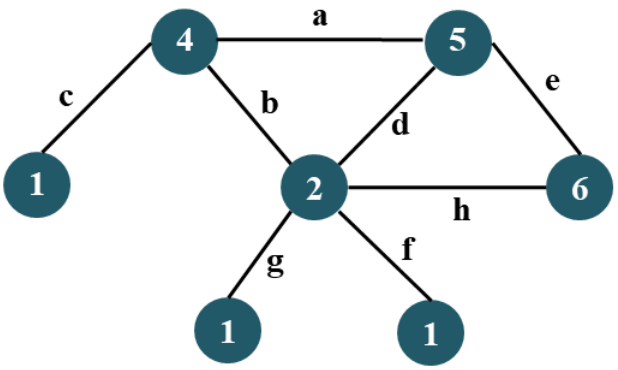
Costs : $w_i, i \epsilon V$

Sets: $\{u, v\}, where\,(u, v)\,\epsilon E$

Primal and Dual equation for above problem -

Primal: $min \sum_{u \epsilon V} w_v x_v$ such that $x_u + x_v \geq 1$and $x_u \geq 0$

Dual: $max \sum_{e \epsilon E} y_e$ such that $\sum_{e, v \epsilon e} y_e \leq w_v$ and $y_e \geq 0$



The Complementary Slackness conditions are as follows :

PCS - $x_v > 0 \Rightarrow \sum_{e \epsilon E} y_e = w_v,$

DCS - $y_e > 0 \Rightarrow x_u + x_v = 1.$

These conditions will guide us to design an approximate algorithm. Let us see how to interpret these conditions. A primal dual algorithm will construct primal and dual feasible solutions simultaneously. To ensure that these solutions are optimal, the primal condition says that a vertex should be picked only if this vertex is saturated by the matching, and the dual condition says that an edge should be picked only if exactly one vertex is chosen. Dual condition is difficult because we can pick one vertex for each edge in the matching but still cover all the edges in the graph. So let us relax the dual condition by a factor of 2, i.e. set $\beta = 2$:

$$y_e > 0 \Rightarrow x_u + x_v \leq 2.$$

This relaxation now makes the problem much easier because the dual condition is now satisfied automatically as $x_u and x_v$ are at most 1. Now, we just need to construct primal and dual solutions satisfying the primal complementary slackness condition only. This can be achieved by the following simple algorithm :

1. Initialization: x=0,y=0.

2. When there is an uncovered edge:

(a)Pick an uncovered edge, and increase y_e until some vertices go tight.

(b)Add all tight vertices to the vertex cover.

3. Output the vertex cover.

Clearly this algorithm will produce a feasible solution for the vertex cover problem, and also satisfy the primal complementary slackness condition.

Steps -

1)Increase y for a $\Rightarrow x_4 = 1 \Rightarrow$Freeze a, b and c.

2)Increase y for e to 1 $\Rightarrow x_5 = 1 \Rightarrow$Freeze d and e.

3)Increase $y_g to 2 \Rightarrow x_2 = 1 \Rightarrow x_2 = 1 \Rightarrow$Freeze f, g and h.

Here, $1 \leq x_u + x_v \leq \beta$, where $\beta = 2$.

Above algorithm is PD based approximate algorithm.

In summary, to solve a problem by P-D approach,

i) start with X^0, such that dual is feasible, then

ii) find Z, such that dual remains feasible

iii) find $X^{i+1} \leftarrow X^i + Z$

19.8.5 Weighted Vertex Cover via Primal-Dual method

Weighted Vertex Cover (WVC): Given an undirected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$ and a cost function on vertices : $C \rightarrow Q^+$, find a subset $C \subseteq V$ such that every edge $e \in E$ has at least one end point in C and C has minimum cost.
Formulate vertex cover as following IP :
For each vertex $i \in V (V = (1, 2, 3, ..., n))$
Let $x_i \in 0, 1$ be variables such that: $x_i = 1$ if $i \in C$, otherwise $x_i = 0$.
We have:

$$min \sum_{i=1}^{n} c_i x_i$$
$$s.t : x_i + x_j \geq 1 \qquad \forall (i, j) \in E$$
$$x_i \in 0, 1 \qquad \forall i \in V$$

The corresponding LP P of above IP:

$$min \sum_{i=1}^{n} C_i x_i$$
$$s.t : x_i + x_j \geq 1 \qquad \forall (i, j) \in E$$
$$x_i \geq 0 \qquad \forall i \in V$$

Assign a dual variable y_{ij} to the constraint $x_i + x_j \geq 1$, We have the corresponding dual D:

$$max \sum_{(i,j) \in E} y_{ij}$$
$$s.t : \sum_{j:(i,j) \in E} y_{ij} \leq C_i \qquad \forall i \in V$$

$$y_{ij} \geq 0 \qquad \forall (i,j) \in E$$

Let us choose $\alpha = 1$ and $\beta > 1$. To ensure primal conditions and suitably relax dual conditions:
For each vertex $i \in E$: either $x_i = 0$ or $\sum_{j:(i,j)\in E} y_{ij} = C_i$
For each edge $(i,j) \in E$: either $y_{ij} = 0$ or $1 \leq x_i + x_j \leq \beta$ where $\beta > 1$
Therefore, when $x_i \neq 0$,then $\sum_{j:(i,j)\in E} y_{ij} = C_i$. When $\sum_{j:(i,j)\in E} y_{ij} = C_i$ for some i, we say that this constraint goes tight.

19.8.6 Primal-Dual algorithm for WVC

step 1. Initialise $x = 0$ and $y = 0$

step 2. While $E \neq \phi$ do

a) Select non-empty set $E' \subseteq E$

b) Raise y_{ij} for each edge $(i,j) \in E'$ until some dual constraints goes tight i.e $\sum_{j:i\in(i,j)} y_{ij} = C_i$ for some i.

c) Let S be the set of vertices i corresponding to the dual constraints that just went tight.

d) For each$i \in S$ set $x_i = 1$ and delete all edges (i,j) from E i.e. delete all edges incident to vertices in S.

step 3. End while

step 4. Return set $C = i|x_i = 1$.

19.8.7 WVC :Ver3

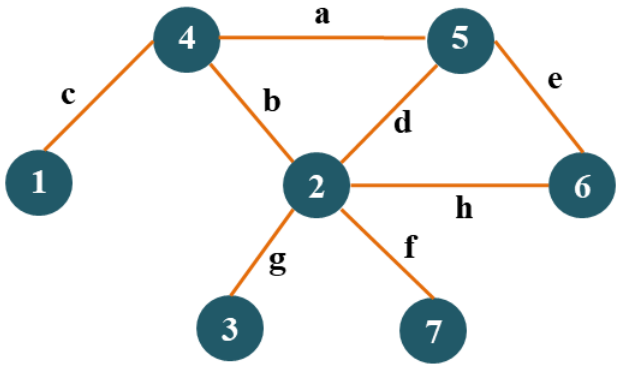


Figure 19.3: An example graph for WVC algorithm

Step 1. Start with $x = [00...0]$ and $y = [00...0]$

Step 2. Increase y for a i.e. $x_4 = 1$. This implies that x_4 can be in the vertex cover. Freeze a, b, c.

Step 3. Increase y for e i.e. $x_5 = 1$.Freeze d, e.

Step 4. Increase y for h i.e. $x_2 = 1$.Freeze f, g, h.

Now, we have got a solution which satisfies P and D both.

Lemma 19.8.1. Let x and y be the solutions obtained from the above algorithm, then x is primal feasible and y is dual feasible.

PROOF. Note that each edge (i,j) removed from E is incident on some vertex i s.t. $x_i = 1$. Additionally,the loop is terminated when all edges have been removed.Therefore, $x_i + x_j \geq 1 \forall (i,j) \in E$ i.e. x is feasible to P.
Likewise, once the constraint goes tight for some i i.e. $\sum_{j:(i,j)\in E} y_{ij} = C_i$, the algorithm removes these edges. Therefore, none of the values of y_{ij} exceed the value of C_i.Hence, y is feasible to D. □

Theorem 19.8.2. The above algorithm produces a vertex cover C with an approximation ratio 2.

PROOF. Let OPT be the cost of vertex cover. We have:

$$cost(C) = \sum_{i \in C} C_i = \sum_{i \in C} (\sum_{j:(i,j) \in E} y_{ij})$$

The last equality follows from the fact that we set $x_i = 1$ only for vertices corresponding to tight dual constraints i.e. $i \in C$.This implies $\sum_{j:(i,j) \in E} y_{ij} = C_i$.
Also note that:

$$\sum_{i \in C} (\sum_{j:(i,j) \in E} y_{ij}) = \sum_{(i,j) \in E} (\sum_{i \in C \cap (i,j)} y_{ij}) \leq 2 \sum_{(i,j) \in E} y_{ij}$$

Last inequality follows from the fact that$|(i,j)| = 2$. for all edge (i,j) i.e each edge has 2 endpoints, so we may count y_{ij} twice for each edge (i,j).
Therefore, we conclude that:

$$cost(C) = 2 \sum_{j:(i,j) \in E} y_{ij} \leq 2OPT(D) \leq 2OPT$$

□

19.9 Example: Minimum Steiner Forest via Primal-Dual method

Minimum Steiner Forest (MSF): Given an undirected graph $G = (V, E)$, a cost function $c : E \to Q^+$ and a collection of disjoint subsets of V :$S_1, ..., S_n$, find a minimum cost subgraph of G in which each pair of vertices belonging to the same S_i is connected, such a subgraph is cycle free and is called a steiner forest.

For any $X \subseteq V$, we set $f(X) = 1$ iff there exists a $u \in X$ and $v \in \bar{X}$ such that u and v belong to some set S_i, otherwise $f(X) = 0$.
Let $\delta(X)$ be the set of edges with exactly one endpoint in X. Let a binary variable x_e indicate whether the edge is chosen in the subgraph for each edge $e \in E$.

The problem can be formulated as an IP:

$$min \sum_{e \in E} c_e x_e$$

$$s.t : \sum_{e:e \in \delta(X)} x_e \geq f(X) \qquad \forall X \subseteq V$$

$$x_e \in 0, 1$$

The corresponding LP P is:

$$min \sum_{e \in E} c_e x_e$$

$$s.t : \sum_{e:e \in \delta(X)} x_e \geq f(X) \qquad \forall X \subseteq V$$

$$x_e \geq 0$$

The dual D will be:

$$max \sum_{X \in V} f_X y_X$$

$$s.t : \sum_{X:e \in \delta(X)} y_X \leq c_e \qquad \forall e \in E$$

$$y_X \geq 0$$

19.9.1 Approximation Algorithm for Minimum Steiner Forest via Primal-Dual method

Step 1. initialize $F \leftarrow \phi, y \leftarrow 0, j \leftarrow 0$

Step 2. while F is in-feasible do:

a) let χ be the set of all active X

b) simultaneosly increase y_X at same rate for each active set $\chi \in X$ until some edge e goes tight

c) refer e as e_j

d) $F \leftarrow F \cup e_j$

e) $j++$

Step 3. end while

Step 4. Reverse Delete Step: for $j = |F| to 1$ do:
if $F - e_j$ is primal feasible, then $F \leftarrow F - e_j$

Step 5. refer this set as F'

Step 6. return F'

Step 7. end

19.10 WEIGHTED SET COVER:

Given a universe $U = 1, ..., m$, a collection of subsets of U, $S = S_1, ..., S_n$ and a cost function $c : S \rightarrow Q^+$, find a minimum cost sub collection $C = S_j | 1 \leq j \leq n$ such that C covers all the elements of U.

Let c_j be the weight of subset S_j. Let x_j be a binary variable such that $x_j = 1$ if $S_j \in C$, otherwise $x_j = 0$. We have the following IP:

$$min \sum_{j=1}^{n} c_j x_j$$

$$s.t : \sum_{j:i \in S_j} x_j \geq 1 \qquad \forall i \in 1, ..., m$$

$$x_j \in 0, 1 \qquad \forall j \in 1, ..., n$$

The corresponding LP primal P :

$$min \sum_{j=1}^{n} c_j x_j$$

$$s.t : \sum_{j:i \in S_j} x_j \geq 1 \qquad \forall i \in 1, ..., m$$

$$x_j \geq 1 \qquad \forall j \in 1, ..., n$$

Let y_i be the variable corresponding to the constraint $\sum_{j:i \in S_j} x_j \geq 1$. The corresponding dual D is:

$$max \sum_{i=1}^{m} y_i$$

$$s.t : \sum_{i \in S_j} y_i \leq c_j \qquad \forall j \in 1, ..., n$$

$$y_i \geq 0 \qquad \forall i \in 1, ..., m$$

For some j if the dual constraint $\sum_{i \in S_j} y_i = c_j$ we say that this constraint goes tight and the corresponding S_j is tight. We have the following algorithm:

Step 1. initialize $x = 0, y = 0$

Step 2. while $U \neq \phi$ do:

a) Choose an uncovered element say i and raise yP_i until some set in S goes tight say S_j.

b) Choose all these sets S_j and set $x_j = 1$

c) Remove all the elements in these sets S_j from U

d) Remove all these sets S_j from collection S

Step 3. end while

Step 4. return $C = S_j | x_j = 1$

QUES 2:Derive a Primal-Dual based exact algorithm for minimum spanning tree problem.

The primal LP formulation for MST is denoted as P:

$$min \sum_e c_e x_e$$

$$s.t: \sum_{ecrosses\pi} x_e \geq |\pi| - 1 \qquad \forall \pi$$

$$x_e \geq 0 \qquad \forall e \in E$$

The corresponding dual is:

$$max \sum_\pi y_\pi (|\pi| - 1)$$

$$s.t: \sum_{ecrosses\pi} y_\pi \leq c_e \qquad \forall e \in E$$

$$y_\pi \geq 0 \qquad \forall \pi$$

In the algorithm, first note that we need to maintain the y_π. Initially they are all zeroes. At any stage we need to improve the cost function. In general we can do that by increasing some of the y and decreasing some. To keep things as simple as possible, we will try and do this by only increasing one of them at a time.
Let us assume that all c_e are strictly positive, so initially, none of the constraints are tight. We wish to increase the cost function. To do so we have to increase some y_π. Which one? It seems that we gain the most by increasing the one where each vertex is in a separate partition. So, suppose we start to increase this. How much can we increase it by? We see that we can increase this upto the weight of the minimum weight edge. At this point the inequalities corresponding to all edges of minimum weight will be tight. Let us consider the generic step. So suppose that at some stage we have some y. Let F denote the set of edges which are currently tight (the corresponding inequalities are tight.) We need to increase some y_π such that for each of these edges the sum of the increases in the y that the edge crosses is at most zero. This means we can increase a y_π such that none of the edges of F cross π. How do we find such a π? The most natural is to find the connected components and put each component in one part.
Here then is the algorithm for MST:

1. **INITIALIZATION:** we think of all y_π to be 0. Note that we can not explicitly set them.
2. **ITERATIVE STEP:** let E' denote set of edges which are tight. Find the connected components of the graph $G' = (V, E')$. Increase y_π till some edgebecomes tight when the parts of π are the connected components of G'.
3. previous step terminates when we get one connected component.

Problems

Minimum-cost branching Given a directed graph $G = (V, A)$, root $r \epsilon V$, find a min-cost subgraph such that there is a directed path from r to every other vertex.

Maximum Independent Set Given a directed graph $G = (V, A)$, finding an independent set such that adding any other vertex to the set forces the set to contain an edge.

QUES 1: Derive a Primal-Dual based approximation algorithm for sudoku problem.

QUES 2: Derive a Primal-Dual based exact algorithm for travelling salesman problem.

Chapter 20

Convex Sets and Convex Functions

20.1 Introduction

We have been mostly looking into the linear optimization (linear programming) and the associated concepts from linear algebra. We had also seen how many of these concepts (eg. duality) are more general than linear programming.

We also argued that linear programming is convex, while integer programming is not convex. What does it mean? Why it should matter to us? What are the general class of convex optimization schemes? How does one optimize when problem is non-convex or/and non-linear? We will discuss some of these in the next few lectures. Note that this is a huge area of literature. We will be discussing only a limited set of topics.

20.1.1 Line joining the points

Suppose $x_1 \neq x_2$ are two points in $\mathbb{R}^n$. Points of the form

$$y = \theta x_1 + (1 - \theta)x_2 \tag{20.1}$$

where $\theta \in \mathbb{R}$, form the line passing through x_1 and x_2. The parameter value $\theta = 0$ corresponds to $y = x_2$, and the parameter value $\theta = 1$ corresponds to $y = x_1$. Values of the parameter θ between 0 and 1 correspond to the (closed) line segment between x_1 and x_2.

Expressing y in the form

$$y = x_2 + \theta(x_1 - x_2) \tag{20.2}$$

gives another intepretation: y is the sum of the base point x_2 (corresponding to $\theta = 0$) and the direction $x_1 - x_2$ (which points from x_2 to x_1) scaled by the parameter θ. Thus, θ gives the fraction of the way from x_2 to x_1 where y lies. As θ increases from 0 to 1, the point y moves from x_2 to x_1.

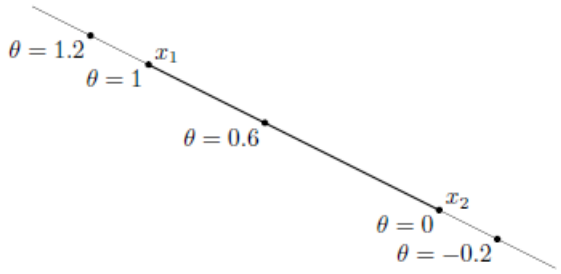


Figure 20.1: The line passing through x_1 and x_2 is described parametrically by $\theta x_1 + (1 - \theta)x_2$, where θ varies over $\mathbb{R}$.

The reason for us to familiarize ourselves with the concept of line segment is going to be clear in the next section, when we dive into convex sets, and convex functions.

20.2 Convex Sets

Definition. A set C is *convex* if the line segment between any two points in C lies in C, *i.e.*, if for any $x_1, x_2 \in C$ and any θ with $0 \leq \theta \leq 1$, we have

$$y = \theta x_1 + (1 - \theta)x_2 \ \in \ C \tag{20.3}$$

.

In other words, a set is convex if every point in the set is *seen* by every other point inside the set.

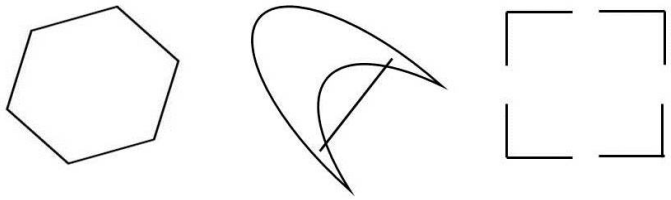


Figure 20.2: Examples of convex and non-convex sets.

Examples: Figure 20.3 shows an example of a convex set, and a non-convex set. Figure 20.2 shows an additional example at the right exterem. The square contains (the right extreme) some boundary points but not all, and is not convex.

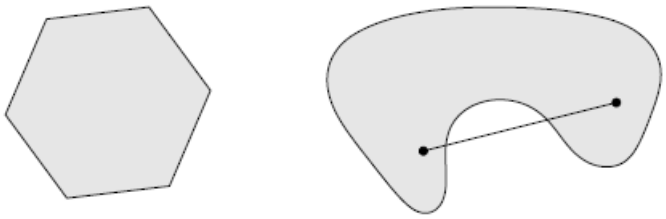


Figure 20.3: *Left*: Hexagon is a convex set, while *Right* The kidney shaped set is clearly not a convex set

Some examples of convex sets are given below:

1. The Real line
2. A line segmet $\alpha \leq x \leq \beta$ $(x \in \mathbb{R})$
3. Empty set, Singleton set $\{x_0\}$,
4. Real space $\mathbb{R}^n$
5. Hyper planes and Half spaces. Hyper planes are represented by
$$\{x \mid x^T a = b\}$$
and half spaces are represented by
$$\{x \mid x^T a \leq b\}$$
6. Euclidean Balls $\mathrm{B}(x_0, \varepsilon) = \{x \mid \|x - x_0\|_2 \leq \varepsilon\}$
7. Solution to an underdetermined system of equations. Let $Ax = b$ with A having more columns than rows (linearly independet).
8. Polyhedron $\mathrm{P} = \{x \mid Ax \leq b \;\; Cx = d\}$
A polyhedron is defined as the solution of a finite number of linear equations & inequalities
$$P = \{x \mid a_i^T x \leq b_i \;\; C_j^T x = d_j\}$$

20.2.1 Properties of Convex sets

1. Interesection of two convex sets is convex
PROOF. Suppose A and B are convex, and let $C = A \cap B$
To show C is convex, we have to show that if x_1 and x_2 are elements of C and θ is any scalar between 0 and 1, then $\theta x_1 + (1 - \theta)x_2$ is also an element of C. Since x_1 and x_2 are in C, x_1 and x_2 are in A and B. Since A is

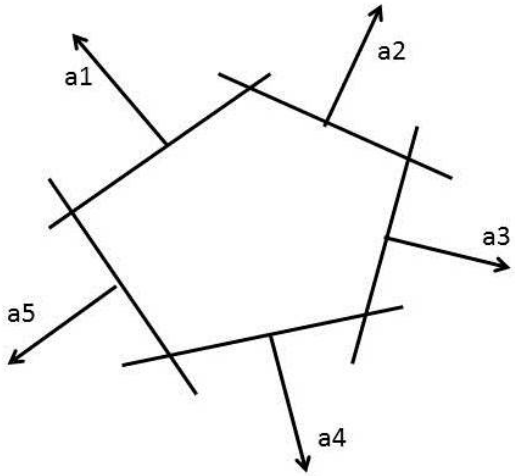


Figure 20.4: Example of a polyhedron

convex, it follows that $\theta x_1 + (1-\theta)x_2$ is in A. Similarly x_1 and x_2 are in B, and B being convex, $\theta x_1 + (1-\theta)x_2$ is in B.

Thus $\theta x_1 + (1-\theta)x_2$ is in A and B, i.e., $\theta x_1 + (1-\theta)x_2$ is in C. □

2. Union of two convex sets need not be convex

   Proof. We can provide a counter example to disprove that union of two convex sets is convex.

   Let us consider two sets on the Real line $\mathbb{R}$ A and B, where $A = [\alpha, \beta]$, and $B = [\gamma, \lambda]$, such that $A \cap B = \emptyset$ *i.e.* they are mutully exclusive. If we consider any linear combination of β and γ, then it might not be on $\mathbb{R}$, hence □

3. If $C = \{x\}$ is a convex set, the $\alpha C = \{\alpha x\}$ is also convex
4. If $C = \{x\}$ is a convex set, the $(C + t) = \{x + t\}$ is also convex
5. If $C = \{x\}$ is a convex set, the $(aC + b) = \{ax + b\}$ is also convex
6. Set Sum: If C_1 and C_2 are convex, $C_1 + C_2 = \{x + y \mid x \in S_1,\ y \in S_2\}$ is also convex.

20.2.2 Convex Hull

Before appreciating the convex hull, we look at a convex combination. We call a point of the form $\theta_1 x_1 + ... + \theta_k x_k = \sum_i \theta_i x_i$, where $\theta \in R$, $\theta_1 + \ldots + \theta_k = 1$ and $\theta_i \geq 0$, $i = 1, \ldots, k$, a *convex combination* of the points $x_1, \ldots, x_k$. A convex combination of points can be thought of as a mixture or weighted average of the points, with θ_i the fraction of x_i in the mixture.

Definition. Convex hull of a set C is the set of all convex combinations of points in C and is represented by $Conv(C)$.

$$Conv(C) = \{\theta_1 x_1 + \theta_2 x_2 + ... + \theta_k x_k\}$$

$$\text{Such that} x_i \in C\ \theta_i \geq 0 \quad \sum_i \theta_i = 1$$

Note that $Conv(C)$ is convex and $C \subseteq Conv(C)$. From the definition of the convex set, one can also see that the convex hull of a convex set is itself.

Convex hull of C is always convex, as any linear combination of points lies in the set only. In fact, it is the smallest convex set that contains C. In other words, if B is any convex set that contains C, then **conv** $C \subseteq B$. Figure 20.6 shows a convex hull of a set of points. As evident from the figure, the upper part of the hull (colored in **blue**) is called the upper hull, while the lower part of the hull (colored in **red**) is called lower hull.

- We know that R is convex. Note that set of integers Z is not. (why? a linear combination can result in a point that is not an integer.)

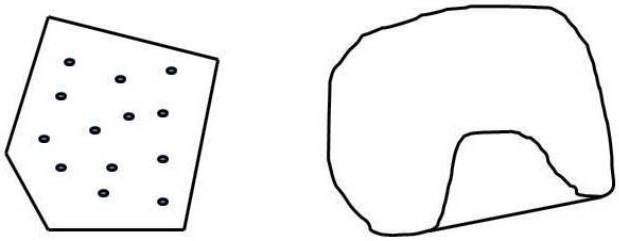


Figure 20.5: The convex hulls of two sets in R^2. Left. The convex hull of a set of thirteen points (shown as dots) is the pentagon (shown shaded). Right. The convex hull of the kidney shaped set is the shaded set.

- You may remember that Linear Programming is convex while integer programming is not. You can connect this to the fact that the real space (over which LP optimizes) is convex. While the set of integers is not.
- What does happen with LP relaxation? We find the convex hull.!!

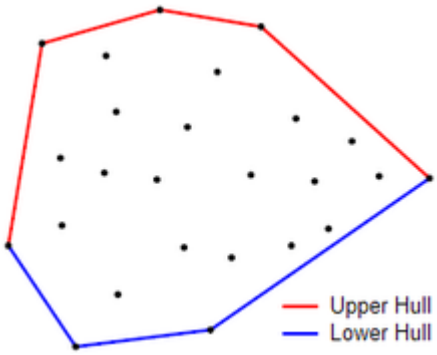


Figure 20.6: Convex hull of a set of points, showing the upper and the lower hull

20.2.3 Separating Hyperplane

Assume we are given two convex sets C and D, such that $C \cap D = \emptyset$. Then we can always find a separating hyperplane, which can separate C and D. To be precise, for any x in C, we have $a^T x \geq b$, and for any x in D, we have $a^T x < b$. In this case, we always have a separating hyperplane to separate the set of points in n-dimensional space. See Figure 20.7 for a visualzation.

If C and D are the convex sets and $C \cap D = \emptyset$, then there exists a non-zero a such that

$$\forall x \in C \quad a^T x \leq b$$

and

$$\forall x \in D \quad a^T x \geq b$$

such a hyperplane is characterized by a and is called the separating hyperplane.

20.3 Definition of Convex Functions

A Convex function can be defined and understood in multiple ways. We now look at some of these definitions in detail below.

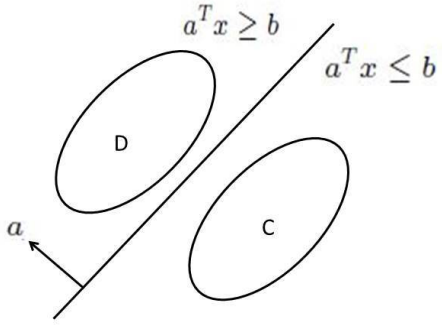


Figure 20.7: Separatimg plane for two convex sets

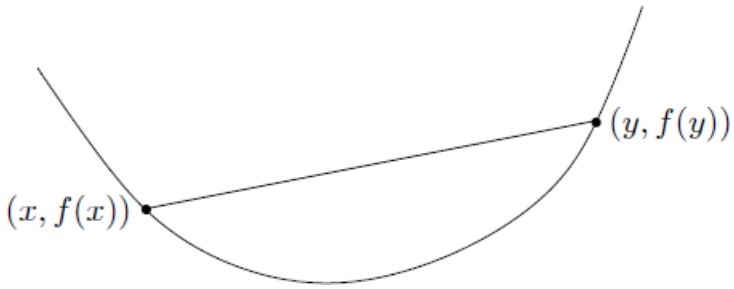


Figure 20.8: Example of a convex function. Chord joining the two points lie above the function

20.3.1 Jensen's inequality

A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if $\mathbf{dom}(\mathbf{f})$ is a convex set and if for all $x, y \in \mathbf{dom}(f)$, and θ with $0 \leq \theta \leq 1$, we have

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y). \tag{20.4}$$

This is called the **Jensen′s inequality**. In other words, it is a function defined on a convex domain such that for any two points in the domain, the segment between the two points lies above the function. Figure 20.8 shows an example of a convex function. In the figure, we find that for any two points x, and y in the domain of f, the line joining the points $(x, f(x))$, and $(y, f(y))$ lie above the function.

20.3.2 Epigraph

Epigraph of a function f is denoted by $\mathbf{epi}(f)$ and is given by

$$\mathbf{epi}(f) = \{(x,t)|x \in \mathbf{dom}(f), t \geq f(x)\} \tag{20.5}$$

The link between convex sets and convex functions is via epigraph: A function is convex if and only if its epigraph is a convex set.

In other words, if a function is convex, then its epigraph is a convex set. Again, if the epigraph of a function is a convex set, then the function is convex. Figure 20.9 (*Left*) shows the epigraph of a function f. As evident from the

figure, the epigraph can be considered as a filled bucket, where the function f can be imagined as a bucket, and the epigraph is the whole space inside the bucket.

Since the epigraph of a function is the part above that function, and if the epigraph is s convex set, that means that any two point inside that the epigraph should lie inside that only. That essentially means that any two points inside the function can be connected by a line segment lying entirely inside the function.

Figure 20.9 (*Right*) shows the epigraph of a non-convex function f. As evident from the figure, the line segment connectting any two points x and y does not entirely lie inside the function. So, there exists atleast one point among the set of points given by $p = \theta x + (1-\theta)y$ for any value of θ such that, $0 \leq \ \theta \ \leq 1$, which does not lie above the function, thus dissatisfying the inequality

$$f(\theta x + (1-\theta)y) \ \leq \ \theta f(x) + (1-\theta)f(y). \tag{20.6}$$

It has to be noted, that the epigraph of the function is not convex set, hence proving that the function is non-convex.

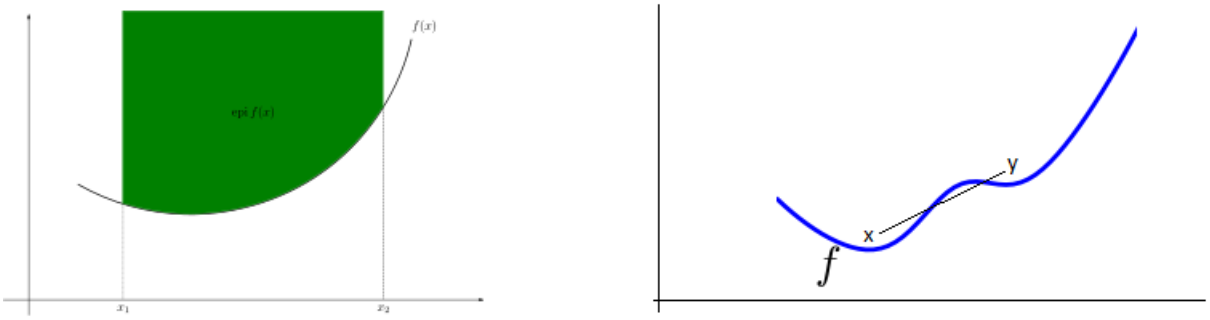


Figure 20.9: *Left*: Epigraph of a convex function. Line joining any two points inside the green region lies inside the epigraph. *Right*: For a non-convex function, line joining two points x and y does not lie completely inside the epigraph of the function

20.3.3 Hessians

A function is convex if its domain is convex and its Hessian is a PSD (Positive Semi Definite) matrix.

$$H \ = \ \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{n\times n}$$

$$H \ = \ \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ & \cdots & & \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}_{n\times n}$$

If you are not familiar with the terminology of Hessian, please read the appendix. More insight into this is seen in the following two subsections.

20.3.4 First-order conditions

If we assume that the function f is differentiable at each point in the domain of f, that is gradient ∇ of f exists. The f is convex if and only if **dom** f is convex and

$$f(y) \ \geq \ f(x) + \nabla f(x)^T (y - x) \tag{20.7}$$

holds for all $x, y \in$ **dom** f.This inequality shows that from the local information of a convex function, we can derive its global properties. This is one of the most important property of a convex function, and is crucial in the field of convex optimizations problems. From the inequality, we can say that if gradient of the function f is 0. In other

words, if $\nabla f(x) = 0$, then for all $y \in \textbf{dom } f$, $f(y) \geq f(x)$, that is , x is a global minimizer of the function f. Hence, value of $f(x)$ is minimum at that point.

This is already known to us, because for any function, we can minimize that function by setting its derivative to 0, then proving that the function is minimized if the double derivative is less than 0. The same thing is valid for convex functions as well.

20.3.5 Second-order conditions

If we assume that f is twice differentiable, that is, its *Hessian* or second-order derivative *i.e.* ∇^2 exists at each point in **dom** f, which is open. Then we can say, that f is convex if and only if **dom** f is convex and its Hessian is positive semidefinite: for all $x \in \textbf{dom}(f)$.

$$\nabla^2 f(x) \geq 0 \tag{20.8}$$

In others words, if the derivative of the function is non-decreasing, then the function is said to be convex. That is, the graph of the function have positive curavature at any point x. In addition to that, the **dom** f should be convex set.

20.4 More on Convex Functions

20.4.1 Strictly convex functions

The above definitions of the convex functions are not for strictly convex functions. If we use $<$ instead of $\leq$ in equation 20.10, we obtain the strictly convex functions.

20.4.2 Sublevel Sets

The $\alpha - \textit{sublevel set}$ of a function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as

$$C_\alpha = \{x \in \textbf{dom}(f) | f(x) \leq \alpha\}. \tag{20.9}$$

In other words, $\alpha - \textit{sublevel set}$ of a function f is a set of all values of x in the domain of f, for which the value of $f(x)$ will be less that or equal to α. Figure 20.10 explains the concept of $\alpha - \textit{sublevel set}$ of a function f.

Lemma 20.4.1. Sublevel sets of a convex function are convex, for any value of α.

Proof. The proof is immediate from the definition of convexity. Let us consider two elements x and y from the set C_α. Then, $f(x) \leq \alpha$ and also, $f(y) \leq \alpha$, and so $f(\theta x + (1-\theta)y) \leq \alpha$ for $0 \leq \theta \leq 1$, and hence $\theta x + (1-\theta)y \, \epsilon \, C_\alpha$. So any linear combination of x and y is present in the set. Since we have considered a general value of α, so we can say that for any convex function, α sublevel sets of the function is also convex, for any value of α. □

But it is to be noted, that the converse is not true: a function can have all its sublevel sets convx, but not be a convex function. An example of such a setup is $f(x) = -e^x$ is not convex on $\mathbb{R}$, although all its sublevel sets are convex. In fact, the function is strictly concave.

20.4.3 Convexity Preserving operations over functions

1. If f is convex, then αf *is convex* for $\alpha \geq 0$
2. If f and g are two convex functions, then
   (a) $f + g$ is convex
   (b) $\max(f, g)$ is convex

   Proof:

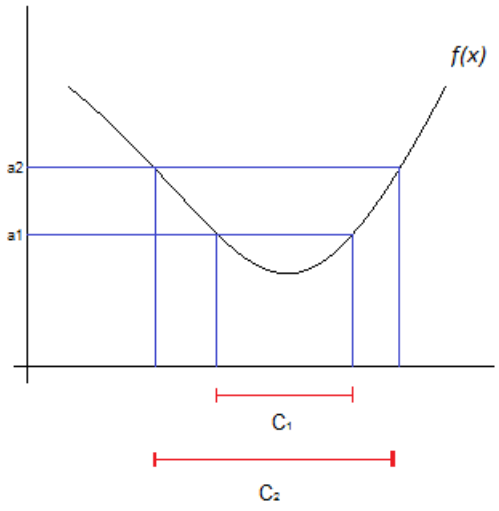


Figure 20.10: $\alpha - sublevel\ set$ of a function $f(x)$. C_1 is the α_1 set of f. C_2 is the α_2 set of f. $C_1, C_2 \in$ **dom** f

(a) Given f and g are convex, let $h(x) = f(x) + g(x)$
since f and g are convex consider x_1 and $x_2 \in dom(f), dom(g)$ and $dom(h)$

$$
\begin{aligned}
f(\theta x_1 + (1-\theta)x_2) &\leq \theta f(x_1) + (1-\theta)f(x_2) \\
\text{Similarly } \quad g(\theta x_1 + (1-\theta)x_2) &\leq \theta g(x_1) + (1-\theta)g(x_2) \\
h(\theta x_1 + (1-\theta)x_2) &= f(\theta x_1 + (1-\theta)x_2) + g(\theta x_1 + (1-\theta)x_2) \\
\implies \quad h(\theta x_1 + (1-\theta)x_2) &\leq \theta(f(x_1) + g(x_1)) + (1-\theta)(f(x_2) + g(x_2)) \\
&= \theta h(x_1) + (1-\theta)h(x_2) \\
\therefore \quad h(\theta x_1 + (1-\theta)x_2) &\leq \theta h(x1) + (1-\theta)h(x_2) \\
&\implies \quad h\ is\ convex
\end{aligned}
$$

Let $F(x) = max(f(x), g(x))$

$$
\begin{aligned}
F(\theta x_1 + (1-\theta)x_2) &= max\{f(\theta x_1 + (1-\theta)x_2), g(\theta x_1 + (1-\theta)x_2)\} \\
&\leq max\{\theta f(x_1) + (1-\theta)f(x_2)\,,\, \theta g(x_1) + (1-\theta)g(x_2)\} \\
&\leq \theta max\{f(x_1), g(x_1)\} + (1-\theta)max\{f(x_2), g(x_2)\} \\
&\leq \theta F(x_1) + (1-\theta)F(x_2) \\
&\implies \quad F\ is\ convex
\end{aligned}
$$

Hence proved.

Note that this property can be generalised to any linear combination of convex functions, not just two.

20.4.4 Concave Functions

A function $f : \mathbb{R}^n \to \mathbb{R}$ is *concave* if **dom**(**f**) is a convex set and if for all $x, y \in$ **dom**(f), and θ with $0 \leq \theta \leq 1$, we have

$$f(\theta x + (1-\theta)y) \geq \theta f(x) + (1-\theta)f(y). \tag{20.10}$$

1. $f(x) = -x^2$
2. $g(x) = \sqrt{x}$
3. sin function is concate in the interval $[0, \pi]$

Question: If a function f is convex, will $-f$ be concave?

Question: Should every function be either convex or concate?

20.4.5 Quasiconvex functions

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is called *quasiconvex* or *unimodal* if its domain and its sublevel sets

$$S_\alpha = \{x \in \mathbf{dom} f | f(x) \leq \alpha\} \tag{20.11}$$

for $\alpha \in \mathbb{R}$, are convex. Figure 20.11 shows a quasiconvex function. The function has all α-sublevel sets as convex sets, but still the function is not a convex function. In other words, the α-sublevel sets should be an interval (which might be an unbounded). In the figure, we see that S_α is the interval $[a, b]$. On the other hand, S_β is an interval $(-\infty, c]$.

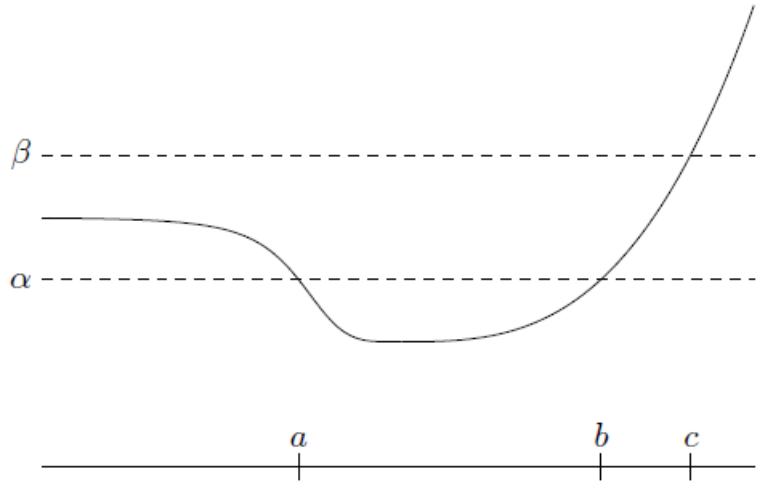


Figure 20.11: Quasiconvex funtion. We find that α-sublevel and β-sublevel sets of the function are convex *i.e.* and interval.

Examples of quasiconvex functions include:

1. $\sqrt{|x|}$ is quasi convex on R.
2. f$(x_1, x_2) = x_1 x_2$ is quasiconcave on R^2_{++}
3. distance ratio:
$$f(x) = \frac{\|x - a\|_2}{\|x - b\|_2}, \quad dom(f) = \{x \mid \|x - a\|_2 \leq \|x - b\|_2\}$$
is quasi-convex

20.5 Example Problems

1. Are the following convex sets? Prove.
   (a) $slab\,\{x \in R^n \mid \alpha \leq a^T x \leq \beta\}$

   A slab is convex, since it is the intersection of two half-spaces
   $$\{x \in R^n \mid a^T x \leq \beta\}\, and\, \{x \in R^n \mid a^T x \geq \alpha\}$$
   and each half-space is a convex set.

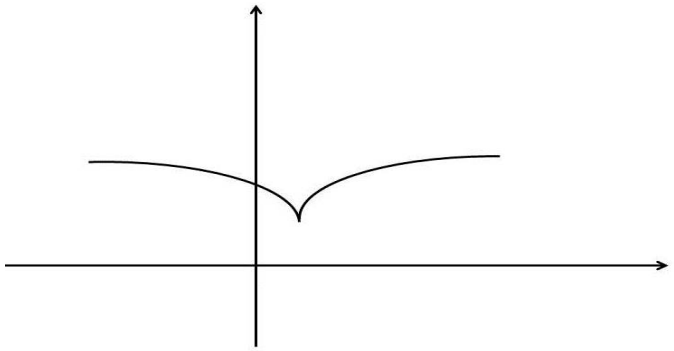


Figure 20.12: An example quasi-convex function.

(b) The set $\{x \,|x \,+\, S_2 \,\subseteq\, S_1\}$ where S_1 and $S_2 \subseteq\, R^n$ *with* S_1 *convex.*

$$x \,+\, S_2 \text{ can be expressed as } \{y \,|\, y \,=\, x \,+ z, z \,\in\, S_2\}$$

Consider a new set that contains x. For any 2 points x_1 and x_2 in this new set, by definition we have

$$x_1 \,+\, z \,\in\, S_1, \forall z \,\in\, S_2$$

$$x_2 \,+\, z \,\in\, S_1, \forall z \,\in\, S_2$$

Consider any $\lambda \,\in\, [0, 1]$

$$[\lambda x_1 \,+ (1 \,-\, \lambda)x_2] \,+\, z \,=\, \lambda(x_1 \,+\, z) \,+\, (1 \,-\, \lambda)(x_2 \,+\, z) \,\epsilon\, S_1 \;\; \forall z \,\epsilon\, S_2$$

because for any $z \,\in\, S_2$ we have $(x_1 \,+\, z), (x_2 \,+\, z)$ are in S_1 and S_1 is a convex set. Therefore the new set containing such x is also convex.

(c) $\{x|||x - x_0||_2 \,\leq\, ||x - y||_2\}$ for all $y \,\epsilon\, S, S \,\subseteq\, \mathbb{R}^n$
Solution : For any fixed value of $y \,\epsilon\, S$, we have the set given by $\{x|||x - x_0||_2 \,\leq\, ||x - y||_2\}$. It can be shown that it is a halfspace. To show that, it is sufficient to show that it can be expressed as $A^T X \,\preceq\, b$. Now x is closer to x_0 that to x_i if and only if,

$||x - x_0||_2 \,\leq\, ||x - x_i||_2$
$\Rightarrow\; (x - x_0)^T(x - x_0) \,\leq\, (x - x_i)^T(x - x_i)$
$\Rightarrow\; x^T x - 2x_0^T x + x_0^T x_0 \,\leq\, x^T x - 2x_i^T x + x_i^T x_i$
$\Rightarrow\; 2(x_i - x_0)^T x \,\leq\, x_i^T x_i - x_0^T x_0,$

which clearly defines a halfspace of the form $\{x|Ax \,\preceq\, b\}$ with

$$A = 2 \begin{bmatrix} x_1 - x_0 \\ x_2 - x_0 \\ \cdots \\ x_K - x_0 \end{bmatrix}, \text{ and } b = \begin{bmatrix} x_1^T x_1 - x_0^T x_0 \\ x_2^T x_2 - x_0^T x_0 \\ \cdots \\ x_K^T x_K - x_0^T x_0 \end{bmatrix}$$

Once we know that the set is a halfspace, we can now say that the given set is convex. The given set can be expressed as,

$$\bigcap_{y \,\epsilon\, S} \{x|||x - x_0||_2 \,\leq\, ||x - y||_2\}, \tag{20.12}$$

i.e. an intersection of halfspaces. We know that intersection of halfspaces is always convex. Hence proved that the given set of convex.

2. Check whether the following functions are convex, concave or quasi-convex

(a) $f(n) \,=\, e^x \,-\, 1 \,on\, R$

Every exponential function is convex. Therefore the given function is strictly convex and every convex function is quasi-convex too.

(b) f$(x_1, x_2) = x_1 x_2$ on R^2_{++}

The Hessian of f is

$$\nabla^2 f(x) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

which is neither positive nor negative semi-definite. Therefore f is neither convex nor concave. It is quasi-convex since its superlevel sets

$$\{(x_1, x_2) \in R^2_{++} \mid x_1 x_2 \geq \alpha\}$$

are convex.

(c) f$(x_1, x_2) = \frac{x_1}{x_2}$ on R^2_{++}

The Hessian of f is

$$\nabla^2 f(x) = \frac{1}{x_1 x_2} \begin{bmatrix} \frac{2}{x_1^2} & \frac{1}{x_1 x_2} \\ \frac{1}{x_1 x_2} & \frac{2}{x_2^2} \end{bmatrix} \geq 0$$

Therefore, f is convex and quasi-convex. It is not concave.

(d) Inverse of an increasing convex function : Suppose f: R $->$ R is increasing and convex on its domain (a,b). Let g denote its inverse i.e., the function with domain (f(a),f(b)) and g(f(x))=x for $a < b$. What can you say about convexity or concavity of g?

(e) Check if the following set is convex or not : The set of points whose distance to a does not exceed a fixed fraction θ of the distance to b i.e.,

$$\{x \mid \|x - a\|_2 \leq \theta \|x - b\|_2\}$$

where a $\neq$ b, $0 \geq \theta \geq 1$, and a,b,x ϵ R

3. **Check whether the following functions are convex, concave, or quasian**

(a) $f(x) = e^x - 1$
Solution : The Hessian of f is $\nabla^2 f(x) = e^x$. We know that $e^x > 0$ for all $x \geq 0$. So we can say, that f is strictly convex, and therefore it is also called quasiconvex.

(b) $f(x_1, x_2) = x_1 x_2$
Solution : The Hessian of f is

$$\nabla^2 f(x) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

which is neither positive semidefinite nor negative semidefinite. Therefore, f is neither convex nor concave. The superlevel sets of f, given by

$$\{(x_1, x_2) \epsilon \mathbb{R}^2_{++} \mid x_1 x_2 \geq \alpha\} \tag{20.13}$$

are convex. So the function f is quasiconvex.

(c) $f(x_1, x_2) = x_1 / x_2$
Solution : The Hessian of f is

$$\nabla^2 f(x) = \begin{bmatrix} 0 & -1/x_2^2 \\ -1/x_2^2 & 2x_1/x_2^3 \end{bmatrix},$$

which is not positive semidefinite nor negative semidefinite. Therefore f is not convex not concave.

20.6 Additional problems

1. Show that the following function $f : \mathbb{R}^n \to \mathbb{R}$ is convex.

$$f(x) = ||Ax - b||, \text{ where } A \in \mathbb{R}^{mXn}, b \in \mathbb{R}^m$$

and $\| \cdot \|$ is a norm on $\mathbb{R}^m$.

2. Which of the following sets are convex

(a) A wedge i.e $\{x \in \mathbb{R}^n \mid a_1^t x \leq \beta_1, a_2^t x \leq \beta_2\}$ where $a1, a2 \in \mathbb{R}^n$ $\beta_1 \beta_2 \in \mathbb{R}$

(b) The set of points whose distance to a doesnot exceed a fixed fraction $\theta \in [0, 1]$ of the distance to b,i.e the set $\{x \in \mathbb{R}^n \mid \|x - a\|_2 \leq \theta \|x - b\|_2\}$, where $a \neq b$ are points in $\mathbb{R}^n$.

20.7 Convex Optimization

Convex Optimization studies the problem of minimizing convex functions over convex sets. Optimization can be made easier by making use of convexity property, for example any local minimum should be global minimum in case of convex functions.

Definition. Convex minimization is the minimization of a convex function over a convex set.

The convexity property can make optimization in some sense easier than the general case - for example, any local minimum must be a global minimum.

Standard form is the usual and most intuitive form of describing a convex minimization problem.

$$Minf_c(x)s.t.$$

$$f_i(c) \leq 0 \text{ where } f_i \text{ is convex}$$

$$h_i(x) = 0 \text{ where } h_i \text{ is linear}$$

$$a_i^T x = b_i$$

Where the functions h_i are affine. In practice, the terms "linear" and "affine" are often used interchangeably. Such constraints can be expressed in the form $h_i(x) = a_i^T x + b_i$, where a_i is a column-vector and b_i a real number.

20.8 Appendix: Gradient, Hessian and Jacobian

20.8.1 Gradient

The gradient of a function $g(x)$ of n variables, at $\hat{x}$, is the vector of first partial derivatives evaluated at $\hat{x}$, and is denoted by $\bigtriangledown g(\hat{x})$:

$$\bigtriangledown g(\hat{x}) = \begin{bmatrix} \frac{\partial g(\hat{x})}{\partial x_1} \\ \frac{\partial g(\hat{x})}{\partial x_2} \\ \vdots \\ \frac{\partial g(\hat{x})}{\partial x_n} \end{bmatrix}$$

20.8.2 Hessian

The Hessian of a function $g(x)$ of n variables, at $\hat{x}$, is the matrix of second partial derivatives evaluated at $\hat{x}$, and is denoted as $\bigtriangledown^2 g(\hat{x})$g(x):

$$H(g) = \begin{bmatrix} \frac{\partial^2 g(\hat{x})}{\partial x_1^2} & \frac{\partial^2 g(\hat{x})}{\partial x_1 \, \partial x_2} & \cdots & \frac{\partial^2 g(\hat{x})}{\partial x_1 \, \partial x_n} \\ \frac{\partial^2 g(\hat{x})}{\partial x_2 \, \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 g(\hat{x})}{\partial x_2 \, \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 g(\hat{x})}{\partial x_n \, \partial x_1} & \frac{\partial^2 g(\hat{x})}{\partial x_n \, \partial x_2} & \cdots & \frac{\partial^2 g(\hat{x})}{\partial x_n^2} \end{bmatrix}.$$

This is a symmetric matrix, because $\frac{\partial^2 g(\hat{x})}{\partial x_i, \partial x_j} = \frac{\partial^2 g(\hat{x})}{\partial x_j, \partial x_i}$

20.8.3 Jacobian

Jacobain is an *nxn* matrix whose entries are the various partial derivative of the components of f. Specifically:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}.$$

J is the derivative matrix (or Jacobian matrix) evaluated at x_0.

Chapter 21

Convex Optimization

21.1 Introduction

Mathematical optimization can be very difficult to solve in general, in terms of computational complexity. However, a class of problems can be solved easily. This include problems such as least-squares problems, linear programming problems, and convex optimization problems. We had seen the least squares problems and linear programming problems in the past. This lecture focuses primarily on an overview of the convex optimization formulations and different variations thereof. Let us start with a brief comparison of convex optimization with the other well-known and, arguably easier to solve, classes of optimization, viz. least squares and linear programs, as shown in Table 21.1.

Property	Least Squares	Linear Programs	Convex Optimization
Formulation	$minimize \|\|Ax-b\|\|_2^2$	$minimize\ c^Tx$ $a_i^Tx \leq b_i,\ i=1,\ldots,m$	$minimize\ f_0(x)$ $f_i(x) \leq b_i,\ i=1,\ldots,m$
Analytical Solution	$x^* = (A^TA)^{-1}A^Tb$	No analytical formula	No analytical formula
Algorithms	Reliable and efficient	Reliable and efficient	Reliable and efficient
Computational Complexity	n^2k	n^2m	$max\{n^3, n^2m, F\}$

Table 21.1: Convex optimization versus least squares and linear programs

21.1.1 Review

- A set C is convex, if for any $x_1, x_2 \in C$, any $x_3 = \theta x_1 + (1-\theta)x_2$ is also in C, where $0 \leq \theta \leq 1$
- Example: R is convex while Z is not convex. Integer set can be convex when it is a singleton set.
- A function f is convex iff
$$f(\theta x_1 + (1-\theta)x_2) \leq \theta f(x_1) + (1-\theta)f(x_2)$$
See the figure 21.1 for a quick understanding.

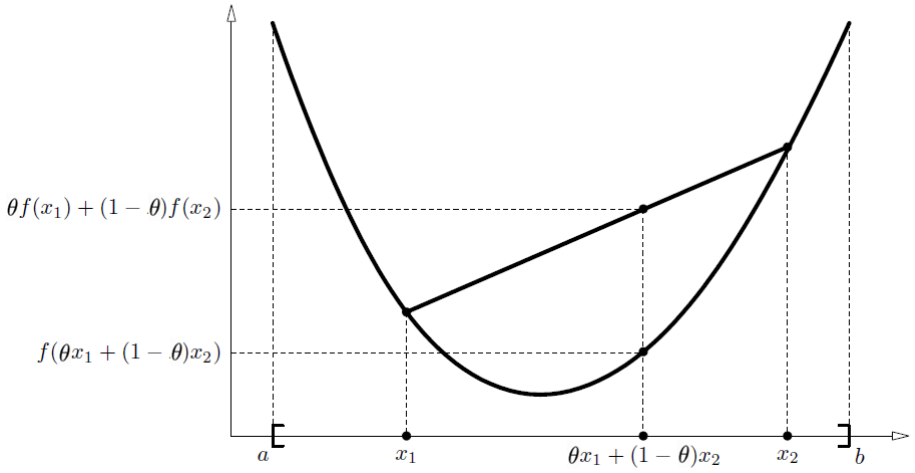


Figure 21.1: Example of a convex function. Note that the function is always below the line joining x_1 and x_2 for all the points between x_1 and x_2.

- Convex optimization is the optimization/minimization of a convex function over a convex set.
- LP is convex. Note that half spaces are convex and LP is an optimization over a set of intersctions of the half spaces. Also linear objective functions are convex functions.

In general, any convex optimization problem can be represented as:

$$\begin{aligned} &minimize\ f_0(x) \\ &subject\ to\ f_i(x) \leq b_i, i=1,\ldots,m \end{aligned} \tag{21.1}$$

where $x = (x_1, \ldots, x_n)$ are the optimization variables, $f_0 = \mathbb{R}^n \to \mathbb{R}$ is the objective function, and $f_i = \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, m$ are the constraint functions. The optimal solution is the one which has the smallest value for the objective function while satisfying all the constraints.

Optimization problems with additional constraints such as in equation 21.1 are often called constrained optimization problems. There are also problems which are unconstrained optimization problems.

21.2 Convex Optimization

21.2.1 Optimization Problem

Given a real vector-space X together with a convex, real-valued function:

$f : \chi \to \mathbb{R}$ defined on a convex subset χ of X

The problem is to find any point x^* in χ for which the number $f(x)$ is smallest, i.e. a point x^* such that $f(x^*) \leq f(x)$ for all $x \in \chi$

Convex minimization has applications in a wide range of disciplines, such as automatic control systems, estimation and signal processing, communications and networks, electronic circuit design, data analysis and modeling, statistics (optimal design), and finance. With recent improvements in computing and in optimization theory, convex minimization is nearly as straightforward as linear programming. Convex optimization has a vast plethora of applications in academic research and industrial engineering, ranging from automatic control systems to signal processing, from data modelling to electronic circuit design and many many more. Recent improvements in computing power and new theoretical breakthroughs have brought convex optimisation almost on par with linear programming in terms of ease of solution.

Convex optimization is the formulation of the general optimization problem over convex sets and convex functions. More specifically, convex optimization refers to the problem of optimising a convex function over a convex set defined by constraints, which are themselves either convex or linear. For a general convex optimization problem, any locally optimal solution is usually the global optimum as well.

Why is it important to know the convex optimization? Solving optimization problems is generally difficult except for some classes of problems that can be solved efficiently and reliably. Examples include:

1. Least square problems
2. Linear programming problems
3. Convex optimization problems.

Is it that most problems are convex? Not really. Many practical problems are non-convex. We will discuss some of these in the next lectures.

21.2.2 Formulation

We use the notation to define the convex optimization problems

$$\begin{aligned} &\textit{minimize } f_0(x) \\ &\textit{subject to } f_i(x) \leq 0, i = 1, \ldots, m \\ &\qquad\qquad h_i(x) = 0, i = 1, \ldots, p \end{aligned} \tag{21.2}$$

Here the vector $x = (x_1, ..., x_n)$ is the optimization variable of the problem, the function $f_0 : R^n -> R$ is the convex objective function, the functions $f_i : R^n \to R, i = 1, ..., m$, are the (inequality) constraint functions. Some times $f_i(x) \leq 0$ is represented as $f_i(x) \leq b_i$.

This refers to the problem of finding the value of x that minimises the function f_0 while simultaneously satisfying the conditions $f_i(x) \leq 0$ and $h_i(x) = 0$ for all $i = 1, \ldots, m$ and $i = 1, \ldots, p$ respectively. The variable, or rather vector, $x \in \mathbb{R}^n$ is the optimization variable, and the function $f_0(x) : \mathbb{R}^n \to \mathbb{R}$ is the cost function or objective function. The constraints $f_i(x) \leq 0, i = 1, \ldots, m$ are the inequality constraints, and the functions $f_i : \mathbb{R}^n \to \mathbb{R}$ are usually convex in nature. If there are no constraints, then we have an unconstrained problem.

Where the functions h_i are affine. In practice, the terms "linear" and "affine" are often used interchangeably. Such constraints can be expressed in the form $h_i(x) = a_i^T x + b_i$, where a_i is a column-vector and b_i a real number.

21.2.3 Implicit and Explicit Constraints

In convex optimization problems, the inequality constraints and the equality constraints are the explicit constraints. However, we can convert all the explicit constraints to implicit ones by redefining their domains. The extreme case here would be coverting a standard convex optimization problem to an unconstrained one $minimize\ F(x)$ where the function F is the same as f_0, but with a redefined domain restricted to the feasible set as $dom\ F = \{x \in dom\ f_0 | f_i(x) \leq 0,\ i = 1, \ldots, m,\ h_i(x) = 0,\ i = 1, \ldots, p\}$. Alternatively, if $f_i(x) \leq 0$ and $h_i(x) = 0$ are the explicit constraints, then the implicit constraints are given by $x \in dom\ f_i,\ x \in dom\ h_i$ and $D = dom\ f_0 \cap \ldots \cap dom\ f_m \cap dom\ h_1 \cap \ldots \cap dom\ h_p$ where D is the domain of the objective function.

21.2.4 Feasibility Problem

In any general convex optimization problem, if the objective function is zero, then the optimal value can be either zero if there is a feasible set, or ∞ if the feasible set is empty. Basically, with $f_0(x) = 0$, the convex optimization problem is reduced to

$$\begin{aligned} &minimize\ 0 \\ &subject\ to\ f_i(x) \leq 0, i = 1, \ldots, m \\ &\qquad\qquad h_i(x) = 0, i = 1, \ldots, p \end{aligned} \tag{21.3}$$

The feasibility problem can then be formulated as follows:

$$\begin{aligned} &minimize\ x \\ &subject\ to\ f_i(x) \leq 0, i = 1, \ldots, m \\ &\qquad\qquad h_i(x) = 0, i = 1, \ldots, p \end{aligned} \tag{21.4}$$

where the goal is to determine whether the constraints are consistent, and find a suitable solution satisfying them. As already mentioned, the optimal value will be 0 if the constraints are feasible, and any value of x satisfying the constraints will be optimal. In case the constraints are infeasible, we obtain an optimal value of ∞.

21.2.5 Optimality Criterion

For any general convex optimization problem, let us suppose that the objective function f_0 is differentiable with respect to x. In that case, for then any value of x, y satisfying the constraints,.i.e. any value of x, y from the feasible se, we have the following relation:

$$f_0(y) \geq f_0(x) + \nabla f_0(x)^T (y - x) \tag{21.5}$$

We can thus conclude that any value x in the feasible set is optimal if and only if the following condition holds

$$\nabla f_0(x)^T (y - x) \geq 0\ \forall feasible\ y \tag{21.6}$$

Geometrically, this optimality criterion tells us that for $\nabla f_0(x) \neq 0$, then it defines a supporting hyperplane to he feasible set at x as shown in Figure 21.2

21.2.6 Equivalent convex problems

Two optimization problems are considered to be equivalent if the solution of one can be obtained from the solution of the other, and vice-versa. In practice, a large number of optimization problems can be converted to a convex optimization problem, and solved. These conversions are usually done using a few common transformations that preserve convexity, viz.:

- Change of variables
- Transformation of objective function
- Transformation of constraint functions
- Eliminating equality constraints

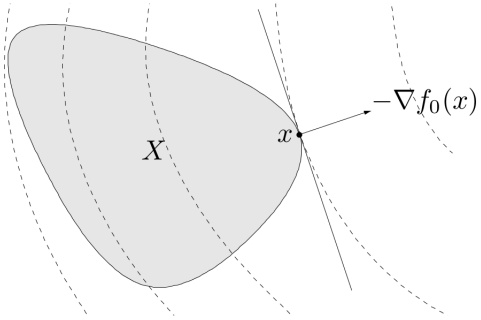


Figure 21.2: Optimality condition shown geometrically. The feasible region is the shaded convex hull X. Possible level curves of f_0 are shown as dashed lines. At the optimal point x, $-\nabla f_0(x)$ defines a supporting hyperplane.

- Introducing equality constraints
- Introducing slack variables
- Epigraph form

21.3 Quasi Convex Optimization

Note: [1]

In mathematics, a quasiconvex function is a real-valued function defined on an interval or on a convex subset of a real vector space such that the inverse image of any set of the form $(-\infty, a)$ is a convex set. Informally, along any stretch of the curve the highest point is one of the endpoints. The negative of a quasiconvex function is said to be quasiconcave.

All convex functions are also quasiconvex, but not all quasiconvex functions are convex, so quasiconvexity is a generalization of convexity. Quasiconvexity and quasiconcavity extend to functions with multiple arguments the notion of unimodality of functions with a single real argument.

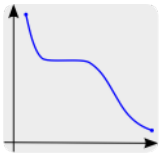


Figure 21.3: A quasilinear function is both quasiconvex and quasiconcave

Figure 21.3 shows a quasiconvex function.

Definition. A function $f : S \to \mathbb{R}$ defined on a convex subset S of a real vector space is quasiconvex if for all x, y $\in S$ and $\lambda \in [0, 1]$ we have:

$f(\lambda x + (1 - \lambda)y) \leq \max\big(f(x), f(y)\big)$.

In words, if f is such that it is always true that a point directly between two other points does not give a higher a value of the function than do both of the other points, then f is quasiconvex. Note that the points x and y, and the point directly between them, can be points on a line or more generally points in n-dimensional space.

[1] Note discussed for S2016

A quasilinear function is both quasiconvex and quasiconcave as shown in Figure 21.3.

The graph of a function that is both concave and quasi-convex on the nonnegative real numbers. An alternative way of defining a quasi-convex function f(x) is to require that each sub-levelset $S_\alpha(f) = \{x|f(x) \leq \alpha\}$ is a convex set.

If furthermore $f(\lambda x + (1-\lambda)y) < \max\big(f(x), f(y)\big)$ for all $f(x) \neq f(y)$ and $\lambda \in (0,1)$, then f is strictly quasiconvex.

That is, strict quasiconvexity requires that a point directly between two other points must give a lower value of the function than one of the other points does.

A quasiconcave function is a function whose negative is quasiconvex, and a strictly quasiconcave function is a function whose negative is strictly quasiconvex.

Equivalently a function f is quasiconcave if $f(\lambda x + (1-\lambda)y) \geq \min\big(f(x), f(y)\big)$

and strictly quasiconcave if $f(\lambda x + (1-\lambda)y) > \min\big(f(x), f(y)\big)$

A (strictly) quasiconvex function has (strictly) convex lower contour sets, while a (strictly) quasiconcave function has (strictly) convex upper contour sets. A function that is both quasiconvex and quasiconcave is quasilinear. A particular case of quasi-concavity is unimodality, in which there is a locally maximal value.

21.3.1 Quasiconvex Standard Form

Let q_t be a family of convex functions that satisfy $f_0(x) \leq t \Leftrightarrow \phi_t(x) \leq 0$

Find x such that $\phi_t(x) \leq 0$
$f_t(x) \leq 0$
$Ax = b$

If this is feasible $p^* < t$ Else if this is infeasible $p^* > t$

21.3.2 Optimization:

A quasiconvex optimization problem is an optimization problem where we seek to minimize(or, alternatively, maximise) a quasiconvex function over a convex set defined by constraints that are either convex or linear. Mathematically, a quasiconvex optimization problem is defined as:

$$\begin{aligned} &minimize\ f_0(x) \\ &subject\ to\ f_i(x) \leq 0, i = 1, \ldots, m \\ &\qquad\qquad Ax = b \end{aligned} \tag{21.7}$$

where the function $f_0 : \mathbb{R}^n \to \mathbb{R}$ is quasiconvex instead of being convex, while the functions $f_i : \mathbb{R}^n \to \mathbb{R}$ are convex, and the functions $h_i : \mathbb{R}^n \to \mathbb{R}$ are linear. It is to be noted that, in the nature of quasiconvex functions, this sort of problem can have locally optimal points that are not globally.

A standard approach to solving quasiconvex optimization problems is representing sublevel sets of a quasiconvex function with a family of convex inequalities. If f_0 be the quasiconvex function, then we have a family of functions ϕ_t such that t-sublevel set of f_0 is 0-sublevel set of ϕ_t, i.e.

$$f_0(x) \leq t \iff \phi_t(x) \leq 0 \tag{21.8}$$

We can then solve the quasiconex otimization problem by solving the associated feasibility problem, viz.

$$\begin{aligned} &\text{find} \quad x \\ &\text{subject} \quad \text{to} \quad \phi_t(x) \leq 0 \\ &\qquad\quad \text{and} \quad f_i(x) \leq 0,\ i = 1, \ldots, m \\ &\qquad\quad \text{and} \quad Ax = b \end{aligned} \tag{21.9}$$

which is a convex feasibility problem in x for fixed t. If this is feasible, then we have $t \geq p^*$, otherwise we will have $t \leq p^*$. We can sole this convex feasibility problem using a variant of the bisection method, as shown below.

minimize $f_0 x$
subject to $f_i x \leqslant 0$ i=1,2,...,m
Ax=b

with $f_0 : R^n \Rightarrow$ R quasiconvex, $f_1 \ldots f_m$ convex
can have locally optimal points that are not (globally) optimal

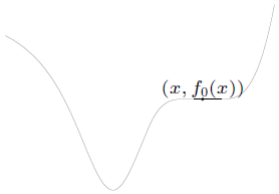


Figure 21.4: QuasiConvex Function

Representation of sublevel sets of f_0

if f_0 is quasiconvex, there exists a family of functions Φ_t such that:
1.Φ_t is convex in x for fixed t
2.t-sublevel set of f_0 is 0-sublevel set of Φ_t , i.e.,

Example

$f_0 x = \frac{p(x)}{q(x)}$
with p convex, q concave, and p(x) $\geq$ 0 , q(x) $>$ 0 on dom f_0
can take Φ_t(x)=p(x)-tq(x):
1. for t $\geq$ 0,Φ_t convex in x
2. p(x)/q(x) $\leq$ t if and only if Φ_t(x) $\leq$ 0

Quasiconvex optimization via convex feasibility problems

$\Phi_t(\mathrm{x}) \leq 0,\ f_i(x) \leq 0, i = 1, 2, ..m, Ax = b \qquad (1)$
1.for fixed t, a convex feasibility problem in x
2.if feasible, we can conclude that t $\geq p^\star$; if infeasible, t $\leq p^\star$

Bisection method for quasiconvex optimization

given l $\leq p^\star$, u $\geq p^\star$, tolerance $\epsilon > 0$
repeat
1. t := (l + u)/2
2. Solve the convex feasibility problem (1)
3. if (1) is feasible, u:= t ; else l:= t
until u - l $\leq \epsilon$
requires exactly $ln((u-l)/\epsilon)$ iterations (where u,l are initial values)

algorithm **Bisection method for quasiconvex feasibility**
Require: $l \leq p^*,\ u \geq p^*,\ tolerance\ \epsilon > 0$
  repeat
    $t \leftarrow (l+u)/2$
    Solve the convex feasibility problem in x and t
    if $nfeasible$ **then**
      $u \leftarrow t$
    else

$l \leftarrow t$
end if
until $u - l \leq \epsilon$

21.4 Variations and Other Formulations

Linear optimization, or rather linear programming problems are simply convex optimization problems where the objective and constraint functions are all affine. A general linear problem has the standard form:

$$\begin{aligned} &minimize\ c^T x + d \\ &subject\ to\ Gx \leq h \\ &\qquad Ax = b \end{aligned} \tag{21.10}$$

where $G \in \mathbb{R}^{m \times n}$ and $A \in \mathbb{R}^{p \times n}$.
The geometric interpretation of a linear program is shown in Figure 21.5. The feasible region in a linear program is, as already covered in previous chapters, a polyhedron formed by the interection of the constraints.

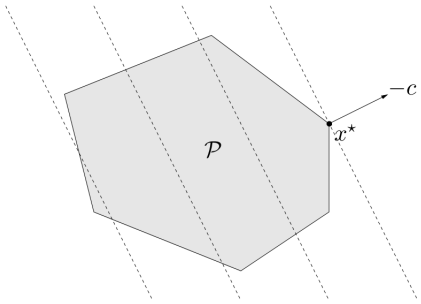


Figure 21.5: A linear program shown geometrically. The polyhedron P is the feasible region. The level curves of the linear objective function $c^T x$ are orthogonal to c, and the point x^* is optimal.

21.4.1 Linear Fractional Programming

A linear-fractional programming problem is another type of optimization problem where the objective is to minimize a ratio of affine functions over a polyhedron formed by the intersection of constraint. The standard form of a linear-fractional program is :

$$\begin{aligned} &minimize\ f_0(x) = \frac{c^T x + d}{e^T x + f} \\ &subject\ to\ Gx \leq h \\ &\qquad Ax = b \end{aligned} \tag{21.11}$$

where $dom f_0 = \{x | e^T x + f > 0\}$. Linear-fractional programs are usually quasiconvex in nature, due to the objective function being quasiconvex.

21.4.2 Quadratic Optimisation

A convex optimization problem with an objective function that is quadratic and constraints that are affine in nature is known as a quadratic optimization problem. In general, a quadratic optimization problem can be expressed as:

$$\begin{aligned} &minimize\ \frac{1}{2} x^T P x + q^T x + r \\ &subject\ to\ Gx \leq h \\ &\qquad Ax = b \end{aligned} \tag{21.12}$$

where $P \in S_+^n$, $G \in \mathbb{R}^{m \times n}$, and $A \in \mathbb{R}^{p \times n}$. Here, S is the vector space of matrices of appropriate dimensionality. The feasible region is usually a polyhedron in case of quadratic optimization, as shown in Figure 21.6.

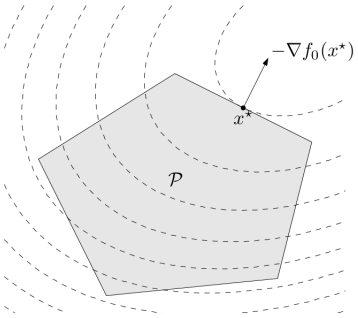


Figure 21.6: A quadratic optimization shown geometrically. The polyhedron P is the feasible region. The level curves of the quadratic objective function are shown as dashed curves. x^* is the optimal point.

Example

Least Squares
minimize $\| Ax - b \|_2^2$

Related Topics

1.Quadratically constrained quadratic program (QCQP)
2.Second-order cone programming

21.4.3 Quadratically Constrained Quadratic Programs

This is a variant of quadratic optimization with constraints that are quadratic in nature as opposed to convex or affine constraints as in plain vanilla quadratic optimization.

$$\begin{aligned} &minimize\ \frac{1}{2}x^T P_0 x + q_0^T x + r_0 \\ &subject\ to\ \frac{1}{2}x^T P_i x + q_i^T x + r_i < 0,\ i = 1, \ldots, m \\ &\qquad Ax = b \end{aligned} \tag{21.13}$$

Here $P_i \in S_+^n,\ i = 0, \ldots, m$, and the feasible region is an intersection of ellipsoids $P_i > 0$. It is to be noted that linear programs are a special case of quadratic programs, where $P = 0$. Further, quadratic programs, and by extension, linear programs, are special cases of quadratically constrained quadratic programs, where in addition to $P = 0$, we also have $P_i = 0,\ i = 1, \ldots, m$.

21.4.4 Second Order Cone Programming

Second order cone programming is a form of convex optimization where the inequalities are second order cone constraints. It closely related to other forms of convex optimization. A second order cone program can be reduced to a QCQP when $c_i = 0,\ i = 1, \ldots, m$ by squaring each of the constraints. Furthermore, if $A_i = 0,\ 1, \ldots, m$, then it is further reduced to a linear program.

21.4.5 SDP

Semidefinite programming is the subfield of convex optimization that deals with the optimization of a linear objective function over the intersection of the cone of positive semidefinite matrices with an affine space. The inequality constraints are called linear matrix inequalities. Like SOCP, it is also closely associated with other forms of convex optimiation. Linear programs and second order cone programs can be converted to semidefinite programs, which are more general in nature.

21.4.6 Hard Variations

Slight modifications of the standard convex optimization problem can yield problems that are quite hard to solve computationally. A few examples of the same include the following:

- Convex Maximization/Concave Minimization

$$maximize\ ||x||\ subject\ to\ Ax \leq b \tag{21.14}$$

- Nonlinear Equality Constraints

$$minimize\ c^T x\ subject\ to\ x^T P_i x + q_i^T x + r_i = 0,\ i = 1, \ldots, k \tag{21.15}$$

- Non-convex Sets, e.g. Boolean Sets

$$find\ x\ subject\ to\ Ax \leq b,\ x_i \in \{0, 1\} \tag{21.16}$$

21.5 Exercise

Exercise. Optimize $f(x, y) = 5x - 3y$ such that $x^2 + y^2 = 136$

$F = 5x - 3y - \lambda x^2 - \lambda y^2 + 136\lambda$

F_x is obtained from differentiating F with respect to x
$F_x = 5 - 2\lambda x = 0$ Therefore, $x = \frac{5}{2\lambda}$

F_y is obtained from differentiating F with respect to y
$F_y = -3 - 2\lambda y = 0$ Therefore, $y = \frac{-3}{2\lambda}$

F_λ is obtained from differentiating F with respect to λ
$F_\lambda = -x^2 - y^2 + 136 = 0$

$F_\lambda = \frac{25}{4\lambda^2} + \frac{9}{4\lambda^2} + 136$

$F_\lambda = \frac{34}{4\lambda^2} = 136$

So therefore λ can be the following two values:

$\lambda = \frac{1}{4}$ then $x = 10$ and $y = -6$ so $f(10, -6) = 68$ (Maximization Problem)

$\lambda = \frac{-1}{4}$ then $x = -10$ and $y = 6$ so $f(-10, 6) = -68$ (Minimization Problem)

Exercise. Consider the optimization problem:

$$\begin{aligned} &minimize\ f_0(x_1, x_2) \\ &subject\ to\ 2x_1 + x_2 \geq 1 \\ &\qquad x_1 + 3x_2 \geq 1 \\ &\qquad x_1, x_2 \geq 0 \end{aligned}$$

Make a sketch of the feasible region/set. For each objective function below, give the optimal set and value.

- $f_0(x_1, x_2) = x_1 + x_2$
- $f_0(x_1, x_2) = max\{x_1, x_2\}$
- $f_0(x_1, x_2) = x_1^2 + 9x_2^2$

The feasible set is given by the intersection of the halfspaces defined by the constraints. More specifically, we have the following point of intersection between the two inequality constraints:

$$\begin{aligned} 2x_1 + x_2 &= 1 \\ 2 * (x_1 + 3x_2) &= 2 \\ \hline 5x_2 &= 1 \\ (x_1,\ x_2) &= (2/5,\ 1/5) \end{aligned}$$

Between each of the inequality constraints and the non-negativity constraints, we have the following points of intersection:

$$(0,\ \infty),\ (0,\ 1),\ (1,\ 0),\ (\infty,\ 0)$$

So, the feasible set or region is given by the convex hull of the polygon with vertices as $(0,\ \infty)$, $(0,\ 1)$, $(2/5,\ 1/5)$, $(1,\ 0)$, $(\infty,\ 0)$, after taking into account the apropriate directions according to the inequalities. This is shown in Figure 21.7.

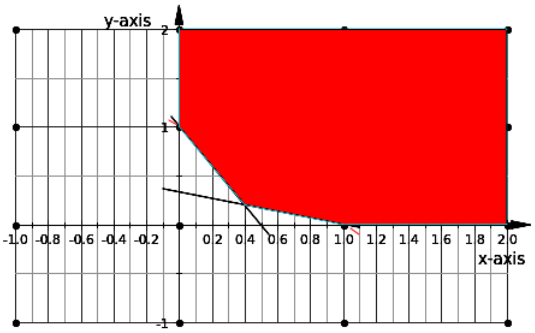


Figure 21.7: Feasible region.

For $f_0(x) = x_1 + x_2$, the optimal set is given by $x^* = (2/5,\ 1/5)$, and the optimal value is $3/5$.

For $f_0(x) = max\{x_1,\ x_2\}$, the optimal set is given by $x^* = (1/3,\ 1/3)$, and the optimal value is $1/3$.

For $f_0(x) = x_1^2 + 9x_2^2$, the optimal set is given by $x^* = (1/2,\ 1/6)$, and the optimal value is $1/2$.

Exercise. Prove that $x^* = [1, 1/2, -1]^T$ is optimal for the problem:

$$\begin{aligned}
&minimize\ \frac{1}{2}x^T Px + q^T x + r \\
&subject\ to\ -1 \leq x_i \leq 1\ \forall i = 1, 2, 3 \\
&P = \begin{bmatrix} 13 & 12 & -2 \\ 12 & 17 & 6 \\ -2 & 6 & 12 \end{bmatrix} q = \begin{bmatrix} -22.0 \\ -14.5 \\ 13.0 \end{bmatrix} r = 1
\end{aligned}$$

In order to minimize $\frac{1}{2}x^T Px + q^T x + r$ the first thing we need to do is find its gradient. Hence we diferentiate the objective function with respect to x. This gives us:

$$\nabla f_0(x) = x^T P + q^T$$

In order for the given x^* to be the optimal solution, it needs to satisfy the optimality condition specified earlier, which is given again as follows:

$$\nabla f_0(x)^T (y - x) \geq 0\ \forall feasible\ y$$

Now, for $x^* = [1, 1/2, -1]^T$, the gradient of the objective function attains a value of:

$$\begin{aligned}
\nabla f_0(x^*) &= \{x^*\}^T P + q^T \\
&= [-1, 0, 2]
\end{aligned}$$

which is obtained by plugging in the values of P and q respectively. Therefore the optimality condition, for any $y = [y_1, y_2, y_3]^T$ is reduced to:

$$\begin{aligned}
\nabla f_0(x^*)^T (y - x^*) &= -1(y_1 - 1) + 0(y_2 - 1/2) + 2(y_2 + 1) \\
&= -y_1 + 1 + 2y_2 + 2 \\
&= 3 + 2y_2 - y_1 \\
&\geq 0,\ \forall y\ such\ that\ -1 \leq y_i \leq 1
\end{aligned}$$

Since the optimality condition is satisfied along with the feasibility constraints of $-1 \leq y_i \leq 1$, we can conclude that $x^* = [1, 1/2, -1]^T$ is indeed an optimal solution for the quadratic optimization probem of minimizing $\frac{1}{2}x^T Px + q^T x + r$

Exercise. Consider the following optimization problem:

$$\begin{aligned} &\textit{minimize } x_1 + x_2 \\ &\textit{subject to } -x_1 \leq 0 \\ &\qquad \textit{and } -x_2 \leq 0 \\ &\qquad \textit{and } 1 - x_1 x_2 \leq 0 \end{aligned}$$

Prove that the feasible set is a half-hyperboloid, with optimal value 2 at optimal point $x^* = (1, 1)$.

Chapter 22

Optimization Problem: Support Vector Machines

22.1 Introduction

max Margin and SVM problem

22.2 Primal problem

22.3 Dual problem

22.4 Problems

Chapter 23

SDP and MaxCut

23.1 Introduction

Chapter 24

Nonlinear Optimization: Solving $f(x) = 0$

24.1 Introduction

In this lecture we discuss various methods to solve m nonlinear equations in n variables.

$$\begin{aligned} f_1(x_1, x_2, ..., x_n) &= 0 \\ f_2(x_1, x_2, ..., x_n) &= 0 \\ f_3(x_1, x_2, ..., x_n) &= 0 \\ &. \\ &. \\ &. \\ f_m(x_1, x_2, ..., x_n) &= 0 \end{aligned}$$

We can also write this system as $f(x) = 0$ where

$$f(x) = \begin{bmatrix} f_1(x_1, x_2, ..., x_n) \\ f_2(x_1, x_2, ..., x_n) \\ f_3(x_1, x_2, ..., x_n) \\ . \\ . \\ . \\ f_m(x_1, x_2, ..., x_n) \end{bmatrix} \text{ and , } x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ x_m \end{bmatrix}$$

Often we make two Assumptions:

1. f is continuous, (i.e. $\lim_{y \to x} f(y) = f(x)$ for all x) and sometimes differentiable.
2. f is available in explicit form (Eg. $x_1^2 + x_2^2 + x_3^2 = 10$) or as black-box (input: x, output : $f(x)$).

We are interested in two problems (in the unconstrained setting):

1. Solve $f(x) = 0$. (You may recollect that we solved $Ax - \mathbf{b} = 0$ in the first part of the course). Indeed this problem is more complex than the linear systems that we had discussed. Also the optimality criteria (i.e., derivative is zero) results in this problem for many optimization tasks.
2. Minimize $||f(x)||$. (You may also recollect that we solved the linear least squares problem in the past i.e., minimize $||Ax - \mathbf{b}||$.).

24.1.1 Iterative Algorithms

Nonlinear equations usually have to be solved by iterative algorithms, that generate a sequence of points $x^{(0)}, x^{(1)}, x^{(2)}, \ldots$ with $f(x^{(k)}) \to 0$, as $k \to \infty$. The vector $x^{(0)}$ is called the *starting point* of the algorithm, and $x^{(k)}$ is called the *kth iterate*. Moving from $x^{(k)}$ to $x^{(k+1)}$ is called an *iteration* of the algorithm. The algorithm is terminated when $||f(x^{(k)})|| \leqslant \epsilon$, where $\epsilon > 0$ is some specified tolerance, or when it is determined that the sequence is not converging.

The general structure of this class of algorithim is:

1. Choose a starting point x_0.
2. Compute the next iterate x_{k+1} using x_k and information about the function at x_k.
3. We find x_{k+1} untill a solution point with sufficent accuracy is found or no further progress can be made.

Some questions that we are interesting in answering in this regard are as follows

- How many iterations are required?
- How fast does it converge?
- Is it converging to a local or a global maximum?

24.2 Bisection Method

This method is applicable when we wish to solve the equation $f(x) = 0$ for the real variable x, where f is a continuous function defined on an interval $[a, b]$ and $f(a)$ and $f(b)$ have opposite signs. In this case a and b are said to bracket a root since, by the intermediate value theorem, the f must have at least one root in the interval (a, b).

At each step the method divides the interval in two by computing the midpoint $c = (a + b)/2$ of the interval and the value of the function $f(c)$ at that point. Unless c is itself a root (which is very unlikely, but possible) there are now two possibilities: either $f(a)$ and $f(c)$ have opposite signs and bracket a root, or $f(c)$ and $f(b)$ have opposite signs and bracket a root. The method selects the subinterval that is a bracket as a new interval to be used in the next step. In this way the interval that contains a zero of f is reduced in width by 50% at each step. The process is continued until the interval is sufficiently small.

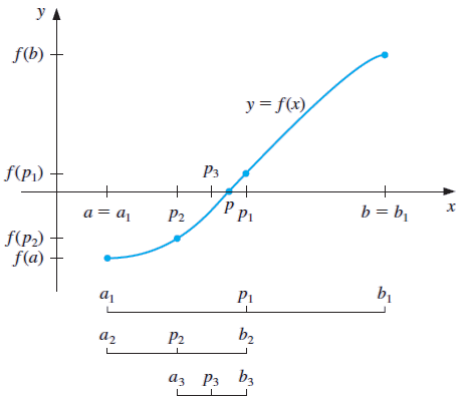


Figure 24.1: Bisection Method.

Figure 24.1 shows how the bisection method converges.
Source : https://www.math.ust.hk/ mamu/courses/231/Slides/ch02_1.pdf

The Bisection Method is basically a numerical method for estimating the roots of a polynomial $f(x)$. Consider an equation $f(x) = 0$ which has a zero in the interval $[a, b]$ and $f(a) * f(b) < 0$. This method computes the zero, say p, by repeatedly having the interval $[a, b]$. Starting with $p = (a + b)/2$. This step is like computing x_0. Now, the next step is to compute the next iterate. The interval $[a, b]$ is replaced by $[p, b]$ if $f(p) * f(b) < 0$ or with $[a, p]$ if $f(a) * f(p) < 0$. This process is continued untill the zero is obtained i.e, $f(x_{k+1}) \to 0$ or $|a - b| < \epsilon$.

24.2.1 Algorithim

Consider an example in Figure 24.2. We start with a_1, b_1 such that $f(a_1).f(b_1) < 0$

The number of iterations taken by this procedure to converge is $log_2 \frac{|a_1 - b_1|}{\epsilon}$

Given: A function f(x) continuous on an interval [a,b] and $f(a) * f(b) < 0$

while ($|a - b| > \epsilon$)
{
    p = (a+b)/2
    if($f(a) * f(p) < 0$)
        b = p
    else
        a = p
}

Example 24. Question: Find the root of $\boldsymbol{f(x) = x^2 - 3}$, given $\boldsymbol{\epsilon = 0.01}$.
Answer: We start with the interval **[1,2]**. Iterations are shown in Table 24.1.

Thus, with the seventh iteration, we note that the final interval, [1.7266, 1.7344], has a width less than 0.01 and $|f(1.7344)| < 0.01$, and therefore we chose b = 1.7344 to be our approximation of the root.

Example 25. Suppose we apply bisection method to find a root of the polynomial

$$f(x) = x^3 - x - 2.$$

For $a_1 = 1$ and $b_1 = 2$ we have $f(a_1) = -2$ and $f(b_1) = +4$. Since the function is continuous, the root must lie within the interval $[1, 2]$.

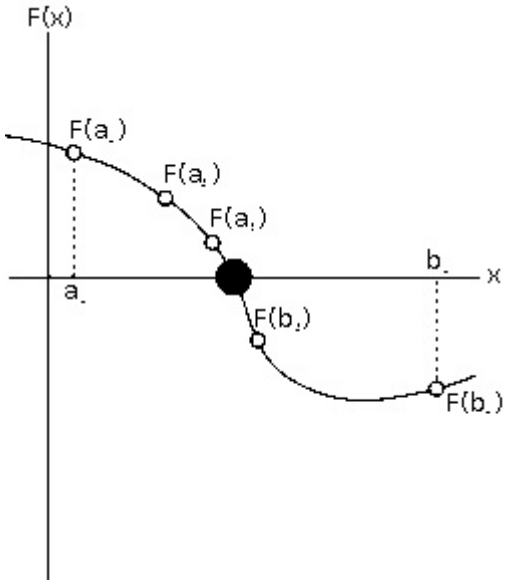


Figure 24.2: An Example

a	b	f(a)	f(b)	p=(a+b)/2	f(p)	Update	new (b-a)
1.0	2.0	-2.0	1.0	1.5	-0.75	a = p	0.5
1.5	2.0	-0.75	1.0	1.75	0.062	b = p	0.25
1.5	1.75	-0.75	0.0625	1.625	-0.359	a = p	0.125
1.625	1.75	-0.3594	0.0625	1.6875	-0.1523	a = p	0.0625
1.6875	1.75	-0.1523	0.0625	1.7188	-0.0457	a = p	0.0313
1.7188	1.75	-0.0457	0.0625	1.7344	0.0081	b = p	0.0156
1.71988	1.7344	-0.0457	0.0081	1.7266	-0.0189	a = p	0.0078

Table 24.1: Example for Bisection

Midpoint is

$$c_1 = \frac{2+1}{2} = 1.5.$$

Function value at the midpoint is $f(c_1) = -0.125$. Because it is negative, $a_2 = c_1 = 1.5$ and $b_2 = b_1 = 2$, to ensure they have opposite signs at the next iteration.

Table 24.2 shows how the method converges gradually to the solution.

After 13 iterations, it becomes apparent that there is a convergence to about 1.521: a root for the polynomial.

24.2.2 Convergence Analysis

Bisection method is based on the Intermediate Value Theorem (IVT). It is guaranteed to converge to a root of f if f is a continuous function on the interval $[a, b]$, and $f(a)$ and $f(b)$ have opposite signs. The absolute error is halved at each step so the method converges linearly, which is comparatively slow.

Specifically, if $c_1 = (a + b)/2$ is the midpoint of the initial interval, and c_n is the midpoint of the interval in the nth step, then the difference between c_n and a solution c is bounded by

$$|c_n - c| \leqslant \frac{|b - a|}{2^n}$$

This formula can be used to determine in advance the number of iterations that the bisection method would need to converge to a root to within a certain tolerance. The number of iterations needed, n, to achieve a given error (or tolerance), ϵ is given by

$$n = \log_2\left(\frac{\epsilon_0}{\epsilon}\right) = \frac{\log \epsilon_0 - \log \epsilon}{\log 2}$$

Iteration	a_n	b_n	c_n	$f(c_n)$
1	1	2	1.5	-0.125
2	1.5	2	1.75	1.6093750
3	1.5	1.75	1.625	0.6660156
4	1.5	1.625	1.5625	0.2521973
5	1.5	1.5625	1.5312500	0.0591125
6	1.5	1.5312500	1.5156250	-0.0340538
7	1.5156250	1.5312500	1.5234375	0.0122504
8	1.5156250	1.5234375	1.5195313	-0.0109712
9	1.5195313	1.5234375	1.5214844	0.0006222
10	1.5195313	1.5214844	1.5205078	-0.0051789
11	1.5205078	1.5214844	1.5209961	-0.0022794
12	1.5209961	1.5214844	1.5212402	-0.0008289
13	1.5212402	1.5214844	1.5213623	-0.0001034
14	1.5213623	1.5214844	1.5214233	0.0002594
15	1.5213623	1.5214233	1.5213928	0.0000780

Table 24.2: Bisection Method Iterations

where, $\epsilon_0 = b - a$ (inital bracket size). Therefore the linear convergence is expressed by $\epsilon_{n+1} = constant \times \epsilon_n^m, m = 1$.

24.2.3 Discussion

1. The method is guaranteed to converge.
2. The error bound decreases by half with each iteration.
3. The bisection method is robust and simple, but converges very slowly.
4. It is often used to obtain a rough approximation to a solution which is then used as a starting point for more rapidly converging methods.
5. The bisection method cannot detect multiple roots.

Advantages

- The procedure is simple
- We don't need the explicit form of the function, i.e. a blackbox representation is sufficient
- Guaranteed to converge

Disadvantages

- Two variable initialization is required
- Very slow

24.3 Fixed point iteration

A point where $x = g(x)$ is called the fixed point of the function g(x). We can make use of this concept to solve for problems of the sort $f(x) = 0$ as follows

To solve $f(x) = 0$ we create a function $g(x)$ such that the solution to $f(x) = 0$ is a fixed point of $g(x)$ algorithm
Iterative Fixed Point Algorithm
$x_0 \leftarrow random()$
While$|g(x_k) - x_k| > \epsilon$
$x_{k+1} \leftarrow g(x_k)$

Fixed-point iteration is a method of computing fixed points of iterated functions. A fixed-point (also known as invariant point) of a function is an element of the function's domain that is mapped to itself by the function. For example, if g is defined on the real numbers by $g(x) = x^2 - 3x + 4$, then 2 is a fixed point, since $g(2) = 2$.

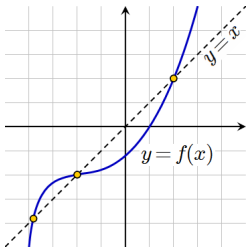


Figure 24.3: A function with three fixed points.

Figure 24.3 shows a function with three fixed points.
Source : http://en.wikipedia.org/wiki/Fixed_point_(mathematics)

More specifically, given a function g defined on the real numbers with real values and given a point x_0 in the domain of g, the fixed-point iteration is

$$x_{n+1} = g(x_n), n = 0, 1, 2, \ldots$$

which gives rise to a sequence $x_0, x_1, x_2, \ldots$ which is hoped to converge to a point x. If g is continuous, then one can prove that the obtained x is a fixed point of g, i.e. $g(x) = (x)$.

Given a root-finding problem $f(p) = 0$, there are many g with fixed-points at p. If g has fixed-points at p, then $f(x) = x - g(x)$ has zero at p.

Fixed-point iteration is a method of computing fixed points of iterated functions. More specifically, given a function f(x) defined on the real numbers with real values and given a point x_0 in the domain of f, the fixed point iteration is $x_{n+1} = f(x_n)$, n=0, 1, 2, ... which gives rise to the sequence $x_0, x_1, x_2, \ldots$ which is hoped to converge to a point x. If f is continuous, then one can prove that the obtained x is a fixed point of f(x), i.e., f(x)=x. More generally, the function f can be defined on any metric space with values in that same space.

24.3.1 Algorithim

Given: An equation f(x) = 0
Convert f(x) = 0 into the form x = g(x)
Let the initial guess be x0
Do

x_{i+1}= g(x_i)

while(none of the convergence criterion C1 or C2 is met)

1. C1. Fixing apriori the total number of iterations N .
2. C2. By testing the condition $|x_i + 1 - g(x_i)|$ (where i is the iteration number) less than some tolerance limit, say ϵ, fixed apriori.

Example 26. Question: Find a root of $\boldsymbol{x^4 - x - 10 = 0}$
Answer: Consider $\boldsymbol{g1(x) = 10/(x^3 - 1)}$ and the fixed point iterative scheme
$\boldsymbol{x_{i+1} = 10/(x_i^3 - 1)}$, i = 0, 1, 2, ...
Let the initial guess $\boldsymbol{x_0}$ be **2.0**

i	0	1	2	3	4	5	6	7	8
x_i	2	1.429	5.214	0.071	-10.004	-9.978E-3	-10	-9.99E-3	-10

So the iterative process with $\boldsymbol{g1}$ gone into an infinite loop without converging.
Consider another function $\boldsymbol{g2(x) = (x + 10)^{1/4}}$ and the fixed point iterative scheme $\boldsymbol{x_{i+1} = (x_i + 10)^{1/4}}$, i = 0, 1, 2, . . .
let the initial guess $\boldsymbol{x_0}$ be **1.0, 2.0 and 4.0**

i	0	1	2	3	4	5	6
x_i	1.0	1.82116	1.85424	1.85553	1.85558	**1.85558**	
x_i	2.0	1.861	1.8558	1.85559	1.85558	**1.85558**	
x_i	4.0	1.93434	1.85866	1.8557	1.85559	1.85558	**1.85558**

That is for **g2** the iterative process is converging to **1.85558** with any initial guess.
Consider $\boldsymbol{g3(x) = (x+10)^{1/2}/x}$ and the fixed point iterative scheme

$\boldsymbol{x_{i+1} = (x_i + 10)^{1/2}/x_i}$, i = 0, 1, 2, …let the initial guess $\boldsymbol{x_0}$ be **1.8**,

i	0	1	2	3	4	5	6	...	98
x_i	1.8	1.9084	1.80825	1.90035	1.81529	1.89355	1.82129	...	**1.8555**

That is for **g3** with any initial guess the iterative process is converging but very slowly to
Geometric interpretation of convergence with **g1, g2** and **g3**.

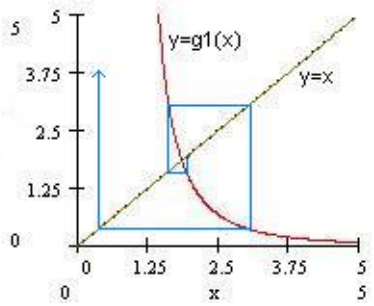


Figure 24.4: Using **g1** the iterative process does not converge for any initial approximation.

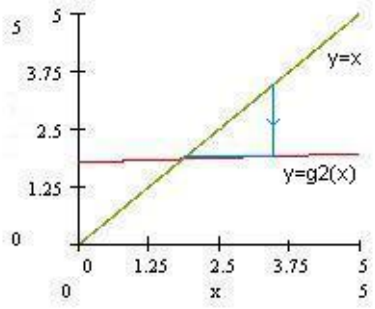


Figure 24.5: Using **g2**, the iterative process converges very quickly to the root which is the intersection point of $\mathbf{y = x}$ **and** $\mathbf{y = g2(x)}$ as shown in the figure.

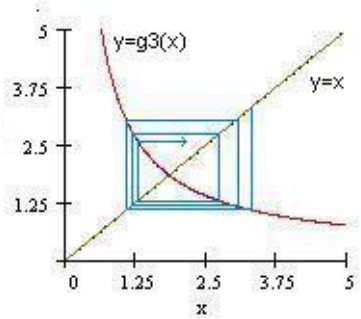


Figure 24.6: Using **g3**, the iterative process converges but very slowly.

Example 27. *Consider* $f(x) = x^2 - x - 2$ Suppose that we want to solve $f(x) = 0$
To solve this by Iterative Fixed Point Algorithm we can think of 4 possible candidates for g(x) which can be derived as follows

1. $x^2 - x - 2 = 0 \implies x = x^2 - 2 \implies g(x) = x^2 - 2$ (for $x = g(x)$)

2. $x^2 - x - 2 = 0 \implies x^2 = x + 2 \implies x = \sqrt{x+2} \implies g(x) = \sqrt{x+2}$
3. $x^2 - x - 2 = 0 \implies x(x-1) = 2 \implies x - 1 = \frac{2}{x} \implies x = 1 + \frac{2}{x} \implies g(x) = 1 + \frac{2}{x}$
4. $x^2 - x - 2 = 0 \implies 2x^2 - x = x^2 + 2 \implies x(2x-1) = x^2 + 2 \implies x = \frac{x^2+2}{2x-1} \implies g(x) = \frac{x^2+2}{2x-1}$

Suppose we choose the candidate $g(x) = 1 + \frac{2}{x}$. Then the steps in the solution are as follows

$g(x) = 1 + \frac{2}{x}$
Let $x^0 = 1$
$x^1 = 1 + \frac{2}{1} = 3$
$x^2 = 1 + \frac{2}{3} = \frac{5}{3}$
$x^3 = 1 + \frac{2}{\frac{5}{3}} = \frac{11}{5}$
$x^4 = 1 + \frac{2}{\frac{11}{5}} = \frac{21}{11}$

Now suppose we choose the candidate $g(x) = x^2 - 2$. Then the steps in the solution are as follows

$g(x) = x^2 - 2$
Let $x^0 = 2.5$
$x^1 = 2.5^2 - 2 = 4.25$
$x^2 = 4.25^2 - 2 = 16.0625$
$x^3 = 16.0625^2 - 2 = 256.004$

Clearly, the series is not converging

Note

- Not all $g(x)$ are good as they may not converge to a solution
- We have to analyze and see what makes a particular $g(x)$ good or bad for this purpose
- If $|g'(x*)| < 1$ then the fixed point iteration is locally convergent
- E.g.
  For $g(x) = 1 + \frac{2}{x}$, we have $g'(x) = -\frac{2}{x^2}$, so for $x = 2, g'(x) = -\frac{2}{4} = -\frac{1}{2} < 1$
  For $g(x) = x^2 - 2$, we have $g'(x) = 2x$, so for $x = 2, g'(x) = 2 \times 2 = 4 > 1$

Example 28. Find $\sqrt{a}$ with the help of fixed point iteration
Solve $x^2 - a = 0$
We need to put this in the form $x = g(x)$ for using fixed point iteration
$x^2 - a = 0 \implies 2x^2 = a + x^2 \implies x = \frac{1}{2}(\frac{a}{x} + x) \implies g(x) = \frac{1}{2}(\frac{a}{x} + x)$

Using the above formulation, let us obtain $\sqrt{5}$, i.e. $a = 5$. With a starting value of $x_0 = 1$ we get:

Iteration	x_n	$g(x_n)$	x_{n+1}
1	1	3	3
2	3	2.333	2.333
3	2.333	2.238	2.238
4	2.238	2.236	2.236
5	2.236	2.236	2.236

Table 24.3: Fixed-Point Iterations for $\sqrt{5}$

Table 24.3 shows the fixed-point iteration convergence. Thus we obtain $\sqrt{5} = 2.236$.

Example 29. Suppose we want to solve the equation

$$f(x) = \cos x - x$$

Iteration	x_n	$g(x_n)$	x_{n+1}
1	0.3	0.92106099400289	0.92106099400289
2	0.92106099400289	0.60497568726594	0.60497568726594
3	0.60497568726594	0.82251592555039	0.82251592555039
4	0.82251592555039	0.68037954156567	0.68037954156567
5	0.68037954156567	0.77733400966246	0.77733400966246
6	0.77733400966246	0.71278594551835	0.71278594551835
7	0.71278594551835	0.75654296195845	0.75654296195845

Table 24.4: Fixed-Point Iterations for $\cos x$

There are many ways to convert $f(x)$ to the fixed-point form $x = g(x)$. Suppose we choose

$$\cos x = x \text{ such that } g(x) = \cos x$$

as our fixed-point equation. With a starting value of $x_0 = 0.3$ we get:

Table 24.4 shows the fixed-point iteration convergence.

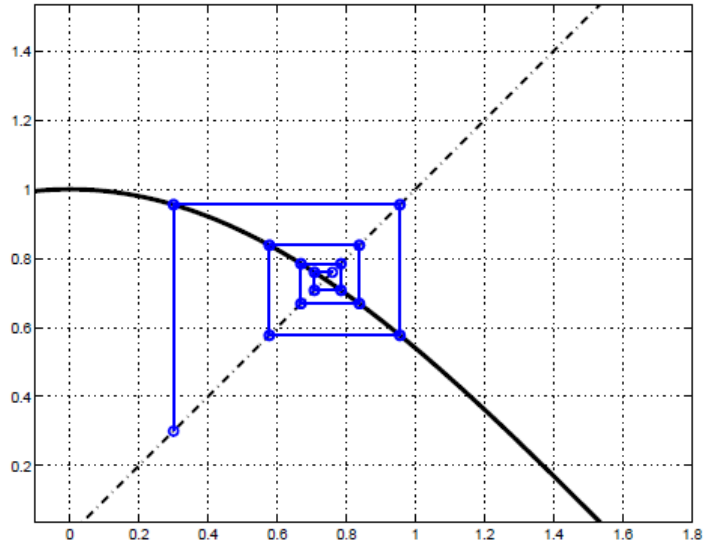


Figure 24.7: Convergence of the cosine function.

Figure 24.7 shows the iterations of the $\cos x$ function.
Source : http://www-rohan.sdsu.edu/ jmahaffy/courses/s10/math541/lectures/pdf/week03/lecture-static-04.pdf

24.3.2 Convergence Analysis

We prove the convergence of fixed-point iteration using the following three theorems.

Theorem 24.3.1. If $f \in C[a,b]$ and $f(x) \in [a,b], \forall x \in [a,b]$, then f has a fixed point $p \in [a,b]$. (Brouwer fixed point theorem)

PROOF. If $f(a) = a$, or $f(b) = b$, then we are done.
Otherwise, $f(a) > a$ and $f(b) < b$.
We define a new function $h(x) = f(x) - x$.
Since both $f(x)$ and x are continuous, we have $h(x) \in C[a,b]$. Further $h(a) > 0$, and $h(b) < 0$ by construction.
Now, the **intermediate value theorem** guarantees $\exists p^* \in (a,b) : h(p^*) = 0$.
We have $0 = h(p^*) = f(p^*) - p^*$ or $p^* = f(p^*)$. □

Theorem 24.3.2. If, in addition, the derivative $f'(x)$ exists on (a,b) and $|f'(x)| \leqslant k < 1, \forall x \in (a,b)$, then the fixed-point is unique.

PROOF. $|f'(x)| \leqslant k < 1$. Suppose we have two fixed points $p^* \neq q^*$.
Without loss of generality we may assume $p^* < q^*$.
The **mean value theorem** tells $\exists r \in (p^*, q^*)$:

$$f'(r) = \frac{f(p^*) - f(q^*)}{p^* - q^*}$$

Now,

$$\begin{aligned} |p^* - q^*| &= |f(p^*) - f(q^*)| \\ &= \left|f'(r)\right| . |p^* - q^*| \\ &\leqslant k\,|p^* - q^*| \\ &< |p^* - q^*| \end{aligned}$$

The contradiction $|p^* - q^*| < |p^* - q^*|$ shows that the supposition $p^* \neq q^*$ is false. Hence, the fixed point is unique. □

Theorem 24.3.3. Then, for any number $p_0 \in [a,b]$, the sequence defined by

$$p_n = f(p_{n-1}), n = 1, 2, \ldots, \infty$$

converges to the unique fixed point $p^* \in [a,b]$.

PROOF. The proof is straight-forward:

$$\begin{aligned} |p_n - p^*| &= |f(p_{n-1}) - f(p^*)| \\ &= \left|f'(r)\right| . |p_{n-1} - p^*| \text{ by MVT} \\ &= k\,|p_{n-1} - p^*| \text{ from Theorem 4.2} \end{aligned}$$

Since $k < 1$, the distance to the fixed-point is shrinking in every iteration.
In fact,

$$|p_{n-1} - p^*| \leqslant k^n\,|p_{n-1} - p^*| \leqslant k^n max\{p_0 - a, b - p_0\}.$$

□

24.4 Newton's Method

Newton's method (also known as the Newton–Raphson method), named after Isaac Newton and Joseph Raphson, is a method for finding successively better approximations to the roots (or zeroes) of a real-valued function.

The idea behind the algorithm is a simple one. We begin with an initial guess for the root we wish to find. This often can be determined from the graph of the function. We then calculate the coordinates of the point on the graph of our function that has for its x-value the initial guess. The equation of the tangent line at this point is computed, then the point at which the tangent line intercepts the x-axis is noted. This usually serves as a better estimate of the zero we seek. Given a function f defined over the reals x, and its derivative f' , we begin with a first guess x_0 for a root of the function f. Provided the function satisfies all the assumptions made in the derivation of the formula, a better approximation x_1 is

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Geometrically, (x_1, 0) is the intersection with the x-axis of the tangent to the graph of f at (x_0, f (x_0)). The process is repeated as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until a sufficiently accurate value is reached.

We can arrive at this from Taylors series. If $f \in C^2[a,b]$, and we know $x^* \in [a,b]$, then we can formally Taylor expand around a point x close to the root:

$$0 = f(x) = f(x) + (x^* - x)\,f'(x) + \frac{(x - x^*)^2}{2} f''(\xi(x)), \xi(x) \in [x, x^*]$$

If we are close to the root, then $|x - x^*|$ is small, which means that $|x - x^*|^2 \ll |x - x^*|$, hence we make the approximation:

$$0 \approx f(x) + (x^* - x)\, f'(x), \leftrightarrow x^* \approx x - \frac{f(x)}{f'(x)}$$

Newton's Method for root finding is based on the approximation:

$$x^* \approx x - \frac{f(x)}{f'(x)}$$

which is valid when x is close to x^*.
Given an approximation x_{n-1}, we get an improved approximation x_n by computing

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

Geometrically Newton's method can be interpreted as follows. Let's suppose that we want to approximate the solution to $f(x) = 0$ and let's also suppose that we have somehow found an initial approximation to this solution say, x_0. This initial approximation is probably not all that good and so we would like to find a better approximation. This is easy enough to do. First we will get the tangent line to $f(x)$ at x_0.

$$y = f(x_0) + f'(x_0)(x - x_0)$$

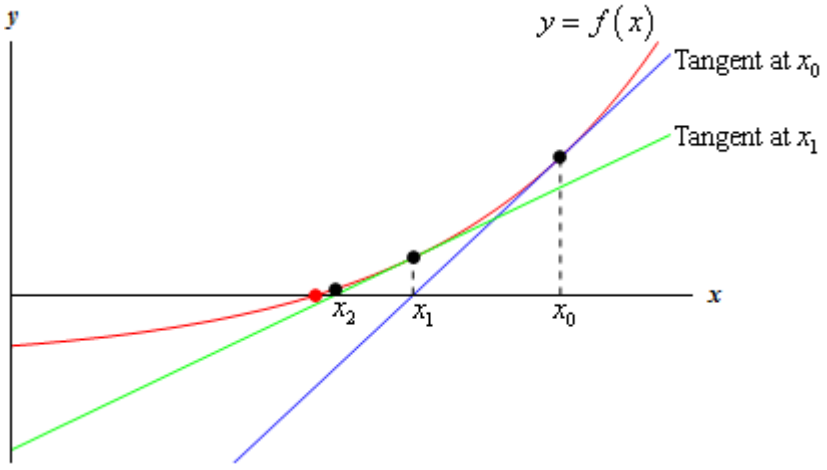


Figure 24.8: Newton's method.

Figure 24.8 shows the geometric interpretation of Newton's method.
Source : http://tutorial.math.lamar.edu/Classes/CalcI/NewtonsMethod.aspx

Now, from the graph shown, the blue line is the tangent line at x_0. We can see that this line will cross the x-axis much closer to the actual solution to the equation than x_0 does. This point is x_1, where $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$. Similarly $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$. Generalizing this, we we get a sequence of numbers that are getting very close to the actual solution. The sequence is given by the Newton's method, i.e.

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

Newton's method is a special case of the fixed point iteration given by

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$$

Here $g(x) = x - \frac{f(x)}{f'(x)}$
$\therefore g'(x) = 1 - \theta$, where $\theta > 0$. So we can prove that

$$|g'(x*)| < 1$$

This means that Newton's method is locally converging. It is also faster than the bisection method

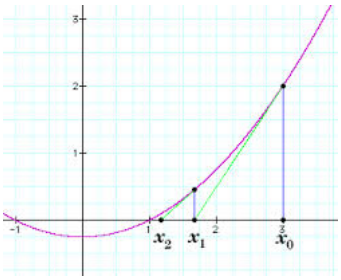


Figure 24.9: Newtons Method

Example 30. Find $\sqrt{a}$ using Newton's method
We have to solve for $x^2 - a = 0 \therefore f(x) = x^2 - a \implies f'(x) = 2x$

$$\implies x^{k+1} = x^k - \frac{x^{k^2} - a}{2x^k} = \frac{1}{2}(x^k + \frac{a}{x^k})$$

$$\implies g(x) = \frac{1}{2}(\frac{a}{x} + x)$$

We see that this is the same as what we got earlier using the fixed point method

24.4.1 Algorithm

1. Declare Variables
2. Set Maximum number of iterations to perform.
3. Set tolerance to small value
4. Set an initial guess x_0.
5. Set the counter of number of iterations to zero.
6. Begin Loop:
   (a) Find next guess $x_1 = x_0 - \frac{f(x_0)}{f'x_0}$.
   (b) If $|f(x_0)| <$ tolerance, then exit loop
   (c) Increment the count of number of iterations.
   (d) If number of iteratations > max allowed, then exit.
7. If root was not found in the max number of iterations, then print warning message.
8. Print the value of root and number of iterations performed.

Example 31. Let us now consider an example in which we want to find the roots of the polynomial $f(x) = x^3 - x + 1 = 0$
The sketch of the graph will tell us that this polynomial has exactly one real root. We first need a good guess as to where it is. This generally requires some creativity, but in this case, notice that $f(-2) = -5$ and $f(-1) = 1$. This tells us that the root is between -2 and -1. We might then choose $x_0 = -1$ for our initial guess.
To perform the iteration, we need to know the derivative $f'(x) = 3x^2 - 1$ so that

$$x_{n+1} = x_n - \frac{x_n^3 - x_n + 1}{3x_n^2 - 1}$$

With our initial guess of $x_0 = -1$, we can produce the following values:

x_0	-1
x_1	-1.500000
x_2	-1.347826
x_3	-1.325200
x_4	-1.324718
x_5	-1.324717
x_6	-1.324717

Notice how the values for x_n become closer and closer to the same value. This means that we have found the approximate solution to six decimal places. In fact, this was obtained after only five relatively painless steps.

Example 32. As another example, let's try and find a root to the equation $f(x) = e^x - 2x = 0$. Notice that $f'(x) = e^x - 2$ so that

$$x_{n+1} = x_n - \frac{e^{x_n} - 2x_n}{e^{x_n} - 2}$$

If we try an initial value $x_0 = 1$, we find that

$$x_1 = 0, \quad x_2 = 1, \quad x_3 = 0, \quad x_4 = 1, ...$$

In other words, Newton's Method fails to produce a solution. Why is this? Because there is no solution to be found! We could rewrite a solution as $e^x = 2x$.

Example 33. Let's solve the previous root-finding problem using Newton's method. We want to find $\sqrt{a}, \forall a \in \mathbb{Z}$
Suppose $a = 612$, i.e. we want to find solution for $x^2 = 612$
Then $f(x) = x^2 - 612$ and derivative $f'(x) = 2x$. With starting point $x_0 = 10$ we obtain the following sequence:
Table 24.5 shows the convergence using Newton's method of iteration. Thus we obtain $\sqrt{612} = 24.7386$.

Iteration	x_n	$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
1	10	35.6
2	35.6	26.3955
3	26.3955	24.7906
4	24.7906	24.7387
5	24.7387	24.7386

Table 24.5: Newton's method for $\sqrt{6}12$

24.4.2 Discussion

1. **Fast Convergence:** Newton's method converges fastest of the methods discussed (Quadratic convergence).
2. **Expensive:** We have to compute the derivative in every iteration, which is quite expensive.
3. **Starting point:** We argued that when $|x - x^*|$ is small, then $|x - x^*|^2 \ll |x - x^*|$, and we can neglect the second order term in the Taylor expansion. In order for Newton's method to converge we need a good starting point.

   Theorem 24.4.1. Let $f(x) \in C^2[a, b]$. If $x^* \in [a, b]$ such that $f(x^*) = 0$ and $f'(x^*) \neq 0$, then there exists a $\delta > 0$ such that Newton's method generates a sequence $\{x_n\}_{n=1}^{\infty}$ converging to x^* for any initial approximation $x_0 \in [x^* - \delta, x^* + \delta]$.
4. **Newton's method as a fixed-point iteration:** $x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$. Then (the fixed point theorem), we must find an interval $[x^* - \delta, x^* + \delta]$ that g maps into itself, and for which $|g'(x)| \leqslant k < 1$.

   $$g'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2} = \frac{f(x)f''(x)}{[f'(x)]^2}$$

   By assumption $f(x^*) = 0, f'(x^*) \neq 0$, so $g'(x^*) = 0$. By continuity $|g'(x)| \leqslant k < 1$ for some neighborhood of x^*. Hence the fixed-point iteration will converge.

Other issues:

1. **Choose a wise starting point**: We need to choose a good starting point such that $|x_{i+1} - x_i|$ is very small.
2. **Faster Convergence** : It converges fastest of the methods discussed so far.
   Convergence is quadratic: as the method converges on the root, the difference between the root and the approximation is squared (the number of accurate digits roughly doubles) at each step.
3. **Expensive** : We have to calculate values of two function for each iteration, one of f(x) and other of $f'(x)$.
4. **Difficulties with this method**
   (a) Difficulty in calculating derivative of a function.
   (b) The method may overshoot, and diverge from that root.
   (c) A large error in the initial estimate can contribute to non-convergence of the algorithm.
   (d) If the root being sought has multiplicity greater than one, the convergence rate is merely linear.

24.5 Secant Method

Newton's method was based on using the line tangent to the curve of y = f(x), with the point of tangency $(x_0, f(x_0))$. When $x_0 \to \alpha$, the graph of the tangent line is approximately the same as the graph of y = f(x) around $x = \alpha$. We then used the root of the tangent line to approximate α.
Consider using an approximating line based on interpolation. We assume we have two estimates of the root α, say x_0 and x_1. Then we produce a linear function $q(x) = a_0 + a_1 x$ with $q(x_0) = f(x_0), q(x_0) = f(x_0)$
This line is sometimes called a secant line. Its equation is given by
$q(x) = \frac{(x_1-x)f(x_0)+(x-x_0)f(x_1)}{x_1-x_0}$
We now solve the equation q(x) = 0, denoting the root by x2. This yields
$x_{n+1} = x_n - f(x_n)/\frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}$, n=1,2,3,...
This is called the secant method for solving f(x)=0.

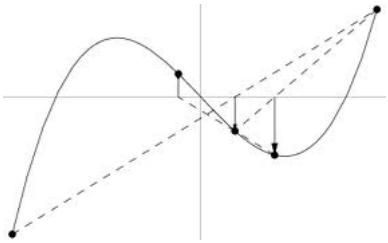


Figure 24.10: Secant Method

We get the secant method when we substitute the differential term in the Newton's method with a finite difference term. As a result the secant method will clearly be a bit slower than Newton's method. However it is still faster than the bisection method and is useful in situations where we have a blackbox representation of the function but not of its differential Substituting $f'(x^k) = \frac{f(x^k)-f(x^{k-1})}{x^k-x^{k-1}}$, we get

$$x^{k+1} = x^k - f(x^k)\frac{x^k - x^{k-1}}{f(x^k) - f(x^{k-1})}$$

The main weakness of Newton's method is the need to compute the derivative, $f'()$, in each step. Many times $f'()$ is far more difficult to compute and needs more arithmetic operations to calculate than $f(x)$.
By definition

$$f'(x_{n-1}) = \lim_{x \to x_{n-1}} \frac{f(x) - f(x_{n-1})}{x - x_{n-1}}$$

Let $x = x_{n-2}$, and approximate

$$f'(x_{n-1}) \approx \frac{f(x_{n-2}) - f(x_{n-1})}{x_{n-2} - x_{n-1}}$$

using this approximation for the derivative in Newton'e method, gives us the Secant Method

$$\begin{aligned} x_n &= x_{n-1} - \frac{f(x_{n-1})}{\left[\frac{f(x_{n-2})-f(x_{n-1})}{x_{n-2}-x_{n-1}}\right]} \\ &= x_{n-1} - \frac{f(x_{n-1})\left[x_{n-2} - x_{n-1}\right]}{f(x_{n-2}) - f(x_{n-1})} \end{aligned}$$

24.5.1 Algorithim

Given an equation f(x) = 0
Let the initial guesses be x0 and x1
Do

$$x_{n+1} = x_n - f(x_n)/\frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}, \text{ n=1,2,3...}$$

while (none of the convergence criterion C1 or C2 is met)

1. C1. Fixing apriori the total number of iterations N.

2. C2. By testing the condition $|x_{i+1} - x_i|$ (where i is the iteration number) less than some tolerance limit, say ϵ, fixed apriori.

Example 34. Find the root of 3x+sin[x]-exp[x] = 0.
Solution: Let the initial guess be 0.0 and 1.0
f(x) = 3x+sin[x]-exp[x]

i	0	1	2	3	4	5	6
x_i	0	1	0.471	0.308	0.363	0.36	0.36

So the iterative process converges to 0.36 in six iterations.

24.6 Additional Problems

Exercise. Write Newton's iterations for solving each of the following non-linear equations

1. $x^3 - 2x - 5 = 0$
Soln. $f(x) = x^3 - 2x - 5 \implies f'(x) = 3x^2 - 2$

$$\therefore x^{k+1} = x^k - \frac{{x^k}^3 - 2x^k - 5}{3{x^k}^2 - 2} = \frac{2{x^k}^3 + 5}{3{x^k}^2 - 2}$$

Let $x^0 = 2$

$$x^1 = \frac{2.2^3 + 5}{3.2^2 - 2} = 2.1$$

$$x^2 = \frac{2.(2.1)^3 + 5}{3.(2.1)^2 - 2} = 2.094$$

$$x^3 = \frac{2.(2.094)^3 + 5}{3.(2.094)^2 - 2} = 2.09455$$

2. $e^{-x} = x$
Soln. $f(x) = x - e^- x \implies f'(x) = 1 + e^- x$

$$\therefore x^{k+1} = x^k - \frac{x^k - e^{-x^k}}{1 + e^{-x^k}} = e^{-x^k}\frac{x^k + 1}{1 + e^{-x^k}}$$

Let $x^0 = 0$

$$x^1 = e^0\frac{0 + 1}{1 + e^0} = 0.5$$

$$x^2 = e^{-0.5}\frac{0.5 + 1}{1 + e^{-0.5}} = 0.566$$

$$x^3 = e^{-0.566}\frac{0.566 + 1}{1 + e^{-0.566}} = 0.567$$

3. $xsinx = 1$
Soln. $f(x) = xsinx - 1 \implies f'(x) = sinx + xcosx$

$$\therefore x^{k+1} = x^k - \frac{x^k sinx^k - 1}{sinx^k + x^k cosx^k} = \frac{{x^k}^2 cosx^k + 1}{sinx^k + x^k cosx^k}$$

Let $x^0 = \frac{\pi}{2}$

$$x^1 = \frac{(\frac{\pi}{2})^2 cos(\frac{\pi}{2}) + 1}{sin(\frac{\pi}{2}) + \frac{\pi}{2}cos(\frac{\pi}{2})} = 1$$

$$x^2 = \frac{1^2.cos(1) + 1}{sin(1) + 1.cos(1)} = 1.115$$

$$x^3 = \frac{1.115^2.cos(1.115) + 1}{sin(1.115) + 1.115.cos(1.115)} = 1.114$$

Exercise. A calculator is defective: it can only add, subtract, and multiply. Use the equation $\frac{1}{x} = \frac{1}{37}$, the Newton Method, and the defective calculator to find $\frac{1}{37}$ correct to 8 decimal places
Soln.For convenience we write a instead of $\frac{1}{37}$. Then $\frac{1}{a}$ is the root of the equation $f(x) = 0$ where

$$f(x) = a - \frac{1}{x}$$

We have $f'(x) = \frac{1}{x^2}$, and therefore the Newton Method yields the iteration

$$x_{n+1} = x_n - \frac{a - \frac{1}{x_n}}{\frac{1}{{x_n}^2}} = x_n - {x_n}^2(a - \frac{1}{x_n}) = x_n(2 - ax_n)$$

Note that the expression $x_n(2ax_n)$ can be evaluated on our defective calculator, since it only involves multiplication and subtraction.
Pick x_0 reasonably close to $\frac{1}{37}$. The choice $x_0 = 1$ would work out fine, but we will start out a little closer, maybe by noting that 1.37 is about $\frac{4}{3}$ so its reciprocal is about $\frac{3}{4}$. Choose $x_0 = 0.75$
We get $x_1 = x_0(2\frac{1}{37}x_0) = 0.729375$. Similarly $x_2 = 0.729926589$, and $x_3 = 0.729927007$. It turns out that $x_4 = x_3$ to 9 decimal places. So we can be reasonably confident that $\frac{1}{37}$ is equal to 0.72992701 to 8 decimal places

24.6.1 Additional Problem

1. Find a root of $x - sin(x) - (1/2) = 0$ using Secant Method.
2. Consider the function $f(x) = x^3 - 3x - 3$. Find all extrema and points of inflection. Is this function odd, even or neither? Sketch a graph of this function. Use Newton's Method to approximate the value of the x-intercept. Start with $x_0 = 2$ and perform four iterations.
3. Show that, Newton's method fails and the iterations diverge to infinity for every $f(x) = |x|^\alpha$ where $0 < \alpha < \frac{1}{2}$.

Chapter 25

More about Non Linear Optimization

25.1 Introduction

Last class, we introduced the problems of finding a solution of m nonlinear equations in n variables (though our discussions were only the case when $n = 1$)

$$\begin{aligned} f_1(x_1, x_2, ...x_n) &= 0 \\ f_2(x_1, x_2, ...x_n) &= 0 \\ &\vdots \\ f_m(x_1, x_2, ...x_n) &= 0 \end{aligned}$$

To simplify, we expressed the above set of equations as $f(x) = 0$ where

$$f(x) = \begin{bmatrix} f_1(x_1, x_2, ...x_n) \\ f_2(x_1, x_2, ...x_n) \\ \vdots \\ f_m(x_1, x_2, ...x_n) \end{bmatrix}, \text{ and } x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

We are interested in solving two categories of problems:

- **Problem 1:** Solve $f(x) = 0$
- **Problem 2:** Min $||f(x)||$

Here the nature of f can be,

- $f : R \to R$: Non-linear. Eg, $f(x) = x^2 + 2x + 1 = 0$
- $f : R^n \to R$: $f(x_1, x_2, ...x_n)$
- $f : R^n \to R^m$: $f_1(x_1, x_2, ...x_n)$
  $f_2(x_1, x_2, ...x_n)$
  .
  .
  $f_m(x_1, x_2, ...x_n)$

- f : convex $\to$ Local Minima = Global Minima
- f : non-convex $\to$ Local Minima $\neq$ Global Minima

25.2 Review of Iterative Scehemes

In the Last class, various iterative methods were discussed for solving the nonlinear equations where $f : R^n \to R$ is differentiable. *Did we make any assumption of convexity?* (we did not really talk about the case of $f : R^n \to R^m$)

The methods discussed include:

Bisection Method

We start with an interval $[a, b]$ that satisfies $f(a) \cdot f(b) < 0$ (the function values at the end points of the interval have opposite signs). Since f is continuous, this guarantees that the interval contains at least one solution of $f(x) = 0$.

In each iteration we evaluate f at the midpoint $p = (a + b)/2$ of the interval, and depending on the sign of $f(p)$, replace a or b with p. If $f(p)$ has the same sign as $f(a)$, we replace a with p. Otherwise we replace b. Thus we obtain a new interval that still satisfies $f(a)f(b) <0$. The method is called bisection because the interval is replaced by either its left or right half at each iteration.

Example 35. Let us find a root of $f(x) = 3x + sin(x) - exp(x) = 0$. The graph of this eqution is given in the Figure 25.1. It is clear from the graph that there are two roots, one lies between 0 and 0.5 and the other lies between 1.5 and 2.0. Consider the function $f(x)$ in the interval $[0, 0.5]$ since $f(0) * f(0.5) <0$. Then the bisection iterations are given in Table **??**

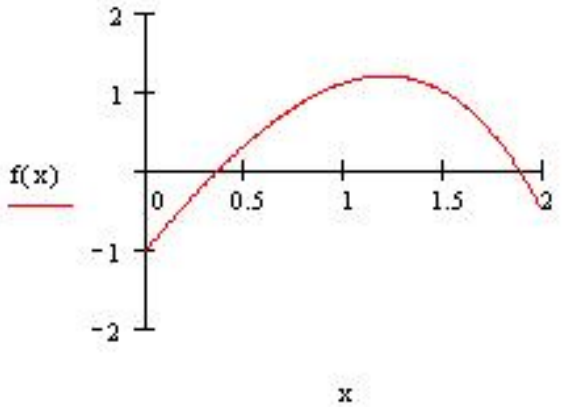


Figure 25.1: Graph of the equation f(x)

Iteration No	a	b	c	$f(a) * f(c)$
1	0	0.5	0.25	0.287 (+ve)
2	0.25	0.50	0.393	-0.015 (-ve)
3	0.65	0.393	0.340	9.69 E-3 (+ve)
4	0.34	0.393	0.367	-7.81 E-4 (-ve)
5	0.34	0.367	0.354	8.9 E-4 (+ve)
6	0.35	0.367	0.360	-3.1 E-6 (-ve)

Table 25.1: Bisection Method Iterations

Newtons Method

In numerical analysis, Newton's method (also known as the Newton-Raphson method), named after Isaac Newton and Joseph Raphson, is a method for finding successively better approximations to the roots (or zeroes of a real-valued function.

$$x : f(x) = 0\,.$$

The Newton-Raphson method in one variable is implemented as follows. Given a function f defined over the real x, and its derivative f', we begin with a first guess x_0 for a root of the function f. Provided the function satisfies all the assumptions made in the derivation of the formula, a better approximation x_1 is

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}\,.$$

Geometrically, $(x_1, 0)$ is the intersection with the x-axis of the tangent to the graph of f at $(x_0, f(x_0))$.

The process is repeated as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until a sufficiently accurate value is reached. This method can also be extended to complex functions and to systems of equations.

Example 36. Let us now consider an example in which we want to find the roots of the polynomial $f(x) = x^3 - x + 1 = 0$.

The sketch of the graph will tell us that this polynomial has exactly one real root. We first need a good guess as to where it is. This generally requires some creativity, but in this case, notice that $f(-2) = -5$ and $f(-1) = 1$. This tells us that the root is between -2 and -1. We might then choose $x_0 = -1$ for our initial guess.

To perform the iteration, we need to know the derivative $f'(x) = 3x^2 - 1$ so that

$$x_{n+1} = x_n - \frac{x_n^3 - x_n + 1}{3x_n^2 - 1}$$

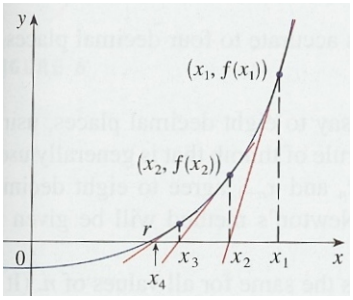


Figure 25.2: Demonstration of Newtons Method

x_0	-1
x_1	-1.500000
x_2	-1.325200
x_3	-1.324718
x_4	-1.324717
x_5	-1.324717
x_6	-1.324717

Table 25.2: Newton Method Iterations

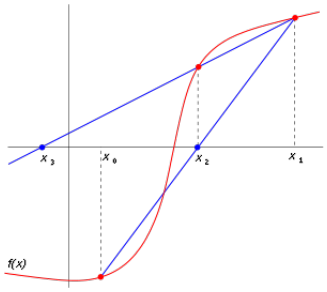


Figure 25.3: Demonstration of Secant Method

With our initial guess of $x_0 = -1$, we can produce the following values as in Table **??**.

Notice how the values for x_n become closer and closer to the same value. This means that we have found the approximate solution to six decimal places. In fact, this was obtained after only five relatively painless steps.

Secant Method

In numerical analysis, the secant method is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function f. The secant method can be thought of as a finite difference approximation of Newton's method. However, the method was developed independently of Newton's method, and predated the latter by over 3,000 years.

The secant method is defined by the recurrence relation

$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})} = \frac{x_{n-2}f(x_{n-1}) - x_{n-1}f(x_{n-2})}{f(x_{n-1}) - f(x_{n-2})} \tag{25.1}$$

As can be seen from the recurrence relation, the secant method requires two initial values, x_0 and x_1, which should ideally be chosen to lie close to the root.

Example 37. Lets try to find the root of $3x + sin(x) - exp(x) = 0$, using secant method. Let the initial guess be 0.0 and 1.0 Following the secant method's recurrence relation, we will get the Table **??** As can be seen the secant method converges to 0.36 in six iterations.

x_0	0
x_1	1
x_2	0.471
x_3	0.308
x_4	0.363
x_5	0.36
x_6	0.36

Table 25.3: Secant Method Iterations

25.3 Gradient, Hessian and Jacobian

We had mostly focussed on valued functions of a real variable in the last class. We also know, real valued functions of multiple variables and vector valued functions are of equal interest to us. Often we would be using terms such as Gradient ($\nabla f(x)$) , Hessian ($\nabla^2 f(x)$) and Jacobian (J) while dealing with nonlinear functions and their optimization. Following are the basic definitions of each of them.

25.3.1 Basic Terminology

Gradient: The gradient of a function $f(x)$ of n variables, at x^*, is the vector of first partial derivatives evaluated at x^*, and is denoted as $\nabla f(x^*)$:

$$\nabla f(x^*) = \begin{bmatrix} \frac{\partial f(x^*)}{\partial x_1} \\ \frac{\partial f(x^*)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x^*)}{\partial x_n} \end{bmatrix} \tag{25.2}$$

Hessian: The Hessain of a function $f(x)$ of n variables, at x^*, is the matrix of second partial derivatives evaluated at x^*, and is denoted as $\nabla^2 f(x^*)$:

$$\nabla^2 f(x^*) = \begin{bmatrix} \frac{\partial^2 f(x^*)}{\partial x_1^2} & \frac{\partial^2 f(x^*)}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f(x^*)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x^*)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x^*)}{\partial x_2^2} & \dots & \frac{\partial^2 f(x^*)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x^*)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x^*)}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f(x^*)}{\partial x_n^2} \end{bmatrix} \tag{25.3}$$

Hessian is a square matrix of second-order partial derivatives of a function. It describes the local curvature of a function of many variables. The Hessian of a function $f(x)$ of n variables, at x^*, is the matrix of second partial derivatives evaluated at x^*, and is denoted as $\bigtriangledown^2 f(x^*)$. This is a symmetric matrix, because $\frac{\partial^2 f(x^*}{\partial x_i, \partial x_j} = \frac{\partial^2 f(x^*)}{\partial x_j, \partial x_i}$

Jacobian: Given a set of m equations $y_i = f_i(x)$ in n variables $x_1, ..., x_n$, the Jacobian is defined as:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{25.4}$$

Looking into the above definitions one can observe a simple relation between $\nabla f(x)$, $\nabla^2 f(x)$ and J. The jacobian of gradient is Hessian and is given by:

$$J(\nabla f(x)) = \nabla^2 f(x); \tag{25.5}$$

25.4 Approximating Functions and Taylor's Series

Taylor series is a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots .$$

Let $x \to y$ and $a \to x$

$$f(y) = f(x) + \frac{f'(x)}{1!}(y-x) + \frac{f''(x)}{2!}(y-x)^2 + \frac{f^{(3)}(x)}{3!}(y-x)^3 + \cdots .$$

It is common practice to approximate a function by using a finite number of terms of its Taylor series. Taylor's theorem gives quantitative estimates on the error in this approximation. Any finite number of initial terms of the Taylor series of a function is called a Taylor polynomial.

The Taylor series of a function is the limit of that function's Taylor polynomials, provided that the limit exists. A function may not be equal to its Taylor series, even if its Taylor series converges at every point. A function that is equal to its Taylor series in an open interval (or a disc in the complex plane) is known as an analytic function.

Example 38. Let's compute the Taylor series for $f(x) = e^x$ with center $x_0 = 0$. All derivatives are of the form e^x , so at $x_0 = 0$ they evaluate to 1. Thus the Taylor series has the form:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Commonly Used Taylor Series are $\frac{1}{1-x}, cosx, sinx, e^x, ln(1+x), etc.$

25.4.1 Orders of Approximation

Orders of approximation refer to formal or informal terms for how precise an approximation is, and to indicate progressively more refined approximations: in increasing order of precision, a zeroth-order approximation, a first-order approximation, a second-order approximation, and so forth.

Formally, an nth-order approximation is one where the order of magnitude of the error is at most x^{n+1}, or in terms of big O notation, the error is $O(x^{n+1})$. In suitable circumstances, approximating a function by a **Taylor polynomial** of degree n yields an nth-order approximation, by **Taylor's theorem**: a first-order approximation is a linear approximation, and so forth.

Example 39. Figure 25.4 is an accurate approximation of sin(x) around the point x = 0. The pink curve is a polynomial of degree seven:

$$sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}.$$

The error in this approximation is no more than $\frac{|x|^9}{9!}$. In particular, for $1 < x < 1$, the error is less than 0.000003.

To view the usefulness of Taylor series, Figures 25.5, 25.6, and 25.7 show the zeroth, first, and second order Taylor series approximations of the exponential function $f(x) = e^x$ at $x = 0$.

Zero Order Approximation

The approximation $f(x+h) = f(x)$ is called a zeroth-order Taylor-series approximation.

25.4.2 First order approximation

The first order approximation takes the first two terms in the series and approximates the function

$$f(y) = f(x) + \tfrac{f'(x)}{1!}(y-x)$$

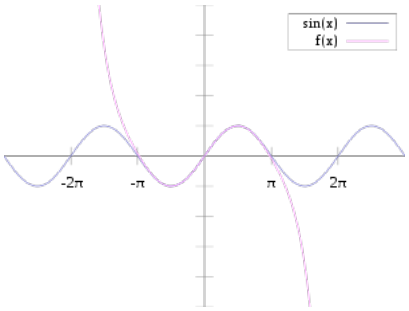


Figure 25.4: The sine function (blue) is closely approximated by its Taylor polynomial of degree 7 (pink) for a full period centered at the origin.

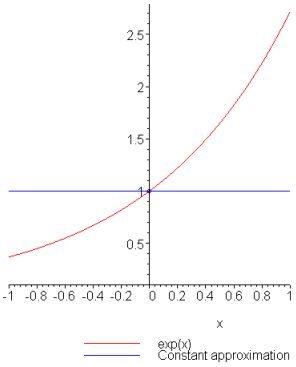


Figure 25.5: The zeroth-order Taylor series approximation of e^x around x = 0.

Let $f(y) = 0$, the true solution

then $y - x = \frac{-f(x)}{f'(x)}$

$\Rightarrow y = x - \frac{f(x)}{f'(x)}$

$\Rightarrow y = x - \frac{f(x_0)}{f'(x_0)}$

The first order approximation is the equation of a line with a slope of $f'(x_0)$. So, the first two terms of the Taylor Series gives us the equation of the line tangent to our function at the point (x_0, y_0) .

We can now develop an iterative algorithm by replacing y with x^{k+1} and x with x^k. At each iteration we will get a better solution then the previous iteration.

$$x^{k+1} = x^k - f(x)/f'(x)$$

Function of the type $f: \mathbf{R^n} \to \mathbf{R}$

The first order approximation can be given as

$$f(y) = f(x) + \nabla f(x)^T (y - x) \tag{25.6}$$

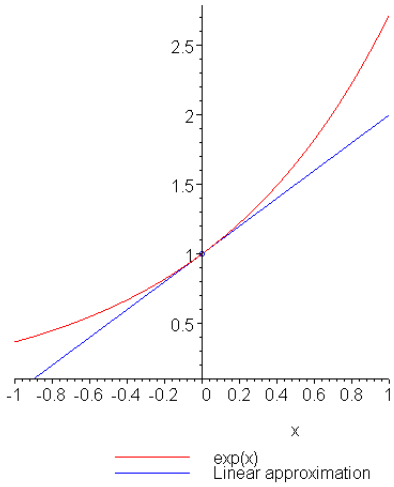


Figure 25.6: The first-order Taylor series approximation of e^x around x = 0.

Where $\nabla f(x)$ can be given as

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

$$\Rightarrow f(y) = f(x) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(y_i - x_i)$$

Putting $f(y) = 0$ in Equation 25.6 we get

$$0 = f(x) + \nabla f(x)^T (y - x)$$

Let $y - x = s$

$$0 = f(\overline{x}) + \nabla f(\overline{x})^T \overline{s}$$

$$\Rightarrow \nabla f(\overline{x})^T \overline{s} = -f(\overline{x}) \tag{25.7}$$

Iterative algorithm to solve $f(\overline{x}) = 0$ can be given as

algorithm
  $x^0 \leftarrow$ Critical guess
  for $k = 0, 1, 2, \cdots$ **do**
    solve $\nabla f(x_k)^T \overline{s} = -f(x_k)$ for s
    $x_{k+1} = x_k + s$
  end for

Unfortunately this method does now work. Why?

Function of the type f: $\mathbf{R^n} \rightarrow \mathbf{R^m}$

The first order approximation can be given as

$$\overline{f}(y) = \overline{f}(x) + J_x(y - x) \tag{25.8}$$

Where Jacobian J is an $m \times n$ matrix and can be given as

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

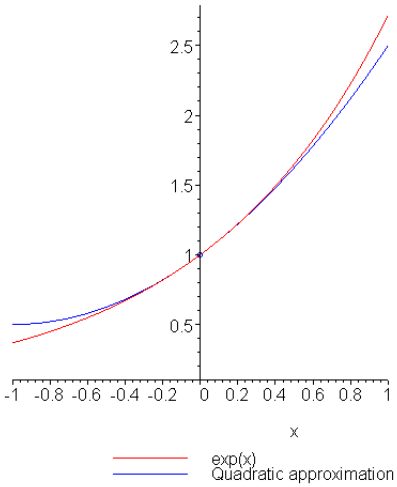


Figure 25.7: The second-order Taylor series approximation of e^x around x = 0.

Putting $f(y) = 0$ in equation 26.1 we get

$$\overline{0} = \overline{f}(\overline{x}) + J_x(\overline{y} - \overline{x})$$

Let $\overline{y} - \overline{x} = \overline{s}$

$$\begin{aligned} 0 &= f(x) + J_x s \\ \Rightarrow J_x s &= -f(x) \end{aligned} \tag{25.9}$$

Iterative algorithm to solve $\overline{f}(\overline{x}) = \overline{0}$ can be given as algorithm

$\overline{x}^0 \leftarrow$ Critical guess
for $k = 0, 1, 2, \cdots$ **do**
  solve $J_{x_k} s = -f(x_k)$ for s
  $x_{k+1} = x_k + s$
end for

The convergance is guaranteed if spectral radius of J and maximam eigen value of J is less then 1.

25.4.3 second order approximation

$$f(y) = f(x) + f'(x)(y - x) + \frac{f''(x)(y - x)^2}{2!}$$

We can find better solution then the first order approximation but it is harder to compute.

As we notice, the second order approximation uses the first three terms of the Taylor Series.

$$f(y) = f(x) + \frac{f'(x)}{1!}(y - x) + \frac{f''(x)}{2!}(y - x)^2$$

Let $f(y) = 0$, the true solution

$$\text{then } 0 = f(x) + \frac{f'(x)}{1!}(y - x) + \frac{f''(x)}{2!}(y - x)^2$$

In second order approximation, better solution can be obtained, but more difficult to compute.

Analysis While the approximation in Figure 1 becomes poor very quickly, it is quite apparent that the linear, or 1st-order, approximation in Figure 2 is already quite reasonable in a small interval around $x = 0$. The quadratic, or 2nd-order, approximation in Figure 3 is even better. However, as the degree of approximation increases, computation also increases. So, there's a tradeoff!

25.5 Optimality Conditions

Let, $f : R^n \to R$ be scalar valued function of n variables $x = (x_1, x_2, ..., x_n)$. We say $x^* = (x_1^*, x_2^*, ...x_n^*)$ minimizes f if $f(x^*) \leq f(x)$ for all $x \in R^n$. We use the notation *min f(x)* to denote the problem of finding an x^* that minimizes f. A vector x^* is a local minimum if there exists a neighbourhood around x^* in which $f(x^*) \leq f(x)$. With minimum we refer to a global minimum. We can have case with finite *min f(x)* and no x^* with $f(x^*) = minf(x)$, in such a case optimal value is not attained. It is also possible that f(x) is unbounded below, in which case we define the optimal value as $minf(x) = -\infty$.

25.5.1 Optimality conditions

Global optimality A function g is convex if $\bigtriangledown^2 g(x)$ is a positive semi definite ever where.If g is convex, then x is a minimum if and only if

$$\bigtriangledown g(x) = 0$$

Means there are no other local minima, i.e. every local minimum is global.

Local optimality It is much harder to characterize optimality if g is not convex (i.e., if there are points where the Hessian is not positive semidefinite). It is not sufficient to set the gradient equal to zero, because such a point might correspond to a local minimum, a local maximum, or a saddle point . However, we can state some simple conditions for local optimality.

1. Necessary condition. If x is locally optimal, then $\bigtriangledown g(x) = 0$and $\bigtriangledown^2 g(x)$ is positive semidefinite.
2. Sufficient condition. If $\bigtriangledown g(x) = 0$ and $\bigtriangledown^2 g(x)$ is positive definite, then x is locally optimal.

Example 40. Lets try to find the local extrema of $f(x_1, x_2) = x_1^3 + x_2^3 - 3x_1x_2$.

This function is everywhere differentiable, so extrema can only occur at points x^* such that its gradient ($\bigtriangledown f(x^*) = 0$.)

$$\bigtriangledown f(x) = \begin{bmatrix} 3x_1^2 - 3x_2 \\ 3x_1^2 - 3x_1 \end{bmatrix}$$

This equals 0 iff (x_1, x_2)= $(0, 0)$ or $(1, 1)$. The Hessian is

$$H(x) = \begin{bmatrix} 6x_1 & -3 \\ -3 & 6x_2 \end{bmatrix}$$

So,

$$H(0, 0) = \begin{bmatrix} 0 & -3 \\ -3 & 0 \end{bmatrix}$$

Let H_1 denote the first principal minor of $H(0, 0)$ and let H_2 denote its second principal minor. Then $det(H_1) = 0$ and $det(H_2) = -9$.
Therefore, $H(0, 0)$ is neither positive nor negative definite.

$$H(1, 1) = \begin{bmatrix} 6 & -3 \\ -3 & 6 \end{bmatrix}$$

Its first principal minor has $det(H_1) = 6 > 0$ and its second principal minor has $det(H_2) = 25 > 0$. Therefore, $H(1, 1)$ is positive definite, which implies that $(1, 1)$ is a local minimum.

25.6 Additional Examples

Exercise. Derive gradient and Hessian of the function $g(x) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2 + 1}$. Also, show that $\bigtriangledown g(x)^2$ is Positive Definite.

Calculation of Gradient

$$\bigtriangledown g(x) = \begin{bmatrix} \frac{\partial g(x)}{\partial x_1} \\ \frac{\partial g(x)}{\partial x_2} \\ \vdots \\ \frac{\partial g(x)}{\partial x_n} \end{bmatrix} \tag{25.10}$$

$$= \begin{bmatrix} \frac{x_1}{\sqrt{x1^2 + x2^2 + \cdots + xn^2 + 1}} \\ \frac{x_2}{\sqrt{x1^2 + x2^2 + \cdots + xn^2 + 1}} \\ \vdots \\ x_n\sqrt{x1^2 + x2^2 + \cdots + xn^2 + 1} \end{bmatrix} \tag{25.11}$$

$$= \frac{1}{g(x)} \begin{bmatrix} x1 \\ x2 \\ \vdots \\ xn \end{bmatrix} \tag{25.12}$$

Calculation of Hessian

$$\bigtriangledown^2 g(x) = \begin{bmatrix} \frac{x_2^2 + \cdots + x_n^2 + 1}{(x_1^2 + x_2^2 + \cdots + x_n^2 + 1)^{3/2}} & \frac{-x_1x_2}{(x_1^2 + x_2^2 + \cdots + x_n^2 + 1)^{3/2}} & \cdots & \frac{-x_1x_n}{(x_1^2 + x_2^2 + \cdots + x_n^2 + 1)^{3/2}} \\ \frac{-x_2x_1}{(x_1^2 + x_2^2 + \cdots + x_n^2 + 1)^{3/2}} & \frac{x_1^2 + x_3^2 + \cdots + x_n^2 + 1}{(x_1^2 + x_2^2 + \cdots + x_n^2 + 1)^{3/2}} & \cdots & \frac{-x_2x_n}{(x_1^2 + x_2^2 + \cdots + x_n^2 + 1)^{3/2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{-x_nx_1}{(x_1^2 + x_2^2 + \cdots + x_n^2 + 1)^{3/2}} & \frac{-x_nx_2}{(x_1^2 + x_2^2 + \cdots + x_n^2 + 1)^{3/2}} & \cdots & \frac{x_1^2 + \cdots + x_{n-1}^2 + 1}{(x_1^2 + x_2^2 + \cdots + x_n^2 + 1)^{3/2}} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{x_2^2 + \cdots + x_n^2 + 1}{g^3(x)} & \frac{-x_1x_2}{g^3(x)} & \cdots & \frac{-x_1x_n}{g^3(x)} \\ \frac{-x_2x_1}{g^3(x)} & \frac{x_1^2 + x_3^2 + \cdots + x_n^2 + 1}{g^3(x)} & \cdots & \frac{-x_2x_n}{g(x)^3} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{-x_nx_1}{g^3(x)} & \frac{-x_nx_2}{g^3(x)} & \cdots & \frac{x_1^2 + \cdots + x_{n-1}^2 + 1}{g^2(x)} \end{bmatrix} \tag{25.13}$$

$$= \frac{1}{g(x)} \begin{bmatrix} \frac{x_2^2 + \cdots + x_n^2 + 1}{g^2(x)} & \frac{-x_1x_2}{g^3(x)} & \cdots & \frac{-x_1x_n}{g^2(x)} \\ \frac{-x_2x_1}{g^2(x)} & \frac{x_1^2 + x_3^2 + \cdots + x_n^2 + 1}{g^2(x)} & \cdots & \frac{-x_2x_n}{g^2(x)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{-x_nx_1}{g^2(x)} & \frac{-x_nx_2}{g^2(x)} & \cdots & \frac{x_1^2 + \cdots + x_{n-1}^2 + 1}{g^2(x)} \end{bmatrix}$$

The above Hessian Matrix can be expressed as

$$\bigtriangledown^2 g(x) = \frac{1}{g(x)}(I - uu^T) \tag{25.14}$$

, by considering $u = \frac{1}{g(x)} \begin{bmatrix} x1 & x2 & \cdots & xn \end{bmatrix}$

Now we will prove that, $\bigtriangledown^2 g(x) = \frac{1}{g(x)}(I - uu^T)$ is positive definite matrix.

- We can see that, it is symmetric.
- As u is a n-vector with norm less than 1, all its eigen values are positive.
- Since all the eigen values are positive and the matrix is real symmetric, the matrix is positive definite.

Exercise. Consider the system of equations

$$x_1 - 1 = 0$$
$$x_1 x_2 - 1 = 0$$

Suggest a starting point where Newton's method may fail. Why?
Soln. For a system of equations, Newton's method is represented as

$$\vec{x}^{k+1} = \vec{x}^k - {\mathbf{J_F}}^{-1} \vec{F}(\vec{x}^k)$$

where $\mathbf{J_F}$ is the Jacobian and $\vec{F}$ is the function vector. In this case, we have

$$\vec{F} = \begin{pmatrix} x_1 - 1 \\ x_1 x_2 - 1 \end{pmatrix}, \mathbf{J_F} = \begin{pmatrix} 1 & 0 \\ x_2 & x_1 \end{pmatrix}$$

Clearly for $x_1 = 1$ and any value of x_2, $|\mathbf{J_F}| = 0 \implies {\mathbf{J_F}}^{-1}$ is not defined
$\therefore$ If we start with $x_1 = 1$ any value of x_2 Newton's method fails

Exercise. Use Newton's method to find all solutions of the system

$$(x_1 - 1)^2 + 2(x_2 - 1) = 1$$
$$3(x_1 + x_2 - 2)^2 + (x_1 - x_2 - 1)^2 = 2$$

in two variables x_1 and x_2
Soln. For the above system of equations, we have

$$\vec{F} = \begin{pmatrix} (x_1 - 1)^2 + 2(x_2 - 1) - 1 \\ 3(x_1 + x_2 - 2)^2 + (x_1 - x_2 - 1)^2 - 2 \end{pmatrix}, \mathbf{J_F} = \begin{pmatrix} 2x_1 - 2 & 2 \\ 8x_1 + 4x_2 - 14 & 4x_1 + 8x_2 - 10 \end{pmatrix}$$

We start with one seed value $x_1 = 2, x_2 = 1$. In this case we get $\mathbf{J_F} = \begin{pmatrix} 2 & 2 \\ 6 & 6 \end{pmatrix}$. Clearly this does not have an inverse so the method fails.
We try another seed $x_1 = 1, x_2 = 1$. We get

$$\mathbf{J_F} = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}, \vec{F} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$$

$$\therefore \vec{x}^1 = \vec{x}^0 - \begin{pmatrix} 0 \\ -\frac{1}{2} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 \\ -\frac{1}{2} \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{3}{2} \end{pmatrix}$$

Calculating $\mathbf{J_F}$, we get $\mathbf{J_F} = \begin{pmatrix} 0 & 2 \\ 0 & 6 \end{pmatrix}$. Clearly this does not have an inverse, so the method fails again. This leads us to believe that the system of equations has no real solutions.

25.6.1 Additional Problem

- Derive gradient and Hessian of the function g(x) = $\sqrt{||Cx||^2 + 1}$, where C is a left-invertible $m \times n$ matrix.
- Let f(x) be a strictly convex twice-continuously differentiable and x* be a point for which $\Delta f(x^*) = 0$. Suppose that H(x) satisfies the following conditions:
  a.) there exists a scalar $h > 0$ for which $\|H(x^*)^{-1}\| \leq \frac{1}{h}$
  b.) there exists scalars $\beta > 0$ and $L > 0$ for which $\|H(x) - H(x^*)\| \leq L\|x - x^*\|$
  Let x satisfy $\|x - x^*\| < \gamma := min\{\beta, \frac{2h}{3L}\}$ and $x_N := x - H(x)^{-1} \bigtriangledown f(x)$, then prove following:
  (i)$\|x_N - x^*\| \leq \|x - x^*\|^2 \frac{L}{2(h - L\|x - x^*\|)}$
  (ii)$\|x_N - x^*\| < \|x - x^*\| < \gamma$
  (iii)$\|x_N - x^*\| \leq \|x - x^*\|^2 \frac{3L}{2h}$
  Hint: Ref. Quadratic Convergence Theorem of Newton's Method

- Show that the eigen value-vector problem can be casted as *root-finding* problem i.e.;
  $(A - \lambda I)v = 0; v^T v = 1$
  Derive the Newton's iteration to solve the above problem.

Chapter 26

More about Non Linear Optimization

In the last lecture, we had seen how the Newtons method come out of the Taylors series expansion. We had also briefly seen the situation where we solve $f(x) = 0$ where $f : R^m \to R^n$.

We now complete the need for studying this problem for optimization by connecting to the situation of minimizinh $g(x)$ where x is a vector. The conditions for optimality in the' previous lecture pointed to the need of solving $\nabla g = 0$ or else, we have now set of equations to simultaneously solve.

26.1 Newton's Method For Sets of Nonlinear Equations

In the single variable case, Newton's method was derived by considering the linear approximation of the function f at the initial guess x_0. From Calculus, the following is the linear approximation of f at x_0, for vectors and vector-valued functions:

$$f(x) \approx f(x_0) + J(x - x_0).$$

Here J is an $n \times n$ matrix whose entries are the various partial derivative of the components of f, i.e. :

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}.$$

J is the derivative matrix (or Jacobian matrix, explained elsewhere) evaluated at x_0

That is,

$$\overline{f}(y) = \overline{f}(x) + J_x(y - x) \tag{26.1}$$

Where Jacobian J is an $m \times n$ matrix and can be given as

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Putting $f(y) = 0$ in equation 26.1 we get

$$\overline{0} = \overline{f}(\overline{x}) + J_x(\overline{y} - \overline{x})$$

Let $\overline{y} - \overline{x} = \overline{s}$

$$\begin{aligned} 0 &= f(x) + J_x s \\ \Rightarrow J_x s &= -f(x) \end{aligned} \tag{26.2}$$

Iterative algorithm to solve $\overline{f}(\overline{x}) = \overline{0}$ can be given as algorithm

$\overline{x}^0 \leftarrow$ Critical guess
for $k = 0, 1, 2, \cdots$ **do**
  solve $J_{x_k} s = -f(x_k)$ for s
  $x_{k+1} = x_k + s$
end for

The convergance is guaranteed if spectral radius of J and maximam eigen value of J is less then 1.

Example 41. As an example, lets take a problem with two equations and two variables

$$f_1(x_1, x_2) = log(x_1^2 + 2x_2^2 + 1) - 0.5, \qquad f_2(x_1, x_2) = -x_1^2 + x_2 + 0.2 \tag{26.3}$$

The derivative matrix is $J = \begin{bmatrix} \frac{2x_1}{x_1^2 + 2x_2^2 + 1} & \frac{4x_2}{x_1^2 + 2x_2^2 + 1} \\ -2x_1 & 1 \end{bmatrix}$.

After finding the derivative matrix, we can proceed with the Newtons method discussed in the last class.

There are two solutions, (0.70,0.29) and (−0.70,0.29).

Example 42. Suppose we need to solve the following system of non linear equations.
$F(x) = \begin{bmatrix} x_1 + x_2 - 3 \\ x_1^2 + x_2^2 - 9 \end{bmatrix}$

1. Let the initial guess be $x_\circ = \begin{bmatrix} 1 & 5 \end{bmatrix}$

2. Iteration 1
   Solving $J(x_\circ).s_\circ = -F(x_\circ)$
   $\begin{bmatrix} 1 & 1 \\ 2 & 10 \end{bmatrix}.s_\circ = -\begin{bmatrix} 3 \\ 17 \end{bmatrix}$
   $s_\circ = \begin{bmatrix} \frac{13}{8} \\ \frac{11}{8} \end{bmatrix}$ $x_1 = x_\circ + s_\circ = \begin{bmatrix} -0.625 & 3.625 \end{bmatrix}$

3. Iteration 2
   Solving $J(x_1).s_1 = -F(x_1)$
   $\begin{bmatrix} 1 & 1 \\ -\frac{5}{4} & \frac{29}{4} \end{bmatrix}.s_1 = \begin{bmatrix} 0 \\ \frac{145}{32} \end{bmatrix}$
   $s_1 = \begin{bmatrix} \frac{145}{272} \\ -\frac{145}{272} \end{bmatrix}$
   $x_2 = x_1 + s_1 = \begin{bmatrix} 0.092 & 3.092 \end{bmatrix}$. The actual solution to the above problem is $\begin{bmatrix} 0 & 3 \end{bmatrix}$. In just two iterations algorithm has reached quite close.

26.2 Newton's method for minimizing a convex function

If the objective function g is convex we can find it's minimum by solving $\bigtriangledown g(x) = 0$. This is a set of n nonlinear equations in n variables that we can solve using any method of nonlinear equations. If we linearise the optimality condition $\bigtriangledown g(x) = 0$ near x° we obtain

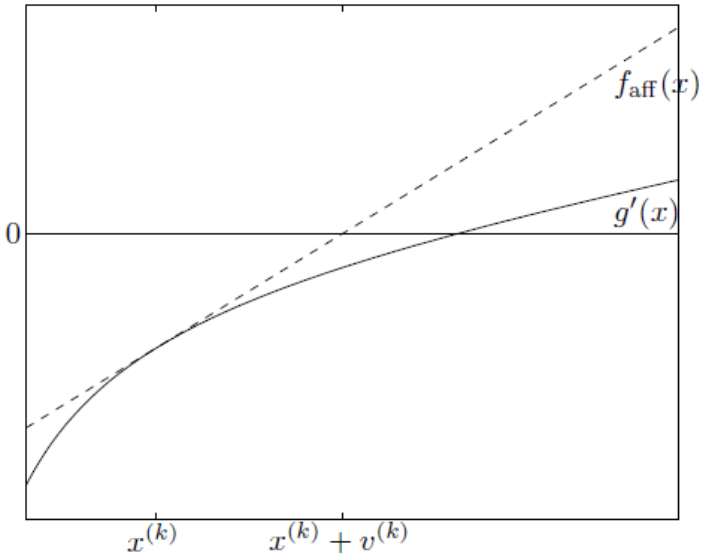
$$\bigtriangledown g(x) \approx \bigtriangledown g(x^\circ) + \bigtriangledown^2 g(x)(x - x^\circ) = 0$$

This is a linear equation in x, with solution

$$x = x^\circ - \bigtriangledown^2 g(x^\circ)^{-1} g(x^\circ)$$

The above step is called Newton step.

When $n = 1$ this interpretation is particularly simple. The solution of the linearized optimality condition is the zero-crossing of the derivative $g(x)$, which is monotonically increasing since $g(x) > 0$. Given our current approximation $x(k)$ of the solution, we form a first-order Taylor approximation of $g(x)$ at $x(k)$. The zero crossing of this approximation is then $x(k) + v(k)$. This interpretation is illustrated in the below figure.



The solid curve is the derivative $g(x)$ of the function g. $f_{aff}(x) = g(x^{(k)}) + g(x^{(k)})(x - x^{(k)})$ is the affine approximation of $g(x)$ at $x^{(k)}$. The Newton step $v(k)$ is the difference between the root of f_{aff} and the point $x^{(k)}$.

ALGORITHM for Newton's method for unconstrained optimization
given
initial x, tolerance $\varepsilon > 0$
repeat
1. Evaluate $\nabla g(x)$ and $\nabla^2 g(x)$.
2. if $\|\nabla g(x)\| \leq \varepsilon$, return x.
3. Solve $\nabla^2 g(x)v = \nabla g(x)$.
4. $x := x + v$.

until a limit on the number of iterations is exceeded

Example 43. Let's try to find the minimum of this function $f(x) = 7x - lnx$

$\bigtriangledown f(x) = f'(x) = 7 - \frac{1}{x}$

$Hessian = H(x) = f''(x) = \frac{1}{x^2}$

It is not hard to check that $x^* = \frac{1}{7} = 0.142857143$ is the unique global minimizer.

From Egn 5, the Newton Direction is $-H(x)^{-1} \bigtriangledown f(x) = x - 7x^2$, and is defined so long as $x > 0$ (Domain of $f(x)$ is $x > 0$)

So, Newton's method will generate the sequence of iterates as:

$$\begin{aligned} x_{k+1} &= x_k + (x_k - 7{x_k}^2) \\ &= 2x_k - 7x_k^2 \end{aligned}$$

Below are some examples of the sequences generated by this algorithm for different starting points

Note that the iterate in the first column is not in the domain of the objective function, so the algorithm has to terminate.

As we can see, the above algorithm converges only when started near the solution, as expected based on the general properties of the method.

26.3 Newton's method with backtracking

Newton's method always tries to move in the correct direction. The only factor by which it may fail to converge is if the step size is too big. In this case the function value increases from iteration to iteration.

k	x_k	x_k	x_k
0	1	0.1	0.01
1	-5	0.13	0.0193
2		0.1417	0.03599257
3		0.14284777	0.062916884
4		0.142857142	0.098124028
5		0.142857143	0.128849782
6			0.1414837
7			0.142843938
8			0.142857142
9			0.142857143
10			0.142857143

Table 26.1: Newton Method Iterations for different Starting Points

It is quite evident that Newton's method will converge only when started near the solution. We need a modification that makes its globally convergent. A closer look into the method shows that problem doesn't lie with the direction of step but it's the step size that creates a problem. A step-size too large (might miss global optima) or even too small (method could be too slow) is a problem; we need to select a perfect step size for a given problem.

To overcome this problem at each iteration first take a full Newton step and evaluate the function at that point. If the function value $g(x^k + v^k)$ if higher than $g(x^k)$ it's rejected and $x^k + \frac{1}{2}v^k$ is tried, If the function is still higher than $g(x^k)$ than $x^k + \frac{1}{4}v^k$ is tried and so on un til a value of t is found with $x^k + tv^k < g(x^k)$

Newton's method with backtracking is the solution to above problem. The idea is to use the direction of the chosen step, but to control the length. Following algorithm is used for same.

The purpose of backtracking is to avoid a situation in which, the function values increase from iteration to iteration and never converges. Analysis shows that there is actually nothing wrong with the direction of the Newton step, since it always points in the direction of decreasing g. The problem is that we step too far in that direction. So the remedy is quite obvious.

At each iteration, we first attempt the step 4 of previous algorithm x ($x^{k+1} = x^k + v^k$), and evaluate g at that point. If the function value $g(x^k + v^k)$ is higher than $g(x^k)$, we reject the update, and try $x^{k+1} = x^k + (1/2)v^k$, instead. If the function value is still higher than g(x), we try $x^{k+1} = x^k + (1/4)v^k$, and so on, until a value of t is found with $g(x^k + tv^k) < g(x^k)$. We then take $x^{k+1} = x^k + tvk$.

In practice, the backtracking idea is often implemented as shown in the following algorithm: **Algorithm**
given initial x, tolerance $\epsilon > 0$, parameter $\alpha \in (0, 1/2)$.

repeat

1. Evaluate $\bigtriangledown g(x)$ and $\bigtriangledown^2 g(x)$.

2. If $||\bigtriangledown g(x)|| < \epsilon$, then return x

3. Solve $\bigtriangledown^2 g(x)v = - \bigtriangledown g(x)$

4. t :=1

while $g(x + tv) > g(x) + \alpha t \bigtriangledown g(x)^T v$, t := t/2.

5. $x = x + v$

untill a limit on the number of iterations is exceeded

The parameter α is usually chosen quite small ($e.g, \alpha = 0.01$)

Figure 26.2 shows the iterations in Newton's method with backtracking, applied to the previous example, starting from $x(0) = 4$. As expected the convergence problem has been resolved. From the plot of the step sizes we note that the method accepts the full Newton step ($t = 1$) after a few iterations. This means that near the solution the algorithm works like the pure Newton method, which ensures fast (quadratic) convergence.

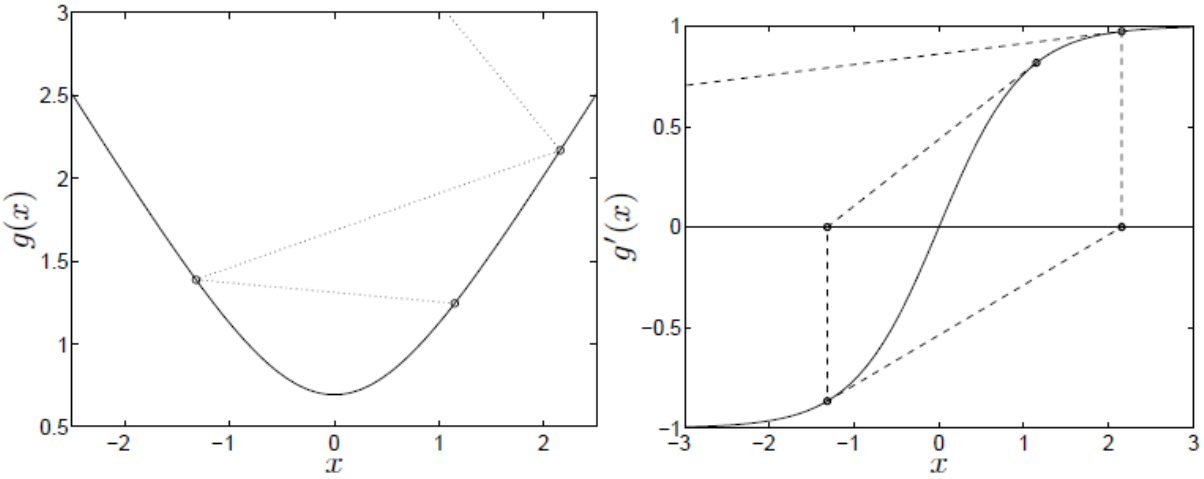


Figure 26.1: The solid line in the left figure is $g(x) = \log(e^x + e^x)$. The circles indicate the function values at the successive iterates in Newton's method, starting at $x(0) = 1.15$. The solid line in the right figure is the derivative $g(x)$. The dashed lines in the right-hand figure illustrate the first interpretation of Newton's method.

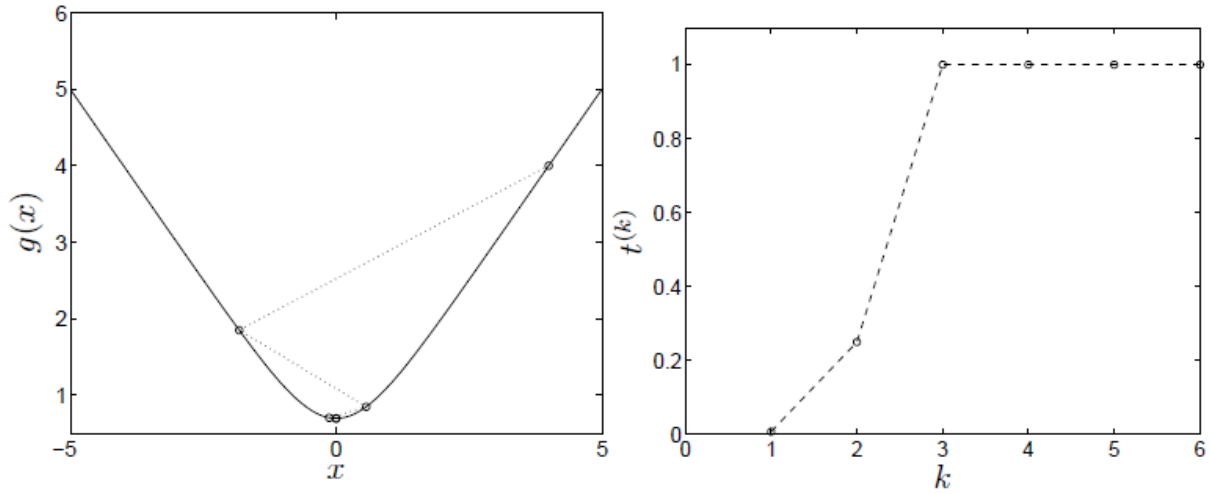


Figure 26.2: Newton

26.4 Example Problems

Exercise. A loan of A dollars is repaid by making 'n' equal monthly payments of 'M' dollars, starting a month after the loan is made. It can be shown that if the monthly interest rate is r, then

$Ar = M(1 - \frac{1}{(1+r)^n})$

A car loan of \$10000 was repaid in 60 monthly payments of \$250. Use the Newton Method to find the monthly interest rate correct to 4 significant figures.

Even quite commonplace money calculations involve equations that cannot be solved by 'exact' formulas. Let r be the interest rate. Then

$10000r = 250(1 - \frac{1}{(1+r)^{60}})$
$f(r) = 40r + \frac{1}{(1+r)^{60}} - 1$
$\Rightarrow f'(r) = 40 - \frac{60}{(1+r)^{61}}$

Using Newton's Method:

$r_(n+1) = r_n - \frac{40r_n + \frac{1}{(1+r_n)^{60}} - 1}{40 - \frac{60}{(1+r_n)^{61}}}$

If the interest rate were 2.5% a month, the monthly interest on $10,000 would be $250, and so with monthly payments of $250 we would never pay off the loan. So the monthly interest rate must have been substantially under 2.5%. A bit of trying suggests taking $r_0 = 0.015$. We then find that $r_1 = 0.014411839, r_2 = 0.014394797 and r_3 = 0.01439477$. This suggests that to four significant figures the monthly interest rate is 1.439%.

Exercise. Derive Newton equation for unconstrained minimization problem $min(\frac{1}{2}x^T x + \log \sum_{i=1}^{n} \exp(a_i^T x + b_i))$

Give an efficient method for solving mXn Newton system assuming matrix $A \varepsilon \Re^{mXn}$ (with rows a_i^T) is dense with $m \ll n$. Give an approximate FLOP count of your method.

Sol. 2:

$f(x) = \frac{1}{2}x^T x + \log \sum_{i=1}^{n} \exp(a_i^T x + b_i)$

$$\nabla f(x) = x + \frac{1}{\sum_{i=1}^{n} \exp(a_i^T x + b_i)} \frac{\partial \sum_{i=1}^{n} \exp(a_i^T x + b_i)}{\partial x} = x + \frac{\sum_{i=1}^{n} a_i^T \exp(a_i^T x + b_i)}{\sum_{i=1}^{n} \exp(a_i^T x + b_i)}$$

$$\nabla^2 f(x) = 1 + \frac{(\sum_{i=1}^{n} \exp(a_i^T x + b_i))(\sum_{i=1}^{n} (a_i^T a_i) \exp(a_i^T x + b_i)) - (\sum_{i=1}^{n} a_i^T \exp(a_i^T x + b_i))(\sum_{i=1}^{n} (a_i^T) \exp(a_i^T x + b_i))}{(\sum_{i=1}^{n} \exp(a_i^T x + b_i))^2}$$

We now solve for $v = -\nabla f^2(x_k)^{-1} \nabla f(x_k)$

Since $\nabla f^2(x_k)$ is is positive definite, we can use the Cholesky/QR factorization.

Cholesky: $nm2 + \frac{1}{2}m^3(cholesky) + 2m^2(bakward/forwardsubstitution)) + 2mn(matrixvectorproduct) \approx nm^2 + \frac{1}{3}m^3$

QR Decomposition: $2nm^2(QR) + m^2(forwardsubstituion) + 2mn(matrixproduct) \approx 2nm^2$

The QR method is slower than Cholesky (by a factor of about two if n m), but it is more accurate. It is the preferred method if n and m are not too large. For very large sparse problems, the Cholesky factorization is useful.

Exercise. Solve the unconstrained minimization problem $min(\sum_{i=1}^{m} \log \exp(a_i^T x - b_i) + \exp(-a_i^T x + b_i))$.A is m $\times$ n-matrix and b is m dimension vector.

Solution Let, $g(x) = \sum_{i=1}^{m} \log \exp(a_i^T x - b_i) + \exp(-a_i^T x + b_i)$

We express g as $g(x) = f(Ax - b)$,i.e.$f(y) = \sum_{i=1}^{m} \log \exp(y_i) + \exp(-y_i)$

$\nabla f(y) = \frac{\exp(y_i) - \exp(-y_i)}{\exp(y_i) + \exp(-y_i)}$

$\nabla^2 f(y) = \frac{4}{(\exp(y_i)+\exp(-y_i))^2} for i = j, 0 for i \neq j$

Once we have the gradient and Hessian, the implementation of Newton's method is straightforward.Using Hessian and gradient of f compute hessian and gradiant of g as shown below:

$\nabla g(x) = A^T \nabla f(y)$

$\nabla^2 g(x) = A^T \nabla^2 f(y) A^T$

We start with $x_0 = (1, 1, 1...1)$ and set $\alpha = 0.01$ terminate if $\|\nabla f(y)\| \leq 10^{-5}$

Note: The number of iterations in iterative algorithm depends on the problem parameters and on the starting point hence its efficiency isn't expressed by giving its flop count, rather by giving upper bounds on the number of iterations to reach a given accuracy.

26.5 Additional Problems

1. Let f(x) be a strictly convex twice-continuously differentiable and x* be a point for which $\Delta f(x^*) = 0$. Suppose that H(x) satisfies the following conditions:
   (a) there exists a scalar $h > 0$ for which $\|H(x^*)^{-1}\| \leq \frac{1}{h}$
   (b) there exists scalars $\beta > 0$ and $L > 0$ for which $\|H(x) - H(x^*)\| \leq L\|x - x^*\|$

2. Let x satisfy $\|x - x^*\| < \gamma := min\{\beta, \frac{2h}{3L}\}$ and $x_N := x - H(x)^{-1} \bigtriangledown f(x)$, then prove following:
   (i)$\|x_N - x^*\| \leq \|x - x^*\|^2 \frac{L}{2(h - L\|x - x^*\|)}$
   (ii)$\|x_N - x^*\| < \|x - x^*\| < \gamma$
   (iii)$\|x_N - x^*\| \leq \|x - x^*\|^2 \frac{3L}{2h}$
   Hint: Ref. Quadratic Convergence Theorem of Newton's Method

3. Show that the eigen value-vector problem can be casted as *root-finding* problem i.e.;
   $(A - \lambda I)v = 0; v^T v = 1$
   Derive the Newton's iteration to solve the above problem.

4. Use Newton's method to find all the solution of the two equations
   $$x_1^2 + x_2^2 = 16$$
   $$(x_1 - 2)^2 + (x_2 - 3)^2 = 25$$

5. Derive gradient and Hessian of the function g(x) = $\sqrt{||Cx||^2 + 1}$, where C is a left-invertible $m \times n$ matrix.
   Hint: Use the result of Homework Problem1 and the expression $\bigtriangledown^2 g(x) = C^T \bigtriangledown^2 h(Cx + d)C$ for the Hessian of the function $g(x) = h(Cx + d)$

Chapter 27

Claoser Look at Gradient Descent Optimization

27.1 Introduction

Chapter 28

Optimization in Deep Neural Networks

Chapter 29

Optimization in Support Vector Machines

Chapter 30

Regression and Regularization

30.1 Introduction

Chapter 31

Sparse Coding and Dictionary Learning

31.1 Representation and Coding

For many problems in engineering, building a numerical representation has been the first step. Often this representation is a vector of n elements. It could be some measurements of the phyisical phenomena. Such representations are understood as an element in a vector space. Often there is also a set of basis functions, such as Fourier basis, for this representation

In this lecture, we are interested in an over-complete basis. This leads to representations that can be "sparse".

There are many aspects for looking for a right representation. It could be the lenth of the representation (number of elements in the vector.) It could also be the compressibility of the representation. It could also be the number of bits required to store. Beyond these computational requirements, one is also (obviously) interested in building a representation that is useful for solving the problem. One useful representaion scheme is based on sparse linear combination of some basis functions. Sparse coding, that is modelling data vectors as sparse linear combinations of basis elements—is widely used in machine learning, neuroscience, signal processing, and statistics. This focuses on learning the basis set, also called dictionary, to adapt it to specific data, an approach that has recently proven to be very effective for signal reconstruction and classification in the audio and image processing domains.

Sparseness is one of the reasons for the extensive use of popular signal transforms such as the Discrete Fourier Transform(DFT), the wavelet transform(WT) and the Singular Value Decomposition(SVD). These transforms often reveal certain structures of a signal and are used to represent these structures in a compact and sparse representation. Sparse representations have therefore increasingly become recognized as providing extremely high performance for applications as diverse as: noise reduction, compression, feature extraction, pattern classification and blind source separation. Sparse approximation techniques also build the foundations of wavelet denoising and methods in pattern classification, such as in the Support Vector Machine(SVM).

1. Sparse modeling calls for constructing efficient representations of data as a (often linear) combination of a few typical patterns (atoms) learned from the data itself. Significant contributions to the theory and practice of learning such collections of atoms (usually called dictionaries or codebooks), and of representing the actual data in terms of them, leading to state-of-the-art results in many signal and image processing and analysis tasks. The first critical component of this topic is how to sparsely encode a signal given the dictionary.
2. The actual dictionary plays a critical role, and it has been shown once and again that learned dictionaries significantly outperforms off-the-shelf ones such as wavelets.
3. There are numerous applications where the dictionary is not only adapted to reconstruct the data, but also learned for a specific task, such as classification, edge detection and compressed sensing .

There are many formulations and algorithms for sparse coding in the literature. Let yR^n be a data point in a data set Y .

Assume that y can be expressed as

$$y = Dx \tag{31.1}$$

where:

- x is a $m \times 1$ column vector
- D is a $m \times p$ matrix
- a is a $k \times 1$ column vector
- $k \ll m$

We discussed how we can build a model based on large number of observations. Using sparse coding we try to represent the new sample in terms of a sparse set of input sample. The large set of sample that is available to us is called dictionary. Each sample i.e each column of the matrix is known as atom. The dictionary is over complete and the rows and columns are not required to be linearly independent. There are a lot of advantages working with sparse vectors. For example calculations involving multiplying a vector by a matrix take less time to compute in general if the vector is sparse. Also sparse vectors require less space when being stored on a computer as only the position and value of the entries need to be recorded.

Problem 1 - Dictionary Learning

Given y find(learn) D.

This problem can be simply stated as:

$$\text{Given } x \text{ find(learn) } D.$$

This is generally a harder problem comparedto the one given below and the major focus in these notes will be on the second problem that is discussed hereafter.

Problem 2 - Sparse Coding Sparse coding that is, modelling data vectors as sparse linear combinations of basis elements—is widely used in machine learning, neuroscience, signal processing, and statistics. This focuses on learning the basis set, also called dictionary, to adapt it to specific data, an approach that has recently proven to be very effective for signal reconstruction and classification in the audio and image processing domains.

31.2 Problem of Sparse Coding

Sparse Coding: Sparse coding is a class of unsupervised methods for learning sets of over-complete bases to represent data efficiently. Formally , The aim of sparse coding is to find a set of basis vectors ϕ_i such that we can represent an input vector x as a linear combination of these basis vectors:

$$x = \sum_{i=1}^{k} (a_i \phi_i)$$

The advantage of having an over-complete basis is that our basis vectors are better able to capture structures and patterns inherent in the input data.

Sparse Representation of a Signal The problem of finding the sparse representation of a signal in a given overcomplete dictionary can be formulated as follows. Given a $N \times M$ matrix A containing the elements of an overcomplete dictionary in its columns, with $M > N$ and a signal $y \in R^N$, the problem of sparse representation is to find an $M \times 1$ coefficient vector x, such that $y = Ax$ and $||x||_0$ is minimized, i.e.

$$x = \min_{x'} ||x'||_0$$

s.t. $y = Ax$

In general the above problem is computationally intractable. Therefore we go for sparse approximation.

Sparse Coding

Traditionally a sample was transformed into a sparse vector based on particular basis. Representing a sample in a particular basis involves finding a unique set, for that sample,of expansion coefficients in that basis. There are a lot of advantages working with sparse vectors. For example calculations involving multiplying a vector by a matrix take less time to compute in general if the vector is sparse. Also sparse vectors require less space when being stored on a computer as only the position and value of the entries need to be recorded.

The main disadvantage of using orthogonal basis to represent a new sample, is that specific basis works for specific type of samples and may not work with other types. For example, smooth continuous signals are sparsely represented in a Fourier basis, while impulses are not. On the other hand, a smooth signal with isolated discontinuities is sparsely represented in a wavelet basis, while a wavelet basis is not efficient at representing a signal whose Fourier transform has narrow high frequency support.

Real world observations often contain features that prohibit sparse representation in any single basis. To ensure that we are able represent every vector in the space, the dictionary of vectors we need to choose from, must span the space. How ever, because the set is not limited to a single basis, the dictionary is not linearly independent.

Because the vectors in the dictionary are not a linearly independent set, the sample representation in the dictionary may not unique. However, by creating a redundant dictionary (over defined dictionary), we can expand our sample in a set of vectors that is a union of several bases. You are free to create a dictionary consisting of the union of several

bases. For example to represent an arbitrary document our dictionary may be defined as a union of various different type of document bases like sports document, musical document, culinary document etc

$$\begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n-1} & d_{1n} \\ \vdots & \ddots & \vdots & & \\ d_{m1} & d_{m2} & \cdots & d_{mn-1} & x_{mn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n - 1 \\ x_n \end{bmatrix}$$

Note on Solving Hard Problems

1. Sparse Coding Given a dictionary D, and a new observation y find a sparse representation of y as x Minimize.$||x||_0$ such that y= Dx or Minimize $||x||_0$ such that $||y - Dx|| <= \epsilon$
2. Dictionary learning Given a lot of samples $y_1 ... y_n$ find the dictionary D

Sparse Coding requires to minimize the L_0 norm which is a hard problem so there are two practical ways to solve the problem as shown in the image.

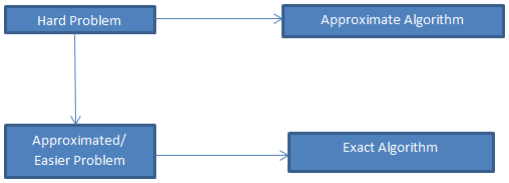


Figure 31.1: Practical approaches to solve a hard problem

The eccentricities of sparse coding can be enumerated as follows:

1. Given x and D , the task is to find a such that it best approximates the signal or data using a highly sparse vector.
2. Consider a linear system of equations $x = Da$, where D is an underdetermined $m \times p$ $(m \ll n)$ matrix and $x \in \mathbb{R}^m$,$a \in \mathbb{R}^p$, called as the dictionary or the design matrix, is given. The problem is estimation of the signal a, subject to it being sparse. Sparse decomposition helps in such a way that even though the observed values are in high-dimensional($\mathbb{R}^m$) space, the actual signal is organized in some lower-dimensional subspace($\mathbb{R}^k$), $(k \ll m)$.
3. It is evident that only a few components of a are non-zero and the rest are zero as it is a sparse vector. This implies that x can be decomposed as a linear combination of only a few $m \times 1$ vectors in D, these vectors are known as atoms. D itself is over-complete $(m \ll p)$. Such vectors(atoms) are called as the basis of x. Here though, unlike other dimensionality reducing decomposition techniques such as Principal Component Analysis, the basis vectors are not required to be mutually orthogonal.

31.2.1 The Sparse Coding Problem Formulated

The sparse decomposition problem is represented as:

$$\min_{\alpha \in \mathbb{R}^p} ||a||_0 \mid x = Da \tag{31.2}$$

where $||a||_0$ is a pseudo-norm, l_0, which counts the number of non-zero components of a. A convex relaxation of the problem can instead be obtained by taking the l_1 norm instead of the l_0 norm, where $||a||_1 = \sum_{i=1}^{p} |a_i|$. The l_1 norm induces sparsity under certain conditions.

The need to learn dictionary(D)

The linear decomposition of a signal using a few atoms of a learned dictionary instead of a predefined one has led to the better results for numerous tasks .For example low-level image processing tasks such as denoising as well as higher-level tasks such as classification showing that sparse learned models are well adapted to natural data. Dictionary you are trying to learn should be specific to the subject .

31.2.2 What are sparse representations/approximations good for?

Sparseness is one of the reasons for the extensive use of popular transforms such as the Discrete Fourier Transform, the wavelet transform and the Singular Value Decomposition. The aim of these transforms is often to reveal certain structures of a signal and to represent these structures in a compact and sparse representation. Sparse representations have therefore increasingly become recognized as providing extremely high performance for applications as diverse as: noise reduction, compression, feature extraction, pattern classification and blind source separation. Sparse representation ideas also build the foundations of wavelet denoising and methods in pattern classification, such as in the Support Vector Machine and the Relevance Vector Machine, where sparsity can be directly related to learnability of an estimator.

Sparse signal representations allow the salient information within a signal to be conveyed with only a few elementary components, called atoms. The goal of sparse coding is to represent input vectors approximately as a weighted linear combination of a small number of (unknown) "basis vectors". These basis vectors thus capture high-level patterns in the input data. Sparse coding is, modelling data vectors as sparse linear combinations of basis elements, it is widely used in machine learning, neuroscience, signal processing, and statistics. It is proven to be very effective for signal reconstruction and classification in the audio and image processing domains.

31.2.3 Application

1. When a sparse coding algorithm is applied to natural images, the learned bases resemble the receptive fields of neurons in the visual cortex.
2. Sparse coding produces localized bases when applied to other natural stimuli such as speech and video.
3. There are many applications of sparse coding in Seismic Imaging linear regression and Transform Coding.

31.2.4 Important aspects related to Sparse Coding

1. **Signal and image processing:** Restoration, reconstruction
   (a) Image Denoising
   (b) Inpainting
   (c) Demosaicking
   (d) Video Processing
   (e) Other Applications
2. **Sparse Linear Models and Dictionary Learning:**
   (a) The machine learning point of view
   (b) Why does the l_1-norm induce sparsity?
   (c) Dictionary Learning and Matrix Factorization
   (d) Group Sparsity
   (e) Structure Sparsity
3. **Computer Vision Applications :**
   (a) Learning codebooks for image classification
   (b) Modelling the local appearance of image patches
   (c) Background subtraction with structured sparsity
4. **Optimization for sparse methods**
   (a) Greedy algorithms
   (b) l_1 approximations
   (c) Online dictionary learning .

31.3 BP, MP and OMP

The sparse decomposition problem is represented as:

$$\min_{\alpha \in \mathbb{R}^p} ||x||_0 \mid y = Dx \tag{31.3}$$

where $||x||_0$ is a pseudo-norm, l_0, which counts the number of non-zero components of x. A convex relaxation of the problem can instead be obtained by taking the l_1 norm instead of the l_0 norm, where $||x||_1 = \sum_{i=1}^{p} |x_i|$. The l_1 norm induces sparsity under certain conditions.

31.3.1 Overview

Basis Pursuit The idea of Basis Pursuit is to replace the difficult sparse problem with an easier optimization problem. The difficulty with the above problem is the L0 norm. Basis Pursuit replaces the L_0 norm with the L_1 to make the problem easier to work with. Basis Pursuit: min $||x||_1$ subject to Ax = b

Matching pursuit (MP) A greedy iterative algorithm for approximately solving the original l_0 pseudo-norm problem. Matching pursuit works by finding a basis vector in D that maximizes the correlation with the residual (initialized to y), and then recomputing the residual and coefficients by projecting the residual on all atoms in the dictionary using existing coefficients.

Matching pursuit (MP) Matching pursuit is a greedy iterative algorithm for approximately solving the original l_0 pseudo-norm problem. Matching pursuit works by finding a basis vector in D that maximizes the correlation with the residual (initialized to x), and then recomputing the residual and coefficients by projecting the residual on all atoms in the dictionary using existing coefficients. It can be seen as an expectation maximization[**?**] problem.

Orthogonal matching pursuit(OMP) It is similar to Matching Pursuit[**?**], except that an atom once picked, cannot be picked again. The algorithm maintains an active set of atoms already picked, and adds a new atom at each iteration. The residual is projected on to a linear combination of all atoms in the active set, so that an orthogonal updated residual is obtained. Both Matching Pursuit and Orthogonal Matching Pursuit use the l_2 norm.

31.3.2 Basic Matching Pursuit

Matching pursuit is a greedy algorithm that computes the best nonlinear approximation to a sample in a complete, redundant dictionary. Matching pursuit builds a sequence of sparse approximations to the signal step-wise. Let $\phi = \varphi_k$ denote a dictionary of unit-norm atoms. Let f be your signal.

1. Start by defining $R^0 f = f$
2. Begin the matching pursuit by selecting the atom from the dictionary that maximizes the absolute value of the inner product w ith $R^0 f = f$. Denote that atom by φ_p.
3. Form the residual $R^1 f$ by subtracting the orthogonal projection of $R^0 f$ onto the space spanned by φ_p.

$$R^1 f = R^0 f - < R^0 f, \phi_p > \phi_p \tag{31.4}$$

4. Iterate by repeating steps 2 and 3 on the residual.

$$R^(m+1) f = R^m f - < R^m f, \phi_k > \phi_k \tag{31.5}$$

5. Stop the algorithm w hen you reach some specified stopping criterion.

In nonorthogonal (or basic) matching pursuit, the dictionary atoms are not mutually orthogonal vectors. Therefore, subtracting subsequent residuals from the previous one can reintroduce components that are not orthogonal to the span of the previously included atoms. The next algorithm Orthogonal Matching Pursuit handles this problem.

Drawbacks

Matching pursuit has a drawback that an atom can be selected multiple times, orthogonal matching pursuit[**?**] gets rid of this drawback in its implementation.

Algorithm : Matching Pursuit

Objective Function:

$$\min |||x - Da||_2^2 \quad | \quad ||x||_0 < N \tag{31.6}$$

Steps:

1. $x_0 = b$, $t = 0$, $v_0 = \phi$
2. Let $v_t = i$ is the vector closest/most correalted to vector r_t, pick v_i
   $v_t = v_{t+1} \cup v_i$
3. Solve $\min ||b - \sum_{i=1}^{t} \alpha_k \alpha_{v_i}||$
4. $t + 1 \leftarrow t$; Go to step 1.

31.3.3 Orthogonal Matching Pursuit

In orthogonal matching pursuit (OMP), the residual is always orthogonal to the atoms already selected. This means that the same atom can never be selected twice and results in convergence for a n-dimensional vector after at most n steps.

The OMP algorithm is:

1. Denote your signal by f. Initialize the residual $R^0 f = f$
2. Select the atom that maximizes the absolute value of the inner product w ith $R^0 f = f$. Denote that atom by ϕ_p.
3. Form a matrix, Φ, with previously selected atoms as the columns. Define the orthogonal projection operator onto the span of the columns of Φ.
   $$P = \Phi(\Phi * \Phi)^{-}1\Phi^* \tag{31.7}$$
4. Apply the orthogonal projection operator to the residual.
5. Update the residual.
   $$R^{m+1} f = (I - P)R^m f \tag{31.8}$$

I is the identity matrix.

Orthogonal matching pursuit ensures that the previously selected atoms are not chosen again in subsequent steps.

The other approach is to solve an approximated problem in an exact manner. Taking example of the sparse coding problem, we have seen that it is hard problem , but the problem can be approximated to an easier problem that can be solved exactly. In the easier version we try to find the L1 norm instead of L0 norm. This algorithm is known as Basis pursuit.

Algorithm : Orthogonal Matching Persuit

Objective function

$$\min |||y - Dx||_2^2 \quad | \quad ||x||_0 < L \tag{31.9}$$

Steps:

1. $\lambda = \phi$
2. for iter = 1,2 .....L do

3. Select the atom which most reduces the objective
4. Let $v_t = i$ is the vector closest/most correalted to vector r_t, pick v_i
   $v_t = v_{t-1} \cup v_i$
5. Update the active set.
6. Update the residual .
7. Update the coefficients.

It is similar to Matching Pursuit, except that an atom once picked, cannot be picked again. The algorithm maintains an active set of atoms already picked, and adds a new atom at each iteration. The residual is projected on to a linear combination of all atoms in the active set, so that an orthogonal updated residual is obtained. Both Matching Pursuit and Orthogonal Matching Pursuit use the norm.Contrary to MP, an atom can only be selected one time with OMP. It is,however, more difficult to implement efficiently. The keys for a goodimplementation in the case of a large number of signals are

Sparse Approximations: This problem is NP-hard in general. Therefore various relaxed sparsity measures have been presented to make the problem tractable. Two commonly used methods used to solve sparse approximation problems are

1. Basic pursuit

2. Orthogonal matching pursuit

Basic Pursuit: The idea of Basis Pursuit is to replace the difficult sparse problem with an easier optimization problem. Below the formal definition of the sparse problem is given :

$$Sparse\, problem: \ min||x||_0$$

subject to $Ax = b$, where $||\ ||_0$ is count of the number of non-zero coefficients. The difficulty with the above problem is the L_0 norm. Basis Pursuit replaces the L_0 norm with the L_1 to make the problem easier to work with.

$$Basis\, Pursuit: \ min||x||_1$$

subject to $Ax = b$, where $||\ ||_1$ is used for measuring the reconstruction error.

BP can be solved using linear programming, Projected gradient or interior-point methods.

Orthogonal Matching Pursuit: Orthogonal matching pursuit (OMP) constructs an approximation by going through an iteration process. At each iteration the locally optimum solution is calculated. This is done by finding the column vector in A which most closely resembles a residual vector r. OMP is based on a variation of an earlier algorithm called Matching Pursuit (MP). MP simply removes the selected column vector from the residual vector at each iteration.

$$r_{t=r_{t-1}-} < a_{OP}, r_{t-1} > r_{t-1}$$

Where a OP is the column vector in A which most closely resembles r_{t-1} . OMP uses a least-squares step at each iteration to update the residual vector in order to improve the approximation.

Algorithm for OMP:

Input : signal b, matrix A and stopping criterion

Output : Approximation vector c.

Algorithm:

1. Start by setting the residual $r_0 = b$, the time $t = 0$ and index set $V_0 = \emptyset$
2. Let $v_t = i$, where a_i gives the solution of $max < rt, ak >$ where a_k are the row vectors of A.
3. Update the set V_t with $v_t : V_t = V_{t-1} \cup \{v_t\}$
4. Solve the least-squares problem

$$\min_{c \in C^{V_t}} ||b - \sum_{j=1}^{t} c(v_j) a_{v_j}||_2$$

5. Calculate the new residual using c

$$r_t = r_{t-1} - \sum_{j=1}^{t} c(v_j) a_{v_j}$$

6. Set $t \leftarrow t + 1$
7. Check stoping criterion, if the criterion has not been satisfied then return to step 2.

31.3.4 Discussions

Uniqueness of Sparse approximation A sparse represntation need not necessarily be unique. For uniqueness it has to satisfy a certain condition. A sparse representation x of b is unique if

$$x_0 < Spark(A)/2$$

where,

$Spark(A)$ is defined the size of the smallest set of linearly dependant vectors of A.

Sparse coding using Fixed Dictionary To solve an unconstrained optimization problem, Let $y \in R^d$ and $x \in R^N$ (where d < N) be the input and the coefficient vectors and let the matrix $D \in R^{dxN}$ be the dictionary .

$$\min_x \phi(x); \phi(x) = ||y - Dx||^2 + \lambda ||x||_0$$

where

$||x||_0$ is the sparsity measure (which counts the number of non-zero coefficients)

λ is constant multiplier

Upon applying basis pursuit on $||x||_0$ the problem becomes an L_1 regularized linear least-squares problem. A number of recent methods for solving this type of problems are based on coordinate descent with soft thresholding. When the columns of the dictionary have low correlation, these simple methods have proven to be very efficient.

31.4 Dictionary Learning

31.4.1 Why to learn Dictionary?

The linear decomposition of a signal using a few atoms of a learned dictionary instead of a predefined one has led to the better results for numerous tasks .For example low-level image processing tasks such as denoising as well as higher-level tasks such as classification showing that sparse learned models are well adapted to natural data. Dictionary you are trying to learn should be specific to the subject .

Dictionary Learning However, the columns of learned dictionaries are in general highly correlated and thus more preferably used.

Let $D = [d1, ..., dp] \in R^{mxp}$ be a set of normalized basis vectors and . We call it dictionary and let Y be a set of n-dimensional N input signals. D is adapted to y if it can represent it with a few basis vectors, that is, there exists a sparse vector α in R^p . We call α the sparse code

$$(y) \approx (d1|d2|...|dp)(\frac{\frac{\alpha[1]}{\alpha[2]}}{\frac{...}{\alpha[3]}})$$

Learning a reconstructive dictionary with K items for sparse representation of Y can be accomplished by solving the following problem:

$$< D, X >= arg\,min||Y - DX||_2^2$$

s.t. $\forall i, ||x||_0 \leq T$

where $D = [d1...dK] \in R^{nxK}(K > n)$, making the dictionary over-complete) is the learned dictionary, $X = [x1, ..., xN] \in R^{KxN}$ are the sparse codes of input signals Y , and T is a sparsity constraint factor (each signal has fewer than T items in its decomposition).

The construction of D is achieved by minimizing the reconstruction error and satisfying the sparsity constraints. The K-SVD algorithm is an iterative approach to minimize the energy and learns a reconstructive dictionary for sparse representations of signals. It is highly efficient and works well in applications such as image restoration and compression.The term $||Y - DX||_2^2$ denotes the reconstruction error.

31.4.2 Dictionary Learning

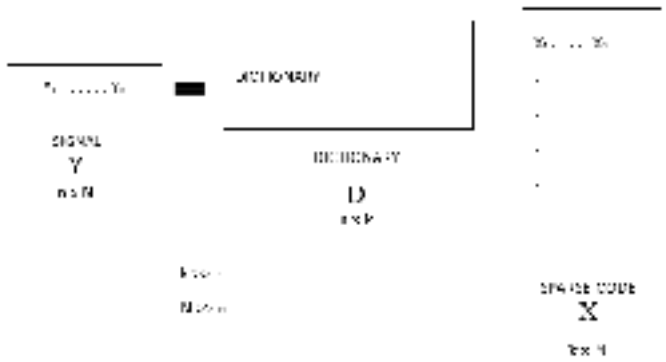
In Dictionary Learning, we consider the problem of finding a few representatives for a dataset, i.e., a subset of data points that efficiently describes the entire dataset. The problem is computationally costly.

$min \parallel x \parallel_0$

s.t $\parallel Y - DX \parallel_2 < \varepsilon$

Dictionary Learning : Given $Y_1....Y_N$, Find D

There will be N vectors for each



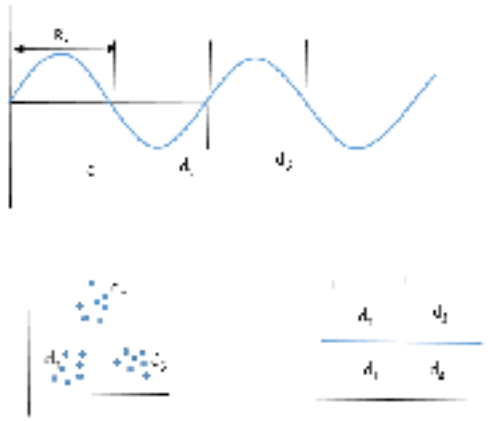
Given a set Y=$(y_j)_{j=1}^m \in \mathbb{R}^{nXm}$of m signals $y_j \in R^m$, dictionary learning aims at finding the best dictionary D=$(d_i)_{i=1}^p$ of p atoms $d_i \in R^n$ to sparse code all the data.

If Dictionary can be adopted to the domain then it can be much faster.

For example:

Let us consider Dictionary for sound signal

For sound signal there are some salient features, that can be used to reconstruct the signal using only few elementry components, called 'atoms'. For signal to be encoded first its Vector Quantization is done and then a Dictionary can be built using its salient features which are domain specific. Successful application of a sparse decomposition depends on the dictionary used, and whether it matches the signal features.



In some case it is possible that original signal cannot be reconstructed using Dictionary, this error can be ignored to some threshold value.

31.4.3 Dictionary Learning Methods:

Dictionary learning, one often starts with some initial dictionary and finds sparse approximations of the set of training signals while keeping the dictionary fixed. This is followed by a second step in which the sparse coefficients are kept fixed and the dictionary is optimized. This algorithm runs for a specific number of alternating optimizations or until a specific approximation error is reached. Most of these algorithms have been derived for dictionary learning in a noisy sparse approximation setting.

31.4.4 k-means:[Randomly Partition Y to k subsets]

k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining.

k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

$min_{X,D} \| Y - DX \|_2$ s.t. $\| x_i \|_0 \leq T_0 \forall$i , if i =1 -->K - means

X is a k vector

D is a full rank matrix, D rank =N

31.4.5 Dictionary Learning via k-means

Data points Y= { $Y_1 \cdots Y_N$} will be mapped to $C_1 \cdots C_k$ clusters

C is given dictionary

C=[$C_1 \cdots C_k$]

$Y_i = Cx_i$; $x_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ s.t. $x_i = e_j$

s.t. $\| y_i - Ce_j \| \leq \|Y_i - Ce_k \| \forall k \neq j$

$min \| Y - CX \|$s.t. $x_i = e_k$for some 'k'

$$\|Y - DX\| = \| Y - \sum_{j=1}^{k} d_j x_T^j \|$$

Algorithm:

1. Find C_i, i=$1 \cdots k$ as mean of member in subset 'i'.
2. Repartition, such that every signal is assign to the nearest subset.

$$E = \sum_{i=1}^{N} e_i^2 = \| Y - CX \|^2$$

31.5 K-SVD

K-SVD[**?**] is a dictionary learning algorithm for creating a dictionary for sparse representations, via a singular value decomposition approach. K-SVD is a generalization of the k-means clustering method, and it works by iteratively alternating between sparse coding the input data based on the current dictionary, and updating the atoms in the dictionary to better fit the data, just like in expectation maximization. K-SVD can be found widely in use in applications such as image processing, audio processing, biology, and document analysis.

Problem Description

Given an overcomplete dictionary matrix $D \in \mathbb{R}^{\aleph \times \mathbb{K}}$ that contains K signal-atoms for each columns, a signal $x \in \mathbb{R}^n$ can be represented as a linear combination of these atoms. To represent x, the sparse representation a should satisfy the exact condition $x = Da$, or approximate condition $X \approx DA$, or $|||x - Da||_p \leq \epsilon$. The vector $x \in \mathbb{R}^K$ contains the representation coefficients of the signal x. Typically, norm p is selected as $1, 2$ or ∞.

If $n < K$ and D is a full-rank matrix, an infinite number of solutions are available for the representation problem, Hence, constraints should be set on the solution. Also, to ensure sparsity, the solution with the fewest number of nonzero coefficients is preferred. Thus, the sparsity representation is the solution of either

$$(P_0) \quad \min_a ||a||_0 \quad | \quad x = Da \tag{31.10}$$

OR

$$(P_0, \epsilon) \quad \min_a ||a||_0 \quad | \quad ||x - Da||_2 \leq \epsilon \tag{31.11}$$

where the L_0 norm counts the nonzero entries of a vector.

Algorithm: K-SVD

K-SVD is a kind of generalization of K-means, as follows.
The k-means clustering can be also regarded as a method of sparse representation. That is, finding the best possible codebook to represent the data samples $\{x_i\}_{i=1}^M$ by nearest neighbor, by solving

$$\min_{D,A} \{||X - DA||_F^2\} \quad | \quad \forall i, a_i = e_k \exists \ k \tag{31.12}$$

or equivalently

$$\min_{D,A} \{||X - DA||_F^2\} \quad | \quad \forall i, ||a_i||_0 = 1 \tag{31.13}$$

The sparse representation term $a_i = e_k$(where e_k is a column vector vector from a $m \times m$ permutation matrix) enforces K-means algorithm to use only one atom(column) in dictionary D. To relax this constraint, target of the K-SVD algorithm is to represent signal as a linear combination of atoms in D. The K-SVD algorithm follows the construction flow of K-means algorithm. However, In contrary to K-means, in order to achieve linear combination of atoms in D, sparsity term of the constrain is relaxed so that nonzero entries of each column x_i can be more than 1, but less than a number T_0.

So, the objective function becomes

$$\min_{D,A} \{||X - DA||_F^2\} \quad | \quad \forall i, ||a_i||_0 \leq T_0 \tag{31.14}$$

or in another objective form

$$\min_{D,A} \sum_i ||a_i||_0 \quad | \quad \forall i, ||X - DA||_F^2 \tag{31.15}$$

In the K-SVD algorithm, the dictionary D is first set to be fixed and the best coefficient matrix X is calculated. As finding the truly optimal X is impossible, we use an approximation pursuit method. Any such algorithm as OMP, the orthogonal matching pursuit can be used for the calculation of the coefficients, as long as it can supply a solution with a fixed and predetermined number of nonzero entries T_0.

After the sparse coding task, the next job is to search for a better dictionary D. However, finding the whole dictionary all at a time is impossible, so the process then updates only one column of the dictionary D each time while we fix X. The update of the k^{th} column is done by rewriting the penalty term as

$$||X - DA||_F^2 = ||X - \sum_{j=1}^{k} d_j a_j^T||_F^2 = ||(X - \sum_{j \neq k} d_j x_j^T) - d_k x_k^T||_F^2 = ||E_k - d_k x_k^T||_F^2 \tag{31.16}$$

where x_k^T denotes the k-th column of X(the super-script denotes transpose for vector multiplication).

By decomposing the multiplication DX into sum of K rank 1 matrices, we can assume the other $K-1$ terms are fixed, and the k^{th} column remains unknown. After this step, we can solve the minimization problem by approximating the E_k term with a $rank-1$ matrix using singular value decomposition, then update d_k with it. However, the new solution of vector x_k^T is very likely to be filled, because the sparsity constraint is not made compulsory.

K-SVD

K-SVD is a dictionary learning algorithm for creating a dictionary for sparse representations, via a singular value decomposition approach. K-SVD is a generalization of the k-means clustering method, and it works by iteratively alternating between sparse coding the input data based on the current dictionary, and updating the atoms in the dictionary to better fit the data. K-SVD can be found widely in use in applications such as image processing, audio processing, biology, and document analysis.

Problem Description

Given an overcomplete dictionary matrix $D \in \mathbb{R}^{\kappa \times \mathbb{K}}$ that contains K signal-atoms for each columns, a signal $y \in \mathbb{R}^n$ can be represented as a linear combination of these atoms. To represent y, the sparse representation x should satisfy the exact condition $y = Dx$, or approximate condition $Y \approx DX$, or $|||y - Dx||_p \leq \epsilon$. The vector $y \in \mathbb{R}^K$ contains the representation coefficients of the signal y. Typically, norm p is selected as $1, 2$ or ∞.

If $n < K$ and D is a full-rank matrix, an infinite number of solutions are available for the representation problem, Hence, constraints should be set on the solution. Also, to ensure sparsity, the solution with the fewest number of nonzero coefficients is preferred. Thus, the sparsity representation is the solution of either

$$(P_0) \quad \min_x ||x||_0 \quad | \quad y = Dx \tag{31.17}$$

OR

$$(P_0, \epsilon) \quad \min_x ||x||_0 \quad | \quad ||y - Dx||_2 \leq \epsilon \tag{31.18}$$

where the L_0 norm counts the nonzero entries of a vector.

The Algorithm

K-SVD is a kind of generalization of K-means, as follows.
The k-means clustering can be also regarded as a method of sparse representation. That is, finding the best possible codebook to represent the data samples $\{y_i\}_{i=1}^M$ by nearest neighbor, by solving

$$\min_{D,X} \{||Y - DX||_F^2\} \quad | \quad \forall i, x_i = e_k \exists\, k \tag{31.19}$$

or can be written as

$$\min_{D,X} \{||Y - DX||_F^2\} \quad | \quad \forall i, ||x_i||_0 = 1 \tag{31.20}$$

The sparse representation term $x_i = e_k$ enforces K-means algorithm to use only one atom (column) in dictionary D. To relax this constraint, target of the K-SVD algorithm is to represent signal as a linear combination of atoms in D. The K-SVD algorithm follows the construction flow of K-means algorithm. However, In contrary to K-means, in order to achieve linear combination of atoms in D, sparsity term of the constrain is relaxed so that nonzero entries of each column x_i can be more than 1, but less than a number T_0.

So, the objective function becomes

$$\min_{D,X} \{||Y - DX||_F^2\} \quad | \quad \forall i, ||x_i||_0 \leq T_0 \tag{31.21}$$

or in another objective form

$$\min_{D,X} \sum_i ||x_i||_0 \quad | \quad \forall i, ||Y - DX||_F^2 \tag{31.22}$$

In the K-SVD algorithm, the dictionary D is first set to be fixed and the best coefficient matrix X is calculated. As finding the truly optimal X is impossible, we use an approximation pursuit method. Any such algorithm as OMP, the orthogonal matching pursuit in can be used for the calculation of the coefficients, as long as it can supply a solution with a fixed and predetermined number of nonzero entries T_0.

After the sparse coding task, the next is to search for a better dictionary D. However, finding the whole dictionary all at a time is impossible, so the process then updates only one column of the dictionary D each time while we fix X. The update of the k^{th} column is done by rewriting the penalty term as

$$||Y - DX||_F^2 = |Y - \sum_{j=1}^{k} d_j x_T^j|_F^2 = |(Y - \sum_{j \neq k} d_j x_T^j) - d_k x_T^k|_F^2 = ||E_k - d_k x_T^k||_F^2 \tag{31.23}$$

where x_T^k denotes the k-th row of X.

By decomposing the multiplication DX into sum of K rank 1 matrices, we can assume the other $K - 1$ terms are assumed fixed, and the k^{th} column remains unknown. After this step, we can solve the minimization problem by approximating the E_k term with a $rank - 1$ matrix using singular value decomposition, then update d_k with it. However, the new solution of vector x_T^k is very likely to be filled, because the sparsity constraint is not enforced.

31.5.1 K- SVD

K-SVD is a generalization of the k-means clustering method, and it works by iteratively alternating between sparse coding the input data based on the current dictionary, and updating the atoms in the dictionary to better fit the data.

Given an over complete dictionary matrix D$\in \mathbb{R}^{nXK}$ that contains K signal-atoms for each columns, a signal y$\in \mathbb{R}^n$ can be represented as a linear combination of these atoms. To represent y, the sparse representation x should satisfy the exact condition $Y = Dx$, or approximate condition $y \approx Dx$, or $\|y - Dx\|_p \leq \epsilon$. The vector $x \in \mathbb{R}^K$ contains the representation coefficients of the signal y . Typically, norm P is selected as 1, 2 or ∞. If n < K and D is a full-rank matrix, an infinite number of solutions are available for the representation problem, Hence, constraints should be set on the solution. Also, to ensure sparsity, the solution with the fewest number of nonzero coefficients is preferred. Thus, the sparsity representation is the solution of either

$(P_0) \quad \min\limits_{x} \quad \| x \|_0$ subject to $y = Dx$

or

$(P_{0,e}) \quad \min\limits_{x} \quad \| x \|_0$ subject to $\|y - Dx\|_2 \leq \epsilon$

where L_0 the norm counts the nonzero entries of a vector.

Algorithm:

Finding the whole dictionary all at a time is impossible, so the process then update only one column of the dictionary D each time while fix X. The update of $k - th$ is done by rewriting the penalty term as

$$\|Y - DX\| = \| Y - \sum_{j=1}^{K} d_j x_T^j \|$$

$$\|(Y - \sum_{j \neq k} d_i x_T^i) - d_k x_T^k\| = \| E_k - d_k x_T^k \|$$

x_T^kdenotes the $k-th$ row of X

Over complete Dictionary

SVD(E_K)=UDV^T

if $E_K$is Rank 1 => D is a diagonal with only one non-zero principal diagonal element.

UDV can be factored as $U_1(d_{11},x_1)+ U_2(d_{22}, x_2)^T \cdots$

this will get closest to Rank 1

now $U_1 \to d_k$

$(d_{11}v_1)^T \to x_T^K$

Convergence of this algorithm is guaranteed, since both the steps going down direction.

Note

k -th vector of D is used by very few people (X). Since X is highely sparse. So instead of using the whole bigger problem, we can use the smaller problem where only columns corrospoding to the values that are present with non zero (in X) are used.

31.6 Homework Problems:

31.6.1 Homework

Design a method to fit a line to a set of points in 2D such that orthogonal distance to the line is minimized.

The residuals of the best-fit line for a set of n points using unsquared perpendicular distances d_i of points (x_i, y_i) are given by

$$R_\perp = \sum_{i=1}^{n} d_i \tag{31.1}$$

Since the perpendicular distance from a line y=a+ b x to point i is given by

$$d_i = \frac{|y_i - (a + bx_i)|}{\sqrt{1 + b^2}}, \tag{31.2}$$

the function to be minimized is

$$R_\perp = \sum_{i=1}^{n} \frac{|y_i - (a + bx_i)|}{\sqrt{1 + b^2}} \tag{31.3}$$

The absolute value function does not have continuous derivatives, minimizing R is not amenable to analytic solution. However, if the square of the perpendicular distances

$$R_\perp^2 = \sum_{i=1}^{n} \frac{[y_i - (a + bx_i)]^2}{1 + b^2} \tag{31.4}$$

is minimized instead, the problem can be solved in closed form. R^2 is a minimum when

$$= \frac{2}{1+b^2}\sum_{i=1}^{n}[y_i-(a+bx_i)](-1)=0 \tag{31.5}$$

and

$$= \frac{2}{1+b^2}\sum_{i=1}^{n}[y_i-(a+bx_i)](-x_i)+\sum_{i=1}^{n}\frac{[y_i-(a+bx_i)]^2(-1)(2b)}{(1+b^2)^2}=0 \tag{31.6}$$

The former gives

$$a = \frac{\sum_{i=1}^{n} y_i - b\sum_{i=1}^{n} x_i}{n} \tag{31.7}$$

$$= y - bx, \tag{31.8}$$

and the latter

$$(1+b^2)\sum_{i=1}^{n}[y_i-(a+bx_i)]x_i + b\sum_{i=1}^{n}[y_i-(a+bx_i)]^2=0 \tag{31.9}$$

But

$$[y-(a+bx)]^2 = y^2-2(a+bx)y+(a+bx)^2 \tag{31.10}$$

$$y^2-2ay-2bxy+a^2+2abx+b^2x^2, \tag{31.11}$$

so (10) becomes

$$\begin{aligned}(1+b^2)(\sum_{i=1}^{n}x_iy_i - a\sum_{i=1}^{n}x_i - b\sum_{i=1}^{n}x_i^2)+b(\sum_{i=1}^{n}y_i^2 - \\ 2a\sum_{i=1}^{n}y_i - 2b\sum_{i=1}^{n}x_iy_i + a^2\sum_{i=1}^{n}1 + 2ab\sum_{i=1}^{n}x_i + b^2\sum_{i=1}^{n}x_i^2)=0\end{aligned} \tag{31.12}$$

$$\begin{aligned}[(1+b^2)(-b)+b(b^2)]\sum_{i=1}^{n}x_i^2 + [(1+b^2)-2b^2]\sum_{i=1}^{n}x_iy_i + b\sum_{i=1}^{n}y_i^2 + \\ [-a(1+b^2)+2ab^2]\sum_{i=1}^{n}x_i - 2ab\sum_{i=1}^{n}y_i + ba^2n = 0\end{aligned} \tag{31.13}$$

$$-b\sum_{i=1}^{n}x_i^2 + (1-b^2)\sum_{i=1}^{n}x_iy_i + b\sum_{i=1}^{n}y_i^2 + a(b^2-1)\sum_{i=1}^{n}x_i - 2ab\sum_{i=1}^{n}y_i + ba^2n = 0 \tag{31.14}$$

Plugging the value into (14) then gives

$$\begin{aligned}-b\sum_{i=1}^{n}x_i^2 + (1-b^2)\sum_{i=1}^{n}x_iy_i + b\sum_{i=1}^{n}y_i^2 + \frac{1}{n}(b^2-1)[\sum_{i=1}^{n}y_i - b\sum_{i=1}^{n}x_i]\sum_{i=1}^{n}x_i - \\ \frac{2}{n}[\sum_{i=1}^{n}y_i - b\sum_{i=1}^{n}x_i]b\sum_{i=1}^{n}y_i + \frac{b}{n}[\sum_{i=1}^{n}y_i - b\sum_{i=1}^{n}x_i]^2\end{aligned} \tag{31.15}$$

On simplifying the equations we get

$$b^2 + \frac{\sum_{i=1}^{n}y_i^2 - \sum_{i=1}^{n}x_i^2 + \frac{1}{n}[(\sum_{i=1}^{n}x_i)^2 - (\sum_{i=1}^{n}y_i)^2]}{\frac{1}{n}\sum_{i=1}^{n}x_i\sum_{i=1}^{n}y_i - \sum_{i=1}^{n}x_1y_i}b - 1 = 0 \tag{31.16}$$

So define

$$B = \frac{1}{2}\frac{[\sum_{i=1}^{n}y_i^2 - \frac{1}{n}(\sum_{i=1}^{n}y_i)^2] - [\sum_{i=1}^{n}x_i^2 - \frac{1}{n}(\sum_{i=1}^{n}x_i)^2]}{\frac{1}{n}\sum_{i=1}^{n}x_i\sum_{i=1}^{n}y_i - \sum_{i=1}^{n}x_1y_i} \tag{31.17}$$

$$\frac{1}{2}\frac{(\sum_{i=1}^{n} y_i^2 - ny^2) - [\sum_{i=1}^{n} x_i^2 - nx^2}{nxy - \sum_{i=1}^{n} x_i y_i}, \tag{31.18}$$

and the quadratic formula gives

$$b = -B + -\sqrt{B^2 + 1}, \tag{31.19}$$

1 . Under what condition does MP/OMP gives optimal answer?

Ans:Orthogonal Matching Pursuit will recover the optimum representation if

$$\max_{a_i} ||B^+ a_i||_1 < 1$$

where a_i is the column vectors of A that are not in B and B^+ is the pseudo- inverse of B

2. Under what conditions does BP yields solution to the original problem?

Ans: Let A consist of the columns of two orthonormal basis with coherence μ. Then if a representation x where

$$||x||_0 < \frac{1}{2}(\mu^{-1} + 1)$$

where $||\ ||_0$ is count of the number of non-zero coefficients. Then x is the unique solution

Model Fitting

Once the data representation part is done, next we try to build a model with our data. Here we try to model the target variable as function of features.

Lets represent the height of child as y, age of child as x_1, weight of child x_2, height of parents x_3 and x_4. We may create a linear model like

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_4 \tag{31.20}$$

or a polynomial model like

$$y = a_0 + a_1 x_1 + a_2 {x_1}^2 + a_3 x_2 + a_4 x_2^2 + a_5 x_3 x_4 \tag{31.21}$$

or a even complex one like

$$y = a_0 + a_1 log x_1 + a_2 e_2^x + a_3 x_3 {x_4}^{\frac{1}{5}} \tag{31.22}$$

Problem Solver

In this step we may try to either predict new values based on our model (prediction) or may try to classify a new observation into a some class (classification).

Example - Linear Model (Line Fitting).

Taking the example of modelling the height of child, lets say our model is very simple and it says the height of a the child is only dependent on the age of child linearly. Mathematically height of child y, age of child x_1

$$y = a_0 + a_1 x_1 \tag{31.23}$$

or more generally $y = Ax$

where x is a n dimensional vector corresponding to (n-1) features

As can be seen this is the equation of a line and we have a lot sample of height and age of child. Now the problem statement is to find the equation of line using all these samples that minimizes the error. The error can be defined in may forms like

1. lleast square error $||y - Ax||^2$, fit the line such that sum distance of point form line is minimized

2. L_0 norm $||y - Ax||_0$, i.e fit the line such that number of points not on line are minimized
3. least square error $||y - Ax||^2$, fit the line such that sum of orthogonal distance of point form line is minimized.

References

1. Francis Bach, Julien Mairal, Jean Ponce and Guillermo Sapiro Sparse Coding and Dictionary Learning for Image Analysis, ICCVâ 09 tutorial, Kyoto, 28th September 2009
2. Yuchen Xie On A Nonlinear Generalization of Sparse Coding and Dictionary Learning , Qualcomm Technologies, Inc., San Diego, CA 92121 USA
3. Philip Breen Algorithms for Sparse Approximation , University of Edinburgh 2009
4. Philip Breen, Algorithms for Sparse Approximation, School of Mathematics, University of Edinburgh ,2009
5. Shaobing Chen, David Donoho, Basis Pursuit, Statistics Department, Stanford University, Stanford, CA 94305
6. Holger Boche, Robert Calderbank, Gitta Kutyniok, and Jan Vybiral,Survey of Compressed Sensing
7. The EM Algorithm November 20, 2005
8. IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 57, NO. 7, JULY 2011 Orthogonal Matching Pursuit for Sparse Signal Recovery With Noise
9. *T. Tony Cai and Lie Wang*
10. IEEE TRANSACTIONS ON SIGNAL PROCESSING, VOL. 54, NO. 11, NOVEMBER 2006 4311 K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation *Michal Aharon, Michael Elad, and Alfred Bruckstein*
11. KSVD https://en.wikipedia.org/wiki/K-SVD
12. M. Aharon, M. Elad, and A. M. Bruckstein. The K-SVD: An algorithm for designing of overcomplete dictionaries for sparse representations. IEEE Transactions on Signal Processing, 54(11):4311–4322, November 2006.
13. Orthogonal Matching Pursuit for Sparse SignalRecovery With Noise*T. Tony Cai and Lie Wang*
14. Francis Bach, Julien Mairal, Jean Ponce and Guill ermo Sapiro, Sparse Coding and Dictionary Learning for Image Analysis (Part III: Optimization for Sparse Coding and Dictionary Learning), CVPR'10 tutorial, San Francisco, 14th June 2010.