

DM587
Scientific Programming

Linear Programming

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Leontief Input Output Models
2. Production Planning
3. Diet Problem
4. Budget Allocation

Outline

Leontief Input Output Models
Production Planning
Diet Problem
Budget Allocation

1. Leontief Input Output Models

2. Production Planning

3. Diet Problem

4. Budget Allocation

Closed Model

- Economic system consisting of a finite number of industries, $1, 2, \dots, k$.
- Over some fixed period of time, each industry produces output of some good or service that is completely utilized in a predetermined manner by the k industries.
- Find suitable prices to be charged for these k outputs so that for each industry, total expenditures equal total income.
- Such a price structure represents an equilibrium position for the economy.

Closed Model

Three homeowners – a carpenter, an electrician, and a plumber – agree to make repairs in their three homes. They agree to work a total of 10 days each according to the following schedule:

	Work Performed by		
	Carpenter	Electrician	Plumber
Days of Work in Home of Carpenter	2	1	6
Days of Work in Home of Electrician	4	5	1
Days of Work in Home of Plumber	4	4	3

For tax purposes, they must report and pay each other a reasonable daily wage, even for the work each does on his or her own home.

Their normal daily wages are about \$100, but they agree to adjust their respective daily wages so that each homeowner will come out even—that is, so that the total amount paid out by each is the same as the total amount each receives.

What should be the **prices** of their work?

Closed Model

-

$$2p_1 + p_2 + 6p_3 = 10p_1$$

$$4p_1 + 5p_2 + p_3 = 10p_2$$

$$4p_1 + 4p_2 + 3p_3 = 10p_3$$

- $(I - A)p = 0$. If $\det(I - A) \neq 0$ then non trivial solution. Moreover, it can be shown that for exchange matrices A that are stochastic matrices the solution p is such that its elements are non-negative and if A^m are positive for all m positive integer then all p entries are positive.

Open Model

- Consider a market with n industries producing n different commodities.
- The market is interdependent, meaning that each industry requires input from the other industries and possibly even its own commodity.
- In addition, there is an outside demand for each commodity that has to be satisfied.
- We wish to determine the amount of output of each industry which will satisfy all demands exactly; that is, both the demands of the other industries and the outside demand.

Open Model

- Let $n = 3$ and let a_{ij} indicate the amount of commodity i , $i = 1, 2, 3$ necessary to produce one unit of commodity j , $j = 1, 2, 3$.

- a_{ij} are given in monetary terms:

a_{ij} cost of the commodity i necessary to produce one unit profit of commodity j .

↪ Hence, we will assume that $a_{ij} \geq 0$

Example: to produce an amount of commodity j worth 100 dkk, one needs an amount of commodity i worth 30 dkk.

For the sake of simplicity we scale all these values such that the profit of each commodity is 1 unity of currency.

↪ Hence, for each commodity j its production is not profitable unless

$$\sum_{i=1}^3 a_{ij} < 1.$$

- d_i be the demand of commodity i expressed in units of currency.

For each commodity, the outside demand is covered by the production of the commodity after the subtraction of the amount of commodity that has to go in the other industries and the amount that has to go in the same industry.

Hence:

$$x_i - \sum_{j=1}^n a_{ij}x_j = d_i$$

For $n = 3$ we have:

$$x_1 - a_{11}x_1 - a_{12}x_2 - a_{13}x_3 = d_1$$

$$x_2 - a_{21}x_1 - a_{22}x_2 - a_{23}x_3 = d_2$$

$$x_3 - a_{31}x_1 - a_{32}x_2 - a_{33}x_3 = d_3$$

In matrix terms:

$$Ix - Ax = d \quad \text{or} \quad (I - A)x = d$$

which is a system of linear equations. To make sense the solution x must be non-negative. It can be shown that under the conditions expressed above the solution to the system is unique and non-negative.

Open Model

- Unique non-negative solution for x if and only if $(I - A)^{-1}$ exists and $(I - A)^{-1} \geq 0$.
- The matrix A such that $(I - A)^{-1}$ exists and $(I - A)^{-1} \geq 0$ is called **productive**.
- The matrix A is productive \iff there exists $x \geq 0$ such that $x > Ax$
 $\iff \sum_{j=1}^n a_{ij} < 1$ (row sums)
 that is, there is some production plan such that each industry produces (monetarily) more than it consumes.
- A matrix is productive $\iff \sum_{i=1}^m a_{ij} < 1$ (column sums)
 that is, the j th industry is **profitable** if the total value of the outputs of all m industries needed to produce one unit of value of output of the industry j is less than one.

Decision Support Tools

- So far we considered a full economic system (country, region) and the decision making from the point of view of a Government planning
IO Models and Linear Systems of Equations
- Now, let's consider the Planning of Activities by a single Firm. Eg: Supply chain management, logistics, production scheduling
Linear Programming

A firm can produce their items in many different ways. The planning problem is characterized by a large number of feasible ways of providing the same output.

Outline

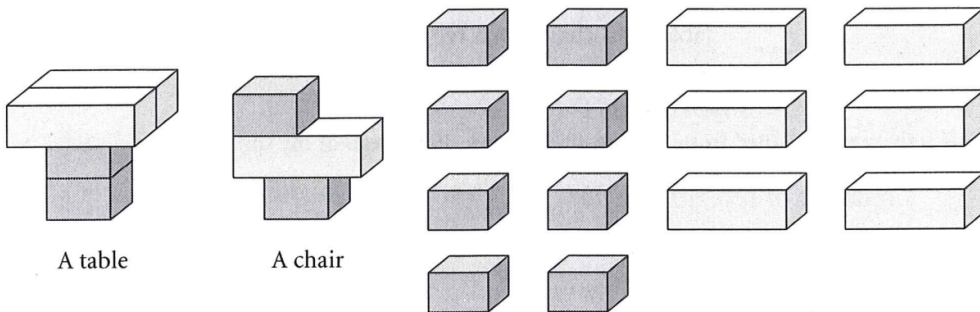
1. Leontief Input Output Models
2. Production Planning
3. Diet Problem
4. Budget Allocation

Production Planning

Suppose a company produces only **tables** and **chairs**.

A table is made of 2 large Lego pieces and 2 small pieces, while a chair is made of 1 large and 2 small pieces.

The resources available are 8 small and 6 large pieces.



The profit for a table is 1600 dkk and for a chair 1000 dkk. What product mix maximizes the company's profit using the available resources?

Mathematical Model

	Tables	Chairs	Capacity
Small Pieces	2	2	8
Large Pieces	2	1	6
Profit	16	10	

Decision Variables

$x_1 \geq 0$ units of small pieces

$x_2 \geq 0$ units of large pieces

Object Function

$\max 16x_1 + 10x_2$ maximize profit

Constraints

$2x_1 + 2x_2 \leq 8$ small pieces capacity

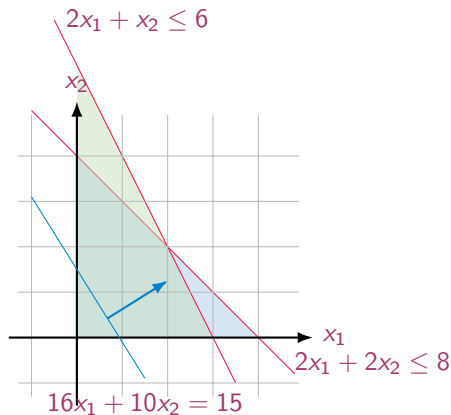
$2x_1 + x_2 \leq 6$ large pieces capacity

Mathematical Model

Materials A and B
Products 1 and 2

$$\begin{array}{ll}\max & 16x_1 + 10x_2 \\ & 2x_1 + 2x_2 \leq 8 \\ & 2x_1 + x_2 \leq 6 \\ & x_1 \geq 0 \\ & x_2 \geq 0\end{array}$$

Graphical Representation:



Resource Allocation - General Model

Managing a production facility

$1, 2, \dots, n$ products

$1, 2, \dots, m$ materials

b_i units of raw material at disposal

a_{ij} units of raw material i to produce one unit of product j

σ_j market price of unit of j th product

ρ_i prevailing market value for material i

$c_j = \sigma_j - \sum_{i=1}^n \rho_i a_{ij}$ profit per unit of product j

x_j amount of product j to produce

$$\begin{aligned}
 &\max \quad c_1 x_1 + c_2 x_2 + c_3 x_3 + \dots + c_n x_n = z \\
 &\text{subject to} \quad a_{11} x_1 + a_{12} x_2 + a_{13} x_3 + \dots + a_{1n} x_n \leq b_1 \\
 &\quad \quad \quad a_{21} x_1 + a_{22} x_2 + a_{23} x_3 + \dots + a_{2n} x_n \leq b_2 \\
 &\quad \quad \quad \dots \\
 &\quad \quad \quad a_{m1} x_1 + a_{m2} x_2 + a_{m3} x_3 + \dots + a_{mn} x_n \leq b_m \\
 &\quad \quad \quad x_1, x_2, \dots, x_n \geq 0
 \end{aligned}$$

Notation

$$\begin{aligned}
 \max \quad & c_1x_1 + c_2x_2 + c_3x_3 + \dots + c_nx_n = z \\
 \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n \leq b_1 \\
 & a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \dots + a_{2n}x_n \leq b_2 \\
 & \dots \\
 & a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \dots + a_{mn}x_n \leq b_m \\
 & x_1, x_2, \dots, x_n \geq 0
 \end{aligned}$$

$$\begin{aligned}
 \max \quad & \sum_{j=1}^n c_jx_j \\
 & \sum_{j=1}^n a_{ij}x_j \leq b_i, \quad i = 1, \dots, m \\
 & x_j \geq 0, \quad j = 1, \dots, n
 \end{aligned}$$

In Matrix Form

$$\begin{aligned}
 \max \quad & c_1x_1 + c_2x_2 + c_3x_3 + \dots + c_nx_n = z \\
 \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n \leq b_1 \\
 & a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \dots + a_{2n}x_n \leq b_2 \\
 & \dots \\
 & a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \dots + a_{mn}x_n \leq b_m \\
 & x_1, x_2, \dots, x_n \geq 0
 \end{aligned}$$

$$c^T = [c_1 \ c_2 \ \dots \ c_n]$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{31} & a_{32} & \dots & a_{mn} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$\begin{aligned}
 \max \quad & z = c^T x \\
 & Ax \leq b \\
 & x \geq 0
 \end{aligned}$$

Vector and Matrices in Excel

$$\sum_{j=1}^n c_j = c_1 + c_2 + \dots + c_n$$

SUM(B5 : B14)

Scalar product

$$\begin{aligned} u \cdot v &= u_1 v_1 + u_2 v_2 + \dots + u_n v_n \\ &= \sum_{j=1}^n u_j v_j \end{aligned}$$

SUMPRODUCT(B5 : B14, C5 : C : 14)

Our Numerical Example

$$\begin{aligned} \max \quad & \sum_{j=1}^n c_j x_j \\ & \sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, \dots, m \\ & x_j \geq 0, \quad j = 1, \dots, n \end{aligned}$$

$$\begin{aligned} \max \quad & 16x_1 + 10x_2 \\ & 2x_1 + 2x_2 \leq 8 \\ & 2x_1 + x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

$$\begin{aligned} \max \quad & c^T x \\ & Ax \leq b \\ & x \geq 0 \end{aligned}$$

$$x \in \mathbb{R}^n, c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$$

$$\max \quad \begin{bmatrix} 16 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 8 \\ 6 \end{bmatrix}$$

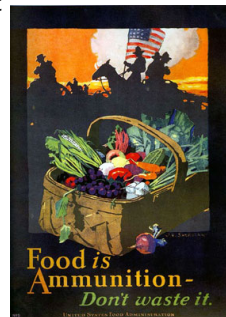
$$x_1, x_2 \geq 0$$

Outline

1. Leontief Input Output Models
2. Production Planning
3. Diet Problem
4. Budget Allocation

The Diet Problem (Blending Problems)

- Select a set of foods that will satisfy a set of daily nutritional requirement at minimum cost.
- Motivated in the 1930s and 1940s by US army.
- Formulated as a **linear programming problem** by George Stigler
- First **linear programming problem**
- (programming intended as planning not computer code)



min cost/weight

subject to nutrition requirements:

eat enough but not too much of Vitamin A

eat enough but not too much of Sodium

eat enough but not too much of Calories

...

The Diet Problem

Suppose there are:

- 3 foods available, corn, milk, and bread,
- there are restrictions on the number of calories (between 2000 and 2250) and the amount of Vitamin A (between 5000 and 50,000)

Food	Corn	2% Milk	Wheat bread
Vitamin A	107	500	0
Calories	72	121	65
Cost per serving	\$0.18	\$0.23	\$0.05

The Mathematical Model

Parameters (given data)

F = set of foods

N = set of nutrients

a_{ij} = amount of nutrient j in food i , $\forall i \in F, \forall j \in N$

c_i = cost per serving of food i , $\forall i \in F$

F_{mini} = minimum number of required servings of food i , $\forall i \in F$

F_{maxi} = maximum allowable number of servings of food i , $\forall i \in F$

N_{minj} = minimum required level of nutrient j , $\forall j \in N$

N_{maxj} = maximum allowable level of nutrient j , $\forall j \in N$

Decision Variables

x_i = number of servings of food i to purchase/consume, $\forall i \in F$

The Mathematical Model

Objective Function: Minimize the total cost of the food

$$\text{Minimize } \sum_{i \in F} c_i x_i$$

Constraint Set 1: For each nutrient $j \in N$, at least meet the minimum required level

$$\sum_{i \in F} a_{ij} x_i \geq N_{minj}, \quad \forall j \in N$$

Constraint Set 2: For each nutrient $j \in N$, do not exceed the maximum allowable level.

$$\sum_{i \in F} a_{ij} x_i \leq N_{maxj}, \quad \forall j \in N$$

Constraint Set 3: For each food $i \in F$, select at least the minimum required number of servings

$$x_i \geq F_{mini}, \quad \forall i \in F$$

Constraint Set 4: For each food $i \in F$, do not exceed the maximum allowable number of servings.

$$x_i \leq F_{maxi}, \quad \forall i \in F$$

The Mathematical Model

system of equalities and inequalities

$$\begin{aligned} \min \quad & \sum_{i \in F} c_i x_i \\ \sum_{i \in F} a_{ij} x_i & \geq N_{\min j}, \quad \forall j \in N \\ \sum_{i \in F} a_{ij} x_i & \leq N_{\max j}, \quad \forall j \in N \\ x_i & \geq F_{\min i}, \quad \forall i \in F \\ x_i & \leq F_{\max i}, \quad \forall i \in F \end{aligned}$$

Outline

1. Leontief Input Output Models
2. Production Planning
3. Diet Problem
4. Budget Allocation

Budget Allocation

- A company has six different **opportunities** to invest money.
- Each opportunity requires a certain investment over a period of 6 years or less.

<i>Expected Investment Cash Flows and Net Present Value</i>							
	Opp. 1	Opp. 2	Opp. 3	Opp. 4	Opp. 5	Opp. 6	Budget
Year 1	-\$5.00	-\$9.00	-\$12.00	-\$7.00	-\$20.00	-\$18.00	\$45.00
Year 2	-\$6.00	-\$6.00	-\$10.00	-\$5.00	\$6.00	-\$15.00	\$30.00
Year 3	-\$16.00	\$6.10	-\$5.00	-\$20.00	\$6.00	-\$10.00	\$20.00
Year 4	\$12.00	\$4.00	-\$5.00	-\$10.00	\$6.00	-\$10.00	\$0.00
Year 5	\$14.00	\$5.00	\$25.00	-\$15.00	\$6.00	\$35.00	\$0.00
Year 6	\$15.00	\$5.00	\$15.00	\$75.00	\$6.00	\$35.00	\$0.00
NPV	\$8.01	\$2.20	\$1.85	\$7.51	\$5.69	\$5.93	

- The company has an investment budget that needs to be met for each year.
- It also has the wish of investing in those opportunities that maximize the combined **Net Present Value** (NPV) after the 6th year.

Digression: What is the Net Present Value?

- P : value of the original payment presently due
- the debtor wants to delay the payment for t years,
- let r be the market rate of return that the creditor would obtain from a similar investment asset
- the future value of P is $F = P(1 + r)^t$

Viceversa, consider the task of finding:

- the present value P of \$100 that will be received in five years, or equivalently,
- which amount of money today will grow to \$100 in five years when subject to a constant discount rate.

Assuming a 5% per year interest rate, it follows that

$$P = \frac{F}{(1 + r)^t} = \frac{\$100}{(1 + 0.05)^5} = \$78.35.$$

Budget Allocation

Net Present Value calculation:

for each opportunity we calculate the NPV at time zero (the time of decision) as:

$$P_0 = \sum_{t=1}^5 \frac{F_t}{(1 + 0.05)^t}$$

Expected Investment Cash Flows and Net Present Value	Opp. 1	Opp. 2	Opp. 3	Opp. 4	Opp. 5	Opp. 6	Budget
Year 1	-\$5.00	-\$9.00	-\$12.00	-\$7.00	-\$20.00	-\$18.00	\$45.00
Year 2	-\$6.00	-\$6.00	-\$10.00	-\$5.00	\$6.00	-\$15.00	\$30.00
Year 3	-\$16.00	\$6.10	-\$5.00	-\$20.00	\$6.00	-\$10.00	\$20.00
Year 4	\$12.00	\$4.00	-\$5.00	-\$10.00	\$6.00	-\$10.00	\$0.00
Year 5	\$14.00	\$5.00	\$25.00	-\$15.00	\$6.00	\$35.00	\$0.00
Year 6	\$15.00	\$5.00	\$15.00	\$75.00	\$6.00	\$35.00	\$0.00
NPV	\$8.01	\$2.20	\$1.85	\$7.51	\$5.69	\$5.93	

Budget Allocation - Mathematical Model

- Let B_t be the budget available for investments during the years $t = 1..5$.
- Let a_{tj} be the cash flow for opportunity j and c_j its NPV
- Task: choose a set of opportunities such that the budget is never exceeded and the expected return is maximized. Consider both the case of indivisible and divisible opportunities.

Variables $x_j = 1$ if opportunity j is selected and $x_j = 0$ otherwise, $j = 1..6$

Objective

$$\max \sum_{j=1}^6 c_j x_j$$

Constraints

$$\sum_{j=1}^6 a_{tj} x_j + B_t \geq 0 \quad \forall t = 1..5$$

DM587
Scientific Programming

Affine Scaling Method

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Theory
Example

1. Theory

2. Example

Outline

Theory
Example

1. Theory

2. Example

Interior point methods in linear programming are classified as:

- affine scaling methods
- potential reduction methods, and
- central path methods (or central trajectory methods)

and for almost every approach one can consider

- a primal version,
- a dual version,
- a primal–dual version, or
- a self-dual version.

- affine scaling by Dikin
- logarithmic barrier algorithm by Fiacco and McCormick
- ellipsoid algorithm by Khachian
- projective method by Karmarkar \equiv logarithmic barrier
- primal-dual logarithmic barrier
- primal-dual barrier algorithm, combined with Mehrotra's predictor-corrector method

Applied with success also to semidefinite programming and other important classes of optimization problems, such as convex quadratic programming.

Interior point algorithm with affine scaling

Theory
Example

- Historically, one of the first interior-point methods to be invented
- No longer considered the method of choice for practical implementations
- Concept 1: Shoot through the interior of the feasible region toward an optimal solution.
- Concept 2: Move in a direction that improves the objective function value at the fastest possible rate.
- Concept 3: Transform the feasible region to place the current trial solution near its center, thereby enabling a large improvement when concept 2 is implemented.

Affine Scaling Method — Phase II

Theory
Example

$$\begin{aligned} &\text{maximize } \mathbf{c}^T \mathbf{x} \\ &\text{subject to } A\mathbf{x} = \mathbf{b} \\ &\quad \mathbf{x} \geq 0 \end{aligned}$$

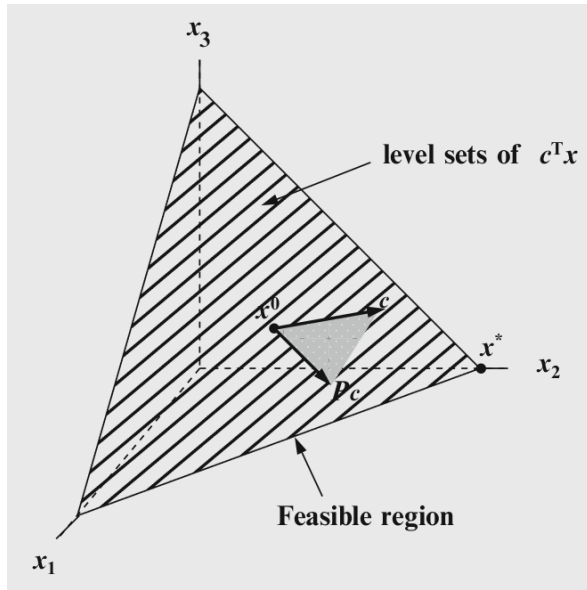
Two-phase method:

- Phase I uses only the feasibility direction
- Phase II uses only the optimality direction

Phase II

We assume that we have a feasible initial starting point, \mathbf{x}_0 that lies in the strict interior of the feasible set. That is:

$$A\mathbf{x}_0 = \mathbf{b} \quad \text{and} \quad \mathbf{x}_0 > 0$$



Hence, the steepest ascent direction will almost surely cause a move to infeasible points. This is also clear algebraically. Indeed,

$$A(x^0 + \Delta x) = Ax^0 + A\Delta x = b + Ac \neq b$$

(unless $Ac = 0$ which is not likely).

To see how to find a better direction, let us first review in what sense the gradient is the steepest ascent direction. The *steepest ascent direction* is defined to be the direction that gives the greatest increase in the objective function subject to the constraint that the displacement vector has unit length. That is, the steepest ascent direction is the solution to the following optimization problem:

$$(21.2) \quad \begin{array}{ll} \text{maximize} & c^T(x^0 + \Delta x) \\ \text{subject to} & \|\Delta x\|^2 = 1. \end{array}$$

We can solve this problem using Lagrange multipliers. Indeed, if we let λ denote the Lagrange multiplier for the constraint, the problem becomes

$$\max_{\Delta x, \lambda} c^T(x^0 + \Delta x) - \lambda(\Delta x^T \Delta x - 1).$$

Differentiating with respect to Δx and setting the derivative to zero, we get

$$c - 2\lambda\Delta x = 0,$$

which implies that

$$\Delta x = \frac{1}{2\lambda}c \propto c.$$

Then differentiating the Lagrangian with respect to λ and setting that derivative to zero, we see that

$$\|\Delta x\|^2 - 1 = 0,$$

which implies that

$$\|\Delta x\| = \pm 1.$$

Hence, the steepest ascent direction points in the direction of either c or its negative. Since the negative is easily seen not to be an ascent direction at all, it follows that the steepest ascent direction points in the direction of c .

The problem with the steepest ascent direction is that it fails to preserve feasibility. That is, it fails to preserve the equality constraints $Ax = b$. To remedy this problem, let's add these constraints to (21.2) so that we get the following optimization problem:

$$\begin{array}{ll}\text{maximize} & c^T(x^0 + \Delta x) \\ \text{subject to} & \|\Delta x\|^2 = 1 \\ & A(x^0 + \Delta x) = b.\end{array}$$

Again, the method of Lagrange multipliers is the appropriate tool. As before, let λ denote the Lagrange multiplier for the norm constraint, and now introduce a vector y containing the Lagrange multipliers for the equality constraints. The resulting unconstrained optimization problem is

$$\max_{\Delta x, \lambda, y} c^T(x^0 + \Delta x) - \lambda(\Delta x^T \Delta x - 1) - y^T(A(x^0 + \Delta x) - b).$$

$$\max_{\Delta x, \lambda, y} c^T(x^0 + \Delta x) - \lambda(\Delta x^T \Delta x - 1) - y^T(A(x^0 + \Delta x) - b).$$

Differentiating this Lagrangian with respect to Δx , λ , and y and setting these derivatives to zero, we get

$$c - 2\lambda\Delta x - A^T y = 0$$

$$\|\Delta x\|^2 - 1 = 0$$

$$A(x^0 + \Delta x) - b = 0.$$

The second equation tells us that the length of Δx is one. Since we are interested in the direction of Δx and are not concerned about its length, we ignore this second equation. The first equation tells us that Δx is proportional to $c - A^T y$, and again, since we aren't concerned about lengths, we put $\lambda = 1/2$ so that the first equation reduces to

$$(21.3) \quad \Delta x = c - A^T y.$$

Since $Ax^0 = b$, the third equation says that

$$A\Delta x = 0.$$

Substituting (21.3) into this equation, we get

$$Ac - AA^T y = 0,$$

which, assuming that AA^T has full rank (as it should), can be solved for y to get

$$y = (AA^T)^{-1}Ac.$$

Now, substituting this expression into (21.3), we see that

$$\Delta x = c - A^T(AA^T)^{-1}Ac.$$

It is convenient to let P be the matrix defined by

$$P = I - A^T(AA^T)^{-1}A.$$

With this definition, Δx can be expressed succinctly as

$$\Delta x = Pc.$$

We claim that P is the matrix that maps any vector, such as c , to its orthogonal projection onto the null space of A . To justify this claim, we first need to define some of the terms we've used. The *null space* of A is defined as $\{d \in \mathbb{R}^n : Ad = 0\}$. We shall denote the null space of A by $N(A)$. A vector \tilde{c} is the *orthogonal projection* of c onto $N(A)$ if it lies in the null space,

$$\tilde{c} \in N(A),$$

and if the difference between it and c is orthogonal to every other vector in $N(A)$. That is,

$$d^T(c - \tilde{c}) = 0, \quad \text{for all } d \in N(A).$$

Hence, to show that Pc is the orthogonal projection of c onto the null space of A , we simply check these two conditions. Checking the first, we see that

$$APc = Ac - AA^T(AA^T)^{-1}Ac,$$

which clearly vanishes. To check the second condition, let d be an arbitrary vector in the null space, and compute

$$d^T(c - Pc) = d^T A^T (AA^T)^{-1} Ac,$$

which also vanishes, since $d^T A^T = (Ad)^T = 0$. The orthogonal projection Pc is shown in Figure [21.1](#).

- If it is close to a “wall”, the overall increase in one step will be small
- scale the variables in the problem so that the current feasible solution is far from the walls, compute the step direction as the projected gradient in the scaled problem, and then translate this direction back into the original system.
- scale each variable in such a manner that its initial value gets mapped to 1. That is, for each $j = 1, 2, \dots, n$, we introduce new variables given by

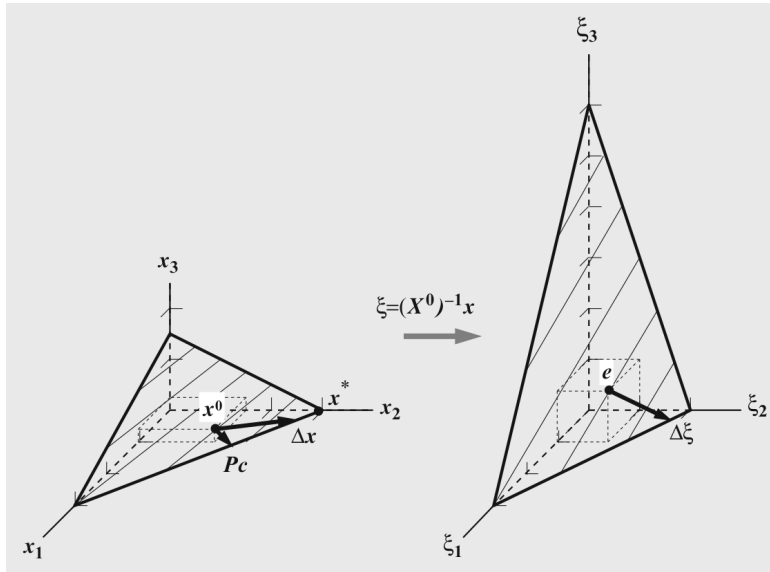
$$\xi_j = \frac{x_j}{x_j^0} \quad \implies \quad x_j = x_j^0 \xi_j$$

In matrix notation:

$$\mathbf{x} = \mathbf{X}^0 \boldsymbol{\xi}$$

(\mathbf{X}^0 diagonal matrix of diagonal \mathbf{x}^0)

under this change of variables, the initial solution \mathbf{x}_0 gets mapped to the vector \mathbf{e} of all ones, which is at least one unit away from each wall.



After substitution:

$$\begin{aligned} & \text{maximize} && \mathbf{c}^T \mathbf{X}^0 \boldsymbol{\xi} \\ & \text{subject to} && \mathbf{A} \mathbf{X}^0 \boldsymbol{\xi} = \mathbf{b} \\ & && \boldsymbol{\xi} \geq 0 \end{aligned}$$

It is a linear programming problem in standard form with constraint matrix $\mathbf{A} \mathbf{X}^0$ and vector of objective function coefficients $(\mathbf{c}^T \mathbf{X}^0)^T = \mathbf{X}^0 \mathbf{c}$ (since \mathbf{X}^0 is diagonal matrix).

We can determine the steepest ascent direction applying what seen in the previous slides:

$$\Delta \boldsymbol{\xi} = \left(\mathbf{I} - \mathbf{X}^0 \mathbf{A}^T (\mathbf{A} \mathbf{X}^{0^2} \mathbf{A}^T)^{-1} \mathbf{A} \mathbf{X}^0 \right) \mathbf{X}^0 \mathbf{c}.$$

and

$$\boldsymbol{\xi}^1 = \boldsymbol{\xi}^0 + \Delta \boldsymbol{\xi} = \mathbf{e} + \Delta \boldsymbol{\xi}$$

Transforming the new point ξ^1 back into the original unscaled variables, we get a new point \mathbf{x}^1 :

$$\mathbf{x}^1 = X^0 \xi^1 = X^0(\mathbf{e} + \Delta \xi) = \mathbf{x}^0 + X^0 \Delta \xi$$

The difference between \mathbf{x}^1 and \mathbf{x}^0 is the step direction in the original variables. Denoting this difference by $\Delta \mathbf{x}$, we see that:

$$\begin{aligned} \Delta \mathbf{x} &= \mathbf{x}_1 - \mathbf{x}_0 = \mathbf{x}^0 + X^0 \Delta \xi - \mathbf{x}^0 = \\ &= X^0 \left(I - X^0 A^T (A X^{0^2} A^T)^{-1} A X^0 \right) X^0 \mathbf{c} = \\ &= (E - E A^T (A E A^T)^{-1} A E) \mathbf{c} \end{aligned}$$

where $E = X^{0^2}$.

This expression for $\Delta \mathbf{x}$ is called **affine-scaling step direction**.

Recall that we worked with $\|\Delta \xi\| = 1$. We can then choose step lengths in such a manner as to ensure “strict” feasibility of each iteration.

If all components of $\Delta\xi$ are nonnegative, then we can increase the objective function without bound; stop the algorithm with an unbounded solution.

Otherwise, if the step were chosen so that the new point were to lie exactly on the boundary of the feasible region, then the multiplier for $\Delta\xi$ in the transformed system would be equal to the inverse of the absolute value of the negative component of $\Delta\xi$ with the largest absolute value. That is:

$$\xi^1 = \xi^0 + \theta \Delta\xi = \mathbf{e} + \theta \Delta\xi$$

$$\theta = \frac{1}{\max_j \{-\Delta\xi_j\}} = \frac{1}{\max_j \{-\Delta x_j / x_j^0\}}$$

With this definition of θ the smallest component of ξ^1 goes to zero.

We then shorten the step by introducing a parameter $0 < \alpha < 1$ and setting the iterations of the affine-scaling algorithm defined by

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \theta \Delta \mathbf{x}$$

Theorem (Convergence Results)

1. *If the problem and its dual are nondegenerate, then for every $\alpha < 1$, the sequence generated by the algorithm converges to the optimal solution.*
2. *For $\alpha \leq 2/3$, the sequence generated by the algorithm converges to an optimal solution (regardless of degeneracy).*
3. *There exists an example and an associated $\alpha < 1$ for which the algorithm converges to a nonoptimal solution.*

As for the speed of convergence, there is no example but it is believed that the Klee–Minty problem might arise here as well.

At iteration k :

1. **Centering** Let $D = X^k = \text{Diag}(\mathbf{x}^k)$. Rescale the problem to center the current interior feasible solution by letting $\tilde{A} = AD$, $\tilde{\mathbf{c}}^T = \mathbf{c}^T D$. Hence, $\boldsymbol{\xi}^k = D^{-1}\mathbf{x}^k = \mathbf{e}$, the vector consisting of all 1's. Note that $\tilde{A}\boldsymbol{\xi}^k = \mathbf{b}$.
2. **Search Direction Computation** For the rescaled problem, project the steepest ascent direction $\tilde{\mathbf{c}}^T$ onto the null space of the constraint matrix \tilde{A} , resulting in the search direction $\Delta\boldsymbol{\xi} = (I - \tilde{A}^T(\tilde{A}\tilde{A}^T)^{-1}\tilde{A})\tilde{\mathbf{c}}$.
3. **Step Length** Add a positive multiple θ of the search direction $\Delta\boldsymbol{\xi}$ to the scaled interior feasible point, by computing $\boldsymbol{\xi}^{k+1} = \mathbf{e} + \theta\Delta\boldsymbol{\xi}$. If $\Delta\boldsymbol{\xi} \geq 0$, then $\boldsymbol{\xi}^{k+1}$, and hence \mathbf{x}^{k+1} , can increase without bound; stop the algorithm with an unbounded solution. Otherwise, because $\tilde{A}\Delta\boldsymbol{\xi} = 0$ and $\tilde{A}\boldsymbol{\xi}^{k+1} = \mathbf{b}$, then θ must be chosen to ensure that $\boldsymbol{\xi}^{k+1} > 0$, avoiding the border. For any constant α such that $0 < \alpha < 1$, the update $\boldsymbol{\xi}^{k+1} = \mathbf{e} + \left(\frac{\alpha}{\max_j \{-\Delta\xi_j\}} \right) \Delta\boldsymbol{\xi}$ suffices.
4. **Optimality Test** Unscale the problem, setting $\mathbf{x}^{k+1} = D\boldsymbol{\xi}^{k+1}$. Test \mathbf{x}^{k+1} for optimality by checking whether $\|\mathbf{x}^{k+1} - \mathbf{x}^k\|$ is small. If \mathbf{x}^{k+1} is optimal, stop the algorithm. Otherwise, return to Step 1 with feasible interior point solution \mathbf{x}^{k+1} .

Outline

Theory
Example

1. Theory

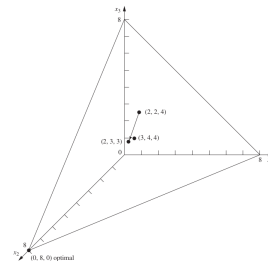
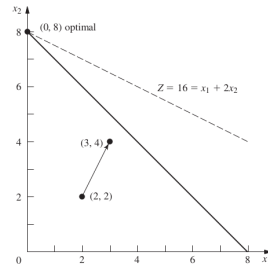
2. Example

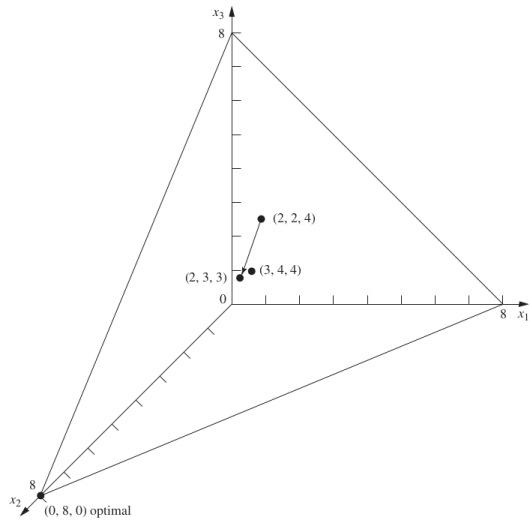
Example

$$\begin{aligned}\max z &= x_1 + 2x_2 \\ x_1 + x_2 &\leq 8 \\ x_1 &\geq 0, x_2 \geq 0\end{aligned}$$

In equational standard form:

$$\begin{aligned}\max z &= x_1 + 2x_2 \\ x_1 + x_2 + x_3 &= 8 \\ x_1 &\geq 0, x_2 \geq 0, x_3 \geq 0\end{aligned}$$





Affine-Scaling Method

1. Given the current trial solution (x_1, x_2, \dots, x_n) , set

$$\mathbf{D} = \begin{bmatrix} x_1 & 0 & 0 & \cdots & 0 \\ 0 & x_2 & 0 & \cdots & 0 \\ 0 & 0 & x_3 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & x_n \end{bmatrix}$$

2. Calculate $\tilde{\mathbf{A}} = \mathbf{AD}$ and $\tilde{\mathbf{c}} = \mathbf{Dc}$.
3. Calculate $\mathbf{P} = \mathbf{I} - \tilde{\mathbf{A}}^T(\tilde{\mathbf{A}}\tilde{\mathbf{A}}^T)^{-1}\tilde{\mathbf{A}}$ and $\mathbf{c}_p = \mathbf{P}\tilde{\mathbf{c}}$.
4. Identify the negative component of \mathbf{c}_p having the largest absolute value, and set v equal to this absolute value. Then calculate

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + \frac{\alpha}{v} \mathbf{c}_p,$$

where α is a selected constant between 0 and 1 (for example, $\alpha = 0.5$).

5. Calculate $\mathbf{x} = \mathbf{D}\tilde{\mathbf{x}}$ as the trial solution for the next iteration (step 1). (If this trial solution is virtually unchanged from the preceding one, then the algorithm has virtually converged to an optimal solution, so stop.)

Watch out the change of notation:

$$\Delta\xi \rightarrow \mathbf{c}_p$$

$$\xi \rightarrow \tilde{\mathbf{x}}$$

$$\theta = \frac{1}{v} = \frac{1}{\max_j \{-\Delta\xi_j\}} = \frac{1}{\max_j \{-\Delta x_j / x_j^k\}}$$

$$\mathbf{D} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix}.$$

The rescaled variables then are the components of

$$\tilde{\mathbf{x}} = \mathbf{D}^{-1} \mathbf{x} = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{x_1}{2} \\ \frac{x_2}{2} \\ \frac{x_3}{4} \end{bmatrix}.$$

In these new coordinates, \mathbf{A} and \mathbf{c} have become

$$\tilde{\mathbf{A}} = \mathbf{A}\mathbf{D} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 4 \end{bmatrix},$$

$$\tilde{\mathbf{c}} = \mathbf{D}\mathbf{c} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix}.$$

Therefore, the projection matrix is

$$\begin{aligned} \mathbf{P} &= \mathbf{I} - \tilde{\mathbf{A}}^T(\tilde{\mathbf{A}}\tilde{\mathbf{A}}^T)^{-1}\tilde{\mathbf{A}} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix} \left(\begin{bmatrix} 2 & 2 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix} \right)^{-1} \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix} \end{aligned}$$

$$\mathbf{P} = \mathbf{I} - \tilde{\mathbf{A}}^T(\tilde{\mathbf{A}}\tilde{\mathbf{A}}^T)^{-1}\tilde{\mathbf{A}}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix} \left(\begin{bmatrix} 2 & 2 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix} \right)^{-1} \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{1}{24} \begin{bmatrix} 4 & 4 & 8 \\ 4 & 4 & 8 \\ 8 & 8 & 16 \end{bmatrix} = \begin{bmatrix} \frac{5}{6} & -\frac{1}{6} & -\frac{1}{3} \\ -\frac{1}{6} & \frac{5}{6} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & \frac{1}{3} \end{bmatrix},$$

so that the projected gradient is

$$\mathbf{c}_p = \mathbf{P}\tilde{\mathbf{c}} = \begin{bmatrix} \frac{5}{6} & -\frac{1}{6} & -\frac{1}{3} \\ -\frac{1}{6} & \frac{5}{6} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ -2 \end{bmatrix}.$$

Define v as the *absolute value* of the *negative* component of \mathbf{c}_p having the *largest* absolute value, so that $v = |-2| = 2$ in this case. Consequently, in the current coordinates, the algorithm now moves from the current trial solution $(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3) = (1, 1, 1)$ to the next trial solution

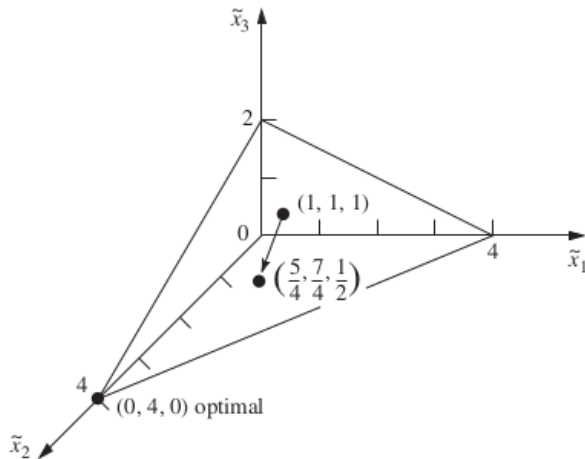
$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \frac{\alpha}{v} \mathbf{c}_p = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \frac{0.5}{2} \begin{bmatrix} 1 \\ 3 \\ -2 \end{bmatrix} = \begin{bmatrix} \frac{5}{4} \\ \frac{7}{4} \\ \frac{1}{2} \end{bmatrix},$$

as shown in Fig. 8.5. (The definition of v has been chosen to make the smallest component of $\tilde{\mathbf{x}}$ equal to zero when $\alpha = 1$ in this equation for the next trial solution.) In the original coordinates, this solution is

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \mathbf{D}\tilde{\mathbf{x}} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} \frac{5}{4} \\ \frac{7}{4} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{7}{2} \\ 2 \end{bmatrix}.$$

This completes the iteration, and this new solution will be used to start the next iteration. These steps can be summarized as follows for any iteration.

Example



Iteration 2*Step 1:*

Given the current trial solution $(x_1, x_2, x_3) = (\frac{5}{2}, \frac{7}{2}, 2)$, set

$$\mathbf{D} = \begin{bmatrix} \frac{5}{2} & 0 & 0 \\ 0 & \frac{7}{2} & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

(Note that the rescaled variables are

$$\begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \end{bmatrix} = \mathbf{D}^{-1} \mathbf{x} = \begin{bmatrix} \frac{2}{5} & 0 & 0 \\ 0 & \frac{2}{7} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{2}{5}x_1 \\ \frac{2}{7}x_2 \\ \frac{1}{2}x_3 \end{bmatrix},$$

so that the BF solutions in these new coordinates are

$$\tilde{\mathbf{x}} = \mathbf{D}^{-1} \begin{bmatrix} 8 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{16}{5} \\ 0 \\ 0 \end{bmatrix}, \quad \tilde{\mathbf{x}} = \mathbf{D}^{-1} \begin{bmatrix} 0 \\ 8 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{16}{7} \\ 0 \end{bmatrix},$$

and

$$\tilde{\mathbf{x}} = \mathbf{D}^{-1} \begin{bmatrix} 0 \\ 0 \\ 8 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix},$$

as depicted in Fig. 8.6.)

Step 2:

$$\tilde{\mathbf{A}} = \mathbf{A}\mathbf{D} = \begin{bmatrix} \frac{5}{2} & \frac{7}{2} & 2 \end{bmatrix} \quad \text{and} \quad \tilde{\mathbf{c}} = \mathbf{D}\mathbf{c} = \begin{bmatrix} \frac{5}{2} \\ 7 \\ 0 \end{bmatrix}.$$

Step 3:

$$\mathbf{P} = \begin{bmatrix} \frac{13}{18} & -\frac{7}{18} & -\frac{2}{9} \\ -\frac{7}{18} & \frac{41}{90} & -\frac{14}{45} \\ -\frac{2}{9} & -\frac{14}{45} & \frac{37}{45} \end{bmatrix} \quad \text{and} \quad \mathbf{c}_p = \begin{bmatrix} -\frac{11}{12} \\ \frac{133}{60} \\ -\frac{41}{15} \end{bmatrix}.$$

Step 4:

$|\frac{41}{15}| > |-\frac{11}{12}|$, so $v = \frac{41}{15}$ and

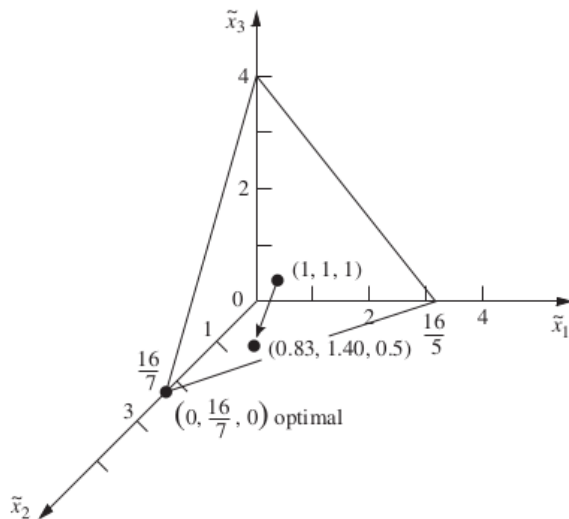
$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \frac{0.5}{\frac{41}{15}} \begin{bmatrix} -\frac{11}{12} \\ \frac{133}{60} \\ -\frac{41}{15} \end{bmatrix} = \begin{bmatrix} \frac{273}{328} \\ \frac{461}{328} \\ \frac{1}{2} \end{bmatrix} \approx \begin{bmatrix} 0.83 \\ 1.40 \\ 0.50 \end{bmatrix}.$$

Step 5:

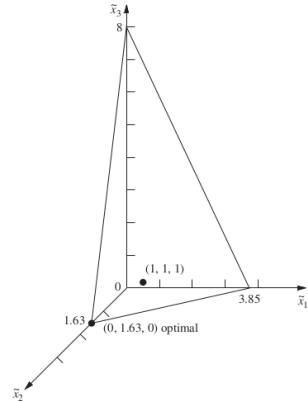
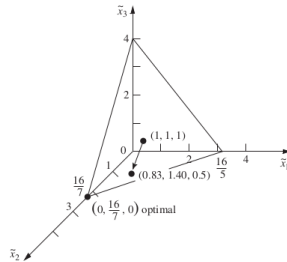
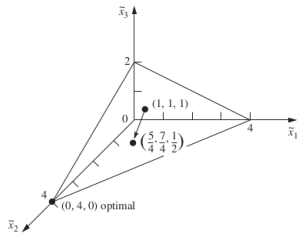
$$\mathbf{x} = \mathbf{D}\tilde{\mathbf{x}} = \begin{bmatrix} \frac{1365}{656} \\ \frac{3227}{656} \\ 1 \end{bmatrix} \approx \begin{bmatrix} 2.08 \\ 4.92 \\ 1.00 \end{bmatrix}$$

is the trial solution for iteration 3.

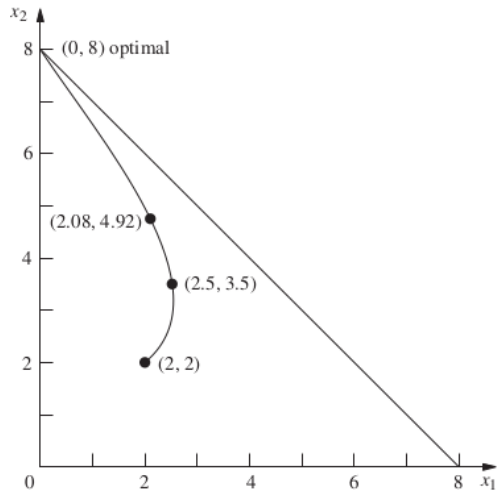
Example



Example



Example



Projection matrix:

$$P = (I - \tilde{A}^T(\tilde{A}\tilde{A}^T)^{-1}\tilde{A})$$

$$\Delta\xi = P\tilde{c} = \tilde{c} - \tilde{A}^T(\tilde{A}\tilde{A}^T)^{-1}\tilde{A}\tilde{c}$$

Solved by:

- $\tilde{A}\tilde{c} = \mathbf{v}$
- $\mathbf{w} = (\tilde{A}\tilde{A}^T)^{-1}\mathbf{v}$ solved as $(\tilde{A}\tilde{A}^T)\mathbf{w} = \mathbf{v}$ by Cholesky decomposition
- $\tilde{c} - \tilde{A}^T\mathbf{w}$

Phase I: Feasibility Direction

To derive a Phase I procedure for the affine-scaling algorithm, we consider a starting point \mathbf{x}^0 that has strictly positive components but does not necessarily satisfy the equality constraints $A\mathbf{x} = \mathbf{b}$. We then let

$$\boldsymbol{\rho} = \mathbf{b} - A\mathbf{x}^0$$

denote the vector of infeasibilities.

Auxiliary problem:

$$\max -x_0$$

$$A\mathbf{x} + x_0\boldsymbol{\rho} = \mathbf{b}$$

$$\mathbf{x} \geq 0, x_0 \geq 0$$

$\begin{bmatrix} \mathbf{x}^0 \\ 1 \end{bmatrix}$ is feasible solution

If $x_0^* > 0$, then the original problem is infeasible.

If $x_0^* = 0$, then the optimal solution to the auxiliary problem provides a feasible starting solution to the original problem (it may not be a strictly interior feasible solution)

Let us now derive a specific formula for the step direction vector in the auxiliary problem. The vector of objective coefficients is

$$\begin{bmatrix} 0 \\ -1 \end{bmatrix},$$

the constraint matrix is

$$\begin{bmatrix} A & \rho \end{bmatrix},$$

and the “current” solution can be denoted as

$$\begin{bmatrix} x \\ x_0 \end{bmatrix}.$$

Substituting these three objects appropriately into (21.5), we get

$$\begin{aligned} \begin{bmatrix} \Delta x \\ \Delta x_0 \end{bmatrix} &= \left(\begin{bmatrix} X^2 & \\ & x_0^2 \end{bmatrix} - \begin{bmatrix} X^2 & \\ & x_0^2 \end{bmatrix} \begin{bmatrix} A^T \\ \rho^T \end{bmatrix} \right. \\ &\quad \left. \left(\begin{bmatrix} A & \rho \end{bmatrix} \begin{bmatrix} X^2 & \\ & x_0^2 \end{bmatrix} \begin{bmatrix} A^T \\ \rho^T \end{bmatrix} \right)^{-1} \right. \\ &\quad \left. \begin{bmatrix} A & \rho \end{bmatrix} \begin{bmatrix} X^2 & \\ & x_0^2 \end{bmatrix} \right) \begin{bmatrix} 0 \\ -1 \end{bmatrix}. \end{aligned}$$

which yields $\Delta x = X^2 A^T (A X^2 A^T)^{-1} \rho$.

6. Cholesky factorization

- triangular matrices
- forward and backward substitution
- the Cholesky factorization
- solving $Ax = b$ with A positive definite
- inverse of a positive definite matrix
- permutation matrices
- sparse Cholesky factorization

Triangular matrix

a square matrix A is **lower triangular** if $a_{ij} = 0$ for $j > i$

$$A = \begin{bmatrix} a_{11} & 0 & \cdots & 0 & 0 \\ a_{21} & a_{22} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & 0 \\ a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} & 0 \\ a_{n1} & a_{n2} & \cdots & a_{n,n-1} & a_{nn} \end{bmatrix}$$

A is **upper triangular** if $a_{ij} = 0$ for $j < i$ (A^T is lower triangular)

a triangular matrix is **unit upper/lower triangular** if $a_{ii} = 1$ for all i

Forward substitution

solve $Ax = b$ when A is lower triangular with nonzero diagonal elements

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

algorithm:

$$x_1 := b_1/a_{11}$$

$$x_2 := (b_2 - a_{21}x_1)/a_{22}$$

$$x_3 := (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33}$$

$$\vdots$$

$$x_n := (b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{nn}$$

cost: $1 + 3 + 5 + \cdots + (2n - 1) = n^2$ flops

Back substitution

solve $Ax = b$ when A is upper triangular with nonzero diagonal elements

$$\begin{bmatrix} a_{11} & \cdots & a_{1,n-1} & a_{1n} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix}$$

algorithm:

$$\begin{aligned} x_n &:= b_n / a_{nn} \\ x_{n-1} &:= (b_{n-1} - a_{n-1,n}x_n) / a_{n-1,n-1} \\ x_{n-2} &:= (b_{n-2} - a_{n-2,n-1}x_{n-1} - a_{n-2,n}x_n) / a_{n-2,n-2} \\ &\vdots \\ x_1 &:= (b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n) / a_{11} \end{aligned}$$

cost: n^2 flops

Inverse of a triangular matrix

triangular matrix A with nonzero diagonal elements is nonsingular

- $Ax = b$ is solvable via forward/back substitution; hence A has full range
- therefore A has a zero nullspace, is invertible, etc. (see p.4-8)

inverse

- can be computed by solving $AX = I$ column by column

$$A \begin{bmatrix} X_1 & X_2 & \cdots & X_n \end{bmatrix} = \begin{bmatrix} e_1 & e_2 & \cdots & e_n \end{bmatrix}$$

- inverse of lower triangular matrix is lower triangular
- inverse of upper triangular matrix is upper triangular

Cholesky factorization

every positive definite matrix A can be factored as

$$A = LL^T$$

where L is lower triangular with positive diagonal elements

cost: $(1/3)n^3$ flops if A is of order n

- L is called the *Cholesky factor* of A
- can be interpreted as ‘square root’ of a positive definite matrix

Cholesky factorization algorithm

partition matrices in $A = LL^T$ as

$$\begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} l_{11} & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix}$$
$$= \begin{bmatrix} l_{11}^2 & l_{11}L_{21}^T \\ l_{11}L_{21} & L_{21}L_{21}^T + L_{22}L_{22}^T \end{bmatrix}$$

algorithm

1. determine l_{11} and L_{21} :

$$l_{11} = \sqrt{a_{11}}, \quad L_{21} = \frac{1}{l_{11}}A_{21}$$

2. compute L_{22} from

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

this is a Cholesky factorization of order $n - 1$

proof that the algorithm works for positive definite A of order n

- step 1: if A is positive definite then $a_{11} > 0$
- step 2: if A is positive definite, then

$$A_{22} - L_{21}L_{21}^T = A_{22} - \frac{1}{a_{11}}A_{21}A_{21}^T$$

is positive definite (see page 4-23)

- hence the algorithm works for $n = m$ if it works for $n = m - 1$
- it obviously works for $n = 1$; therefore it works for all n

Example

$$\begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

- first column of L

$$\begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} = \begin{bmatrix} 5 & 0 & 0 \\ 3 & l_{22} & 0 \\ -1 & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} 5 & 3 & -1 \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

- second column of L

$$\begin{bmatrix} 18 & 0 \\ 0 & 11 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \end{bmatrix} \begin{bmatrix} 3 & -1 \end{bmatrix} = \begin{bmatrix} l_{22} & 0 \\ l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{22} & l_{32} \\ 0 & l_{33} \end{bmatrix}$$

$$\begin{bmatrix} 9 & 3 \\ 3 & 10 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 1 & l_{33} \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 0 & l_{33} \end{bmatrix}$$

- third column of L : $10 - 1 = l_{33}^2$, *i.e.*, $l_{33} = 3$

conclusion:

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

Solving equations with positive definite A

$$Ax = b \quad (A \text{ positive definite of order } n)$$

algorithm

- factor A as $A = LL^T$
- solve $LL^T x = b$
 - forward substitution $Lz = b$
 - back substitution $L^T x = z$

cost: $(1/3)n^3$ flops

- factorization: $(1/3)n^3$
- forward and backward substitution: $2n^2$

Inverse of a positive definite matrix

suppose A is positive definite with Cholesky factorization $A = LL^T$

- L is invertible (its diagonal elements are nonzero)
- $X = L^{-T}L^{-1}$ is a right inverse of A :

$$AX = LL^T L^{-T} L^{-1} = LL^{-1} = I$$

- $X = L^{-T}L^{-1}$ is a left inverse of A :

$$XA = L^{-T}L^{-1}LL^T = L^{-T}L^T = I$$

- hence, A is invertible and

$$A^{-1} = L^{-T}L^{-1}$$

Summary

if A is positive definite of order n

- A can be factored as LL^T
- the cost of the factorization is $(1/3)n^3$ flops
- $Ax = b$ can be solved in $(1/3)n^3$ flops
- A is invertible with inverse: $A^{-1} = L^{-T}L^{-1}$

Sparse positive definite matrices

- a matrix is *sparse* if most of its elements are zero
- a matrix is *dense* if it is not sparse

Cholesky factorization of dense matrices

- $(1/3)n^3$ flops
- on a current PC: a few seconds or less, for n up to a few 1000

Cholesky factorization of sparse matrices

- if A is very sparse, then L is often (but not always) sparse
- if L is sparse, the cost of the factorization is much less than $(1/3)n^3$
- exact cost depends on n , #nonzero elements, sparsity pattern
- very large sets of equations ($n \sim 10^6$) are solved by exploiting sparsity

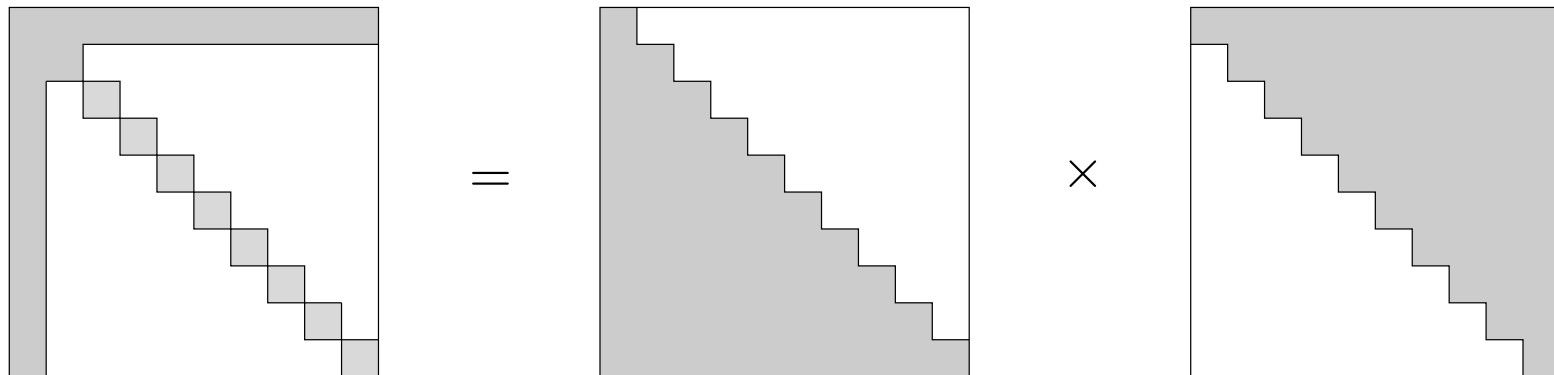
Effect of ordering

sparse equation (a is an $(n - 1)$ -vector with $\|a\| < 1$)

$$\begin{bmatrix} 1 & a^T \\ a & I \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$$

factorization

$$\begin{bmatrix} 1 & a^T \\ a & I \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ a & L_{22} \end{bmatrix} \begin{bmatrix} 1 & a^T \\ 0 & L_{22}^T \end{bmatrix} \text{ where } I - aa^T = L_{22}L_{22}^T$$



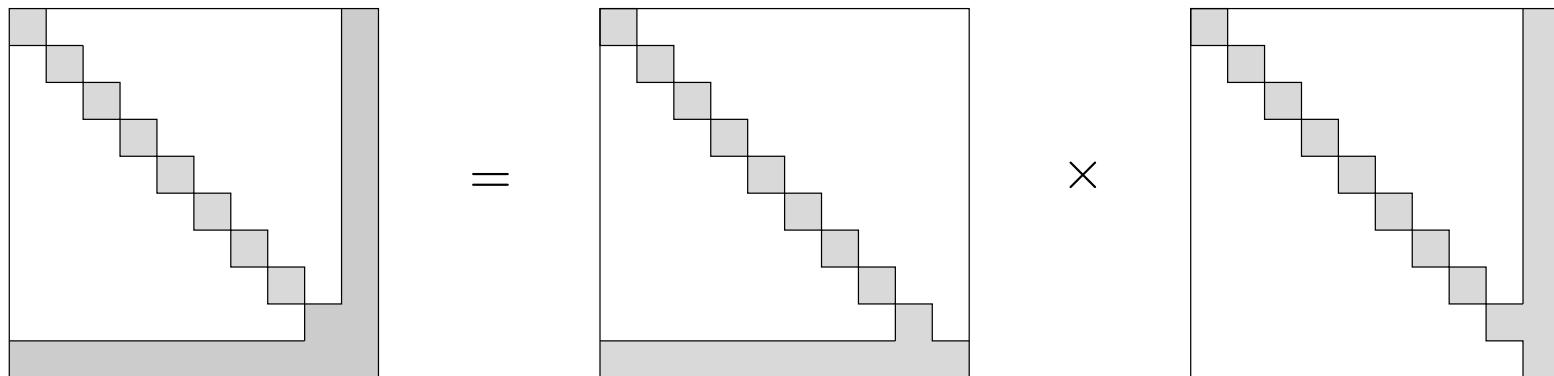
factorization with 100% fill-in

reordered equation

$$\begin{bmatrix} I & a \\ a^T & 1 \end{bmatrix} \begin{bmatrix} v \\ u \end{bmatrix} = \begin{bmatrix} c \\ b \end{bmatrix}$$

factorization

$$\begin{bmatrix} I & a \\ a^T & 1 \end{bmatrix} = \begin{bmatrix} I & 0 \\ a^T & \sqrt{1 - a^T a} \end{bmatrix} \begin{bmatrix} I & a \\ 0 & \sqrt{1 - a^T a} \end{bmatrix}$$



factorization with zero fill-in

Permutation matrices

a *permutation matrix* is the identity matrix with its rows reordered, *e.g.*,

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

- the vector Ax is a permutation of x

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ x_1 \end{bmatrix}$$

- $A^T x$ is the inverse permutation applied to x

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_3 \\ x_1 \\ x_2 \end{bmatrix}$$

- $A^T A = AA^T = I$, so A is invertible and $A^{-1} = A^T$

Solving $Ax = b$ when A is a permutation matrix

the solution of $Ax = b$ is $x = A^T b$

example

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 10.0 \\ -2.1 \end{bmatrix}$$

solution is $x = (-2.1, 1.5, 10.0)$

cost: zero flops

Sparse Cholesky factorization

if A is sparse and positive definite, it is usually factored as

$$A = PLL^T P^T$$

P a permutation matrix; L lower triangular with positive diagonal elements

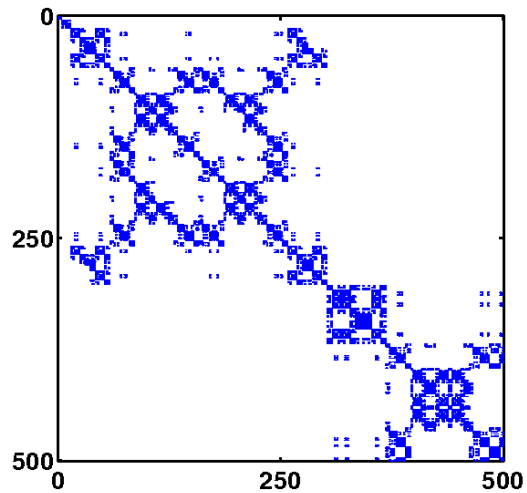
interpretation: we permute the rows and columns of A and factor

$$P^T A P = LL^T$$

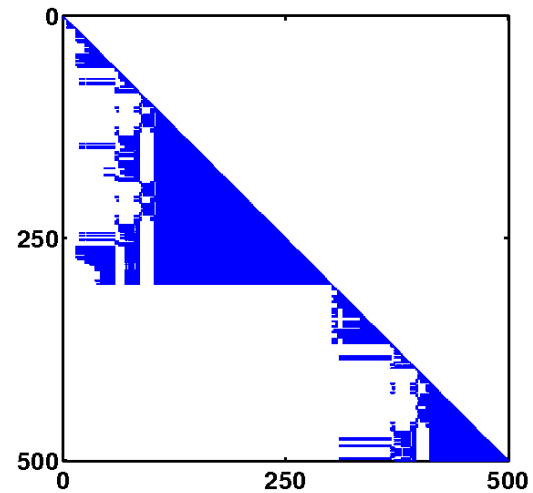
- choice of P greatly affects the sparsity L
- many heuristic methods (that we don't cover) exist for selecting good permutation matrices P

Example

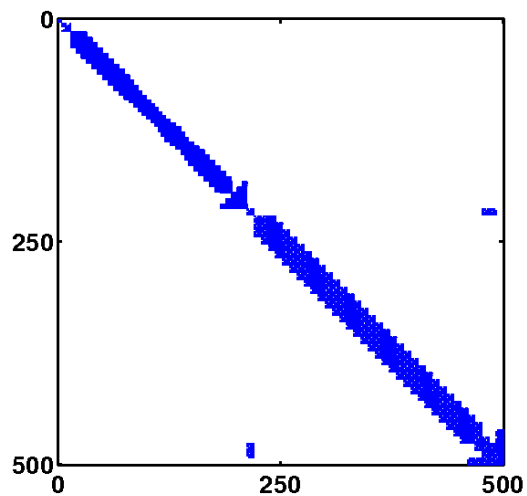
sparsity pattern of A



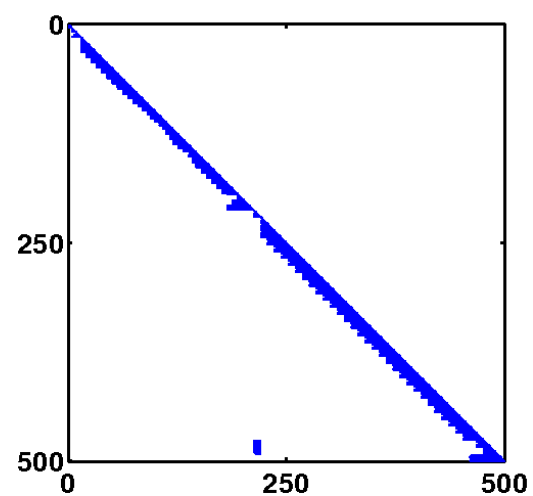
Cholesky factor of A



pattern of $P^T A P$



Cholesky factor of $P^T A P$



Solving sparse positive definite equations

solve $Ax = b$ via factorization $A = PLL^T P^T$

algorithm

1. $\tilde{b} := P^T b$
2. solve $Lz = \tilde{b}$ by forward substitution
3. solve $L^T y = z$ by back substitution
4. $x := Py$

interpretation: we solve

$$(P^T A P) y = \tilde{b}$$

using the Cholesky factorization of $P^T A P$

Interior-point methods
Path-following method

Table of Contents

Barrier Function

Path-Following Method

Implementation example

The KKT System

Table of Contents

Barrier Function

Path-Following Method

Implementation example

The KKT System

Linear Programming

Consider the primal form in linear programming:

$$\begin{array}{ll}\text{maximize} & c^T x \\ \text{subject to} & Ax \leq b \\ & x \geq 0\end{array}$$

And the corresponding dual problem:

$$\begin{array}{ll}\text{minimize} & b^T y \\ \text{subject to} & A^T y \leq c \\ & y \geq 0\end{array}$$

Linear Programming

Both problems can be converted into equality form:

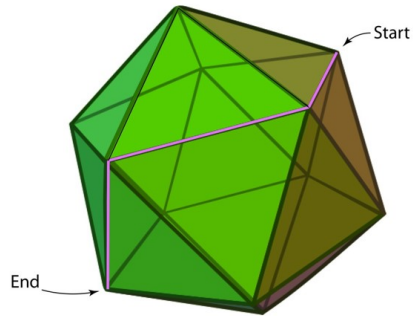
$$\begin{array}{ll}\text{maximize} & c^T x \\ \text{subject to} & Ax + w = b \\ & x, w \geq 0\end{array}$$

And:

$$\begin{array}{ll}\text{minimize} & b^T y \\ \text{subject to} & A^T y + z = c \\ & y, z \geq 0\end{array}$$

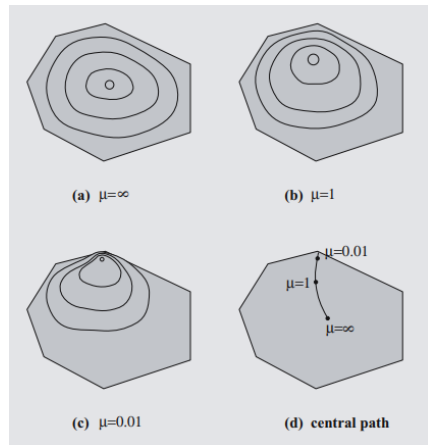
Simplex algorithm

Simplex Method finds the optimal solution by traversing along the edges from one vertex to another.



Interior Point method

The Interior Point method starts inside the polytope and iteratively converges to the optimal solution



Non-standard notation

Non-standard notation ahead! Given a lower-case letter denoting a vector quantity, we also have an upper-case letter denoting a diagonal matrix whose entries corresponds to elements the vector quantity.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \Rightarrow X = \begin{bmatrix} x_1 & & & \\ & x_2 & & \\ & & \ddots & \\ & & & x_n \end{bmatrix}$$

Barrier function

- ▶ How do we avoid converging outside the feasibility region?
- ▶ Barrier problem:

$$BP(\mu) : \begin{array}{ll} \text{maximize} & c^T x + \mu \sum_j \log x_j + \mu \sum_i \log w_i \\ \text{subject to} & Ax + w = b \end{array}$$

- ▶ Nonlinear objective function: logarithmic barrier function
- ▶ Family of problems indexed by parameter $\mu > 0$

Lagrange Multipliers and Barrier Problem

We Lagrange relax to the barrier function, we get the following problem:

$$L(x, w, y) = c^T x + \mu \sum_j \log x_j + \mu \sum_i \log w_i + y^T (b - Ax - w)$$

We get the first-order optimality conditions when we take the derivatives and set them to zero.

$$\frac{\partial L}{\partial x_j} = c_j + \mu \frac{1}{x_j} - \sum_i y_i a_{ij} = 0, \quad j = 1, 2, \dots, n.$$

$$\frac{\partial L}{\partial w_i} = \mu \frac{1}{w_i} - y_i = 0, \quad i = 1, 2, \dots, m.$$

$$\frac{\partial L}{\partial y_i} = b_i - \sum_j a_{ij} x_j - w_i = 0, \quad i = 1, 2, \dots, m.$$

Lagrange Multipliers and Barrier Problem

This can be written in matrix form:

$$A^T y - \mu X^{-1} e = c$$

$$y = \mu W^{-1} e$$

$$Ax + w = b$$

Note: X and W are the diagonal matrices containing diagonal entries.

Lagrange Multipliers and Barrier Problem

When we introduce an extra vector $z = \mu X^{-1}e$, we can rewrite our first-order optimality conditions like this:

$$Ax + w = b$$

$$A^T y - z = c$$

$$z = \mu X^{-1}e$$

$$y = \mu W^{-1}e$$

Important! e is the vector of all ones.

Lagrange Multipliers and Barrier Problem

When we multiply X and W on respectively the third and fourth equation, we get the following equations:

$$Ax + w = b$$

$$A^T y - z = c$$

$$XZe = \mu e$$

$$YWe = \mu e$$

Lagrange Multipliers and Barrier Problem

When we multiply X and W on respectively the third and fourth equation, we get the following equations:

$$Ax + w = b$$

$$A^T y - z = c$$

$$XZe = \mu e$$

$$YWe = \mu e$$

Componentwise, the third and fourth equation can be written like this:

$$x_j z_j = \mu$$

$$j = 1, 2, \dots, n$$

$$y_i w_i = \mu$$

$$i = 1, 2, \dots, m$$

Lagrange Multipliers and Barrier Problem

When we multiply X and W on respectively the third and fourth equation, we get the following equations:

$$Ax + w = b$$

$$A^T y - z = c$$

$$XZe = \mu e$$

$$YWe = \mu e$$

Componentwise, the third and fourth equation can be written like this:

$$x_j z_j = \mu$$

$$j = 1, 2, \dots, n$$

$$y_i w_i = \mu$$

$$i = 1, 2, \dots, m$$

μ -complementarity conditions: $2n + 2m$ equations in $2n + 2m$ unknowns.

Does a solution exist and if so, is it unique?

Lagrange Multipliers and Barrier Problem

Theorem 1

There exists a solution to the barrier problem if and only if both the primal and the dual feasible regions have nonempty interior.

Corollary 2

If a primal feasible set (or, for that matter, its dual) has a nonempty interior and is bounded, then for each $\mu > 0$ there exists a unique solution

$$(x_\mu, w_\mu, y_\mu, z_\mu)$$

Table of Contents

Barrier Function

Path-Following Method

Implementation example

The KKT System

Path-Following Method

1. Estimate appropriate value for μ .
2. Compute step directions $(\Delta x, \Delta w, \Delta y, \Delta z)$ pointing at the point $(x_\mu, w_\mu, y_\mu, z_\mu)$ on the central path.
3. Compute a new step length parameter θ such that the new point:

$$\begin{aligned}\tilde{x} &= x + \theta \Delta x, & \tilde{y} &= y + \theta \Delta y \\ \tilde{w} &= w + \theta \Delta w, & \tilde{z} &= z + \theta \Delta z\end{aligned}$$

4. Replace (x, w, y, z) with the new solution $(\tilde{x}, \tilde{y}, \tilde{w}, \tilde{z})$

Path-Following Method

1. Estimating μ

- ▶ We must find an appropriate value for μ .
- ▶ Too high, we might converge to the analytic center of the feasible set.
- ▶ Too low, we will converge to the edge of the feasible set that might be suboptimal.

$$\mu = \delta \frac{z^T x + y^T w}{n + m}$$

Path-Following Method

2. Compute step directions $(\Delta x, \Delta w, \Delta y, \Delta z)$

Recall the given equations defining the point $(x_\mu, w_\mu, y_\mu, z_\mu)$ on the central path:

$$Ax + w = b$$

$$A^T y - z = c$$

$$XZe = \mu e$$

$$YWe = \mu e$$

Path-Following Method

2. Compute step directions $(\Delta x, \Delta w, \Delta y, \Delta z)$

Given a new point $(x_\mu + \Delta x, w_\mu + \Delta w, y_\mu + \Delta y, z_\mu + \Delta z)$, we get the following equations:

$$A(x + \Delta x) + (w + \Delta w) = b$$

$$A^T(y + \Delta y) - (z + \Delta z) = c$$

$$(X + \Delta X)(Z + \Delta Z)e = \mu e$$

$$(Y + \Delta Y)(W + \Delta W)e = \mu e$$

Path-Following Method

2. Compute step directions $(\Delta x, \Delta w, \Delta y, \Delta z)$

We rewrite the equations so the unknowns are on the left and the data on the right:

$$A\Delta x + \Delta w = b - Ax - w$$

$$A^T \Delta y - \Delta z = c - A^T y + z$$

$$Z\Delta x + X\Delta z + \Delta X\Delta Ze = \mu e - XZe$$

$$W\Delta y + Y\Delta w + \Delta Y\Delta We = \mu e - YWe$$

Path-Following Method

2. Compute step directions $(\Delta x, \Delta w, \Delta y, \Delta z)$

Then we transform the equations into a linear system by dropping the nonlinear terms:

$$A\Delta x + \Delta w = b - Ax - w$$

$$A^T \Delta y - \Delta z = c - A^T y + z$$

$$Z\Delta x + X\Delta z = \mu e - XZe$$

$$W\Delta y + Y\Delta w = \mu e - YWe$$

Path-Following Method

3. Compute a new step length parameter θ

- ▶ Whenever we find the step direction, we need to determine the **step length** θ .
- ▶ Recall that we want to replace (x, w, y, z) with the new solution $(\tilde{x}, \tilde{y}, \tilde{w}, \tilde{z})$ by:

$$\tilde{x} = x + \theta \Delta x, \quad \tilde{y} = y + \theta \Delta y$$

$$\tilde{w} = w + \theta \Delta w, \quad \tilde{z} = z + \theta \Delta z$$

The solution to this system of linear equation corresponds to the application of the Newton method on the primal-dual equations and μ -complementary equations.

Path-Following Method

3. Compute a new step length parameter θ

- We need to guarantee that:

$$x_j + \theta \Delta x > 0, \quad j = 1, 2, \dots, n.$$

Path-Following Method

3. Compute a new step length parameter θ

- ▶ We need to guarantee that:

$$x_j + \theta \Delta x > 0, \quad j = 1, 2, \dots, n.$$

- ▶ Similarly for w , y and z .

Path-Following Method

3. Compute a new step length parameter θ

- We need to guarantee that:

$$x_j + \theta \Delta x > 0, \quad j = 1, 2, \dots, n.$$

- Similarly for w , y and z .

$$\frac{1}{\theta} = \max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\}$$

Path-Following Method

3. Compute a new step length parameter θ

- We need to guarantee that:

$$x_j + \theta \Delta x_j > 0, \quad j = 1, 2, \dots, n.$$

- Similarly for w , y and z .

$$\frac{1}{\theta} = \max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\}$$

- We introduce a parameter r that is close to but strictly less than one.

$$\theta = r \left(\max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\} \right)^{-1} \wedge 1$$

Path-Following Method

4. Replace (x, w, y, z) with $(\tilde{x}, \tilde{y}, \tilde{w}, \tilde{z})$

Now we can replace (x, w, y, z) with $(\tilde{x}, \tilde{y}, \tilde{w}, \tilde{z})$:

$$\tilde{x} = x + \theta \Delta x, \quad \tilde{y} = y + \theta \Delta y$$

$$\tilde{w} = w + \theta \Delta w, \quad \tilde{z} = z + \theta \Delta z$$

Path-Following Method

Pseudo-code of the path-following method

initialize $(x, w, y, z) > 0$

while (not optimal) {

$$\rho = b - Ax - w$$

$$\sigma = c - A^T y + z$$

$$\gamma = z^T x + y^T w$$

$$\mu = \delta \frac{\gamma}{n + m}$$

solve:

$$A\Delta x + \Delta w = \rho$$

$$A^T \Delta y - \Delta z = \sigma$$

$$Z\Delta x + X\Delta z = \mu e - XZe$$

$$W\Delta y + Y\Delta w = \mu e - YWe$$

$$\theta = r \left(\max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\} \right)^{-1} \wedge 1$$

$$x \leftarrow x + \theta \Delta x, \quad w \leftarrow w + \theta \Delta w$$

$$y \leftarrow y + \theta \Delta y, \quad z \leftarrow z + \theta \Delta z$$

}

Optimality

- ▶ We have converged to a solution, but how do we know if it is optimal?

Optimality

- ▶ We have converged to a solution, but how do we know if it is optimal?
- ▶ Recall from the duality theory that we need to meet the following criteria for the solution to be optimal:

Primal feasibility:

$$\|\rho\|_1 = \|b - Ax - w\|_1$$

Dual feasibility:

$$\|\sigma\|_1 = \|c - A^T y + z\|_1$$

Complementarity:

$$\gamma = z^T x + y^T w$$

Optimality

- ▶ When do we stop?
- ▶ Let $\epsilon > 0$ be a small tolerance and $M < \infty$ be a large finite tolerance
- ▶ $\|x\|_{\infty} > M$ then the primal problem is unbounded.
- ▶ $\|y\|_{\infty} > M$ then the dual problem is unbounded.
- ▶ If $\|\rho\|_1 < \epsilon$, $\|\sigma\|_1 < \epsilon$, and $\gamma < \epsilon$ then we found our optimal solution!

Table of Contents

Barrier Function

Path-Following Method

Implementation example

The KKT System

Implementation example

Consider the following LP problem:

Primal problem:

$$\begin{array}{ll}\max & 5x_1 + 4x_2 + 3x_3 \\ \text{s.t.} & 2x_1 + 3x_2 + x_3 \leq 5 \\ & 4x_1 + x_2 + 2x_3 \leq 11 \\ & 3x_1 + 4x_2 + 2x_3 \leq 8 \\ & x_1, x_2, x_3 \geq 0\end{array}$$

Dual problem:

$$\begin{array}{ll}\max & 5y_1 + 11y_2 + 8y_3 \\ \text{s.t.} & 2y_1 + 4y_2 + 3y_3 \leq 5 \\ & 3y_1 + y_2 + 4y_3 \leq 4 \\ & y_1 + 2y_2 + 2y_3 \leq 3 \\ & y_1, y_2, y_3 \geq 0\end{array}$$

Implementation example

Both are converted into equality form:

Primal problem:

$$\begin{array}{ll}\max & 5x_1 + 4x_2 + 3x_3 \\ \text{s.t.} & 2x_1 + 3x_2 + x_3 + w_1 = 5 \\ & 4x_1 + x_2 + 2x_3 + w_2 = 11 \\ & 3x_1 + 4x_2 + 2x_3 + w_3 = 8 \\ & x_1, x_2, x_3, w_1, w_2, w_3 \geq 0\end{array}$$

Dual problem:

$$\begin{array}{ll}\max & 5y_1 + 11y_2 + 8y_3 \\ \text{s.t.} & 2y_1 + 4y_2 + 3y_3 + z_1 \leq 5 \\ & 3y_1 + y_2 + 4y_3 + z_2 \leq 4 \\ & y_1 + 2y_2 + 2y_3 + z_3 \leq 3 \\ & y_1, y_2, y_3, z_1, z_2, z_3 \geq 0\end{array}$$

Implementation example

initialize $(x, w, y, z) > 0$

while (not optimal) {

$$\rho = b - Ax - w$$

$$\sigma = c - A^T y + z$$

$$\gamma = z^T x + y^T w$$

$$\mu = \delta \frac{\gamma}{n + m}$$

solve:

$$A\Delta x + \Delta w = \rho$$

$$A^T \Delta y - \Delta z = \sigma$$

$$Z\Delta x + X\Delta z = \mu e - XZe$$

$$W\Delta y + Y\Delta w = \mu e - YWe$$

$$\theta = r \left(\max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\} \right)^{-1} \wedge 1$$

$$x \leftarrow x + \theta \Delta x, \quad w \leftarrow w + \theta \Delta w$$

$$y \leftarrow y + \theta \Delta y, \quad z \leftarrow z + \theta \Delta z$$

}

Recall that we are following the pseudo-code:

Implementation example

Initialize $(x, w, y, z) > 0$ with arbitrary values:

$$x = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}, w = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}, y = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}, z = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

Implementation example

Initialize $(x, w, y, z) > 0$ with arbitrary values:

$$x = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}, w = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}, y = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}, z = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

Then initialize b, c, A and A^T

$$b = \begin{bmatrix} 5 \\ 11 \\ 8 \end{bmatrix}, c = \begin{bmatrix} 5 \\ 4 \\ 3 \end{bmatrix}, A = \begin{bmatrix} 2, 3, 1 \\ 4, 1, 2 \\ 3, 4, 2 \end{bmatrix}, A^T = \begin{bmatrix} 2, 4, 3 \\ 3, 1, 4 \\ 1, 2, 2 \end{bmatrix}$$

Implementation example

$$\rho = b - Ax - w = \begin{bmatrix} 5 \\ 11 \\ 8 \end{bmatrix} - \begin{bmatrix} 2, 3, 1 \\ 4, 1, 2 \\ 3, 4, 2 \end{bmatrix} \cdot \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} - \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 4.3 \\ 10.2 \\ 7 \end{bmatrix}$$

Implementation example

$$\rho = b - Ax - w = \begin{bmatrix} 5 \\ 11 \\ 8 \end{bmatrix} - \begin{bmatrix} 2, 3, 1 \\ 4, 1, 2 \\ 3, 4, 2 \end{bmatrix} \cdot \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} - \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 4.3 \\ 10.2 \\ 7 \end{bmatrix}$$

$$\sigma = c - A^T y + z = \begin{bmatrix} 5 \\ 4 \\ 3 \end{bmatrix} - \begin{bmatrix} 2, 4, 3 \\ 3, 1, 4 \\ 1, 2, 2 \end{bmatrix} \cdot \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 4.2 \\ 3.3 \\ 2.6 \end{bmatrix}$$

Implementation example

$$\rho = b - Ax - w = \begin{bmatrix} 5 \\ 11 \\ 8 \end{bmatrix} - \begin{bmatrix} 2, 3, 1 \\ 4, 1, 2 \\ 3, 4, 2 \end{bmatrix} \cdot \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} - \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 4.3 \\ 10.2 \\ 7 \end{bmatrix}$$

$$\sigma = c - A^T y + z = \begin{bmatrix} 5 \\ 4 \\ 3 \end{bmatrix} - \begin{bmatrix} 2, 4, 3 \\ 3, 1, 4 \\ 1, 2, 2 \end{bmatrix} \cdot \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 4.2 \\ 3.3 \\ 2.6 \end{bmatrix}$$

$$\gamma = z^T x + y^T w = [0.1, 0.1, 0.1] \cdot \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} + [0.1, 0.1, 0.1] \cdot \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = 0.06$$

Implementation example

$$\rho = b - Ax - w = \begin{bmatrix} 5 \\ 11 \\ 8 \end{bmatrix} - \begin{bmatrix} 2, 3, 1 \\ 4, 1, 2 \\ 3, 4, 2 \end{bmatrix} \cdot \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} - \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 4.3 \\ 10.2 \\ 7 \end{bmatrix}$$

$$\sigma = c - A^T y + z = \begin{bmatrix} 5 \\ 4 \\ 3 \end{bmatrix} - \begin{bmatrix} 2, 4, 3 \\ 3, 1, 4 \\ 1, 2, 2 \end{bmatrix} \cdot \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 4.2 \\ 3.3 \\ 2.6 \end{bmatrix}$$

$$\gamma = z^T x + y^T w = [0.1, 0.1, 0.1] \cdot \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} + [0.1, 0.1, 0.1] \cdot \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = 0.06$$

$$\mu = \delta \frac{\gamma}{n + m} = 0.1 \frac{0.06}{3 + 3} = 0.001$$

Where $\delta = 0.1$, $n = 3$ and $m = 3$.

Implementation example

Crucial part, we create our linear system of equations.

$$A\Delta x + \Delta w = \rho \quad (1)$$

$$A^T \Delta y - \Delta z = \sigma \quad (2)$$

$$Z\Delta x + X\Delta z = \mu e - XZe \quad (3)$$

$$W\Delta y + Y\Delta w = \mu e - YWe \quad (4)$$

We define right-hand side first:

$$rhs1 = \begin{bmatrix} 4.3 \\ 10.2 \\ 7 \end{bmatrix}, rhs2 = \begin{bmatrix} 4.2 \\ 3.3 \\ 2.6 \end{bmatrix}, rhs3 = \begin{bmatrix} -0.009 \\ -0.009 \\ -0.009 \end{bmatrix}, rhs4 = \begin{bmatrix} -0.009 \\ -0.009 \\ -0.009 \end{bmatrix}$$

Recall that $X = \begin{bmatrix} 0.1, 0, 0 \\ 0, 0.1, 0 \\ 0, 0, 0.1 \end{bmatrix}$. Similarly for W , Y , and Z .

Implementation example

Left-hand side, we define M1, M2, M3 and M4 that corresponds to the right-hand side.

$$M1 = \begin{bmatrix} 2, 3, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 \\ 4, 1, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0 \\ 3, 4, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0 \end{bmatrix}$$

$$M2 = \begin{bmatrix} 0, 0, 0, 0, 0, 0, 2, 4, 3, -1, -0, -0 \\ 0, 0, 0, 0, 0, 0, 3, 1, 4, -0, -1, -0 \\ 0, 0, 0, 0, 0, 0, 1, 2, 2, -0, -0, -1 \end{bmatrix}$$

$$M3 = \begin{bmatrix} 0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.1, 0, 0 \\ 0, 0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.1, 0 \\ 0, 0, 0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.1 \end{bmatrix}$$

$$M4 = \begin{bmatrix} 0, 0, 0, 0, 0.1, 0, 0, 0.1, 0, 0, 0, 0, 0 \\ 0, 0, 0, 0, 0, 0.1, 0, 0, 0.1, 0, 0, 0, 0 \\ 0, 0, 0, 0, 0, 0, 0.1, 0, 0, 0.1, 0, 0, 0 \end{bmatrix}$$

Implementation example

Using Python 3.10 and numpy 1.26.4, we solve the system using `np.linalg.solve(M, rhs)` to retrieve the delta values:

$$\Delta x = \begin{bmatrix} 2.011 \\ -0.153 \\ 1.147 \end{bmatrix}, \Delta w = \begin{bmatrix} -0.411 \\ 0.015 \\ -0.716 \end{bmatrix}, \Delta y = \begin{bmatrix} 0.321 \\ -0.105 \\ 0.626 \end{bmatrix}, \Delta z = \begin{bmatrix} -2.101 \\ 0.063 \\ -1.237 \end{bmatrix}$$

Then we retrieve θ :

$$\theta = r \left(\max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\} \right)^{-1} \wedge 1$$
$$= 0.043$$

Implementation example

New solution:

$$x = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} + 0.011 \cdot \begin{bmatrix} 2.011 \\ -0.153 \\ 1.147 \end{bmatrix} = \begin{bmatrix} 0.186 \\ 0.093 \\ 0.149 \end{bmatrix}$$

$$w = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} + 0.011 \cdot \begin{bmatrix} -0.411 \\ 0.015 \\ -0.716 \end{bmatrix} = \begin{bmatrix} 0.082 \\ 0.100 \\ 0.069 \end{bmatrix}$$

$$y = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} + 0.011 \cdot \begin{bmatrix} 0.321 \\ -0.105 \\ 0.626 \end{bmatrix} = \begin{bmatrix} 0.114 \\ 0.095 \\ 0.127 \end{bmatrix}$$

$$z = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} + 0.011 \cdot \begin{bmatrix} -2.101 \\ 0.063 \\ -1.237 \end{bmatrix} = \begin{bmatrix} 0.01 \\ 0.103 \\ 0.047 \end{bmatrix}$$

Implementation example

Is this the optimal solution?

Primal feasibility:

$$\|\rho\|_1 = 20.58$$

Dual feasibility:

$$\|\sigma\|_1 = 9.67$$

Complementarity:

$$\gamma = 0.068$$

As $\|\rho\|_1 \geq \epsilon$, $\|\sigma\|_1 \geq \epsilon$ and $\gamma \geq \epsilon$, we have not found an optimal solution yet. Therefore, we continue on our second iteration with the new x , w , y and z values.

Code

```
r = 0.9
delta = 0.1_# delta is a parameter, but we set it to 0.1 since it works well in practice
tolerance = 0.01
sol = (x, w, y, z)
e = np.ones(len(x))
while True:
    rho = b - np.dot(A, x) - w
    sigma = c - np.dot(A_T, y) + z
    gamma = np.dot(z_t, x) + np.dot(y_t, w)
    mu = delta*(gamma/(len(x)+len(y)))

    # Construct linear system of equation
    rhs1 = rho
    rhs2 = sigma
    rhs3 = mu * np.ones(X.shape[0]) - X @ Z @ np.ones(X.shape[0])
    rhs4 = mu * np.ones(Y.shape[0]) - Y @ W @ np.ones(Y.shape[0])

    n = len(x)
    m = len(y)
    p = len(w)
    q = len(z)
```


Code

```
zero = np.zeros((n, n))
M1 = np.hstack([A, np.eye(A.shape[0]), np.zeros((m, n)), np.zeros((m, n))])
M2 = np.hstack([zero, zero, A.T, -np.eye(n)])
M3 = np.hstack([Z, zero, zero, X])
M4 = np.hstack([zero, Y, W, zero])

M = np.vstack((M1, M2, M3, M4))
rhs = np.concatenate((rhs1, rhs2, rhs3, rhs4))

deltas = np.linalg.solve(M, rhs)

delta_x = deltas[:n]
delta_w = deltas[n:2*n]
delta_y = deltas[2*n:3*n]
delta_z = deltas[3*n:]

max_ratio = max(np.max(-delta_x / x), np.max(-delta_w / w), np.max(-delta_y / y), np.max(-delta_z / z))
inverse_max_ratio = 1 / max_ratio
theta = r * inverse_max_ratio

theta = min(theta, 1)
print(norm(rho, ord=1))
print(norm(sigma, ord=1))
print(gamma)
if norm(rho, ord=1) < tolerance and norm(sigma, ord=1) < tolerance and gamma < tolerance:
    break
```

Code

```
x = x + theta*delta_x  
w = w + theta*delta_w  
y = y + theta*delta_y  
z = z + theta*delta_z  
  
sol = (x, w, y, z)  
print(f'Sol: ', sol)
```

Table of Contents

Barrier Function

Path-Following Method

Implementation example

The KKT System

The KKT System

Given a system of equations:

$$A\Delta x + \Delta w = \rho \quad (5)$$

$$A^T \Delta y - \Delta z = \sigma \quad (6)$$

$$Z\Delta x + X\Delta z = \mu e - XZe \quad (7)$$

$$W\Delta y + Y\Delta w = \mu e - YWe \quad (8)$$

The KKT System

Given a system of equations:

$$A\Delta x + \Delta w = \rho \quad (5)$$

$$A^T \Delta y - \Delta z = \sigma \quad (6)$$

$$Z\Delta x + X\Delta z = \mu e - XZe \quad (7)$$

$$W\Delta y + Y\Delta w = \mu e - YWe \quad (8)$$

In the previous example, it might be possible to solve a linear system of equations in a small problem. But what if the problem is larger? We transform the system into a **symmetric** linear system in matrix form:

$$\left[\begin{array}{cc|cc} -XZ^{-1} & & -I & \\ & & A & I \\ \hline -I & A^T & & \\ & I & & YW^{-1} \end{array} \right] \begin{bmatrix} \Delta z \\ \Delta y \\ \Delta x \\ \Delta w \end{bmatrix} = \begin{bmatrix} -\mu Z^{-1}e + x \\ \sigma \\ \mu W^{-1}e - y \end{bmatrix}$$

This is the Karush-Kuhn-Tucker system (KKT system)

The Reduced KKT System

The KKT system can be reduced further. We solve for Δz and Δw in equations 7 and 8.

$$\Delta z = X^{-1}(\mu e - XZ\epsilon - Z\Delta x)$$

$$\Delta w = Y^{-1}(\mu e - YW\epsilon - W\Delta y)$$

Then we substitute Δz and Δw into equations 5 and 6.

$$A\Delta x - Y^{-1}W\Delta y = \rho - \mu Y^{-1}e + w$$

$$A^T\Delta y - X^{-1}Z\Delta x = \sigma + \mu X^{-1}e - z$$

This gives us the following **reduced KKT System**.

The Reduced KKT System

$$\begin{bmatrix} -Y^{-1}W & A \\ A^T & X^{-1}Z \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x \end{bmatrix} = \begin{bmatrix} b - Ax - \mu Y^{-1}e \\ c - A^T y + \mu X^{-1}e \end{bmatrix}$$

The Reduced KKT System is still symmetric. However, we can keep reducing the system into normal equations.

Normal Equations

Given the reduced KKT System:

$$A\Delta x - Y^{-1}W\Delta y = \rho - \mu Y^{-1}e + w \quad (9)$$

$$A^T\Delta y - X^{-1}Z\Delta x = \sigma + \mu X^{-1}e - z \quad (10)$$

We solve for Δy in equation 9 and eliminate it from 10 OR

We solve for Δx in equation 10 and eliminate it from 9. We choose the latter.

$$\Delta x = -XZ^{-1}(c - A^T y + \mu X^{-1}e - A^T \Delta y)$$

Then we eliminate Δx :

$$\begin{aligned} -(Y^{-1}W + AXZ^{-1}A^T)\Delta y &= b - Ax - \mu Y^{-1}e \\ &\quad - AXZ^{-1}(c - A^T y + \mu X^{-1}e) \end{aligned}$$

Normal Equations

- ▶ This gives us a system of normal equations in primal form.
- ▶ Similarly, if we choose the former option, we get a system of normal equations in dual form.
- ▶ Problem: If A has a dense column, then we end up with a dense matrix which is difficult to solve in primal form.
- ▶ Same problem if A has a dense row, which is also difficult to solve in dual form.
- ▶ Should we then use the reduced KKT matrix? Possible if the matrices are positive definite!