# A First Course in Linear Optimization

— a dynamic book —

*by*
Jon Lee

Second Edition (Version 2.95)

REEX PRESS

Jon Lee

2013, 2014, 2015, 2016

# Go Forward

This is a book on linear optimization, written in LaTeX. I started it, aiming it at the course IOE 510, a masters-level course at the University of Michigan. Use it as is, or adapt it to your course! It is an ongoing project. It is alive! It can be used, modified (the LaTeX source is available) and redistributed as anyone pleases, subject to the terms of the Creative Commons Attribution 3.0 Unported License (CC BY 3.0) . Please take special note that you can *share* (copy and redistribute in any medium or format) and *adapt* (remix, transform, and build upon for any purpose, even commercially) this material, but you must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests that I endorse you or your use. If you are interested in endorsements, speak to my agent.


I started this material, but I don't control so much what you do with it. Control is sometimes overrated — and I am a control freak, so I should know!

I hope that you find this material useful. If not, I am happy to refund what you paid to me.

Jon Lee
University of Michigan
Ann Arbor, Michigan
started March 2013

# Preface

This book is a treatment of linear optimization meant for students who are reasonably comfortable with matrix algebra (or willing to get comfortable rapidly). It is *not* a goal of mine to teach anyone how to solve small problems by hand. My goals are to introduce: (i) the mathematics and algorithmics of the subject at a beginning mathematical level, (ii) algorithmically-aware modeling techniques, and (iii) high-level computational tools for studying and developing optimization algorithms (in particular, `MATLAB` and `AMPL`).

Proofs are given when they are important in understanding the algorithmics. I make free use of the inverse of a matrix. But it should be understood, for example, that $B^{-1}b$ is meant as a mathematical expression for the solution of the square linear system of equations $Bx = b$ . I am not in any way suggesting that an efficient way to calculate the solution of a large (often sparse) linear system is to calculate an inverse! Also, I avoid the dual simplex algorithm (e.g., rather deftly in describing branch-and-bound), preferring to just think about the ordinary simplex algorithm applied to the dual problem. Again, my goal is not to describe the most efficient way to do matrix algebra!

Illustrations are woefully few. Though if Lagrange could not be bothered[1], who am I to aim higher? Still, some of the algorithms are *illustrated* in the modern way, with computer code.

The material that I present was mostly well known by the 1960's. As a student at Cornell in the late 70's and early 80's, I learned and got excited about linear optimization from Bob Bland, Les Trotter and Lou Billera, using [1] and [3]. The present book is a treatment of some of that material, with additional material on integer-linear optimization, mostly which I originally learned from George Nemhauser and Les. There is nothing here on interior-point algorithms and the ellipsoid algorithm; don't tell Mike Todd!

Jon Lee
University of Michigan
Ann Arbor, Michigan
started March 2013

(or maybe really in Ithaca, NY in 1979)

# Serious Acknowledgments

Throw me some funding for this project, and I will acknowledge you. Seriously!

Many of the pictures in this book were found floating around on the web. I am making *"fair use"* of them as they float through this document. Of course, I gratefully acknowledge those who own them.

Hearty thanks to many students and to Prof. Siqian Shen for pointing out typos in an earlier version.

# Dedication

For students *(even Ohio students)*. Not for publishers — not this time. Maybe next time.

# The Nitty Gritty

You can always get all released editions of this book (in `.pdf` format) from my web page and the materials to produce them (LaTeX source, etc.) from me.

If you decide that you need to recompile the LaTeX to change something, then you should be aware of two things. First, there is no user's manual nor help desk. Second, I use a lot of include files, for incorporating code listings and output produced by software like `Matlab` and `AMPL`. Make sure that you understand where everything is pulled from if you recompile the LaTeX. In particular, if you change *anything* in the directory that the LaTeX file is in, something can well change in the `.pdf` output. For example, `Matlab` and `AMPL` scripts are pulled into the book, and likewise for the output of those scripts. If you want to play, and do not want to change those parts of the book, play in another directory.

I make significant use of software. Everything seems to work with:

```
MATLAB R2016b
AMPL 20161130
CPLEX 12.6.2.0
Mathematica 11.0.1.0
WinEdt 10.1
MiKTeX 2.9
```

Use of older versions is inexcusable. Newer versions will surely break things. Nonetheless, if you can report success or failure on newer versions, please let me know.

I use lots of LaTeX packages (which, as you may know, makes things rather fragile). I could not possibly gather the version numbers of those — I do have a day job! (but I do endeavor to keep my packages up to date).

# Contents

# Chapter 1

# Let's Get Started


Our main goals in this chapter are as follows:

- Introduce some terminology associated with linear optimization.
- Describe elementary techniques for transforming any linear-optimization problems to one in a 'standard form.'
- Introduce the Weak Duality Theorem.
- Review ideas from linear algebra that we will make use of later.

## 1.1 Linear Optimization and Standard Form

**Linear optimization** is the study of the mathematics and algorithms associated with minimizing or maximizing a real linear **objective function** of a finite number of real variables, subject to a finite number of **linear constraints**, each being that a real linear function on these variables be $=$ , $\leq$ or $\geq$ a constant. A **polyhedron** is the solution

set of a finite number of linear constraints; so we are studying optimization of a linear function on a polyhedron.

**Definition 1.1**
A **solution** of a linear-optimization problem is an assignment of real values to the variables. A solution is **feasible** if it satisfies the linear constraints. A solution is **optimal** if there is no feasible solution with better objective value. The set of feasible solutions (which is a polyhedron) is the **feasible region**.

It is convenient to put a general linear-optimization problem into a **standard form**

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,, \end{array}$$

where $c \in \mathbb{R}^n$ , $b \in \mathbb{R}^m$ , $A \in \mathbb{R}^{m\times n}$ has *full row rank* $m$ , and $x$ ia a vector of variables in $\mathbb{R}^n$ . That is, *minimization* of a linear function on a finite number number of non-negative real variables, subject to a *non-redundant and consistent* system of linear equations. Note that even though the system of equations, $Ax = b$, has a solution, the problem may not have a feasible solution.

Through a finite sequence of simple transformations, every linear-optimization problem can be brought into an equivalent one in standard form. Specifically, we can apply any of the follow steps, as needed, in the order presented.

- The maximum of $c'x$ is the same as the negative of the minimum of $-c'x$ .
- We can replace any unrestricted variable $x_j$ with the difference of a pair of non-negative variables $x_j^+$ and $x_j^-$ . That is, substituting $x_j^+ - x_j^-$ for $x_j$ . In this way, we can make all variables constrained to be non-negative.
- Next, if we have an inequality $\sum_{j=1}^n \alpha_j x_j \leq \gamma$ , we simply replace it with $\sum_{j=1}^n \alpha_j x_j + s = \gamma$ , where a real **slack variable** $s$ is introduced which is constrained to be non-negative. Similarly, we can replace $\sum_{j=1}^n \alpha_j x_j \geq \gamma$ with $\sum_{j=1}^n \alpha_j x_j - s = \gamma$ , where a real **surplus variable** $s$ is introduced which is constrained to be non-negative.
- Applying these transformations as needed results in a standard-form problem, except possibly for the condition that the matrix of coefficients of the systems of equations have full row rank. But we can realize this last condition by carrying out elementary row operations on the system of equations, resulting in the elimination of any redundant equations or the identification that the system of equations is inconsistent. In the latter case, the linear-optimization problem is infeasible.

## 1.2 A Standard-Form Problem and its Dual

Let $c \in \mathbb{R}^n$ , $b \in \mathbb{R}^m$ , and $A \in \mathbb{R}^{m\times n}$ . Let $x$ be a vector of variables in $\mathbb{R}^n$ . Consider the standard-form problem

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,. \end{array} \qquad \text{(P)}$$

Let $y$ be a vector of variables in $\mathbb{R}^m$ , and consider the linear-optimization problem

$$\begin{array}{lll} \max & y'b & \\ & y'A & \leq \quad c' \ . \end{array} \tag{D}$$

It is worth emphasizing that (P) and (D) are both defined from the same data $A$ , $b$ and $c$ . We have the following very simple but key result, relating the objective values of feasible solutions of the two linear-optimization problems.

**Theorem 1.2 (Weak Duality Theorem)**
If $\hat{x}$ is feasible in (P) and $\hat{y}$ is feasible in (D), then $c'\hat{x} \geq \hat{y}'b$ .

*Proof.*

$$c'\hat{x} \geq \hat{y}'A\hat{x} \ ,$$

because $\hat{y}'A \leq c'$ (feasibility of $\hat{y}$ in (D)) and $\hat{x} \geq \mathbf{0}$ (feasibility of $\hat{x}$ in (P)). Furthermore

$$\hat{y}'A\hat{x} = \hat{y}'b \ ,$$

because $A\hat{x} = b$ (feasibility of $\hat{x}$ in (P)). The result follows. □

## 1.3 Linear-Algebra Review


For a matrix $A \in \mathbb{R}^{m\times n}$ , we denote the entry in row $i$ and column $j$ as $a_{ij}$ . For a matrix $A \in \mathbb{R}^{m\times n}$ , we denote the transpose of $A$ by $A' \in \mathbb{R}^{n\times m}$ . That is, the entry in row $i$ and column $j$ of $A'$ is $a_{ji}$ .

Except when we state clearly otherwise, vectors are "column vectors." That is, we can view a vector $x \in \mathbb{R}^n$ as a matrix in $\mathbb{R}^{n\times 1}$ . Column $j$ of $A$ is denoted by $A_{\cdot j} \in \mathbb{R}^m$ . Row $i$ of $A$ is denoted by $A_{i\cdot}$ , and we view its transpose as a vector in $\mathbb{R}^n$ . We will have far greater occasion to reference columns of matrices rather than rows, so we will often write $A_j$ as a shorthand for $A_{\cdot j}$ , so as to keep notation less cluttered.

For matrices $A \in \mathbb{R}^{m\times p}$ and $B \in \mathbb{R}^{p\times n}$ , the **(matrix) product** $AB \in \mathbb{R}^{m\times n}$ is defined to be the matrix having $\sum_{k=1}^{p} a_{ik}b_{kj}$ as the entry in row $i$ and column $j$ . Note that for the product $AB$ to make sense, the number of columns of $A$ and the number of rows of $B$ must be identical. It is important to emphasize that matrix multiplication is associative; that is, $(AB)C = A(BC)$ , and so we can always unambiguously write the

product of any number of matrices without the need for any parentheses. Also, note that the product and transpose behave nicely together. That is, $(AB)' = B'A'$ .

The **dot product** or **scalar product** of vectors $x, z \in \mathbb{R}^n$ is the scalar $\langle x, z \rangle := \sum_{j=1}^{n} x_j z_j$ , which we can equivalently see as $x'z$ or $z'x$ , allowing ourselves to consider a $1 \times 1$ matrix to be viewed as a scalar. Thinking about matrix multiplication again, and freely viewing columns as vectors, the entry in row $i$ and column $j$ of the product $AB$ is the dot product $\langle (A_{i\cdot})', B_{\cdot j} \rangle$ .

Matrix multiplication extends to "block matrices" in a straightforward manner. If

$$A := \left(\begin{array}{c|c|c} A_{11} & \cdots & A_{1p} \\ \hline A_{21} & \cdots & A_{2p} \\ \hline \vdots & \ddots & \vdots \\ \hline A_{m1} & \cdots & A_{mp} \end{array}\right) \quad \text{and} \quad B := \left(\begin{array}{c|c|c} B_{11} & \cdots & B_{1n} \\ \hline B_{21} & \cdots & B_{2n} \\ \hline \vdots & \ddots & \vdots \\ \hline B_{p1} & \cdots & B_{pn} \end{array}\right),$$

where each of the $A_{ij}$ and $B_{ij}$ are matrices, and we assume that for all $i$ and $j$ the number of columns of $A_{ik}$ agrees with the number of rows of $B_{kj}$ , then

$$AB = \left(\begin{array}{c|c|c} \sum_{k=1}^{p} A_{1k}B_{k1} & \cdots & \sum_{k=1}^{p} A_{1k}B_{kn} \\ \hline \sum_{k=1}^{p} A_{2k}B_{k1} & \cdots & \sum_{k=1}^{p} A_{2k}B_{kn} \\ \hline \vdots & \ddots & \vdots \\ \hline \sum_{k=1}^{p} A_{mk}B_{k1} & \cdots & \sum_{k=1}^{p} A_{mk}B_{kn} \end{array}\right).$$

That is, block $i, j$ of the product is $\sum_{k=1}^{p} A_{ik}B_{kj}$ , and $A_{ik}B_{kj}$ is understood as ordinary matrix multiplication.

For vectors $x^1, x^2, \ldots, x^p \in \mathbb{R}^n$ , and scalars $\lambda_1, \lambda_2, \ldots, \lambda_p$ , $\sum_{i=1}^{p} \lambda_i x^i$ is a **linear combination** of $x^1, x^2, \ldots, x^p$ . The linear combination is **trivial** if all $\lambda_i = 0$ . The vectors $x^1, x^2, \ldots, x^p \in \mathbb{R}^n$ are **linearly independent** if the only representation of the zero vector in $\mathbb{R}^n$ as a linear combination of $x^1, x^2, \ldots, x^p$ is trivial. The set of all linear combinations of $x^1, x^2, \ldots, x^p$ is the vector-space **span** of $\{x^1, x^2, \ldots, x^p\}$ . The **dimension** of a vector space $V$ , denoted $\dim(V)$ , is the maximum number of linearly-independent vectors in it. Equivalently, it is the minimum number of vectors needed to span the space.

A set of $\dim(V)$ linearly-independent vectors that spans a vector space $V$ is a **basis** for $V$ . If $V$ is the vector-space span of $\{x^1, x^2, \ldots, x^p\}$ , then there is a subset of $\{x^1, x^2, \ldots, x^p\}$ that is a basis for $V$ . Moreover, every linearly-independent subset of of such a $V$ can be extended to a basis for $V$ using vectors from $x^1, x^2, \ldots, x^p$ .

The span of the rows of a matrix $A \in \mathbb{R}^{m \times n}$ is the **row space** of $A$ , denoted $\text{r.s.}(A) := \{y'A \ : \ y \in \mathbb{R}^m\}$ . Similarly, the span of the columns of a matrix $A$ is the **column space** of $A$ , denoted $\text{c.s.}(A) := \{Ax \ : \ x \in \mathbb{R}^n\}$ . It is a simple fact that, for a matrix $A$ , the dimension of its row space and the dimension of its column space are identical, this common number being called the **rank** of $A$ . The matrix $A$ has **full row rank** if its number of rows is equal to its rank. That is, if its rows are linearly independent.

Similarly, the matrix $A$ has **full column rank** if its number of columns is equal to its rank. That is, if its columns are linearly independent.

Besides the row and columns spaces of a matrix $A \in \mathbb{R}^{m\times n}$ , there is another very important vector space associated with $A$ . The **null space** of $A$ is the set of vectors having $0$ dot product with all rows of $A$ , denoted n.s.$(A) := \{x \in \mathbb{R}^n \ : \ Ax = \mathbf{0}\}$ .

An important result is the following theorem relating the dimensions of the row and null spaces of a matrix.

**Theorem 1.3 (Rank-Nullity Theorem)**
If $A$ is a matrix with $n$ columns, then

$$\dim(\text{r.s.}(A)) + \dim(\text{n.s.}(A)) = n \ .$$

There are some simple operations on a matrix that preserve its row and null spaces. The following operations are **elementary row operations**:

1. multiply a row by a non-zero scalar;
2. interchange a pair of rows;
3. add a scalar multiple of a row to another row;
4. delete a row that is identically zero.

There is one more operation that we allow, which is really one of convenience rather than mathematics. It is convenient to be able to permute *columns* while also permuting the corresponding column indices. That is, if $A \in \mathbb{R}^{m\times n}$ , we regard the columns as labeled, *in left-to-right order*: $1, 2, \ldots, n$ . So we have

$$A = [A_1, A_2, \ldots, A_n] \ .$$

It can be convenient to have a permutation $\sigma_1, \sigma_2, \ldots, \sigma_n$ of $1, 2, \ldots, n$ , and then write

$$[A_{\sigma_1}, A_{\sigma_2}, \ldots, A_{\sigma_n}] \ .$$

This matrix is really equivalent to $A$ , because we regard its columns as labeled by $\sigma_1, \sigma_2, \ldots, \sigma_n$ rather than $1, 2, \ldots, n$ . Put another way, when we write a matrix, the order of the columns is at our convenience, but the labels of columns is determined by the order that we choose for placing the columns.

The **identity matrix** $\mathbf{I}_r$ in $\mathbb{R}^{r\times r}$ is the matrix having 1 as every diagonal element and 0 as every off-diagonal element. Via elementary row operations, any matrix $A$ , that is not all zero, can be transformed, via elementary row operations, into one of the form

$$[\mathbf{I}_r, M] \ .$$

Using corresponding operations on the associated system of equations, this is known as **Gauss-Jordan elimination**.

For an $r \times r$ matrix $B$ of rank $r$, there is a unique $r \times r$ matrix "$B^{-1}$" such that $B^{-1}B = \mathbf{I}_r$. For this reason, such a matrix $B$ is called **invertible**, and $B^{-1}$ is called the **inverse** of $B$. According to the definition, $B^{-1}B = \mathbf{I}_r$, but we also have $BB^{-1} = \mathbf{I}_r$. Also, $(B')^{-1} = (B^{-1})'$, and if $A$ and $B$ are both invertible, then $(AB)^{-1} = B^{-1}A^{-1}$.

Noting that,

$$B^{-1}[B, \mathbf{I}_r] = [\mathbf{I}_r, B^{-1}] \, ,$$

we see that there is a nice way to compute the inverse of a matrix $B$ using elementary row operations. That is, we perform elementary row operations on

$$[B, \mathbf{I}_r]$$

so that we have the form

$$[\mathbf{I}_r, M],$$

and the resulting matrix $M$ is $B^{-1}$.

Next, we define the **determinant** of a square $r \times r$ matrix $B$, which we denote $\det(B)$. We define the determinant in a non-standard but useful manner, via a recursive formula known as **Laplace expansion**.[2]

If $r = 1$, then $B = (b_{11})$, and we define $\det(B) := b_{11}$. For $r > 1$, choose any fixed column $j$ of $B$, and we define

$$\det(B) = \sum_{i=1}^{r} (-1)^{i+j} \, b_{ij} \det(B^{ij}) \, ,$$

where $B^{ij}$ is the $(r-1) \times (r-1)$ matrix obtained by deleting row $i$ and column $j$ of $B$. It is a fact that this is well defined — that is, the value of $\det(B)$ does not depend on the choice of $j$ (taken at each step of the recursion). Moreover, we have $\det(B') = \det(B)$, so we can could as well choose any fixed *row* $i$ of $B$, and we have

$$\det(B) = \sum_{j=1}^{r} (-1)^{i+j} \, b_{ij} \det(B^{ij}) \, ,$$

resulting in the same value for $\det(B)$.

An interesting observation links $\det(B)$ with elementary row operations. Consider performing elementary row operations on

$$[B, \mathbf{I}_r]$$

to obtain

$$[\mathbf{I}_r, B^{-1}] \, .$$

As we carry out the elementary row operations, we sometimes multiply a row by a non-zero scalar. If we accumulate the product of all of these multipliers, the result is $\det(B^{-1})$; equivalently, the reciprocal is $\det(B)$.

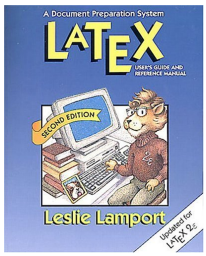
Finally, for an invertible $r \times r$ matrix $B$ and a vector $b$, we can express the unique solution $\bar{x}$ of the system $Bx = b$, via a formula involving determinants. **Cramer's rule** is the following formula:

$$\bar{x}_j = \frac{\det(B(j))}{\det(B)} \, , \text{ for } j = 1, 2, \ldots, r \, ,$$

where $B(j)$ is defined to be the matrix $B$ with its $j$-th column replaced by $b$. It is worth emphasizing that direct application of Cramer's rule is not to be thought of as a useful algorithm for computing the solution of a system of equations. But it can be very useful to have in the proof toolbox.[3]

## 1.4 Exercises

**Exercise 1.0 (Learn LaTeX)**
**Learn to use LaTeX for writing *all* of your homework solutions.** Personally, I use MiKTeX, which is an implementation of LaTeX for Windows. Specifically, within MiKTeX, I am using pdfLaTeX (it only matters for certain things like including graphics and also pdf into a document). I find it convenient to use the editor WinEdt, which is very LaTeX friendly. A good book on LaTeX is




In A.1 there is a template to get started. Also, there are plenty of tutorials and beginner's guides on the web.

**Exercise 1.1 (Convert to standard form)**
Give an original example (i.e., with actual numbers) to demonstrate that you know how to transform a general linear-optimization problem to one in standard form.

**Exercise 1.2 (Weak Duality example)**
Give an original example to demonstrate the Weak Duality Theorem.

**Exercise 1.3 (Convert to $\leq$ form)**
Describe a general recipe for transforming an arbitrary linear-optimization problem into one in which all of the linear constraints are of $\leq$ type.

**Exercise 1.4 ($m + 1$ inequalities)**
Prove that the system of $m$ equations in $n$ variables $Ax = b$ is equivalent to the system $Ax \leq b$ augmented by only *one* additional linear inequality — that is, a total of only $m + 1$ inequalities.

**Exercise 1.5 (Weak duality for another form)**
Give and prove a Weak Duality Theorem for

$$\begin{array}{rrcl} \max & c'x & & \\ & Ax & \leq & b\,; \\ & x & \geq & \mathbf{0}\,. \end{array} \qquad \text{(P}'\text{)}$$

HINT: Convert (P′) to a standard-form problem, and then apply the ordinary Weak Duality Theorem for standard-form problems.

**Exercise 1.6 (Weak duality for a complicated form)**
Give and prove a Weak Duality Theorem for

$$\begin{array}{lccccc} \min & c'x & + & f'w & & \\ & Ax & + & Bw & \leq & b\,; \\ & Dx & & & = & g\,; \\ & x \geq \mathbf{0} & & w \leq \mathbf{0} & & . \end{array} \qquad \text{(P}'\text{)}$$

HINT: Convert (P′) to a standard-form problem, and then apply the ordinary Weak Duality Theorem for standard-form problems.

**Exercise 1.7 (Weak duality for a complicated form — with MATLAB)**
The MATLAB code below makes and solves an instance of (P′) from Exercise 1.6. *Study the code to see how it is works.* Now, extend the code to solve the dual of (P′). Also, after converting (P′) to standard form (as indicated in the HINT for Exercise 1.6), use MATLAB to solve that problem and its dual. Make sure that you get the same optimal value for all of these problems.

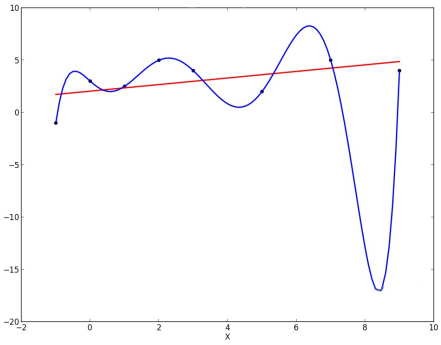
---

```
% DualityWithMatlab1.m // Jon Lee
%

n1=7
n2=15
m1=2
m2=4

rng('default');
rng(1);  % set seed

A = rand(m1,n1);
B = rand(m1,n2);
D = rand(m2,n1);

% Organize the situation
%   so that the problem has a feasible solution
x = rand(n1,1);
w = -rand(n2,1);
b = A*x + B*w + 0.01 * rand(m1,1);
g = D*x;

% Organize the situation
%   so that the dual problem has a feasible solution
y = -rand(m1,1);
pi = rand(m2,1) - rand(m2,1);
c = A'*y + D'*pi + 0.01 * rand(n1,1);
f = B'*y - 0.01 * rand(n2,1);

% Here is how the 'linprog' function works:
%
% [v,z,exitflag] = linprog(c,A,b,Aeq,beq,lb,ub)
%   minimizes c'v ,
%   with constraints A v <= b, Aeq v = beq, and
```

```
%   variable bounds lb <= v <=ub.
%   Some parts of the model can be null: for example,
%   set Aeq = [] and beq = [] if no equalities exist.
%   set ub = [] if no upper bounds exist.
%
% [v,z,exitflag] = linprog(...) returns values:
%     v = solution vector
%     z = minimum objective value
%     exitflag = describes the exit condition:
%        1 optimal solution found.
%       -2 problem is infeasible.
%       -3 problem is unbounded.
%       -5 problem and its dual are both infeasible

% Study the following part carefully. This is how we set up
% a block problem. [c ; f] stacks vertically the vectors c and f.
% [A B] concatenates A and B horizontally. zeros(m2,n2) is an
% m2-by-n2 matrix of 0's. ones(n1,1) is an n1-vector of 1's, and
% Inf is +infinity, so Inf*ones(n1,1) is a vector of length n1
% with all components equal to +infinity. In this way we have
% infinite upper bounds on some variables.
[v,z,exitflag] = linprog([c ; f],[A B],b,[D zeros(m2,n2)],g, ...
    [zeros(n1,1);-Inf*ones(n2,1)],[Inf*ones(n1,1);zeros(n2,1)])

if (exitflag < 1)
     disp('fail 1: LP did not have an optimal solution');
     return;
end;

% Extract the parts of the solution v to the formulation vectors
x = v(1:n1,1)
w = v(n1+1:n1+n2,1)
```

# Chapter 2

# Modeling




Our goals in this chapter are as follows:

- Learn some basic linear-optimization modeling techniques.
- Learn how to use an optimization modeling language like `AMPL`.

## 2.1 A Production Problem

We suppose that a company has $m$ resources, available in quantities $b_i$ , $i = 1, 2, \ldots, m$ , and $n$ production activities, with per-unit profits $c_j$ , $j = 1, 2, \ldots, n$ . Each unit of activity $j$ consumes $a_{ij}$ units of resources $i$ . Each production activity can be carried out

at any non-negative level, as long as the resources availabilities are respected. We assume that any unused resource quantities have no value and can be disposed of at no cost. The problem is to find a profit-maximizing production plan. We can formulate this problem as the linear-optimization problem

$$\begin{array}{lrcl} \max & c'x & & \\ & Ax & \leq & b\,; \\ & x & \geq & \mathbf{0}\,, \end{array} \tag{P}$$

where $b := (b_1, b_2, \ldots, b_m)'$ , $c := (c_1, c_2, \ldots, c_n)'$ , $A \in \mathbb{R}^{m\times n}$ is the matrix of $a_{ij}$, and $x$ is a vector of variables in $\mathbb{R}^n$ .

From the very same data, we can formulate a related linear-optimization problem. The goal now is to set per-unit prices $y_i$ , for the resources $i = 1, 2, \ldots, m$ . The total cost of purchasing the resources from the company is then $y'b$ , and we wish to minimize the total cost of obtaining the resources from the company. We want to set these prices in such a way that the company would never have an incentive to carry out any of the production activities versus selling the resources at the associated resources at these prices. That is, we require that $\sum_{i=1}^{m} y_i a_{ij} \geq c_j$ , for $j = 1, 2, \ldots, n$ . Because of our assumption that the company can dispose of any unused quantities of resources at no cost, we have $y_i \geq 0$ , for $i = 1, 2, \ldots, m$ . All in all, we have the linear-optimization problem

$$\begin{array}{lrcl} \min & y'b & & \\ & y'A & \geq & c'\,; \\ & y & \geq & \mathbf{0}\,, \end{array} \tag{D}$$

Comparing this pair of linear-optimization problem with what you discovered in Exercise 1.5, we see that a Weak Duality Theorem holds: that is, the profit of any feasible production plan is bounded above by the cost of the resources determined by any set of prices that would render all production activities non-profitable.

## 2.2 Norm Minimization


"Norms" are very useful as a measure of the "size" of a vector. In some applications, we are interested in making the "size" small. There are many different "norms"

(for example, the Euclidean norm), but two are particularly interesting for linear optimization.

For $x \in \mathbb{R}^n$ , the **$\infty$-norm** (or **max-norm**) of $x$ is defined as

$$\|x\|_\infty := \max\{|x_j| \ : \ j = 1, 2, \ldots, n\}.$$

We would like to formulate the problem of finding an $\infty$-norm minimizing solution of the system of equations $Ax = b$ . This is quite easy, via the linear-optimization problem:

$$\begin{array}{lrrcl} \min & t & & & \\ & t & - & x_i & \geq 0 \ , \ i = 1, 2, \ldots, n \ ; \\ & t & + & x_i & \geq 0 \ , \ i = 1, 2, \ldots, n \ ; \\ & & & Ax & = b \ , \end{array}$$

where $t \in \mathbb{R}$ is an auxiliary variable. Notice how the minimization "pressure" ensures that an optimal solution $(\hat{x}, \hat{t})$ has $\hat{t} = \max_{j=1}^{n}\{|\hat{x}_j|\} = \|\hat{x}\|_\infty$ . This would not work for maximization!

The **1-norm** of $x$ is defined as

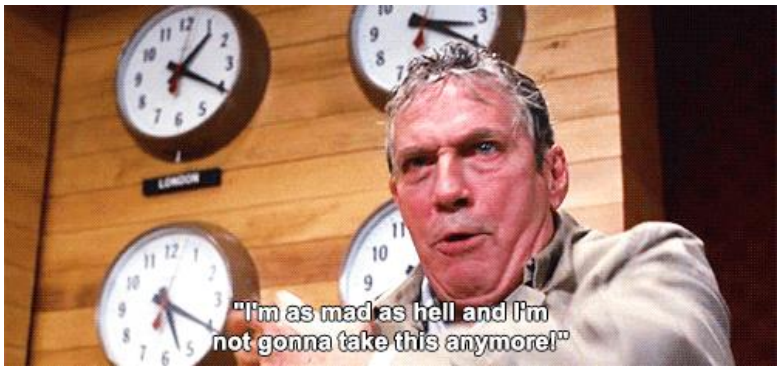
$$\|x\|_1 := \sum_{j=1}^{n} |x_j| \ .$$

Now, we would like to formulate the problem of finding a 1-norm minimizing solution of the system of equations $Ax = b$ . This is quite easy, via the linear-optimization problem:

$$\begin{array}{lrrcl} \min & \sum_{j=1}^{n} t_j & & & \\ & t_j & - & x_j & \geq 0 \ , \ j = 1, 2, \ldots, n \ ; \\ & t_j & + & x_j & \geq 0 \ , \ j = 1, 2, \ldots, n \ ; \\ & & & Ax & = b \ , \end{array}$$

where $t \in \mathbb{R}^n$ is a vector of $n$ auxiliary variables. Notice how the minimization "pressure" ensures that an optimal solution $(\hat{x}, \hat{t})$ has $\hat{t}_j = |\hat{x}_j|$ , for $j = 1, 2, \ldots, n$ (again, this would not work for maximization!), and so we will have $\sum_{j=1}^{n} \hat{t}_j = \|\hat{x}\|_1$ .

## 2.3 Network Flow



A finite **network** $G$ is described by a finite set of **nodes** $\mathcal{N}$ and a finite set $\mathcal{A}$ of **arcs**. Each arc $e$ has two key attributes, namely its **tail** $t(e) \in \mathcal{N}$ and its **head** $h(e) \in \mathcal{N}$. We think of a (single) commodity as being allowed to "flow" along each arc, from its tail to its head. Indeed, we have "flow" variables

$$x_e := \text{ amount of flow on arc } e \,,$$

for $e \in \mathcal{A}$. Formally, a **flow** $\hat{x}$ on $G$ is simply an assignment of *any* real numbers $\hat{x}_e$ to the variables $x_e$, for $e \in \mathcal{A}$. We assume that flow on arc $e$ should be non-negative and should not exceed

$$u_e := \text{ the flow upper bound on arc } e \,,$$

for $e \in \mathcal{A}$. Associated with each arc $e$ is a cost

$$c_e := \text{ cost per-unit-flow on arc } e \,,$$

for $e \in \mathcal{A}$. The (total) **cost** of the flow $\hat{x}$ is defined to be

$$\sum_{e \in \mathcal{A}} c_e x_e \,.$$

We assume that we have further data for the nodes. Namely,

$$b_v := \text{ the net supply at node } v \,,$$

for $v \in \mathcal{N}$. A flow is **conservative** if the net flow out of node $v$, minus the net flow into node $v$, is equal to the net supply at node $v$, for all nodes $v \in \mathcal{N}$.

The **(single-commodity min-cost) network-flow problem** is to find a minimum-cost conservative flow that is non-negative and respects the flow upper bounds on the arcs. We can formulate this as follows:

$$\begin{aligned}
\min \quad & \sum_{e \in \mathcal{A}} c_e x_e \\
& \sum_{\substack{e \in \mathcal{A} : \\ t(e) = v}} x_e \; - \sum_{\substack{e \in \mathcal{A} : \\ h(e) = v}} x_e \;\; = \;\; b_v \,, \quad \forall\, v \in \mathcal{N} \,; \\
& 0 \le x_e \le u_e \,, \qquad\qquad\qquad\qquad \forall\, e \in \mathcal{A} \,.
\end{aligned}$$

## 2.4 An Optimization Modeling Language


Optimization modeling languages facilitate rapid instantiation of mathematical optimization models with data and the subsequent solution by solvers. A well-known example is `AMPL` (see www.ampl.com). The `AMPL` book is available for free download at www.ampl.com/BOOK/download.html.

In a disciplined approach, we set up three files: `*.mod` , `*.dat` and `*.run` . The `*.mod` file sets up the model, the `*.dat` contains the data, and the `*.run` file instantiates the model with the data, passes the instance to the solver, retrieves the solution from the solver, and can be used to report on the solution. In more complex uses, the `*.run` file has control commands that allow for iterative algorithms, possibly instantiating and running many models.

For a simple case, we consider the Production Problem of Section 2.1. Note that (i) on each line of the `AMPL` input files, anything after a symbol "`#`" is treated as a comment, and (ii) commands are terminated with ";" The `*.mod` file can be `production.mod` :

---

```
# production.mod // Jon Lee
#

param m integer > 0;      # NUMBER OF ROWS
param n integer > 0;      # NUMBER OF COLUMNS
set ROWS := {1..m};       # DEFINING THE SET OF ROW INDICES
set COLS := {1..n};       # DEFINING THE SET OF COLUMNS INDICES

param b {ROWS};           # RIGHT-HAND SIDE DATA
param c {COLS};           # OBJECTIVE-FUNCTION DATA
param a {ROWS,COLS};      # CONSTRAINT-MATRIX DATA

var x {j in COLS} >= 0;   # DEFINING THE (NONNEGATIVE) VARIABLES

maximize z:               # CHOOSE MAX/MIN AND NAME THE OBJ. FUNCTION
   sum {j in COLS} c[j] * x [j];  # DEFINING THE OBJECTIVE FUNCTION

subject to Constraints {i in ROWS}: # DEFINING THE CONSTRAINT INDICES
   sum {j in COLS} a[i,j] * x[j] <= b[i]; # DEFINING THE CONSTRAINTS
```

For a sample data set, we can have `production.dat` :

```
# production.dat // Jon Lee
#

param m := 3; # NUMBER OF ROWS

param b :=    # RIGHT-HAND SIDE VECTOR
# i  b(i)
  1   32
  2   33
  3   35 ;

param n := 2; # NUMBER OF COLUMNS

param c :=    # OBJECTIVE VECTOR
# j  c(j)
  1  3
  2  2 ;

param a:       1  2  := # CONSTRAINT MATRIX
# a(i,j):
         1     8  5
         2     8  6
         3     8  7 ;
```

Finally, we can have `production.run` :

```
# production.run // Jon Lee
#

reset;               # CLEAR FROM ANY PREVIOUS SOLVE

option show_stats 1; # SET AMPL OPTION TO CONTROL OUTPUT
option presolve 0;   # COMMENT OUT TO SOLVE BIGGER MODELS
option solver cplex; # AMPL OPTION TO CHOOSE AN AVAILABLE SOLVER
option cplex_options 'display=2 presolve=0'; # SETTING SOLVER OPTIONS
       # DELETING "presolve=0" MAY HELP TO SOLVE BIGGER MODELS

model production.mod; # READ IN THE MODEL
data production.dat;  # READ IN THE DATA AND INSTANTIATE THE MODEL
solve;                # PASS THE INSTANCE TO THE SOLVER

display z > production.out;  # WRITE THE OPTIMAL OBJECTIVE VALUE TO A FILE
                      # NOTE THAT '>' DELETES THE FILE FIRST, IF IT EXISTS
for {j in COLS}
   { # USING C-STYLE COMMANDS TO OUTPUT OPTIMAL SOLUTION TO FILE
     printf "x(%i) = %f\n", j, x[j] >> production.out;
                      # NOTE THAT '>>' APPENDS TO THE FILE, IF IT EXISTS
   };
```

At the University of Michigan, College of Engineering, we have AMPL with the solver CPLEX installed on the CAEN (Computer Aided Engineering Network: caen.engin.umich.edu) machines running Red Hat Linux.

We check that we have the three files needed to run our Production Problem:

```
caen-vnc02% ls

production.dat  production.mod   production.run
```

Next, we invoke AMPL from a command prompt, and invoke the production.run file:

```
caen-vnc02% ampl
ampl: include production.run;

2 variables, all linear
3 constraints, all linear; 6 non-zeros
1 linear objective; 2 non-zeros.

CPLEX 12.2.0.0: display=2
presolve=0
Parallel mode: deterministic, using up to 2 threads for concurrent optimization.
CPLEX 12.2.0.0: optimal solution; objective 12.125
3 dual simplex iterations (1 in phase I)
ampl:
```

Next, we quit out of AMPL from its command prompt:

```
ampl: quit;
```

And we can view the output file:

```
caen-vnc02% cat production.out

z = 12.125

x(1) = 3.375
x(2) = 1.000
caen-vnc02%
```

Next, we see a way to set up a (single-commodity min-cost) network-flow problem. The *.mod file can be flow.mod :

```
# flow.mod // Jon Lee
#

set NODES;

set ARCS within NODES cross NODES;
```

```
    # TELLS AMPL TO CHECK THAT EACH ARC IS AN ORDERED PAIR OF NODES

param b {NODES};

param upper {ARCS};

param c {ARCS};

var x {(i,j) in ARCS} >= 0, <= upper[i,j];

minimize z:
   sum {(i,j) in ARCS} c[i,j] * x[i,j];

subject to Flow_Conservation {i in NODES}:
   sum {(i,j) in ARCS} x[i,j] - sum {(j,i) in ARCS} x[j,i] = b[i];
```

There is a subtlety concerning the summations in the constraints. In each constraint, the node $i$ is fixed. So '`sum{(i,j) in ARCS}`' is really summing over `j in NODES` *such that* `(i,j)` is in `ARCS` ; that is, the arcs whose tail is `i` . Similarly, '`sum{(j,i) in ARCS}`' is really summing over `j in NODES` *such that* `(j,i)` is in `ARCS` ; that is, the arcs whose head is `i` .

The `flow.dat` file can be:

```
# flow.dat // Jon Lee
#

param: NODES: b :=
         1    12
         2    6
         3    -2
         4    0
         5    -9
         6    -7 ;

param: ARCS: upper c :=
        1,2     6  2
        1,3     8  -5
        2,4     5  3
        2,5     7  12
        3,5     5  -9
        4,5     8  2
        4,6     5  0
        5,6     5  4  ;
```

We leave it to the gentle reader to devise an appropriate file `flow.run` (the optimal objective function value is 25).

## 2.5 Exercises

**Exercise 2.1 (Dual in AMPL)**
*Without changing the file* `production.dat` , use AMPL to solve the *dual* of the Production Problem example, as described in Section 2.1. You will need to modify `production.mod` and `production.run` .

**Exercise 2.2 (Sparse solution for linear equations with AMPL)**
In some application areas, it is interesting to find a "sparse solution" — that is, one with few non-zeros — to a system of equations $Ax = b$ . It is well known that a 1-norm minimizing solution is a good heuristic for finding a sparse solution. Using AMPL, try this idea out on several large examples, and report on your results.

HINT: To get an interesting example, try generating a random $m \times n$ matrix $A$ of zeros and ones, perhaps $m = 50$ equations and $n = 500$ variables, maybe with probability $1/2$ of an entry being equal to one. Then choose maybe $m/2$ columns from $A$ and add them up to get $b$ . In this way, you will know that there is a solution with only $m/2$ non-zeros (which is already pretty sparse). Your 1-norm minimizing solution might in fact recover this solution (☺), or it may be sparser (☺☺), or perhaps less sparse (☹).

**Exercise 2.3 (Bloody AMPL)**
A **transportation problem** is a special kind of (single-commodity min-cost) network-flow problem. There are certain nodes $v$ called **supply nodes** which have net supply $b_v > 0$. The other nodes $v$ are called **demand nodes**, and they have net supply $b_v < 0$. There are no nodes with $b_v = 0$ , and all arcs point from supply nodes to demand nodes.

A simplified example is for matching available supply and demand of blood, in types $A$, $B$, $AB$ and $O$ . Suppose that we have $s_v$ units of blood available, in types $v \in \{A, B, AB, O\}$ . Also, we have requirements $d_v$ by patients of different types $v \in \{A, B, AB, O\}$ . It is very important to understand that a patient of a certain type can accept blood not just from their own type. Do some research to find out the compatible blood types for a patient; don't make a mistake — lives depend on this! *In this spirit, if your model misallocates any blood in an incompatible fashion, you will receive a grade of F on this problem.*

Describe a linear-optimization problem that satisfies all of the patient demand with compatible blood. You will find that type $O$ is the most versatile blood, then both $A$ and $B$, followed by $AB$. Factor in this point when you formulate your objective function, with the idea of having the left-over supply of blood being as versatile as possible.

Using AMPL, set up and solve an example of a blood-distribution problem.

**Exercise 2.4 (Mix it up)**

> *"I might sing a gospel song in Arabic or do something in Hebrew. I want to mix it up and do it differently than one might imagine."* — Stevie Wonder

We are given a set of ingredients $1, 2, \ldots, m$ with availabilities $b_i$ and per unit costs $c_i$ . We are given a set of products $j, 2, \ldots, m$ with minimum production requirements $d_j$ and per unit revenues $e_j$ . It is required that product $j$ have at least a *fraction* of $l_{ij}$ of ingredient $i$ and at most a *fraction* of $u_{ij}$ of ingredient $i$ . The goal is to devise a plan to maximize net profit.

Formulate, mathematically, as a linear-optimization problem. Then, model with AMPL, make up some data, try some computations, and report on your results.

**Exercise 2.5 (Task scheduling)**


We are given a set of tasks, numbered $1, 2, \ldots, n$ that should be completed in the minimum amount of time. For convenience, task 0 is a "start task" and task $n + 1$ is an "end task". Each task, except for the start and end task, has a known duration $d_i$ . For convenience, let $d_0 := 0$ . There are precedences between tasks. Specifically, $\Psi_i$ is the set of tasks that must be completed before task $i$ can be started. Let $t_0 := 0$ , and for all other tasks $i$ , let $t_i$ be a decision variable representing its start time.

Formulate the problem, mathematically, as a linear-optimization problem. The objective should be to minimize the start time $t_{n+1}$ of the end task. Then, model the problem with `AMPL`, make up some data, try some computations, and report on your results.

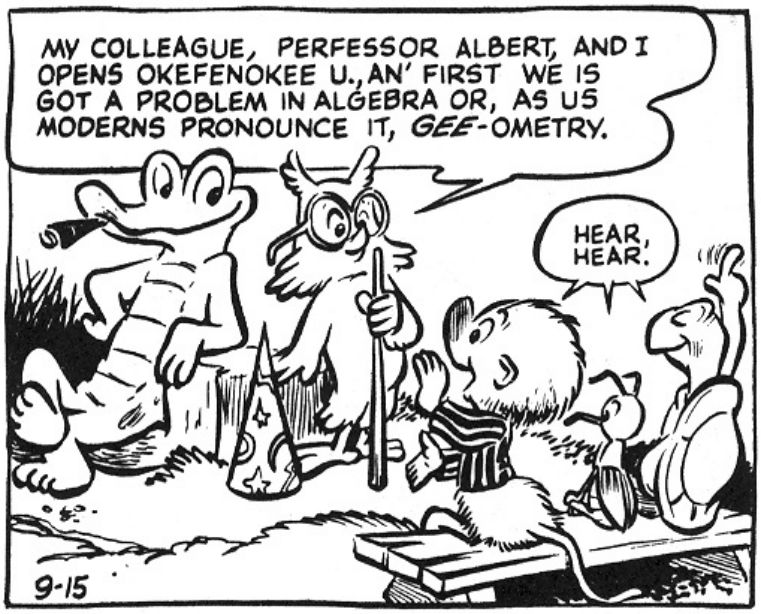
**Exercise 2.6 (Investing wisely)**
Almost certainly, Albert Einstein did *not* say that "compound interest is the most powerful force in the universe."

A company wants to maximize their cash holdings after $T$ time periods. They have an external inflow of $p_t$ dollars at the *start* of time period $t$ , for $t = 1, 2, \ldots, T$ . At the start of each time period, available cash can be allocated to any of $K$ different investment vehicles (in any available non-negative amounts). Money allocated to investment-vehicle $k$ at the start of period $t$ must be held in that investment $k$ for all remaining time periods, and it generates income $v^k_{t,t}$ , $v^k_{t,t+1}$ , …, $v^k_{t,T}$ , per dollar invested. It should be assumed that money obtained from cashing out the investment at the end of the planning horizon (that is, at the end of period $T$) is part of $v^k_{t,T}$ . Note that at the start of time period $t$ , the cash available is the external inflow of $p_t$ , plus cash accumulated from all investment vehicles in prior periods that was not reinvested. Finally, assume that cash held over for one time period earns interest of $q$ percent.

Formulate the problem, mathematically, as a linear-optimization problem. Then, model the problem with `AMPL`, make up some data, try some computations, and report on your results.

# Chapter 3

# Algebra Versus Geometry




Our goals in this chapter are as follows:

- Develop the algebra needed later for our algorithms.
- Develop some geometric understanding of this algebra.

Throughout, we refer to the standard-form problem

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,. \end{array} \tag{P}$$

## 3.1 Basic Feasible Solutions and Extreme Points

A **basic partition** of $A \in \mathbb{R}^{m\times n}$ is a partition of $\{1, 2, \ldots, n\}$ into a pair of ordered sets, the **basis** $\beta = (\beta_1, \beta_2, \ldots, \beta_m)$ and the **non-basis** $\eta = (\eta_1, \eta_2, \ldots, \eta_{n-m})$, so that the **basis**

**matrix** $A_\beta := [A_{\beta_1}, A_{\beta_2}, \ldots, A_{\beta_m}]$ is an invertible $m \times m$ matrix. The connection with the standard "linear-algebra basis" is that the columns of $A_\beta$ form a "linear-algebra basis" for $\mathbb{R}^m$. But for us, "basis" almost always refers to $\beta$.

We associate a **basic solution** $\bar{x} \in \mathbb{R}^n$ with the basic partition via:

$$\begin{array}{rcll} \bar{x}_\eta & := & \mathbf{0} & \in \mathbb{R}^{n-m} \, ; \\ \bar{x}_\beta & := & A_\beta^{-1} b & \in \mathbb{R}^m \, . \end{array}$$

We can observe that $\bar{x}_\beta = A_\beta^{-1} b$ is equivalent to $A_\beta \bar{x}_\beta = b$ , which is the unique way to write $b$ as a linear combination of the columns of $A_\beta$ . Of course this makes sense, because the columns of $A_\beta$ form a "linear-algebra basis" for $\mathbb{R}^m$ .

Note that every basic solution $\bar{x}$ satisfies $A\bar{x} = b$ , because

$$A\bar{x} = \sum_{j=1}^{n} A_j \bar{x}_j = \sum_{j \in \beta} A_j \bar{x}_j + \sum_{j \in \eta} A_j \bar{x}_j = A_\beta \bar{x}_\beta + A_\eta \bar{x}_\eta = A_\beta \left( A_\beta^{-1} b \right) + A_\eta \, \mathbf{0} = b \, .$$

A basic solution $\bar{x}$ is a **basic feasible solution** if it is feasible for (P). That is, if $\bar{x}_\beta = A_\beta^{-1} b \geq \mathbf{0}$ .

It is instructive to have a geometry for understanding the algebra of basic solutions, but for standard-form problems, it is hard to draw something interesting in two dimensions. Instead, we observe that the feasible region of (P) is the solution set, in $\mathbb{R}^n$ , of

$$\begin{array}{rcccl} x_\beta & + & A_\beta^{-1} A_\eta x_\eta & = & A_\beta^{-1} b \, ; \\ x_\beta \geq \mathbf{0} & , & x_\eta \geq \mathbf{0} \, . & & \end{array}$$

Projecting this onto the space of non-basic variables $x_\eta \in \mathbb{R}^{n-m}$ , we obtain

$$\begin{array}{rcl} \left( A_\beta^{-1} A_\eta \right) x_\eta & \leq & A_\beta^{-1} b \, ; \\ x_\eta & \geq & \mathbf{0} \, . \end{array}$$

Notice how we can view the $x_\beta$ variables as slack variables.

**Example 3.1**
For this system, it is convenient to draw pictures when $n - m = 2$ , for example $n = 6$ and $m = 4$ . In such a picture, the basic solution $\bar{x} \in \mathbb{R}^n$ maps to the origin $\bar{x}_\eta = \mathbf{0} \in \mathbb{R}^{n-m}$ , but other basic solutions (feasible and not) will map to other points.

Suppose that we have the data:

$$\begin{array}{rcl} A & := & \begin{pmatrix} 1 & 2 & 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 & 0 \\ 3/2 & 3/2 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} , \\ b & := & (7, 9, 6, 33/10)' \, , \\ \beta & := & (\beta_1 \, , \, \beta_2 \, , \, \beta_3 \, , \, \beta_4) \, = \, (1, 2, 4, 6) \, , \\ \eta & := & (\eta_1 \, , \, \eta_2) \, = \, (3, 5) \, . \end{array}$$

Then

$$\begin{aligned} A_\beta &= [A_{\beta_1}, A_{\beta_2}, A_{\beta_3}, A_{\beta_4}] = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 3 & 1 & 1 & 0 \\ 3/2 & 3/2 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \\ A_\eta &= [A_{\eta_1}, A_{\eta_2}] = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \\ x_\beta &= (x_1, x_2, x_4, x_6)' \\ x_\eta &:= (x_3, x_5)' . \end{aligned}$$

We can calculate

$$\begin{aligned} A_\beta^{-1} A_\eta &= \begin{pmatrix} -1 & 4/3 \\ 1 & -2/3 \\ 2 & -10/3 \\ -1 & 2/3 \end{pmatrix}, \\ A_\beta^{-1} b &:= (1, 3, 3, 3/10)' , \end{aligned}$$

and then we have plotted this in Figure 3.1. The plot has $x_{\eta_1} = x_3$ as the abscissa, and $x_{\eta_2} = x_5$ as the ordinate. In the plot, besides the non-negativity of the variables $x_3$ and $x_4$, the four inequalities of $\left(A_\beta^{-1} A_\eta\right) x_\eta \leq A_\beta^{-1} b$ are labeled with their slack variables — these are the *basic variables* $x_1$ , $x_2$ , $x_4$ , $x_6$ . The correct matching of the basic variables to the inequalities of $\left(A_\beta^{-1} A_\eta\right) x_\eta \leq A_\beta^{-1} b$ is simply achieved by seeing that the $i$-th inequality has slack variable $x_{\beta_i}$ .

The feasible region is colored cyan, while basic feasible solutions project to green points and basic infeasible solutions project to red points. We can see that the basic solution associate with the current basis is feasible, because the origin (corresponding to the non-basic variables being set to 0) is feasible.

A set $S \subset \mathbb{R}^n$ is a **convex set** if it contains the entire line segment between every pair of points in $S$ . That is,

$$\lambda x^1 + (1 - \lambda) x^2 \in S \text{ , whenever } x^1, x^2 \in S \text{ and } 0 < \lambda < 1 .$$

It is simple to check that the feasible region of every linear-optimization problem is a convex set — do it!

For a convex set $S \subset \mathbb{R}^n$ , a point $\hat{x} \in S$ is an **extreme point** of $S$ if it is not on the interior of any line segment wholly contained in $S$ . That is, if we *cannot* write

$$\hat{x} = \lambda x^1 + (1 - \lambda) x^2 \text{ , with } x^1 \neq x^2 \in S \text{ and } 0 < \lambda < 1 .$$

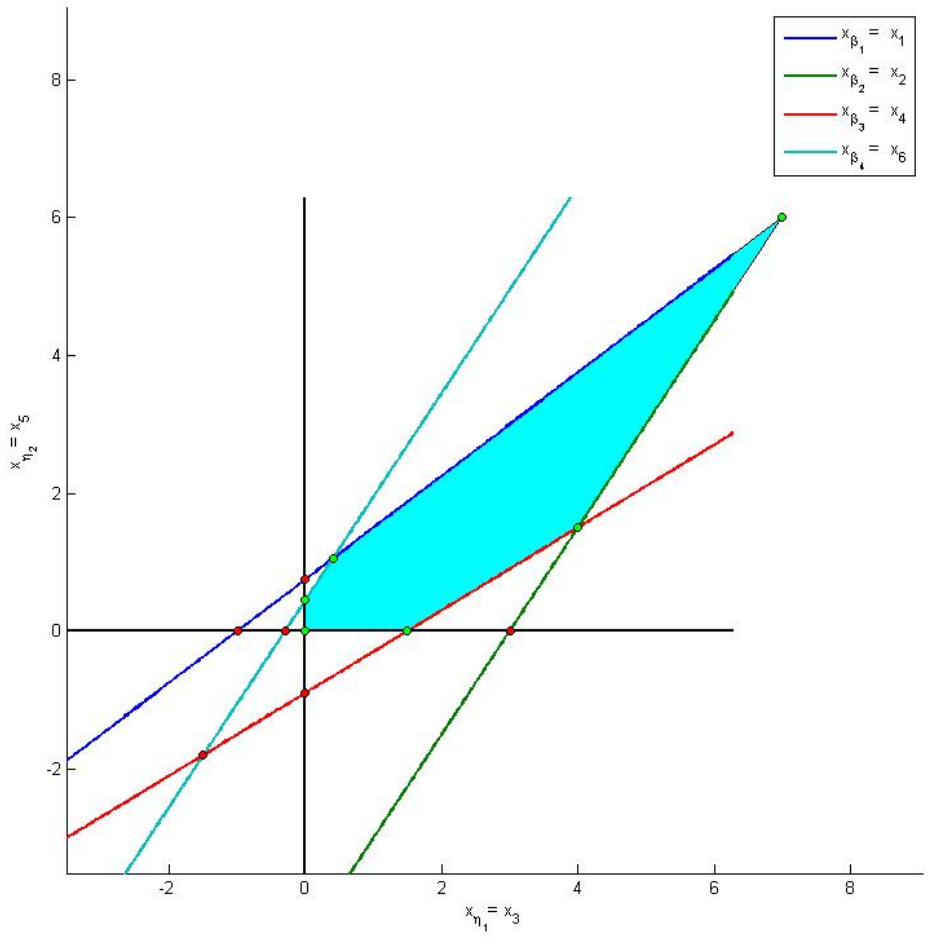



Figure 3.1: Feasible region projected into the space of non-basic variables

**Theorem 3.2**
Every basic feasible solution of standard-form (P) is an extreme point of its feasible region.

*Proof.* Consider the basic feasible solution $\bar{x}$ with

$$\begin{array}{rcll} \bar{x}_\eta & := & \mathbf{0} & \in \mathbb{R}^{n-m} \, ; \\ \bar{x}_\beta & := & A_\beta^{-1} b & \in \mathbb{R}^m \, . \end{array}$$

If

$$\bar{x} = \lambda x^1 + (1-\lambda) x^2 \, , \text{ with } x^1 \text{ and } x^2 \text{ feasible for (P) and } 0 < \lambda < 1 \, ,$$

then $\mathbf{0} = \bar{x}_\eta = \lambda x^1_\eta + (1-\lambda)x^2_\eta$ and $0 < \lambda < 1$ implies that $x^1_\eta = x^2_\eta = \mathbf{0}$. But then $A_\beta x^i_\beta = b$ implies that $x^i_\beta = A_\beta^{-1} b = \bar{x}_\beta$, for $i = 1, 2$. Hence $\bar{x} = x^1 = x^2$ (but we needed $x^1 \neq x^2$), and so we cannot find a line segment containing $\bar{x}$ that is wholly contained in $S$. □

**Theorem 3.3**
Every extreme point of the feasible region of standard-form (P) is a basic solution.

*Proof.* Let $\hat{x}$ be an extreme point of the feasible region of (P). We define

$$\rho := \{j \in \{1, 2, \ldots, n\} \ : \ \hat{x}_j > 0\} .$$

That is, $\rho$ is the list of indices for the positive variables of $\hat{x}$. Also, we let

$$\zeta := \{j \in \{1, 2, \ldots, n\} \ : \ \hat{x}_j = 0\} .$$

That is, $\zeta$ is the list of indices for the zero variables of $\hat{x}$. Together, $\rho$ and $\zeta$ partition $\{1, 2, \ldots, n\}$.

Our goal is to construct a basic partition, $\beta, \eta$, so that the associated basic solution is precisely $\hat{x}$.

The first thing that we will establish is that the columns of $A_\rho$ are linearly independent. We will do that by contradiction. Suppose that they are linearly dependent. That is, there exists $z_\rho \in \mathbb{R}^{|\rho|}$ different from the zero vector, such that $A_\rho z_\rho = \mathbf{0}$. Next we extend $z_\rho$ to a vector $z \in \mathbb{R}^n$, by letting $z_\zeta = \mathbf{0}$. Clearly $Az = A_\rho z_\rho + A_\zeta z_\zeta = \mathbf{0} + A_\zeta \mathbf{0} = \mathbf{0}$; that is, $z$ is in the null space of $A$. Next, let

$$x^1 := \hat{x} + \epsilon z$$

and

$$x^2 := \hat{x} - \epsilon z ,$$

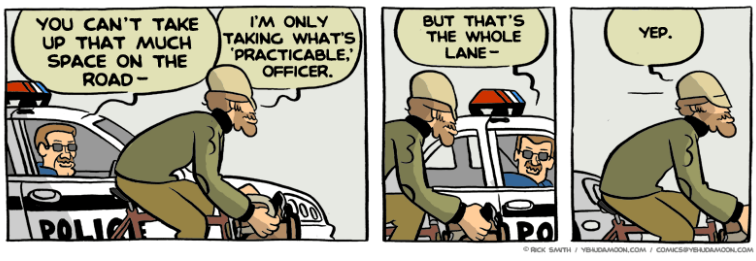
with $\epsilon$ chosen to be sufficiently small so that $x^1$ and $x^2$ are non-negative. Because $z$ is only non-zero on the $\rho$ coordinates (where $\hat{x}$ is positive), we can choose an appropriate $\epsilon$. Notice that $x^1 \neq x^2$, because $z_\rho$ and hence $z$ is not the zero vector. Now, it is easy to verify that $Ax^1 = A(\hat{x} + \epsilon z) = A\hat{x} + \epsilon Az = b + \mathbf{0} = b$ and similarly $Ax^2 = b$. Therefore, $x^1$ and $x^2$ are feasible solutions of (P). Also, $\frac{1}{2}x^1 + \frac{1}{2}x^2 = \frac{1}{2}(\hat{x} + \epsilon z) + \frac{1}{2}(\hat{x} - \epsilon z) = \hat{x}$. So $\hat{x}$ is on the interior (actually it is the midpoint) of the line segment between $x^1$ and $x^2$, in contradiction to $\hat{x}$ being an extreme point of the feasible region of (P). Therefore, it must be that the columns of $A_\rho$ are linearly independent.

In particular, we can conclude that $|\rho| \leq m$, since we assume that $A \in \mathbb{R}^{m \times n}$ has full row rank. If $|\rho| < m$, we choose $m - |\rho|$ columns of $A_\zeta$ to append to $A_\rho$ in such a way as to form a matrix $A_\beta$ having $m$ linearly-independent columns — we note that such a choice is not unique. As usual, we let $\eta$ be a list of the $n - m$ indices not in $\beta$. By definition, the associated basic solution $\bar{x}$ has $\bar{x}_\eta = \mathbf{0}$, and we observe that it is the unique solution to the system of equations $Ax = b$ having $x_\eta = \mathbf{0}$. But $\hat{x}_\eta = \mathbf{0}$ because $\hat{x}_\eta$ is a subvector of $\hat{x}_\zeta = \mathbf{0}$. Therefore, $\hat{x} = \bar{x}$. That is, $\hat{x}$ is a basic solution of (P). □


Taken together, these last two results give us the main result of this section.

**Corollary 3.4**
For a feasible point $\hat{x}$ of standard-form (P), $\hat{x}$ is extreme if and only if $\hat{x}$ is a basic solution.

## 3.2 Basic Feasible Directions




For a point $\hat{x}$ in a convex set $S \subset \mathbb{R}^n$ , a **feasible direction relative to the feasible solution** $\hat{x}$ is a $\hat{z} \in \mathbb{R}^n$ such that $\hat{x} + \epsilon\hat{z} \in S$ , for sufficiently small positive $\epsilon \in \mathbb{R}$ . Focusing now on the standard-form problem (P), for $\hat{z}$ to be a feasible direction relative to the feasible solution $\hat{x}$ , we need $A(\hat{x} + \epsilon\hat{z}) = b$ . But

$$b = A(\hat{x} + \epsilon\hat{z}) = A\hat{x} + \epsilon A\hat{z} = b + \epsilon A\hat{z} \,,$$

so we need $A\hat{z} = \mathbf{0}$ . That is, $\hat{z}$ must be in the null space of $A$ .

Focusing on the standard-form problem (P), we associate a **basic direction** $\bar{z} \in \mathbb{R}^n$ with the basic partition $\beta, \eta$ and a choice of non-basic index $\eta_j$ via

$$\begin{array}{rcll} \bar{z}_\eta & := & \boldsymbol{e_j} & \in \mathbb{R}^{n-m} \,; \\ \bar{z}_\beta & := & -A_\beta^{-1} A_{\eta_j} & \in \mathbb{R}^m \,. \end{array}$$

Note that every basic direction $\bar{z}$ is in the null space of $A$ :

$$A\bar{z} = A_\beta \bar{z}_\beta + A_\eta \bar{z}_\eta = A_\beta \left( -A_\beta^{-1} A_{\eta_j} \right) + A_\eta \boldsymbol{e_j} = -A_{\eta_j} + A_{\eta_j} = \mathbf{0} \,.$$

So

$$A\left(\hat{x} + \epsilon\bar{z}\right) = b \,,$$

for every feasible $\hat{x}$ and every $\epsilon \in \mathbb{R}$. Moving a positive amount in the direction $\bar{z}$ corresponds to increasing the value of $x_{\eta_j}$, holding the values of all other non-basic variables constant, and making appropriate changes in the basic variables so as to maintain satisfaction of the equation system $Ax = b$.

There is a related point worth making. We have just seen that for a given basic partition $\beta, \eta$, *each* of the $n - m$ basic directions is in the null space of $A$ — there is one such basic direction for each of the $n - m$ choices of $\eta_j$. It is very easy to check that these basic directions are linearly independent — just observe that they are columns of the $n \times (n - m)$ matrix

$$\begin{pmatrix} I \\ -A_\beta^{-1} A_\eta \end{pmatrix}.$$

Because the dimension of the null space of $A$ is $n - m$, these $n - m$ basic directions form a basis for the null space of $A$.

Now, we focus on the basic feasible solution $\bar{x}$ determined by the basic partition $\beta, \eta$. The basic direction $\bar{z}$ is a **basic feasible direction relative to the basic feasible solution** $\bar{x}$ if $\bar{x} + \epsilon\bar{z}$ is feasible, for sufficiently small positive $\epsilon \in \mathbb{R}$. That is, if

$$A_\beta^{-1} b - \epsilon A_\beta^{-1} A_{\eta_j} \geq \mathbf{0},$$

for sufficiently small positive $\epsilon \in \mathbb{R}$.

Let $\bar{b} := \bar{x}_\beta = A_\beta^{-1} b$, and let $\bar{A}_{\eta_j} := A_\beta^{-1} A_{\eta_j}$. So, we need that

$$\bar{b} - \epsilon \bar{A}_{\eta_j} \geq \mathbf{0},$$

for sufficiently small positive $\epsilon \in \mathbb{R}$. That is,

$$\bar{b}_i - \epsilon \bar{a}_{i,\eta_j} \geq 0,$$

for $i = 1, 2, \ldots, m$. If $\bar{a}_{i,\eta_j} \leq 0$, for some $i$, then this imposes no restriction at all on $\epsilon$. So, the only condition that we need for $\bar{z}$ to be a *basic feasible direction relative to the basic feasible solution* $\bar{x}$ is that there exists $\epsilon > 0$ satisfying

$$\epsilon \leq \frac{\bar{b}_i}{\bar{a}_{i,\eta_j}}, \text{ for all } i \text{ such that } \bar{a}_{i,\eta_j} > 0.$$

Equivalently, we simply need that

$$\bar{b}_i > 0, \text{ for all } i \text{ such that } \bar{a}_{i,\eta_j} > 0.$$

So, we have the following result:

**Theorem 3.5**
For a standard-form problem (P), suppose that $\bar{x}$ is a basic feasible solution relative to the basic partition $\beta, \eta$. Consider choosing a non-basic index $\eta_j$. Then the associated basic direction $\bar{z}$ is a feasible direction relative to $\bar{x}$ if and only if

$$\bar{b}_i > 0, \text{ for all } i \text{ such that } \bar{a}_{i,\eta_j} > 0.$$

## 3.3 Basic Feasible Rays and Extreme Rays

For a non-empty convex set $S \subset \mathbb{R}^n$ , a **ray** of $S$ is a $\hat{z} \neq \mathbf{0}$ in $\mathbb{R}^n$ such that $\hat{x} + \tau\hat{z} \in S$ , for *all* $\hat{x} \in S$ and *all* positive $\tau \in \mathbb{R}$ .

Focusing on the standard-from problem (P), it is easy to see that $\hat{z} \neq \mathbf{0}$ is a ray of the feasible region if and only if $A\hat{z} = \mathbf{0}$ and $\hat{z} \geq \mathbf{0}$.

Recall from Section 3.2 that for a standard-form problem (P), a basic direction $\bar{z} \in \mathbb{R}^n$ is associated with the basic partition $\beta, \eta$ and a choice of non-basic index $\eta_j$ via

$$\begin{array}{rcll} \bar{z}_\eta & := & \boldsymbol{e_j} & \in \mathbb{R}^{n-m} \ ; \\ \bar{z}_\beta & := & -A_\beta^{-1} A_{\eta_j} & \in \mathbb{R}^m \ . \end{array}$$

If the basic direction $\bar{z}$ is a ray, then we call it a **basic feasible ray**. We have already seen that $A\bar{z} = \mathbf{0}$. Furthermore, $\bar{z} \geq \mathbf{0}$ if and only if $\bar{A}_{\eta_j} := A_\beta^{-1} A_{\eta_j} \leq \mathbf{0}$.

Therefore, we have the following result:

**Theorem 3.6**
The basic direction $\bar{z}$ is a ray of the feasible region of (P) if and only if $\bar{A}_{\eta_j} \leq \mathbf{0}$ .

Recall, further, that $\bar{z}$ is a basic feasible direction *relative to the basic feasible solution* $\bar{x}$ if $\bar{x} + \epsilon\bar{z}$ is feasible, for *sufficiently small positive* $\epsilon \in \mathbb{R}$ . Therefore, if $\bar{z}$ is a basic feasible ray, relative to the basic partition $\beta, \eta$ and $\bar{x}$ is the basic feasible solution relative to the same basic partition, then $\bar{z}$ is a basic feasible direction relative to $\bar{x}$ .

A ray $\hat{z}$ of a convex set $S$ is an **extreme ray** if we *cannot* write

$$\hat{z} = z^1 + z^2 \ , \text{ with } z^1 \neq \mu z^2 \text{ being rays of } S \text{ and } \mu \neq 0 \ .$$

Similarly to the correspondence between basic feasible solutions and extreme points for standard-form problems, we have the following two results.

**Theorem 3.7**
Every basic feasible ray of standard-form (P) is an extreme ray of its feasible region.

**Theorem 3.8**
Every extreme ray of the feasible region of standard-form (P) is a positive multiple of a basic feasible ray.

## 3.4 Exercises

**Exercise 3.1 (Illustrate algebraic and geometric concepts)**
Make a small example, say with six variables and four equations, to fully illustrate all of the concepts in this chapter. In A.2, there are `MATLAB` scripts and functions that could be very helpful here.

**Exercise 3.2 (Basic feasible rays are extreme rays)**
Prove Theorem 3.7.

**Exercise 3.3 (Extreme rays are positive multiples of basic feasible rays)**
If you are feeling very ambitious, prove Theorem 3.8.

**Exercise 3.4 (Dual basic direction — do this if you will be doing Exercise 4.2)**
Let $\beta, \eta$ be a basic partition for our standard-form problem (P). As you will see on the first page of the next chapter, we can associate with the basis $\beta$, a dual solution

$$\bar{y}' := c'_\beta A_\beta^{-1}$$

of

$$\begin{array}{lll} \max & y'b & \\ & y'A & \leq \quad c' \, . \end{array} \tag{D}$$

It is easy to see that $\bar{y}$ satisfies the constraints $y'A_\beta \leq c'_\beta$ (of (D)) with equality; that is, the dual constraints indexed from $\beta$ are "active".

Let us assume that $\bar{y}$ is feasible for (D). Now, let $\eta_l$ be a non-basic index, and let $\bar{w} := H_{\ell \cdot}$ be row $\ell$ of $H := A_\beta^{-1}$ . Consider $\tilde{\bar{y}} := \bar{y} - \lambda \bar{w}$, and explain (with algebraic justification) what is happening to the activity of each constraint of (D), as $\lambda$ increases. HINT: Think about the cases of (i) $i = \ell$, (ii) $i \in \beta$, $i \neq \ell$, and (iii) $j \in \eta$ .

# Chapter 4

# The Simplex Algorithm


Our goal in this chapter is as follows:

- Develop a mathematically-complete Simplex Algorithm for optimizing standard-form problems.

## 4.1 A Sufficient Optimality Criterion

**Definition 4.1**
The dual solution of (D) associated with basis $\beta$ is

$$\bar{y}' := c'_\beta A_\beta^{-1} .$$

**Lemma 4.2**
If $\beta$ is a basis, then the primal basic solution $\bar{x}$ (feasible or not) and the dual solution $\bar{y}$ (feasible or not) associated with $\beta$ have equal objective value.

*Proof.* The objective value of $\bar{x}$ is $c'\bar{x} = c'_\beta \bar{x}_\beta + c'_\eta \bar{x}_\eta = c'_\beta (A_\beta^{-1} b) + c'_\eta \mathbf{0} = c'_\beta A_\beta^{-1} b$ . The objective value of $\bar{y}$ is $\bar{y}'b = (c'_\beta A_\beta^{-1})b = c'_\beta A_\beta^{-1} b$ . □

**Definition 4.3**
The vector of reduced costs associated with basis $\beta$ is

$$\bar{c}' := c' - c'_\beta A_\beta^{-1} A = c' - \bar{y}' A \, .$$

**Lemma 4.4**
The dual solution of (D) associated with basis $\beta$ is feasible for (D) if

$$\bar{c}_\eta \geq \mathbf{0} \, .$$

*Proof.* Using the definitions of $\bar{y}$ and $\bar{c}$ , the condition $\bar{c}_\eta \geq \mathbf{0}$ is equivalent to

$$\bar{y}' A_\eta \leq c'_\eta \, .$$

The definition of $\bar{y}$ gives us

$$\bar{y}' A_\beta = c'_\beta \qquad (\text{equivalently}, \bar{c}_\beta = \mathbf{0}) \, .$$

So we have

$$\bar{y}' A = \bar{y}' [A_\beta, A_\eta] \leq (c'_\beta, c'_\eta) = c' \, .$$

Hence $\bar{y}$ is feasible for (D). □

**Theorem 4.5 (Weak Optimal Basis Theorem)**
If $\beta$ is a feasible basis and $\bar{c}_\eta \geq \mathbf{0}$ , then the primal solution $\bar{x}$ and the dual solution $\bar{y}$ associated with $\beta$ are optimal.

*Proof.* We have already observed that $c'\bar{x} = \bar{y}' b$ for the pair of primal and dual solutions associated with the basis $\beta$ . If these solutions $\bar{x}$ and $\bar{y}$ are feasible for (P) and (D), respectively, then by weak duality these solutions are optimal. □

We can also take (P) and transform it into an equivalent form that is quite revealing. Clearly, (P) is equivalent to

$$\begin{array}{rrcrcl} \min & c'_\beta x_\beta & + & c'_\eta x_\eta & & \\ & A_\beta x_\beta & + & A_\eta x_\eta & = & b \\ & x_\beta \geq \mathbf{0} & , & x_\eta \geq \mathbf{0} \, . & & \end{array}$$

Next, multiplying the equations on the left by $A_\beta^{-1}$ , we see that they are equivalent to

$$x_\beta + A_\beta^{-1} A_\eta x_\eta = A_\beta^{-1} b \, .$$

We can also see this as

$$x_\beta = A_\beta^{-1} b - A_\beta^{-1} A_\eta x_\eta \, .$$

Using this equation to substitute for $x_\beta$ in the objective function, we are led to the linear objective function

$$c'_\beta A_\beta^{-1} b + \min \left( c'_\eta - c'_\beta A_\beta^{-1} A_\eta \right) x_\eta = c'_\beta A_\beta^{-1} b + \min \bar{c}'_\eta x_\eta \ ,$$

which is equivalent to the original one on the set of points satisfying $Ax = b$ . In this equivalent form, it is now solely expressed in terms of $x_\eta$ . Now, if $\bar{c}_\eta \geq \mathbf{0}$ , the best we could hope for in minimizing is to set $x_\eta = \mathbf{0}$ . But the unique solution having $x_\eta = \mathbf{0}$ is the basic *feasible* solution $\bar{x}$ . So that $\bar{x}$ is optimal.

**Example 4.6**
This is a continuation of Example 3.1. In Figure 4.1, we have depicted the sufficient optimality criterion, in the space of a particular choice of non-basic variables — not the choice previously depicted. Specifically, we consider the equivalent problem

$$\begin{array}{lrcl} \min & \bar{c}'_\eta x_\eta & & \\ & \bar{A}_\eta x_\eta & \leq & \bar{b} \, ; \\ & x_\eta & \geq & \mathbf{0} \, . \end{array}$$

This plot demonstrates the optimality of $\beta := (5, 4, 6, 3)$ ($\eta := (2, 1)$). The basic directions available from the basic feasible solution $\bar{x}$ appear as standard unit vectors in the space of the non-basic variables. The solution $\bar{x}$ is optimal because $\bar{c}_\eta \geq \mathbf{0}$ ; we can also think of this as $\bar{c}_\eta$ having a non-negative dot product with each of the standard unit vectors, hence neither direction is improving.

## 4.2 The Simplex Algorithm with No Worries


**Improving direction.** Often it is helpful to directly refer to individual elements of the vector $\bar{c}_\eta$ ; namely,

$$\bar{c}_{\eta_j} = c_{\eta_j} - c'_\beta A_\beta^{-1} A_{\eta_j} = c_{\eta_j} - c'_\beta \bar{A}_{\eta_j} \ , \text{ for } j = 1, 2, \ldots, n - m \ .$$

If the sufficient optimality criterion is not satisfied, then we choose an $\eta_j$ such that $\bar{c}_{\eta_j}$ is negative, and we consider solutions that increase the value of $x_{\eta_j}$ up from $\bar{x}_{\eta_j} = 0$ , changing the values of the basic variables to insure that we still satisfy the equations $Ax = b$ , while holding the other non-basic variables at zero.

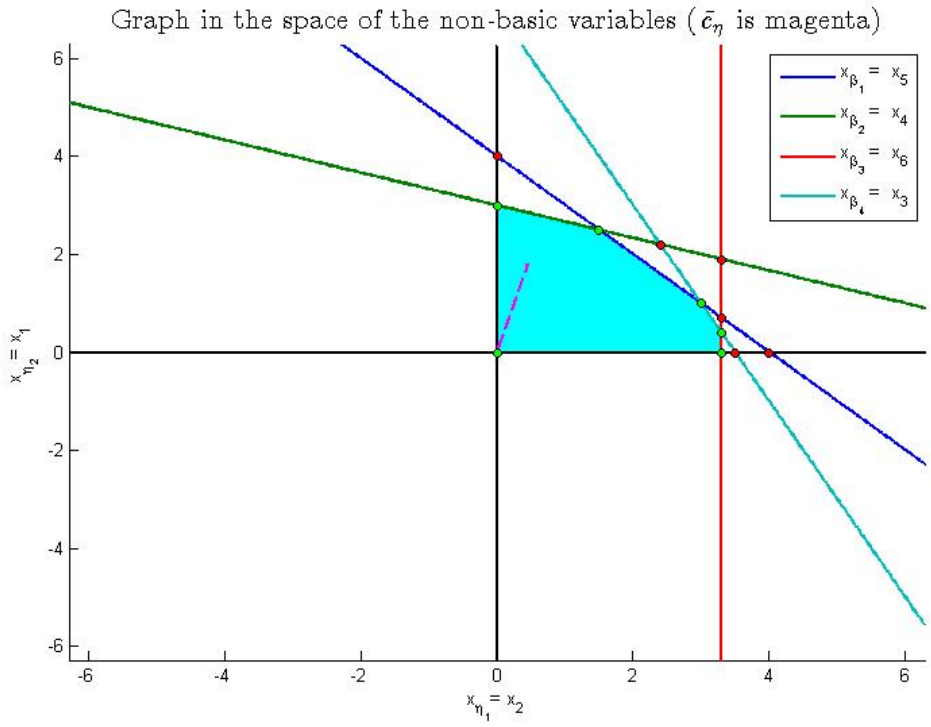



Figure 4.1: Sufficient optimality criterion

Operationally, we take the basic direction $\bar{z} \in \mathbb{R}^n$ defined by

$$\begin{array}{rcll} \bar{z}_\eta & := & \boldsymbol{e_j} & \in \mathbb{R}^{n-m} \; ; \\ \bar{z}_\beta & := & -A_\beta^{-1} A_{\eta_j} = -\bar{A}_{\eta_j} & \in \mathbb{R}^m \; , \end{array}$$

and we consider solutions of the form $\bar{x} + \lambda \bar{z}$ , with $\lambda > 0$ . The motivation is based on the observations that

- $c'(\bar{x} + \lambda \bar{z}) - c'\bar{x} = \lambda c'\bar{z} = \lambda \bar{c}_{\eta_j} < 0$ ;
- $A(\bar{x} + \lambda \bar{z}) = A\bar{x} + \lambda A\bar{z} = b + \lambda \mathbf{0} = b$ .

That is, the objective function changes at the rate of $\bar{c}_{\eta_j}$ , and we maintain satisfaction of the $Ax = b$ constraints.

**Maximum step — the ratio test and a sufficient unboundedness criterion.** By our choice of direction $\bar{z}$, all variables that are non-basic with respect to the current choice of basis remain non-negative ($x_{\eta_j}$ increases from 0 and the others remain at 0). So the only thing that restricts our movement in the direction $\bar{z}$ from $\bar{x}$ is that we have to make sure that the current basic variables remain non-negative. This is easy to take care of. We just make sure that we choose $\lambda > 0$ so that

$$\bar{x}_\beta + \lambda \bar{z}_\beta = \bar{b} - \lambda \bar{A}_{\eta_j} \geq \mathbf{0} \; .$$

Notice that for $i$ such $\bar{a}_{i,\eta_j} \leq 0$ , there is no limit on how large $\lambda$ can be. In fact, it can well happen that $\bar{A}_{\eta_j} \leq \mathbf{0}$ . In this case, $\bar{x} + \lambda\bar{z}$ is feasible for all $\lambda > 0$ and $c'(\bar{x} + \lambda\bar{z}) \to -\infty$ as $\lambda \to +\infty$ , so the problem is unbounded.

Otherwise, to insure that $\bar{x} + \lambda\bar{z} \geq \mathbf{0}$ , we just enforce

$$\lambda \leq \frac{\bar{b}_i}{\bar{a}_{i,\eta_j}} \text{ , for } i \text{ such that } \bar{a}_{i,\eta_j} > 0 \text{ .}$$

Finally, to get the best improvement in the direction $\bar{z}$ from $\bar{x}$, we let $\lambda$ equal

$$\bar{\lambda} := \min_{i \,:\, \bar{a}_{i,\eta_j} > 0} \left\{ \frac{\bar{b}_i}{\bar{a}_{i,\eta_j}} \right\} \text{ .}$$

**Non-degeneracy.** There is a significant issue in even carrying out one iteration of this algorithm. If $\bar{b}_i = 0$ for some $i$ such that $\bar{a}_{i,\eta_j} > 0$ , then $\bar{\lambda} = 0$ , and we are not able to make any change from $\bar{x}$ in the direction $\bar{z}$ . Just for now, we will simply assume away this problem, using the following hypothesis that every basic variable of every basic feasible solution is positive.

**Definition 4.7**
The problem (P) satisfies the **non-degeneracy hypothesis** if for every feasible basis $\beta$, we have $\bar{x}_{\beta_i} := \bar{b}_i > 0$ for $i = 1, 2, \ldots, m$ .

Under the non-degeneracy hypothesis, $\bar{\lambda} > 0$ .

**Another basic feasible solution.** By our construction, the new solution $\bar{x} + \bar{\lambda}\bar{z}$ is feasible and has lesser objective value than that of $\bar{x}$ . We can repeat the construction as long as the new solution is *basic*. If it is basic, there is a natural guess as to what an appropriate basis may be. The variable $x_{\eta_j}$ , formerly non-basic at value $0$ has increased to $\bar{\lambda}$ , so clearly it must become basic. Also, at least one variable that was basic now has value $0$ . In fact, under our non-degeneracy hypothesis, once we establish that the new solution is basic, we observe that *exactly* one variable that was basic now has value $0$. Let

$$i^* := \operatorname*{argmin}_{i \,:\, \bar{a}_{i,\eta_j} > 0} \left\{ \frac{\bar{b}_i}{\bar{a}_{i,\eta_j}} \right\} \text{ .}$$

If there is more than one $i$ that achieves the minimum (which can happen if we do not assume the non-degeneracy hypothesis), then we will see that the choice of $i^*$ can be any of these. We can see that $x_{\beta_{i^*}}$ has value $0$ in $\bar{x} + \bar{\lambda}\bar{z}$ . So it is natural to hope we can replace $x_{\beta_{i^*}}$ as a basic variable with $x_{\eta_j}$ .

Let

$$\tilde{\beta} := (\beta_1, \beta_2, \ldots, \beta_{i^*-1}, \eta_j, \beta_{i^*+1}, \ldots, \beta_m)$$

and

$$\tilde{\eta} := (\eta_1, \eta_2, \ldots, \eta_{j-1}, \beta_{i^*}, \eta_{j+1}, \ldots, \eta_{n-m}) \text{ .}$$

**Lemma 4.8**
$A_{\tilde{\beta}}$ is invertible.

*Proof.* $A_{\tilde{\beta}}$ is invertible precisely when the following matrix is invertible:

$$\begin{aligned} A_\beta^{-1} A_{\tilde{\beta}} &= A_\beta^{-1} \left[ A_{\beta_1}, A_{\beta_2}, \ldots, A_{\beta_{i-1}}, A_{\eta_j}, A_{\beta_{i+1}}, \ldots, A_{\beta_m} \right] \\ &= \left[ \boldsymbol{e_1}, \boldsymbol{e_2}, \ldots, \boldsymbol{e_{i-1}}, \bar{A}_{\eta_j}, \boldsymbol{e_{i+1}}, \ldots, \boldsymbol{e_m} \right] . \end{aligned}$$

But the determinant of this matrix is precisely $\bar{a}_{i,\eta_j} \neq 0$ . □

**Lemma 4.9**
The unique solution of $Ax = b$ having $x_{\tilde{\eta}} = \mathbf{0}$ is $\bar{x} + \bar{\lambda}\bar{z}$ .

*Proof.* $\bar{x}_{\tilde{\eta}} + \bar{\lambda}\bar{z}_{\tilde{\eta}} = \mathbf{0}$ . Moreover, $\bar{x} + \bar{\lambda}\bar{z}$ is the unique solution to $Ax = b$ having $x_{\tilde{\eta}} = \mathbf{0}$ because $A_{\tilde{\beta}}$ is invertible. □

Putting these two lemmata together, we have the following key result.

**Theorem 4.10**
$\bar{x} + \bar{\lambda}\bar{z}$ is a basic solution; in fact, it is the basic solution determined by $\tilde{\beta}$ .

Passing from the partition $\beta, \eta$ to the partition $\tilde{\beta}, \tilde{\eta}$ is commonly referred to as a **pivot**.

## Worry-Free Simplex Algorithm

**Input:** $c \in \mathbb{R}^n$ , $b \in \mathbb{R}^m$ , $A \in \mathbb{R}^{m \times n}$ of full row rank $m$ , for the standard-form problem:

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,, \end{array} \qquad \text{(P)}$$

where $x$ is a vector of variables in $\mathbb{R}^n$ .

0. Start with any basic feasible partition $\beta, \eta$ .
1. Let $\bar{x}$ and $\bar{y}$ be the primal and dual solutions associated with $\beta, \eta$ .
   If $\bar{c}_\eta \geq \mathbf{0}$, then STOP: $\bar{x}$ and $\bar{y}$ are optimal.
2. Otherwise, choose a non-basic index $\eta_j$ with $\bar{c}_{\eta_j} < 0$ .

3. If $\bar{A}_{\eta_j} \leq \mathbf{0}$ , then STOP: (P) is unbounded and (D) is infeasible.

4. Otherwise, let

$$i^* := \underset{i \,:\, \bar{a}_{i,\eta_j} > 0}{\operatorname{argmin}} \left\{ \frac{\bar{b}_i}{\bar{a}_{i,\eta_j}} \right\} ,$$

replace $\beta$ with

$$\left(\beta_1, \beta_2, \ldots, \beta_{i^*-1}, \underline{\underline{\eta_j}}, \beta_{i^*+1}, \ldots, \beta_m\right)$$

and $\eta$ with

$$\left(\eta_1, \eta_2, \ldots, \eta_{j-1}, \underline{\underline{\beta_{i^*}}}, \eta_{j+1}, \ldots, \eta_{n-m}\right) .$$

5. GOTO 1.

**Theorem 4.11**
Under the non-degeneracy hypothesis, the Worry-Free Simplex Algorithm terminates correctly.

*Proof.* Under the non-degeneracy hypothesis, every time we visit Step 1, we have a primal feasible solution with a decreased objective value. This implies that we never revisit a basic feasible partition. But there are only a finite number of basic feasible partitions, so we must terminate, after a finite number of pivots. But there are only two places where the algorithm terminates; either in Step 1 where we correctly identify that $\bar{x}$ and $\bar{y}$ are optimal by our sufficient optimality criterion, or in Step 3 because of our sufficient unboundedness criterion. □


*"The things that should bother me don't—should I be worried?"*
New Yorker 3/3/09

**Remark 4.12**
There are two very significant issues remaining:

- How do we handle degeneracy? (see Section 4.3).
- How do we initialize the algorithm in Step 0? (see Section 4.4).

## 4.3 Anticycling

To handle degeneracy, we will eliminate it with an **algebraic perturbation**. It is convenient to make the perturbation depend on an $m \times m$ non-singular matrix $B$ — eventually we will choose $B$ in a convenient manner. We replace the problem (P) with

$$\begin{array}{lrcll} \min & c'x & & & \\ & Ax & = & b_\epsilon(B)\,; & (\mathrm{P}_\epsilon(B)) \\ & x & \geq_\epsilon & \mathbf{0}_\epsilon\,, & \end{array}$$

where

- $b_\epsilon(B) := b + B\vec{\epsilon}$, and $\vec{\epsilon} := (\epsilon, \epsilon^2, \ldots, \epsilon^m)'$ (these are *exponents* not superscripts);
- the scalar $\epsilon$ is an arbitrarily small indeterminant; $\epsilon$ is *not* given a numerical value; it is simply considered to be a quantity that is positive, yet smaller than any positive real number;
- $\mathbf{0}_\epsilon$ denotes a vector in which all entries are the zero polynomial (in $\epsilon$);
- the variables $x_j$ are polynomials in $\epsilon$ with real coefficients;
- the ordering of polynomials used to interpret the inequality $\geq_\epsilon$ is described next.

The ordering is actually quite simple, but for the sake of precision, we describe it formally.

**An ordered ring.** The set of polynomials in $\epsilon$, with real coefficients, form what is known in mathematics as an "ordered ring". The ordering $<_\epsilon$ is simple to describe. Let $p(\epsilon) := \sum_{j=0}^{m} p_j \epsilon^j$ and $q(\epsilon) := \sum_{j=0}^{m} q_j \epsilon^j$ . Then $p(\epsilon) <_\epsilon q(\epsilon)$ if the least $j$ for which $p_j \neq q_j$ has $p_j < q_j$ . Another way to think about the ordering $<_\epsilon$ is that $p(\epsilon) <_\epsilon q(\epsilon)$ if $p(\epsilon) < q(\epsilon)$ when $\epsilon$ is considered to be an arbitrarily small positive number. Notice how the ordering $<_\epsilon$ is in a certain sense a more refined ordering than $<$ . That is, if $p(0) < q(0)$ , then $p(\epsilon) <_\epsilon q(\epsilon)$ , but we can have $p(0) = q(0)$ without having $p(\epsilon) =_\epsilon q(\epsilon)$ . Finally, we note that the zero polynomial "$0_\epsilon$"(all coefficients equal to 0) is the zero of this ordered ring, so we can speak, for example about polynomials that are positive with respect to the ordering $<_\epsilon$ . Concretely, $p(\epsilon) \neq 0_\epsilon$ is positive if the least $i$ for which $p_i \neq 0$ satisfies $p_i > 0$ . Emphasizing that $<_\epsilon$ is a more refined ordering than $<$ , we see that $p(\epsilon) \geq_\epsilon 0_\epsilon$ implies that $p(0) = p_0 \geq 0$ .

For an arbitrary basis $\beta$, the associated basic solution $\bar{x}^\epsilon$ has $\bar{x}^\epsilon_\beta := A_\beta^{-1}(b + B\vec{\epsilon}) = \bar{b} + A_\beta^{-1}B\vec{\epsilon}$. It is evident that $\bar{x}^\epsilon_{\beta_i}$ is a polynomial, of degree at most $m$, in $\epsilon$ , for each $i = 1, \ldots, m$ . Because the ordering $<_\epsilon$ refines the ordering $<$ , we have that $\bar{x}^\epsilon_\beta \geq_\epsilon \mathbf{0}_\epsilon$ implies that $\bar{x}_\beta \geq \mathbf{0}$ . That is, any basic feasible partition for ($\mathrm{P}_\epsilon(B)$) is a basic feasible partition for (P). This implies that applying the Worry-Free Simplex Algorithm to ($\mathrm{P}_\epsilon(B)$), using the ratio test to enforce feasibility of $\bar{x}$ in ($\mathrm{P}_\epsilon(B)$) at each iteration, implies that each associated $\bar{x}_\beta$ is feasible for (P). That is, the choice of a leaving variable dictated by the ratio test when we work with ($\mathrm{P}_\epsilon(B)$) is valid if we instead do the ratio test working with (P).

The objective value associated with $\bar{x}^\epsilon$ is $c'_\beta A_\beta^{-1}(b+B\vec{\epsilon}) = \bar{y}'b+\bar{y}'B\vec{\epsilon}$, is a polynomial (of degree at most $m$) in $\epsilon$ . Therefore, we can order basic solutions for ($\mathrm{P}_\epsilon(B)$) using $<_\epsilon$ , and that ordering refines the ordering of the objective values of the corresponding basic solution of (P). This implies that if $\bar{x}^\epsilon$ is optimal for ($\mathrm{P}_\epsilon(B)$) , then the $\bar{x}$ associated with the same basis is optimal for (P).

**Lemma 4.13**
The $\epsilon$-perturbed problem ($\mathrm{P}_\epsilon(B)$) satisfies the non-degeneracy hypothesis.

*Proof.* For an arbitrary basis matrix $A_\beta$, the associated basic solution $\bar{x}^\epsilon$ has $\bar{x}^\epsilon_\beta := A_\beta^{-1}(b+B\vec{\epsilon}) = \bar{b} + A_\beta^{-1}B\vec{\epsilon}$. As we have already pointed out, $\bar{x}^\epsilon_{\beta_i}$ is a polynomial, of degree at most $m$, in $\epsilon$ , for each $i = 1, \ldots, m$ . $\bar{x}^\epsilon_{\beta_i} = 0_\epsilon$ implies that the $i$-th row of $A_\beta^{-1}B$ is all zero. But this is impossible for the invertible matrix $A_\beta^{-1}B$ . □

**Theorem 4.14**
Let $\beta^0$ be a basis that is feasible for (P). Then the Worry-Free Simplex Algorithm applied to ($\mathrm{P}_\epsilon(A_{\beta^0})$), starting from the basis $\beta^0$ , correctly demonstrates that (P) is unbounded or finds an optimal basic partition for (P).

*Proof.* The first important point to notice is that we are choosing the perturbation of the original right-hand side to depend on the choice of a basis that is *feasible for* (P). Then we observe that $\bar{x}^{\epsilon}_{\beta^0} := A^{-1}_{\beta^0}(b + A_{\beta^0}\vec{\epsilon}) = A^{-1}_{\beta^0}b + \vec{\epsilon}$. Now because $\bar{x}$ is feasible for (P), we have $A^{-1}_{\beta^0}b \geq \mathbf{0}$ . Then, the ordering $<_\epsilon$ implies that $\bar{x}^{\epsilon}_{\beta^0} = A^{-1}_{\beta^0}b + \vec{\epsilon} \geq_\epsilon \mathbf{0}$ . Therefore, the basis $\beta^0$ is feasible for ($\mathrm{P}_\epsilon(A_{\beta^0})$), and the Worry-Free Simplex Algorithm can indeed be started for ($\mathrm{P}_\epsilon(A_{\beta^0})$) on $\beta^0$ .

Notice that it is only in Step 4 of the Worry-Free Simplex Algorithm that really depends on whether we are considering ($\mathrm{P}_\epsilon(A_{\beta^0})$) or (P). The sufficient optimality criterion and the sufficient unboundedness criterion are identical for ($\mathrm{P}_\epsilon(A_{\beta^0})$) and (P). Because ($\mathrm{P}_\epsilon(A_{\beta^0})$) satisfies the non-degeneracy hypothesis, the Worry-Free Simplex Algorithm correctly terminates for ($\mathrm{P}_\epsilon(A_{\beta^0})$). □

(Pivot.mp4)

Figure 4.2: With some `.pdf` viewers, you can click above to see or download a short video. Or just see it on YouTube (probably with an ad) by clicking here.

## 4.4 Obtaining a Basic Feasible Solution

Next, we will deal with the problem of finding an initial basic feasible solution for the standard-form problem

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,. \end{array} \qquad \text{(P)}$$


### 4.4.1 Ignoring degeneracy

At first, we ignore the degeneracy issue — why worry about two things at once?! The idea is rather simple. First, we choose any basic partition $\tilde{\beta}, \tilde{\eta}$ . If we are lucky, then $A_{\tilde{\beta}}^{-1} b \geq \mathbf{0}$ .


Otherwise, we have some work to do. We define a new non-negative variable $x_{n+1}$, which we temporarily adjoin as an additional non-basic variable. So our basic indices remain as

$$\tilde{\beta} = \left( \tilde{\beta}_1, \tilde{\beta}_2, \ldots, \tilde{\beta}_m \right) \ ,$$

while our non-basic indices are extended to

$$\tilde{\eta} = \left( \tilde{\eta}_1, \tilde{\eta}_2, \ldots, \tilde{\eta}_{n-m}, \ \underline{\tilde{\eta}_{n-m+1} := n+1} \right) \ .$$

The column for the constraint matrix associated with this new variable is defined as $A_{n+1} := -A_{\tilde{\beta}} \mathbf{1}$ . Hence $\bar{A}_{n+1} = -\mathbf{1}$ . Finally, we temporarily put aside the objective function from (P) and replace it with one of minimizing $x_{n+1}$. That is, we consider the so-called **phase-one problem**

$$\begin{array}{lrcrcll} \min & & & & x_{n+1} & & \\ & Ax & + & A_{n+1} x_{n+1} & = & b \, ; & (\Phi) \\ & x & , & x_{n+1} & \geq & \mathbf{0} \, . & \end{array}$$

(With this terminology, the **phase-two problem** is just the original problem (P)).

It is evident that any feasible solution $\hat{x}$ of ($\Phi$) with $\hat{x}_{n+1} = 0$ is feasible for (P). Moreover, if the minimum objective value of ($\Phi$) is greater than 0, then we can conclude that (P) has no feasible solution. So, toward establishing whether or not (P) has a feasible solution, we focus our attention on ($\Phi$). We will soon see that we can easily find a basic feasible solution of ($\Phi$).

**Finding a basic feasible solution of ($\Phi$).** Choose $i^*$ so that $\bar{x}_{\tilde{\beta}_{i^*}}$ is most negative. Then we exchange $\tilde{\beta}_{i*}$ with $\tilde{\eta}_{n-m+1} = n+1$ . That is, our new basic indices are

$$\beta := \left(\tilde{\beta}_1, \tilde{\beta}_2, \ldots, \tilde{\beta}_{i^*-1}, \underline{\underline{n+1}}, \tilde{\beta}_{i^*+1}, \ldots, \tilde{\beta}_m\right) ,$$

and our new non-basic indices are

$$\eta := \left(\tilde{\eta}_1, \tilde{\eta}_2, \ldots, \tilde{\eta}_{n-m}, \underline{\underline{\tilde{\beta}_{i^*}}}\right) .$$

**Lemma 4.15**
The basic solution of ($\Phi$) associated with the basic partition $\beta, \eta$ is feasible for ($\Phi$).

*Proof.* This pivot, from $\tilde{\beta}, \tilde{\eta}$ to $\beta, \eta$ amounts to moving in the basic direction $\bar{z} \in \mathbb{R}^{n+1}$ defined by

$$\begin{array}{rcll} \bar{z}_{\tilde{\eta}} & := & \boldsymbol{e_{n-m+1}} & \in \mathbb{R}^{n-m+1} ; \\ \bar{z}_{\tilde{\beta}} & := & -A_{\tilde{\beta}}^{-1} A_{n+1} = \mathbf{1} & \in \mathbb{R}^m , \end{array}$$

in the amount $\lambda := -\bar{x}_{\tilde{\beta}_{i^*}} > 0$ . That is, $\bar{x} + \lambda\bar{z}$ is the basic solution associated with the basic partition $\beta, \eta$ . Notice how when we move in the direction $\bar{z}$ , all basic variables increase at exactly the same rate that $x_{n+1}$ does. So, using this direction to increase $x_{n+1}$ from 0 to $-\bar{x}_{\tilde{\beta}_{i^*}} > 0$ results in all basic variables increasing by exactly $-\bar{x}_{\tilde{\beta}_{i^*}} > 0$ . By the choice of $i^*$ , this causes all basic variable to become non-negative, and $x_{\tilde{\beta}_{i^*}}$ to become 0 , whereupon it can leave the basis in exchange for $x_{n+1}$ . □

**The end game for ($\Phi$).** If (P) is feasible, then at the very last iteration of the Worry-Free Simplex Algorithm on ($\Phi$), the objective value will drop from a positive number to zero. As this happens, $x_{n+1}$ will be eligible to leave the basis, but so may other variables also be eligible. That is, there could be a tie in the ratio-test of Step 4 of the Worry-Free Simplex Algorithm. As is the case whenever there is a tie, any of the tying indices can leave the basis — all of the associated variables are becoming zero simultaneously. For our purposes, it is critical that if there is a tie, we choose $i^* := n+1$ ; that is, $x_{n+1}$ must be selected to become non-basic. In this way, we not only get a feasible solution to (P), we get a basis for it that does not use the artificial variable $x_{n+1}$ . Now, starting from this basis, we can smoothly shift to minimizing the objective function of (P).

### 4.4.2 Not ignoring degeneracy

**Anticycling for (Φ).** There is one lingering issue remaining. We have not discussed anticycling for (Φ).


But this is relatively simple. We define an $\epsilon$-perturbed version

$$\begin{array}{lccccl} \min & & & & x_{n+1} & \\ & Ax & + & A_{n+1}x_{n+1} & = & b_\epsilon(B)\,; \\ & x & , & x_{n+1} & \geq_\epsilon & \mathbf{0}_\epsilon\,, \end{array} \qquad (\Phi_\epsilon)$$

where $b_\epsilon(B) := b + \vec{\epsilon}$, and $\vec{\epsilon} := (\epsilon, \epsilon^2, \ldots, \epsilon^m)'$. Then we choose $i$ so that $\bar{x}^\epsilon_{\tilde{\beta}_i}$ is most negative with respect to the ordering $<_\epsilon$, and exchange $i$ and $n+1$ as before. Then, as in Lemma 4.15, the resulting basis is feasible for $(\Phi_\epsilon)$.


We do need to manage the final iteration a bit carefully. There are two different ways we can do this.

**"Early arrival".** If (P) has a feasible solution, at some point the value of $x_{n+1}$ will decrease to a *homogeneous* polynomial in $\epsilon$. That is, the constant term will become 0. At this point, although $x_{n+1}$ may not be eligible to leave the basis for $(\Phi_\epsilon)$, it *will* be eligible to leave for (P). So, at this point we let $x_{n+1}$ leave the basis, and we terminate the solution process for $(\Phi_\epsilon)$, having found a feasible basis for (P). In fact, we have just constructively proved the following result.

**Theorem 4.16**
If standard form (P) has a feasible solution, then it has a basic feasible solution.

Note that because $x_{n+1}$ may not have been eligible to leave the basis for $(\Phi_\epsilon)$ when we apply the "early arrival' idea, the resulting basis may not be feasible for $(P_\epsilon)$. So we will have to re-perturb (P) .


**"Be patient".** Perhaps a more elegant way to handle the situation is to fully solve $(\Phi_\epsilon)$. In doing so, if (P) has a feasible solution, then the minimum objective value of $(\Phi_\epsilon)$ will be 0 (i.e., the zero polynomial), and $x_{n+1}$ will necessarily be non-basic. That is because, at every iteration, every basic variable in $(\Phi_\epsilon)$ is *positive*. Because $x_{n+1}$ legally left the basis for $(\Phi_\epsilon)$ at the final iteration, the resulting basis is feasible for $(P_\epsilon)$. So we do not re-perturb (P) , and we simply revert to solving $(P_\epsilon)$ from the final basis of $(\Phi_\epsilon)$ .

## 4.5 The Simplex Algorithm

Too budget-conscious for a Campy switch?
Fix that Simplex Prestige

**Making Simplexes Work**

by Frank Leavitt

I hate plastic. It is ugly and the process by which it is made does horrible damage to the environment. But I have to admit that the plastic Simplex derailleurs which I have owned and worked on have been a joy both to use and to service. From time to time, I hear and read derisive remarks by bikies about these derailleurs. Usually such remarks are based on inadequate knowlege of how to take care of the things.

The cage on the Simplex front changer is attached to a pushrod which is moved by a lever. The lever is moved by a cable housing in compression. You oil the lever by prying off the red plastic disc on the side of the changer body and squirting oil inside. But it is even more important to oil the cable by squirting oil inside the housing and to lubricate the pushrod. I don't think it matters much whether one uses grease or oil on the pushrod. I use whatever may be handy and they seem to work equally well, although I suspect grease may do a slightly better job of keeping out rain. After several hours of riding in the rain, sometimes all the lubricant will wash out no matter what one uses. Then I sometimes find myself shifting from the large to the small chainring by kicking the cage with my heel. I would consider this a perfectly acceptable method of shifting were it not for the fact that it interferes with one's cadence.

When you adjust the Simplex front changer, you adjust the inner travel first by setting the cage at the desired position on the pushrod. If you adjust the outer travel first, then you will have to do it all over again after you have done the inner travel. The same applies for the Campagnolo Valentine front changer.

To adjust the inner travel, you put the chain on the largest (innermost) rear sprocket and on the inner chainring. Loosen the hex-head setscrew on the cage. Set the cage so that it just misses getting rubbed by the chain and tighten down the setscrew. Most instructions I have read say to *center* the cage over the chain. This is a good way to make sure the chain will fall off when you shift, so I always set the cage considerably to the right of center.

It does not matter what kind of wrench you use to tighten down the setscrew. I guess a 4-inch adjustable is the best (this is a very handy size wrench to have around, as it will double as a spoke wrench for off-dite spoke nipples), but I generally use whatever wrench I can grab the quickest. When "use-the-proper-tool-for-the-job" purists are hanging around the shop (you can spot them by their expensive equipment and their inability to hold a wrench without dropping it), I'll tighten the setscrew with a 12-inch adjustable just to shock them. The important thing is that you have enough feel for what you are doing so that you don't snap off bolts or round off heads.

If you do snap off that setscrew, don't despair. An apprentice mechanic must break one or two of them before he can be considered to have the feel of it. (He or she must also break off two stem expander bolts, strip three freewheels, install one rear axle and one bottom bracket spindle backwards, put in two sets of Sturmey-Archer high-gear pawls upside down, destroy the seat tube decals on seven new bicycles and dish three rear wheels to the wrong side before graduating to journeyman mechanic.) If the setscrew is broken off but the cage is adjusted correctly, then don't worry, just leave it there and ride the bike. Of course, if you are working in a shop and it is a customer's bike, you will have to make it look right, so get a tiny drill bit and an "easy-out" and good luck to you.

Notice that even when the setscrew is tightened down properly, you will still be able to rotate the cage a little by hand. This is play in the pushrod: nothing to worry about. Some people mistake this play for a loose setscrew so they tighten the setscrew until it breaks.

The outside travel of the cage is controlled by a screw on the changer body. Loosen this screw. Put the chain on the

*"This is a very complicated case, Maude. You know, a lotta ins, a lotta outs, a lotta what-have-yous. And, uh, a lotta strands to keep in my head, man. Lotta strands in old Duder's head."* — The Dude

Putting everything together, we get a mathematically complete algorithm for linear optimization. That is:

1. Apply an algebraic perturbation to the phase-one problem;
2. Solve the phase-one problem using the Worry-Free Simplex Algorithm, adapted to algebraically perturbed problems, but always giving preference to $x_{n+1}$ for leaving the basis whenever it is eligible to leave for the unperturbed problem. Go to the next step, as soon as $x_{n+1}$ leaves the basis;
3. Starting from the feasible basis obtained for the original standard-form problem, apply an algebraic perturbation (Note that the previous step may have left us with a basis that is feasible for the original unperturbed problem, but infeasible for the original perturbed problem — this is why we apply a perturbation anew);
4. Solve the problem using the Worry-Free Simplex Algorithm, adapted to algebraically perturbed problems.

It is important to know that the Simplex Algorithm will be used, later, to prove the celebrated Strong Duality Theorem. For that reason, it is important that our algorithm be mathematically complete. But from a practical computational viewpoint, there is substantial overhead in working with the $\epsilon$-perturbed problems. Therefore, in practice, no computer code that is routinely applied to large instances worries about the *potential* for cycling associated with degeneracy.

## 4.6 Exercises

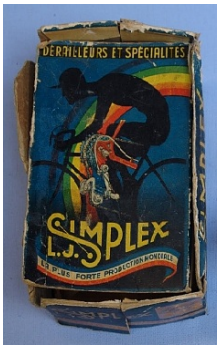
**Exercise 4.1 (Illustrate aspects of the Simplex Algorithm)**
In A.2, there are `MATLAB` scripts and functions which implement the primitive steps of the simplex algorithm. *Using these primitives only*, write a script to carry out the simplex algorithm. Be sure to read in the data using `pivot_setup.m`. Do not worry about degeneracy/anti-cycling. But I do want you to take care of algorithmically finding an initial feasible basis as described in Section 4.4.1. Make some small examples to fully the different possibilities for (P) (i.e., infeasible, optimal, unbounded).

**Exercise 4.2 (Dual change** — first do Exercise 3.4**)**
Let $\beta, \eta$ be any basic partition for the standard-form problem (P). The associated dual solution is $\bar{y}' := c'_\beta A_\beta^{-1}$ . Now, suppose that we pivot, letting $\eta_j$ enter the basis and $\beta_\ell$ leave the basis, so that the new partition $\tilde{\beta}, \tilde{\eta}$ is also a basic partition (in other words,

$A_{\tilde{\beta}}$ is invertible). Let $\tilde{\bar{y}}$ be the dual solution associated with the basic partition $\tilde{\beta}, \tilde{\eta}$ , and let $H_{\ell\cdot}$ be row $\ell$ of $H := A_{\beta}^{-1}$ . Prove that

$$\tilde{\bar{y}} = \bar{y} + \frac{\bar{c}_{\eta_j}}{\bar{a}_{\ell,\eta_j}} H_{\ell\cdot}$$

**Exercise 4.3 (Worry-Free Simplex Algorithm can cycle)**




Let $\theta = 2\pi/k$, with integer $k \geq 5$. The idea is to use the symmetry of the geometric circle, and complete a cycle of the Worry-Free Simplex Algorithm in $2k$ pivots. Choose a constant $\gamma$ satisfying $0 < \gamma < (\sin\theta)/(1-\cos\theta)$ . Let

$$A_1 := \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad A_2 := \begin{pmatrix} 0 \\ \gamma \end{pmatrix} .$$

Let

$$R := \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} .$$

Then, for $j = 3, 4, \ldots, 2k$ , let

$$A_j := \begin{cases} R^{(j-1)/2} A_1 , & \text{for odd } j ; \\ R^{(j-2)/2} A_2 , & \text{for even } j . \end{cases}$$

We can observe that for odd $j$ , $A_j$ is a rotation of $A_1$ by $(j-1)\pi/k$ radians, and for even $j$ , $A_j$ is a rotation of $A_2$ by $(j-2)\pi/k$ radians.

Let $c_j := 1 - a_{1j} - a_{2j}/\gamma$ , for $j = 1, 2, \ldots, 2k$ , and let $b := (0,0)'$ . Because $b = \mathbf{0}$ , the problem is fully degenerate; that is, $\bar{x} = \mathbf{0}$ for all basic solutions $\bar{x}$ . Notice that this implies that either the problem has optimal objective value of zero, or the objective value is unbounded.

For $k = 5$ , check that you can choose $\gamma = \cot\theta$ , and then check that the following is a sequence of bases $\beta$ that are legal for the Worry-Free Simplex Algorithm:

$$\beta = (1,2) \to (2,3) \to (3,4) \to \ldots \to (2k-1, 2k) \to (2k, 1) \to (1,2) .$$

You need to check that for every pivot, the incoming basic variable $x_{\eta_j}$ has negative reduced cost, and that the outgoing variable is legally selected — that is that $\bar{a}_{i,\eta_j} > 0$ . Feel free to use `MATLAB`, `MATHEMATICA`, etc.

If you are feeling ambitious, check that for *all* $k \geq 5$ , we get a cycle of the Worry-Free Simplex Algorithm.

Note that it may seem hard to grasp the picture at all[4]. But see Section 6.1.3 and the following `Mathematica` code. Note that the animation can be activated within the Acrobat Reader using the controls below the figure.

```
(* circle.m // Jon Lee *)
(* *)

t = 2 Pi/5; c = Cos[t]; s = Sin[t]; n = 69; e = .75;
m = {{c, -s, 0}, {s, c, 0}, {(c - 1)/c, s/c, 1}};
x = {{1}, {0}, {0}};
y = {{0}, {Cot[t]}, {0}};
T = Partition[
   Flatten[{x, y, m.x, m.y, m.m.x, m.m.y, m.m.m.x, m.m.m.y, m.m.m.m.x,
      m.m.m.m.y, x}], {3}, {3}];
ListAnimate[
 Table[Graphics3D[
   Table[{GrayLevel[e], Polygon[{{0, 0, 0}, T[[i]], T[[i + 1]]}]},
      {i,10}], Boxed -> False, PlotRangePadding -> 2.5,
   BoxRatios -> {2, 2, 2}, SphericalRegion -> True,
   ViewPoint -> {Sin[2 Pi*t/n], Cos[2 Pi*t/n], 0},
   Lighting -> False], {t, 0, n}]]
```

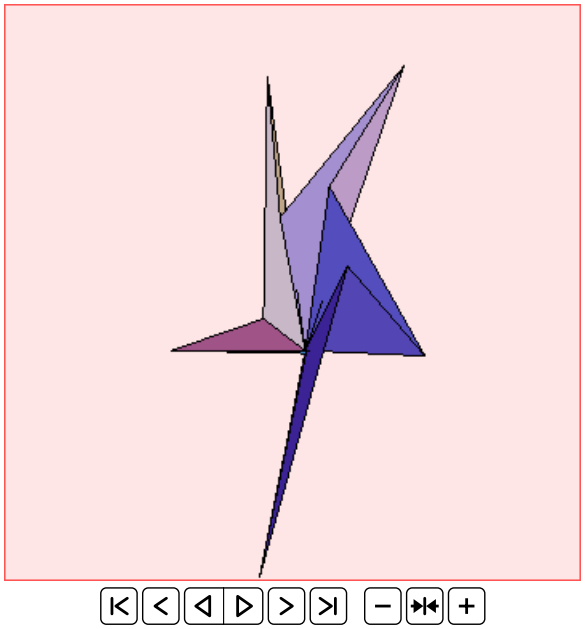


Figure 4.3: A picture of the cycle with $k = 5$

# Chapter 5

# Duality


Our goals in this chapter are as follows:

- Establish the Strong Duality Theorem for the standard-form problem.
- Establish the Complementary Slackness Theorem for the standard-form problem.
- See how duality and complementarity carry over to general linear-optimization problems.
- Learn about "theorems of the alternative."

As usual, we focus on the standard-form problem

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0} \end{array} \tag{P}$$

and its dual

$$\begin{array}{lrcl} \max & y'b & & \\ & y'A & \leq & c'\,. \end{array} \tag{D}$$

## 5.1 The Strong Duality Theorem

We have already seen two simple duality theorems:

- **Weak Duality Theorem.** If $\hat{x}$ is feasible in (P) and $\hat{y}$ is feasible in (D), then $c'\hat{x} \geq \hat{y}'b$ .
- **Weak Optimal Basis Theorem.** If $\beta$ is a feasible basis and $\bar{c}_\eta \geq \mathbf{0}$ , then the primal solution $\bar{x}$ and the dual solution $\bar{y}$ associated with $\beta$ are optimal.

The Weak Duality Theorem directly implies that if $\hat{x}$ is feasible in (P) and $\hat{y}$ is feasible in (D), and $c'\hat{x} = \hat{y}'b$ , then $\hat{x}$ and $\hat{y}$ are optimal. Thinking about it this way, we see that both the Weak Duality Theorem and the Weak Optimal Basis Theorem assert conditions that are sufficient for establishing optimality.

**Theorem 5.1 (Strong Optimal Basis Theorem)**
If (P) has a feasible solution, and (P) is not unbounded, then there exists a basis $\beta$ such that the associated basic solution $\bar{x}$ and the associated dual solution $\bar{y}$ are optimal. Moreover, $c'\bar{x} = \bar{y}'b$ .

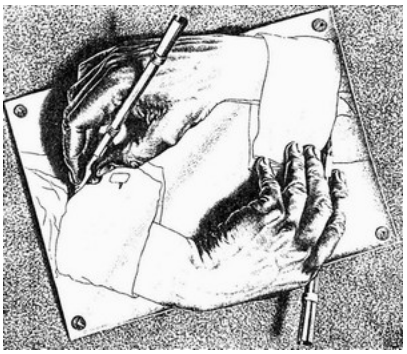
*Proof.* If (P) has a feasible solution and (P) is not unbounded, then the Simplex Algorithm will terminate with a basis $\beta$ such that the associated basic solution $\bar{x}$ and the associated dual solution $\bar{y}$ are optimal. □

As a direct consequence, we have a celebrated theorem.

**Theorem 5.2 (Strong Duality Theorem)**
If (P) has a feasible solution, and (P) is not unbounded, then there exist feasible solutions $\hat{x}$ for (P) and $\hat{y}$ for (D) that are optimal. Moreover, $c'\hat{x} = \hat{y}'b$ .

It is important to realize that the Strong Optimal Basis Theorem and the Strong Duality Theorem depend on the correctness of the Simplex Algorithm — this includes: (i) the correctness of the phase-one procedure to find an initial feasible basis of (P), and (ii) the anti-cycling methodology.

## 5.2 Complementary Slackness



**Definition 5.3**
With respect to the standard-form problem (P) and its dual (D), the solutions $\hat{x}$ and $\hat{y}$ are **complementary** if

$$\begin{aligned}(c_j - \hat{y}' A_{.j})\hat{x}_j &= 0 \text{ , for } j = 1, 2, \ldots, n \text{ ;} \\ \hat{y}_i(A_{i.}\hat{x} - b_i) &= 0 \text{ , for } i = 1, 2, \ldots, m \text{ .}\end{aligned}$$

**Theorem 5.4**
If $\bar{x}$ is a basic solution (feasible or not) of standard-form (P), and $\bar{y}$ is the associated dual solution, then $\bar{x}$ and $\bar{y}$ are complementary.

*Proof.* Notice that if $\bar{x}$ is a basic solution then $A\bar{x} = b$. Then we can see that complementarity of $\bar{x}$ and $\bar{y}$ amounts to

$$\bar{c}_j \bar{x}_j = 0 \text{ , for } j = 1, 2, \ldots, n \text{ .}$$

It is clear then that $\bar{x}$ and $\bar{y}$ are complementary, because if $\bar{x}_j > 0$ , then $j$ is a basic index, and $\bar{c}_j = 0$ for basic indices. □

**Theorem 5.5**
If $\hat{x}$ and $\hat{y}$ are complementary with respect to (P) and (D), then $c'\hat{x} = \hat{y}'b$ .

*Proof.*

$$c'\hat{x} - \hat{y}'b = (c' - \hat{y}'A)\hat{x} + \hat{y}'(A\hat{x} - b) \text{ ,}$$

which is 0 by complementarity. □

**Corollary 5.6 (Weak Complementary Slackness Theorem)**
If $\hat{x}$ and $\hat{y}$ are feasible and complementary with respect to (P) and (D), then $\hat{x}$ and $\hat{y}$ are optimal.

*Proof.* This immediately follows from Theorem 5.5 and the Weak Duality Theorem. □

**Theorem 5.7 (Strong Complementary Slackness Theorem)**
If $\hat{x}$ and $\hat{y}$ are optimal for (P) and (D), then $\hat{x}$ and $\hat{y}$ are complementary (with respect to (P) and (D)).

*Proof.* If $\hat{x}$ and $\hat{y}$ are optimal, then by the Strong Duality Theorem, we have $c'\hat{x} - \hat{y}'b = 0$. Therefore, we have

$$\begin{aligned} 0 &= (c' - \hat{y}'A)\hat{x} + \hat{y}'(A\hat{x} - b) \\ &= \sum_{j=1}^{n}(c_j - \hat{y}'A_{\cdot j})\hat{x}_j + \sum_{i=1}^{m}\hat{y}_i(A_{i\cdot}\hat{x} - b_i) \ . \end{aligned}$$

Next, observing that $\hat{x}$ and $\hat{y}$ are feasible, we have

$$\sum_{j=1}^{n}\underbrace{(c_j - \hat{y}'A_{\cdot j})}_{\geq 0}\underbrace{\hat{x}_j}_{\geq 0} + \sum_{i=1}^{m}\hat{y}_i\underbrace{(A_{i\cdot}\hat{x} - b_i)}_{=0} \ .$$

Clearly this expression is equal to a non-negative number. Finally, we observe that this expression can only be equal to 0 if

$$(c_j - \hat{y}'A_{\cdot j})\hat{x}_j \quad = \quad 0 \ , \text{ for } j = 1, 2, \ldots, n \ .$$

□

## 5.3 Duality for General Linear-Optimization Problems

Thus far, we have focused on duality for the standard-form problem (P). But we will see that *every* linear-optimization problem has a natural dual. Consider the rather general linear minimization problem

$$\begin{array}{lrcrcrcl} \min & c_P'x_P & + & c_N'x_N & + & c_U'x_U & & \\ & A_{GP}x_P & + & A_{GN}x_N & + & A_{GU}x_U & \geq & b_G \ ; \\ & A_{LP}x_P & + & A_{LN}x_N & + & A_{LU}x_U & \leq & b_L \ ; \\ & A_{EP}x_P & + & A_{EN}x_N & + & A_{EU}x_U & = & b_E \ ; \\ & x_P \geq \mathbf{0} \ , & & x_N \leq \mathbf{0} \ . & & & & \end{array} \qquad (\mathcal{G})$$

We will see in the next result that a natural dual for it is

$$\begin{array}{lrcrcrcl} \max & y_G'b_G & + & y_L'b_L & + & y_E'b_E & & \\ & y_G'A_{GP} & + & y_L'A_{LP} & + & y_E'A_{EP} & \leq & c_P' \ ; \\ & y_G'A_{GN} & + & y_L'A_{LN} & + & y_E'A_{EN} & \geq & c_N' \ ; \\ & y_G'A_{GU} & + & y_L'A_{LU} & + & y_E'A_{EU} & = & c_U' \ ; \\ & y_G' \geq \mathbf{0} \ , & & y_L' \leq \mathbf{0} \ . & & & & \end{array} \qquad (\mathcal{H})$$

**Theorem 5.8**

- **Weak Duality Theorem:** If $(\hat{x}_P, \hat{x}_N, \hat{x}_U)$ is feasible in $(\mathcal{G})$ and $(\hat{y}_G, \hat{y}_L, \hat{y}_E)$ is feasible in $(\mathcal{H})$, then $c'_P\hat{x}_P + c'_N\hat{x}_N + c'_U\hat{x}_U \geq \hat{y}'_G b_G + \hat{y}'_L b_L + \hat{y}'_E b_E$ .
- **Strong Duality Theorem:** If $(\mathcal{G})$ has a feasible solution, and $(\mathcal{G})$ is not unbounded, then there exist feasible solutions $(\hat{x}_P, \hat{x}_N, \hat{x}_U)$ for $(\mathcal{G})$ and $(\hat{y}_G, \hat{y}_L, \hat{y}_E)$ for $(\mathcal{H})$ that are optimal. Moreover, $c'_P\hat{x}_P + c'_N\hat{x}_N + c'_U\hat{x}_U = \hat{y}'_G b_G + \hat{y}'_L b_L + \hat{y}'_E b_E$ .

*Proof.* The Weak Duality Theorem for general problems can be demonstrated as easily as it was for the standard-form problem and its dual. But the Strong Duality Theorem for general problems is most easily obtained by converting our general problem $(\mathcal{G})$ to the standard-form

$$
\begin{array}{rcrcrcrcrcl}
\min\ c'_P x_P & - & c'_N \tilde{x}_N & + & c'_U \tilde{x}_U & - & c'_U \tilde{\tilde{x}}_U & & & & \\
A_{GP} x_P & - & A_{GN} \tilde{x}_N & + & A_{GU} \tilde{x}_U & - & A_{GU} \tilde{\tilde{x}}_U & - s_G & & = & b_G\ ; \\
A_{LP} x_P & - & A_{LN} \tilde{x}_N & + & A_{LU} \tilde{x}_U & - & A_{LU} \tilde{\tilde{x}}_U & & + t_L & = & b_L\ ; \\
A_{EP} x_P & - & A_{EN} \tilde{x}_N & + & A_{EU} \tilde{x}_U & - & A_{EU} \tilde{\tilde{x}}_U & & & = & b_E\ ; \\
x_P \geq \mathbf{0}\ , & & \tilde{x}_N \geq \mathbf{0}\ , & & \tilde{x}_U \geq \mathbf{0}\ , & & \tilde{\tilde{x}}_U \geq \mathbf{0}\ , & s_G \geq \mathbf{0}\ , & t_L \geq \mathbf{0}\ . & &
\end{array}
$$

Above, we substituted $-\tilde{x}_N$ for $x_N$ and $\tilde{x}_U - \tilde{\tilde{x}}_U$ for $x_U$ . Taking the dual of this standard-form problem, we obtain

$$
\begin{array}{rcrcrcrcl}
\max & y'_G b_G & + & y'_L b_L & + & y'_E b_E & & & \\
 & y'_G A_{GP} & + & y'_L A_{LP} & + & y'_E A_{EP} & \leq & c'_P & ; \\
- & y'_G A_{GN} & - & y'_L A_{LN} & - & y'_E A_{EN} & \leq & -c'_N & ; \\
 & y'_G A_{GU} & + & y'_L A_{LU} & + & y'_E A_{EU} & \leq & c'_U & ; \\
- & y'_G A_{GU} & - & y'_L A_{LU} & - & y'_E A_{EU} & \leq & -c'_U & ; \\
- & y'_G & & & & & \leq & \mathbf{0} & ; \\
 & & + & y'_L & & & \leq & \mathbf{0} & ,
\end{array}
$$

which is clearly equivalent to $(\mathcal{H})$. □

**Definition 5.9**
With respect to $(\mathcal{G})$ and its dual $(\mathcal{H})$, the solutions $(\hat{x}_P, \hat{x}_N, \hat{x}_U)$ and $(\hat{y}_G, \hat{y}_L, \hat{y}_E)$ are **complementary** if

$$
\begin{aligned}
&\left(c_j - \hat{y}'_G A_{Gj} - \hat{y}'_L A_{Lj} - \hat{y}'_E A_{Ej}\right) \hat{x}_j = 0\ , \text{ for all } j\ ; \\
&\hat{y}_i \left(A_{iP} x_P + A_{iN} x_N + A_{iU} x_U - b_i\right) = 0\ , \text{ for all } i\ .
\end{aligned}
$$

**Theorem 5.10**

- **Weak Complementary Slackness Theorem:** If $(\hat{x}_P, \hat{x}_N, \hat{x}_U)$ and $(\hat{y}_G, \hat{y}_L, \hat{y}_E)$ are feasible and complementary with respect to $(\mathcal{G})$ and $(\mathcal{H})$, then $(\hat{x}_P, \hat{x}_N, \hat{x}_U)$ and $(\hat{y}_G, \hat{y}_L, \hat{y}_E)$ are optimal.

- **Strong Complementary Slackness Theorem:** If $(\hat{x}_P, \hat{x}_N, \hat{x}_U)$ and $(\hat{y}_G, \hat{y}_L, \hat{y}_E)$ are optimal for $(\mathcal{G})$ and $(\mathcal{H})$, $(\hat{x}_P, \hat{x}_N, \hat{x}_U)$ and $(\hat{y}_G, \hat{y}_L, \hat{y}_E)$ are complementary (with respect to $(\mathcal{G})$ and $(\mathcal{H})$).

*Proof.* Similarly to the proof for standard-form (P) and its dual (D), we consider the following expression.

$$
\begin{aligned}
0 \;=\; & \sum_{j\in P} \underbrace{(c_j - \hat{y}_G' A_{Gj} - \hat{y}_L' A_{Lj} - \hat{y}_E' A_{Ej})}_{\geq 0} \underbrace{\hat{x}_j}_{\geq 0} \\
& + \sum_{j\in N} \underbrace{(c_j - \hat{y}_G' A_{Gj} - \hat{y}_L' A_{Lj} - \hat{y}_E' A_{Ej})}_{\leq 0} \underbrace{\hat{x}_j}_{\leq 0} \\
& + \sum_{j\in U} \underbrace{(c_j - \hat{y}_G' A_{Gj} - \hat{y}_L' A_{Lj} - \hat{y}_E' A_{Ej})}_{=0} \hat{x}_j \\
& + \sum_{i\in G} \underbrace{\hat{y}_i}_{\geq 0} \underbrace{(A_{iP}x_P + A_{iN}x_N + A_{iU}x_U - b_i)}_{\geq 0} \\
& + \sum_{i\in L} \underbrace{\hat{y}_i}_{\leq 0} \underbrace{(A_{iP}x_P + A_{iN}x_N + A_{iU}x_U - b_i)}_{\leq 0} \\
& + \sum_{i\in E} \hat{y}_i \underbrace{(A_{iP}x_P + A_{iN}x_N + A_{iU}x_U - b_i)}_{=0} .
\end{aligned}
$$

The results follows easily using the Weak and Strong Duality Theorems for $(\mathcal{G})$ and $(\mathcal{H})$. □

The table below summarizes the duality relationships between the type of each primal constraint and the type of each associated dual variable. Highlighted in yellow are the relationships for the standard-form (P) and its dual (D). It is important to note that the columns are labeled "min" and "max", rather than primal and dual — the table is *not* correct if "min" and "max" are interchanged.

| | min | max | |
|---|---|---|---|
| constraints | $\geq$ | $\geq 0$ | variables |
| | $\leq$ | $\leq 0$ | |
| | $=$ | unres. | |
| variables | $\geq 0$ | $\leq$ | constraints |
| | $\leq 0$ | $\geq$ | |
| | unres. | $=$ | |

## 5.4 Theorems of the Alternative

In this section, we use linear-optimization duality to understand when a linear-optimization problem has a feasible solution. This fundamental result, expounded by Farkas[5], opened the door for studying linear inequalities and optimization.

**Theorem 5.11 (Farkas Lemma)**
Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ be given. Then exactly one of the following two systems has a solution.

$$\begin{array}{rcl} Ax & = & b\,; \\ x & \geq & \mathbf{0}\,. \end{array} \tag{I}$$

$$\begin{array}{rcl} y'b & > & 0\,; \\ y'A & \leq & \mathbf{0}'\,. \end{array} \tag{II}$$

*Proof.* It is easy to see that there cannot simultaneously be a solution $\hat{x}$ to (I) and $\hat{y}$ to (II). Otherwise we would have

$$0 \geq \underbrace{\hat{y}'A}_{\leq \mathbf{0}}\underbrace{\hat{x}}_{\geq \mathbf{0}} = \hat{y}'b > 0\,,$$

which is a clear inconsistency.

Next, suppose that (I) has no solution. Then the following problem is infeasible:

$$\begin{array}{rcl} \min \quad \mathbf{0}'x & & \\ Ax & = & b\,; \\ x & \geq & \mathbf{0}\,. \end{array} \tag{P}$$

Its dual is

$$\begin{array}{rcl} \max \quad y'b & & \\ y'A & \leq & \mathbf{0}'\,. \end{array} \tag{D}$$

Because (P) is infeasible, then (D) is either infeasible or unbounded. But $\hat{y} := \mathbf{0}$ is a feasible solution to (D), therefore (D) must be unbounded. Therefore, there exists a feasible solution $\hat{y}$ to (D) having objective value greater than zero (or even any fixed constant). Such a $\hat{y}$ is a solution to (II). □

**Remark 5.12**
Geometrically, the Farkas Lemma asserts that exactly one of the following holds:

(I) $b$ is in the "cone generated by the columns of $A$" (i.e., $b$ is a non-negative linear combination of the columns of $A$), or

(II) there is $\hat{y} \in \mathbb{R}^m$ that makes an acute angle with $b$ and a non-acute (i.e., right or obtuse) angle with every column of $A$ .

In the case of (II), considering the hyperplane $H$ containing the origin having $\hat{y}$ as its normal vector, this $H$ separates $b$ from the cone generated by the columns of $A$ . So, the Farkas Lemma has the geometric interpretation as a"Separating-Hyperplane Theorem." See Figure 5.1 for an example with $m = 2$ and $n = 4$ . The cone is red and the point $b$ that we separate from the cone is blue. The green point is a solution $\hat{y}$ for (II), and the dashed green line is the separating hyperplane. Notice how the (solid) green vector makes an acute angle with the blue vector and a non-acute angle with all points in the cone.

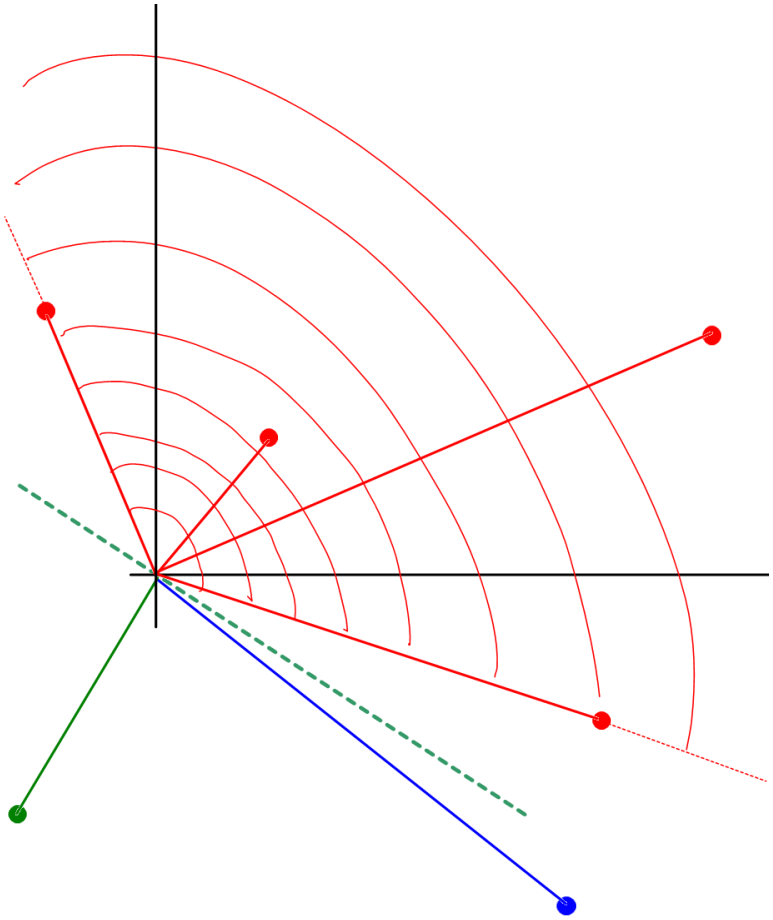


Figure 5.1: Case (II) of the Farkas Lemma

In a similar fashion to the Farkas Lemma, we can develop theorems of this type for feasible regions of other linear-optimization problems.

**Theorem 5.13 (Theorem of the Alternative for Linear Inequalities)**
Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ be given. Then exactly one of the following two systems has a solution.

$$Ax \geq b\,. \tag{I}$$

$$\begin{array}{rcl} y'b & > & 0\,; \\ y'A & = & \mathbf{0}'\,; \\ y & \geq & \mathbf{0}\,. \end{array} \tag{II}$$

*Proof.* It is easy to see that there cannot simultaneously be a solution $\hat{x}$ to (I) and $\hat{y}$ to (II). Otherwise we would have

$$0 = \underbrace{\hat{y}'A}_{=\mathbf{0}}\,\hat{x} \geq \hat{y}'b > 0\,,$$

which is a clear inconsistency.

Next, suppose that (I) has no solution. Then the following problem is infeasible:

$$\begin{array}{rcl} \min & \mathbf{0}'x & \\ & Ax & \geq b\,. \end{array} \tag{P}$$

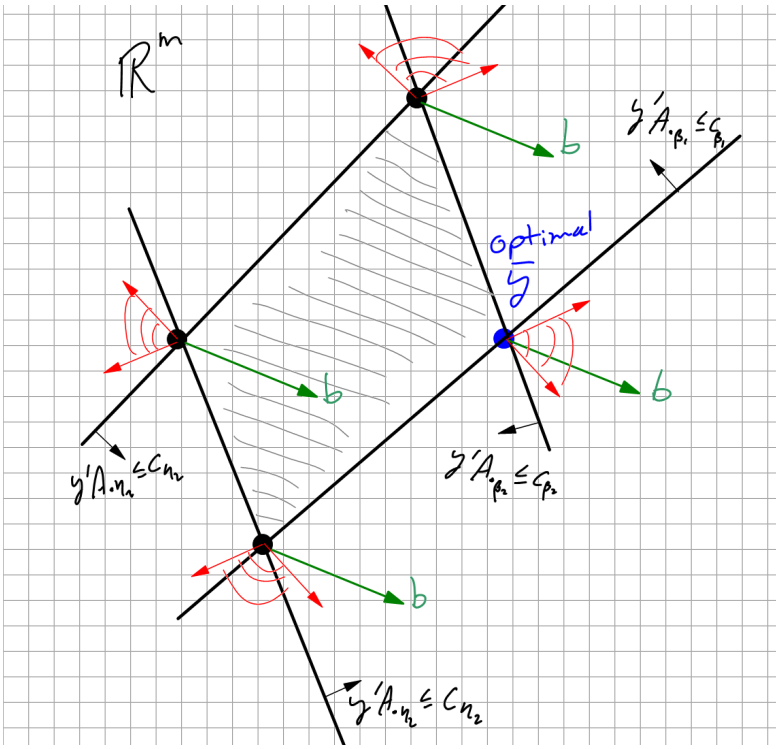
Its dual is

$$\begin{array}{rcl} \max & y'b & \\ & y'A & = \mathbf{0}'\,; \\ & y & \geq \mathbf{0}\,. \end{array} \tag{D}$$

Because (P) is infeasible, then (P) is either infeasible or unbounded. But $\hat{y} := \mathbf{0}$ is a feasible solution to (D), therefore (D) must be unbounded. Therefore, there exists a feasible solution $\hat{y}$ to (D) having objective value greater than zero (or even greater than any fixed constant). Such a $\hat{y}$ is a solution to (II). □

## 5.5 Exercises

**Exercise 5.1 (Dual picture)**
For the standard-form problem (P) and its dual (D), explain aspects of duality and complementarity using this picture:




**Exercise 5.2 (Duality and complementarity with `AMPL`)**
After optimization using `AMPL`, it is easy to get more information regarding primal and dual problems. If the constraints are `Constraints` , then the `AMPL` command '`display Constraints.dual`' displays the associated dual variables and '`display Constraints.slack`' displays the associated slackness. Using these commands, develop an example to verify the concepts of duality and complementarity developed in this chapter.

**Exercise 5.3 (Complementary slackness)**
Construct an example where we are given $\hat{x}$ and $\hat{y}$ and asked to check whether $\hat{x}$ is optimal using complementary slackness. I want your example to have the property that $\hat{x}$ is optimal, $\hat{x}$ and $\hat{y}$ are complementary, but $\hat{y}$ is not feasible.

The idea is to see an example where there is not a unique dual solution complementary to $\hat{x}$ , and so $\hat{x}$ is optimal, but we only verify it with another choice of $\hat{y}$.

**Exercise 5.4 (Over complementarity)**
With respect to the standard-form problem (P) and its dual (D), complementary solutions $\hat{x}$ and $\hat{y}$ are **overly complementary** if *exactly* one of

$$c_j - \hat{y}' A_{\cdot j} \text{ and } \hat{x}_j \text{ is } 0 \ , \text{ for } j = 1, 2, \ldots, n \ .$$

Prove that if (P) has an optimal solution, then there are always optimal solutions for (P) and (D) that are overly complementary.

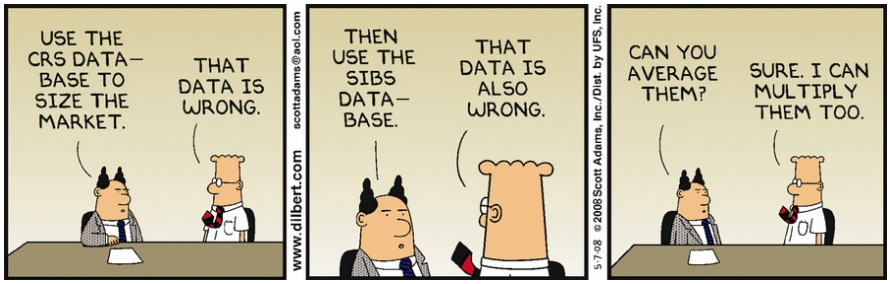
HINT: Let $v$ be the optimal objective value of (P). For each $j = 1, 2, \ldots, n$ , consider

$$\begin{array}{lrcl} \max & x_j & & \\ & c'x & \le & v \\ & Ax & = & b \ . \\ & x & \ge & \mathbf{0} \ . \end{array} \qquad (\mathrm{P}_j)$$

($P_j$) seeks an optimal solution of (P) that has $x_j$ positive. Using the dual of ($P_j$), show that if no optimal solution $\hat{x}$ of (P) has $\hat{x}_j$ positive, then there is an optimal solution $\hat{y}$ of (D) with $c_j - \hat{y}'A_{\cdot j}$ positive. Once you do this you can conclude that, for any fixed $j$, there are optimal solutions $\hat{x}$ and $\hat{y}$ with the property that

$$c_j - \hat{y}'A_{\cdot j} \text{ and } \hat{x}_j \text{ is } 0\ .$$

Take all of these $n$ pairs of solutions $\hat{x}$ and $\hat{y}$ and combine them appropriately to construct optimal $\hat{x}$ and $\hat{y}$ that are overly complementary.




**Exercise 5.5 (Another proof of a Theorem of the Alternative)**
Prove the Theorem of the Alternative for Linear Inequalities directly from the Farkas Lemma, without appealing to linear-optimization duality. HINT: Transform (I) of the Theorem of the Alternative for Linear Inequalities to a system of the form of (I) of the Farkas Lemma.

**Exercise 5.6 (A general Theorem of the Alternative)**
State and prove a "Theorem of the Alternative" for the system:

$$\begin{array}{rcrcrcl} A^G_P x_P & + & A^G_N x_N & + & A^G_U x_U & \geq & b^G\ ; \\ A^L_P x_P & + & A^L_N x_N & + & A^L_U x_U & \leq & b^L\ ; \\ A^E_P x_P & + & A^E_N x_N & + & A^E_U x_U & = & b^E\ ; \\ x_P \geq \mathbf{0}\ , & & x_N \leq \mathbf{0}\ . & & & & \end{array} \tag{I}$$

**Exercise 5.7 (Dual ray)**
Consider the linear-optimization problem

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & \geq & b\ ; \\ & x & \geq & \mathbf{0}\ . \end{array} \tag{P}$$

a) Suppose that (P) is infeasible. Then, by a 'Theorem of the Alternative' there is a solution to what system?

b) Suppose, further, that the dual (D) of (P) is feasible. Take a feasible solution $\hat{y}$ of (D) and a solution $\tilde{y}$ to your system of part (a) and combine them appropriately to prove that (D) is unbounded.

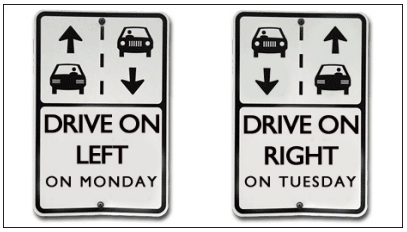
# Chapter 6

# Sensitivity Analysis


Our goal in this chapter is as follows:

- Learn how the optimal value of a linear-optimization problem behaves when the right-hand side vector and objective vector are varied.

## 6.1 Right-Hand Side Changes




We define a function $f : \mathbb{R}^m \to \mathbb{R}$ via

$$\begin{array}{rcll} f(b) & := & \min & c'x \\ & & & Ax = b\,; \\ & & & x \geq \mathbf{0}\,. \end{array} \qquad (\mathrm{P}_b)$$

That is, ($\mathrm{P}_b$) is simply (P) with the optimal objective value viewed as a function of its right-hand side vector $b$ .

### 6.1.1 Local analysis


Consider a fixed basis $\beta$ for ($P_b$). Associated with that basis is the basic solution $\bar{x}_\beta = A_\beta^{-1} b$ and the corresponding dual solution $\bar{y}' = c'_\beta A_\beta^{-1}$ . Let us assume that $\bar{y}$ is feasible for the dual of ($P_b$) — or, equivalently, $c'_\eta - \bar{y}' A_\eta \geq \mathbf{0}'$ . Considering the set $\mathcal{B}$ of $b \in \mathbb{R}^m$ such that $\beta$ is an optimal basis, is it easy to see that $\mathcal{B}$ is just the set of $b$ such that $\bar{b} := A_\beta^{-1} b \geq \mathbf{0}$ . That is, $\mathcal{B} \subset \mathbb{R}^m$ is the solution set of $m$ linear inequalities (in fact, it is a "simplicial cone" — we will return to this point in Section 6.1.3). Now, for $b \in \mathcal{B}$ , we have $f(b) = \bar{y}' b$ . Therefore, $f$ is a linear function on $b \in \mathcal{B}$ . Moreover, as long as $b$ is in the interior of $\mathcal{B}$ , we have $\frac{\partial f}{\partial b_i} = \bar{y}_i$ . So we have that $\bar{y}$ is the gradient of $f$ , as long as $b$ is in the interior of $\mathcal{B}$ . Now what does it mean for $b$ to be in the interior of $\mathcal{B}$ ? It just means that $\bar{b}_i > 0$ for $i = 1, 2, \ldots, m$ .

Let us focus our attention on changes to a single right-hand side element $b_i$ . Suppose that $\beta$ is an optimal basis of (P) , and consider the problem

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b + \Delta_i \boldsymbol{e_i} \, ; \\ & x & \geq & \mathbf{0} \, , \end{array} \qquad (P_i)$$

where $\Delta_i \in \mathbb{R}$ . The basis $\beta$ is feasible (and hence still optimal) for ($P_i$) if $A_\beta^{-1}(b + \Delta_i \boldsymbol{e_i}) \geq \mathbf{0}$ . Let $h^i := A_\beta^{-1} \boldsymbol{e_i}$ . So

$$[h^1, h^2, \ldots, h^m] = A_\beta^{-1} \, .$$

Then, the condition $A_\beta^{-1}(b + \Delta_i \boldsymbol{e_i}) \geq \mathbf{0}$ can be re-expressed as $\bar{b} + \Delta_i h^i \geq \mathbf{0}$ . It is straightforward to check that $\beta$ is feasible (and hence still optimal) for ($P_i$) as long as $\Delta_i$ is in the interval $[L_i, U_i]$ , where

$$L_i := \max_{k \, : \, h^i_k > 0} \left\{ -\bar{b}_k / h^i_k \right\} ,$$

and

$$U_i := \min_{k \, : \, h^i_k < 0} \left\{ -\bar{b}_k / h^i_k \right\} .$$

It is worth noting that it can be the case that $h^i_k \leq 0$ for all $k$, in which case we define $L_i := -\infty$, and it could be the case that $h^i_k \geq 0$ for all $k$, in which case we define $U_i := +\infty$,

In summary, for all $\Delta_i$ satisfying $L_i \leq \Delta_i \leq U_i$ , $\beta$ is an optimal basis of (P) . It is important to emphasize that this result pertains to changing one right-hand side element and holding all others constant. For a result on simultaneously changing all right-hand side elements, we refer to Exercise 6.3.

### 6.1.2 Global analysis


The domain of $f$ is the set of $b$ for which ($P_b$) has an optimal solution. Assuming that the dual of ($P_b$) is feasible (note that this just means that $y'A \leq c'$ has a solution), then ($P_b$) is never unbounded. So the domain of $f$ is just the set of $b \in \mathbb{R}^m$ such that ($P_b$) is feasible.

**Theorem 6.1**
The domain of $f$ is a convex set.

*Proof.* Suppose that $b^j$ is in the domain of $f$ , for $j = 1, 2$ . Therefore, there exist $x^j$ that are feasible for ($P_{b^j}$) , for $j = 1, 2$ . For any $0 < \lambda < 1$ , let $\hat{b} := \lambda b^1 + (1-\lambda)b^2$ , and consider $\hat{x} := \lambda x^1 + (1-\lambda)x^2$ . It is easy to check that $\hat{x}$ is feasible for ($P_{\hat{b}}$) , so we can conclude that $\hat{b}$ is in the domain of $f$ . □

Before going further, we need a few definitions. We consider functions $f : \mathbb{R}^m \to \mathbb{R}$. The domain of $f$ is the subset $S$ of $\mathbb{R}^m$ on which $f$ is defined. We assume that $S$ is a convex set. A function $f : \mathbb{R}^m \to \mathbb{R}$ is a **convex function** on its domain $S$, if

$$f(\lambda u^1 + (1-\lambda)u^2) \leq \lambda f(u^1) + (1-\lambda) f(u^2) ,$$

for all $u^1, u^2 \in S$ and $0 < \lambda < 1$ . That is, $f$ is never underestimated by linear interpolation.

A function $f : \mathbb{R}^m \to \mathbb{R}$ is an **affine function**, if it has the form $f(u_1, \ldots, u_m) = a_0 + \sum_{i=1}^m a_i u_i$ , for constants $a_0, a_1, \ldots, a_m \in \mathbb{R}$ . If $a_0 = 0$ , then we say that $f$ is a **linear function**. Affine (and hence linear) functions are easily seen to be convex.

A function $f : \mathbb{R}^m \to \mathbb{R}$ having a convex set as its domain is a ***convex* piecewise-linear function** if, on its domain, it is the *pointwise maximum* of a finite number of affine functions.

It would be strange to refer to a function as being "convex piecewise-linear" if it were not convex! The next result justifies the moniker.

**Theorem 6.2**
If $\check{f}$ is a convex piecewise-linear function, then it is a convex function.

*Proof.* Let

$$\check{f}(u) := \max_{1 \le i \le k} \{f_i(u)\},$$

for $u$ in the domain of $\check{f}$, where each $f_i$ is an affine function. That is, $\check{f}$ is the pointwise maximum of a finite number ($k$) of affine functions.

Then, for $0 < \lambda < 1$ and $u^1, u^2 \in \mathbb{R}^m$,

$$\begin{aligned}
\check{f}(\lambda u^1 + (1-\lambda)u^2) &= \max_{1 \le i \le k} \left\{ f_i(\lambda u^1 + (1-\lambda)u^2) \right\} \\
&= \max_{1 \le i \le k} \left\{ \lambda f_i(u^1) + (1-\lambda) f_i(u^2) \right\} \text{ (using the definition of affine)} \\
&\le \max_{1 \le i \le k} \left\{ \lambda f_i(u^1) \right\} + \max_{1 \le i \le k} \left\{ (1-\lambda) f_i(u^2) \right\} \\
&= \lambda \max_{1 \le i \le k} \left\{ f_i(u^1) \right\} + (1-\lambda) \max_{1 \le i \le k} \left\{ f_i(u^2) \right\} \\
&= \lambda \check{f}(u^1) + (1-\lambda)\check{f}(u^2) .
\end{aligned}$$

□

**Theorem 6.3**
$f$ is a convex piecewise-linear function on its domain.

*Proof.* We refer to the dual

$$\begin{array}{rcll} f(b) & := & \max & y'b \\ & & & y'A \le c' ; \end{array} \qquad (\mathrm{D}_b)$$

of ($\mathrm{P}_b$).

A basis $\beta$ is feasible or not for ($D_b$), independent of $b$ . Thinking about it this way, we can see that

$$f(b) = \max\left\{\left(c'_\beta A_\beta^{-1}\right) b \ : \ \beta \text{ is a dual feasible basis}\right\},$$

and so $f$ is a convex piecewise-linear function, because it is the pointwise maximum of a finite number of affine (even linear) functions. □

### 6.1.3 A brief detour: the column geometry for the Simplex Algorithm

In this section, we will describe a geometry for visualizing the Simplex Algorithm.[6] The ordinary geometry for a standard-form problem, in the space of the non-basic variables for same choice of basis, can be visualized when $n - m = 2$ or 3. The "column geometry" that we will describe is in $\mathbb{R}^{m+1}$ , so it can be visualized when $m + 1 = 2$ or 3. Note that the graph of the function $f(b)$ (introduced at the start of this chapter) is also in $\mathbb{R}^{m+1}$ , which is why we take the present detour.

We think of the $n$ points

$$\begin{pmatrix} c_j \\ A_j \end{pmatrix},$$

for $j = 1, 2, \ldots, n$ , and the additional so-called *requirement line*

$$\left\{\begin{pmatrix} z \\ b \end{pmatrix} \ : \ z \in \mathbb{R}\right\}.$$

We think of the first component of these points and of the line as the vertical dimension; so the requirement line is thought of as vertical. It of particular interest to think of the cone generated by the $n$ points. That is,

$$K := \left\{\begin{pmatrix} c'x \\ Ax \end{pmatrix} \in \mathbb{R}^{m+1} \ : \ x \geq \mathbf{0}\right\}.$$

Notice how the top coordinate of a point in the cone gives the objective value of the associated $x$ for (P). So the goal of solving (P) can be thought of as that of finding a point on the intersection of the requirement line and the cone that is as low as possible.

Restricting ourselves to a basis $\beta$ , we have the cone

$$K_\beta := \left\{ \begin{pmatrix} c'_\beta x_\beta \\ A_\beta x_\beta \end{pmatrix} \in \mathbb{R}^{m+1} \ : \ x_\beta \geq \mathbf{0} \right\}.$$

The cone $K_\beta$ is an "$m$-dimensional simplicial cone." Next, we observe that if $\beta$ is a feasible basis, then $K_\beta$ intersects the requirement line uniquely at the point

$$\begin{pmatrix} c'_\beta \bar{x}_\beta \\ A_\beta \bar{x}_\beta \end{pmatrix},$$

where $\bar{x}$ is the basic solution associated with $\beta$ .

In a pivot of the Simplex Algorithm from basis $\beta$ to basis $\tilde{\beta}$, we do so with the goal of having $K_{\tilde{\beta}}$ intersect the requirement line at a *lower* point than did $K_\beta$ . In Figure 6.1 ($m = 2$ and the coordinate axes are the red lines), we see an example depicting a single pivot. $K_\beta$ is the yellow cone, intersecting the blue requirement line at the red point. After the pivot (with one cone generator exchanged), we have the green cone $K_{\tilde{\beta}}$ intersecting the requirement line at the pink point.

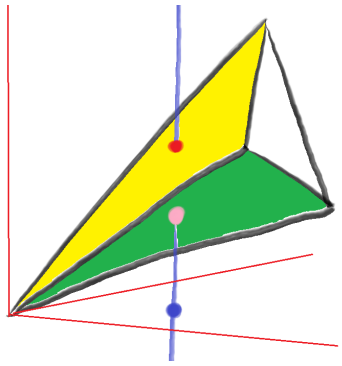


Figure 6.1: A simplex pivot

So at each iteration of the Simplex Algorithm, we exchange a single "generator" of the simplicial cone $K_\beta$ associated with our basis $\beta$, to descend along the requirement line, ultimately finding a point of $K$ that meets the requirement at its lowest point.


### 6.1.4 Reduced costs as dual values

It will turn out that we can regard reduced costs as values of dual variables for non-negativity constraints. To pursue this idea, via the following linear-optimization prob-

lem, we consider changing the lower bound of a single variable $x_k$:

$$\begin{array}{rlllll} v(\delta_k) := & \min & c'x & & & \underline{\text{dual variables}} \\ & & Ax & = & b\,; & y \\ & & x & \geq & \mathbf{0} + \delta_k \boldsymbol{e_k}\,. & w \end{array} \qquad (\mathrm{P}_k(\delta_k))$$

With $\delta_k = 0$, we have the ordinary standard-form problem

$$\begin{array}{rlllll} z := \min & c'x & & & & \underline{\text{dual variables}} \\ & Ax & = & b\,; & & y \\ & x & \geq & \mathbf{0}\,, & & \end{array} \qquad (\mathrm{P})$$

so we can see that $(\mathrm{P}_k(0))$ and (P) are equivalent.

**Theorem 6.4**

(i) $v$ is a convex piecewise-linear function on its domain.

Let $\beta, \eta$ be an optimal basic partition for (P), and assume that $k \in \eta$. Let $\bar{y}$ be the associated dual solution, and let $\bar{c}_\eta$ be the reduced costs for the non-basic variables.

(ii) Defining $\bar{w} \in \mathbb{R}^n$ by

$$\begin{array}{rcll} \bar{w}_\beta & := & \mathbf{0} & \in \mathbb{R}^m\,; \\ \bar{w}_\eta & := & \bar{c}_\eta & \in \mathbb{R}^{n-m}\,, \end{array}$$

we have that together, $\bar{y}$ and $\bar{w}$ are optimal for the dual of $(\mathrm{P}_k(0))$.

(iii) If $\bar{b}_i > 0$ for $i = 1, 2, \ldots, m$, then $\frac{dv}{d\delta_k}\big|_{\delta_k=0} = \bar{c}_k$ .

*Proof.* Part (i) can easily be established in the same manner as Theorem 6.3.

For (ii), we start by observing that the dual of $(\mathrm{P}_k(\delta_k))$ is

$$\begin{array}{rllllll} \max & y'b & + & w_k\delta_k & & & \\ & y'A & + & w' & = & c'\,; & \\ & & & w & \geq & \mathbf{0}\,, & \end{array} \qquad (\mathrm{D}_k(\delta_k))$$

or equivalently

$$\begin{array}{rllllll} \max & y'b & + & w_k\delta_k & & & \\ & y'A_\beta & + & w'_\beta & = & c'_\beta\,; & \\ & y'A_\eta & + & w'_\eta & = & c'_\eta\,; & \\ & & & w & \geq & \mathbf{0}\,. & \end{array} \qquad (\mathrm{D}_k(\delta_k))$$

Recalling that $\bar{y}' = c'_\beta A_\beta^{-1}$, we can easily check that together, $\bar{y}$ and $\bar{w}$ are feasible for $(\mathrm{D}_k(\delta_k))$. We can see that the objective value of $\bar{y}$ and $\bar{w}$ in $(\mathrm{D}_k(0))$ is $\bar{y}'b$ (because $\delta_k = 0$ in $(\mathrm{D}_k(0))$), and so together, $\bar{y}$ and $\bar{w}$ are optimal for $(\mathrm{D}_k(0))$.

For part (iii), consider the point $\hat{x}$ defined by

$$\begin{array}{rcl} \hat{x}_\eta & := & \delta_k \boldsymbol{e_k}\,; \\ \hat{x}_\beta & := & A_\beta^{-1}(b - \delta_k A_k)\,, \end{array}$$

This solution corresponds to starting at $\bar{x}$ and changing the value of $x_k$ from 0 to $\delta_k$, offsetting this change by adjusting the values of the basic variables. For all sufficiently small $\delta_k$ (positive or negative), $\hat{x}$ is feasible for ($P_k(\delta_k)$). The constraints of ($D_k(\delta_k)$) do not depend on $\delta_k$, so together, $\bar{y}$ and $\bar{w}$ are feasible for ($D_k(\delta_k)$). The objective value of $\hat{x}$ in ($P_k(\delta_k)$) is

$$c'_\beta \hat{x}_\beta + c'_\eta \hat{x}_\eta = c'_\beta A_\beta^{-1}(b - \delta_k A_k) + \delta_k c'_\eta \boldsymbol{e_k} = \bar{y}'b + \delta_k \bar{c}_k \ ,$$

which is the objective value of $\bar{y}$ and $\bar{w}$ in ($D_k(\delta_k)$). So we have that, for sufficiently small $\delta_k$, $v(\delta_k) = \bar{y}'b + \delta_k \bar{c}_k = z + \delta_k \bar{c}_k$ . The result follows. □

## 6.2 Objective Changes

> *"Here is what is needed for Occupy Wall Street to become a force for change: a clear, and clearly expressed, objective. Or two."* — Elayne Boosler

We define a function $g : \mathbb{R}^n \to \mathbb{R}$ via

$$\begin{array}{rcl} g(c) := \min & c'x & \\ & Ax = b\,; & \\ & x \geq \mathbf{0}\,. & \end{array} \qquad (P^c)$$

That is, ($P^c$) is simply (P) with the optimal objective value viewed as a function of its objective vector $c$ .

### 6.2.1 Local analysis

Consider a fixed basis $\beta$ for ($P^c$). Associated with that basis is the basic solution $\bar{x}_\beta = A_\beta^{-1}b$ and the corresponding dual solution $\bar{y}' = c'_\beta A_\beta^{-1}$ . Let is assume that $\bar{x}$ is feasible for ($P^c$) — or, equivalently, $A_\beta^{-1}b \geq \mathbf{0}$ . Considering the set $\mathcal{C}$ of $c \in \mathbb{R}^n$ such that $\beta$ is an optimal basis, is it easy to see that this is just the set of $c$ such that $c'_\eta - c'_\beta A_\beta^{-1} A_\eta \geq \mathbf{0}'$ . That is, $\mathcal{C} \subset \mathbb{R}^n$ is the solution set of $n - m$ linear inequalities (in fact, it is a cone). Now, for $c \in \mathcal{C}$ , we have $g(c) = c'_\beta \bar{x}_\beta$ . Therefore, $g$ is a linear function on $c \in \mathcal{C}$ .

### 6.2.2 Global analysis

The domain of $g$ is the set of $c$ for which ($P^c$) has an optimal solution. Assuming that ($P^c$) is feasible, then the domain of $g$ is just the set of $c \in \mathbb{R}^n$ such that ($P^c$) is not unbounded.

Similarly to the case of variations in the right-hand side vector $b$, we have the following two results.

**Theorem 6.5**
The domain of $g$ is a convex set.

A function $g : \mathbb{R}^n \to \mathbb{R}$ is a **concave function** on its domain $S$, if

$$g(\lambda u^1 + (1-\lambda)u^2) \geq \lambda g(u^1) + (1-\lambda)g(u^2) \ ,$$

for all $u^1, u^2 \in S$ and $0 < \lambda < 1$ . That is, $f$ is never overestimated by linear interpolation. The function $g$ is a ***concave* piecewise-linear function** if it is the pointwise *minimum* of a finite number of affine functions.

**Theorem 6.6**
$g$ is a concave piecewise-linear function on its domain.

### 6.2.3 Local sensitivity analysis with a modeling language

To carry out some local sensitivity analysis with AMPL, use the following options

```
option solver cplex;
option cplex_options 'sensitivity';
```

Then use the AMPL command `solve` to solve the problem (in particular, to determine an optimal basis — though AMPL will not tell you the optimal basis that it found).

AMPL has suffixes to retrieve some local sensitivity information from the solver. For a constraint named `<Constraint>`, `<Constraint>.current` gives the current right-hand side value, while `<Constraint>.down` and `<Constraint>.up` give lower and upper limits on that right-hand side value, such that the optimal basis remains optimal. Changing a particular right-hand side value, in this way, assumes that the right-hand side values for the other constraints are held constant (though see Exercise 6.3 for the case of simultaneous changes). In our notation from Section 6.1.1, if `<Constraint>` corresponds to $\sum_{j=1}^{n} a_{ij}x_j = b_i$ (that is, the $i$-th constraint of ($P_b$), then

$$\begin{aligned} b_i &= \texttt{<Constraint>.current} \ , \\ L_i &= \texttt{<Constraint>.down} - \texttt{<Constraint>.current} \ , \\ U_i &= \texttt{<Constraint>.up} - \texttt{<Constraint>.current} \ . \end{aligned}$$

In a similar manner, for a variable named `<Variable>`, we have that `<Variable>.current`, `<Variable>.down` and `<Variable>.up` give the current value, and lower and upper limits on the coefficient of `<Variable>` in the objective function. Additionally, `<Variable>.lrc`

give the reduced cost of `<Variable>`. As we learned in Section 6.1.4, this reduced cost of `<Variable>` is the value of the optimal dual variable for the non-negativity constraint on `<Variable>`. If `<Variable>` also has an explicit upper bound, then `<Variable>.urc` is the value of the dual variable for the upper-bound constraint on `<Variable>`.

## 6.3 Exercises

**Exercise 6.1 (Illustrate local sensitivity analysis)**
Make an original example to illustrate the local sensitivity-analysis concepts of this chapter. Use a combination of hand calculations and `AMPL`.

**Exercise 6.2 (Illustrate global sensitivity analysis)**
Using `AMPL`, make an original example, with at least three constraints, graphing the objective value of $(P)$ , as a single `b[i]` is varied from $-\infty$ to $+\infty$ . As you work on this, bear in mind Theorem 6.3.

**Exercise 6.3 (*"I feel that I know the change that is needed."* — Mahatma Gandhi)**
We are given $2m$ numbers satisfying $L_i \leq 0 \leq U_i$ , $i = 1, 2, \ldots, m$ . Let $\beta$ be an optimal basis for all of the $m$ problems

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b + \Delta_i \boldsymbol{e_i} \; ; \\ & x & \geq & \mathbf{0} \; , \end{array} \qquad (\mathrm{P}_i)$$

for all $\Delta_i$ satisfying $L_i \leq \Delta_i \leq U_i$ . Let's be clear on what this means: For each $i$ *individually*, the basis $\beta$ is optimal when the $i$th right-hand side component is changed from $b_i$ to $b_i + \Delta_i$ , as long as $\Delta_i$ is in the interval $[L_i, U_i]$ (see Section 6.1.1).

The point of this problem is to be able to say something about *simultaneously* changing all of the $b_i$ . Prove that we can simultaneously change $b_i$ to

$$\tilde{b}_i := b_i + \lambda_i \left\{ \begin{array}{c} L_i \\ U_i \end{array} \right\},$$

where $\lambda_i \geq 0$ , when $\sum_{i=1}^{m} \lambda_i \leq 1$ . [Note that in the formula above, for each $i$ we can pick either $L_i$ (a decrease) or $U_i$ (an increase)].

**Exercise 6.4 (Domain for objective variations)**
Prove Theorem 6.5.

**Exercise 6.5 (Concave piecewise-linear function)**
Prove Theorem 6.6.

# Chapter 7

# Large-Scale Linear Optimization

Our goals in this chapter are as follows:

- To see some approaches to large-scale linear-optimization problems
- In particular, to learn about decomposition, Lagrangian relaxation and column generation.
- Also, via a study of the "cutting-stock problem," we will have a first glimpse at some issues associated with integer-linear optimization.

## 7.1 Decomposition

In this section we describe what is usually known as **Dantzig-Wolfe Decomposition**. It is an algorithm aimed at efficiently solving certain kinds of structured linear-optimization problems. The general viewpoint is that we might have a *very* efficient way to solve a certain type of structured linear-optimization problem, if it were not for a small number of constraints that break the structure. For example, the constraint matrix might have the form in Figure 7.1, where if it were not for the top constraints, the optimization problem would separate into many small problems[7].

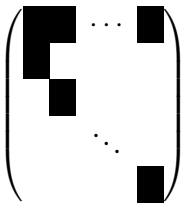


Figure 7.1: Nearly separates

### 7.1.1 The master reformulation

**Theorem 7.1 (The Representation Theorem)**
Let

$$\begin{array}{rcll} \min & c'x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,. \end{array} \qquad \text{(P)}$$

Suppose that (P) has a non-empty feasible region. Let $\mathcal{X} := \{\hat{x}^j \,:\, j \in \mathcal{J}\}$ be the set of basic-feasible solutions of (P), and let $\mathcal{Z} := \{\hat{z}^k \,:\, k \in \mathcal{K}\}$ be the set of basic-feasible rays of (P). Then the feasible region of (P) is equal to

$$\left\{ \sum_{j\in\mathcal{J}} \lambda_j \hat{x}^j + \sum_{k\in\mathcal{K}} \mu_k \hat{z}^k \;:\; \sum_{j\in\mathcal{J}} \lambda_j = 1\,;\; \lambda_j \geq 0\,,\; j \in \mathcal{J}\,;\quad \mu_k \geq 0\,,\; k \in \mathcal{K} \right\}.$$

*Proof.* Let $S$ be the feasible region of (P) . Let

$$S' = \left\{ \sum_{j\in\mathcal{J}} \lambda_j \hat{x}^j + \sum_{k\in\mathcal{K}} \mu_k \hat{z}^k \;:\; \sum_{j\in\mathcal{J}} \lambda_j = 1\,;\; \lambda_j \geq 0\,,\; j \in \mathcal{J}\,;\quad \mu_k \geq 0\,,\; k \in \mathcal{K} \right\}.$$

We will demonstrate that $S = S'$ . It is very easy to check that $S' \subset S$ , and we leave that to the reader. For the other direction, suppose that $\hat{x} \in S$ , and consider the system

$$\begin{array}{rcll} \displaystyle\sum_{j\in\mathcal{J}} \lambda_j \hat{x}^j + \sum_{k\in\mathcal{K}} \mu_k \hat{z}^k & = & \hat{x}\,; & \\ \displaystyle\sum_{j\in\mathcal{J}} \lambda_j & = & 1\,; & \\ \lambda_j \geq 0\,,\; j \in \mathcal{J}\,; \quad \mu_k \geq 0\,,\; k \in \mathcal{K}\,. & & & \end{array} \qquad \text{(I)}$$

Keep in mind that in (I), $\hat{x}$ is fixed as well as are the $\hat{x}^j$ and the $\hat{z}^k$ — the variables are the $\lambda_j$ and the $\mu_k$ . By way of establishing that $S \subset S'$ , suppose that $\hat{x} \notin S'$ — that is, suppose that (I) has no solution. Applying the Farkas Lemma to (I) , we see that the system

$$\begin{array}{rcrcll} w'\hat{x} & + & t & > & 0\,; & \\ w'\hat{x}^j & + & t & \leq & 0\,, & \forall\, j \in \mathcal{J}\,; \\ w'\hat{z}^k & & & \leq & 0\,, & \forall\, k \in \mathcal{K} \end{array} \tag{II}$$

has a solution, say $\hat{w}, \hat{t}$ . Now, consider the linear-optimization problem

$$\begin{array}{rrcl} \min & -\hat{w}'x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,. \end{array} \tag{$\hat{\text{P}}$}$$

($\hat{\text{P}}$) cannot be unbounded, because $-\hat{w}'\hat{z}^k \geq 0$ , for all $k \in \mathcal{K}$ . In addition, every basic feasible solution of ($\hat{\text{P}}$) has objective value at least $\hat{t}$ . By Theorem 5.1 (the Strong Optimal Basis Theorem), this implies that the optimal value of ($\hat{\text{P}}$) is at least $\hat{t}$ . But the objective value $-\hat{w}'\hat{x}$ of $\hat{x}$ is less than $\hat{t}$ . Therefore, $\hat{x}$ cannot be feasible. That is, $\hat{x} \notin S$ . □

**Corollary 7.2 (The Decomposition Theorem)**
Let

$$\begin{array}{rrcl} \min & c'x & & \\ & Ex & = & h\,; \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,. \end{array} \tag{Q}$$

Let $S := \{x \in \mathbb{R}^n \,:\, Ax = b\,, x \geq \mathbf{0}\}$ , let $\mathcal{X} := \{\hat{x}^j \,:\, j \in \mathcal{J}\}$ be the set of basic-feasible solutions $S$, and let $\mathcal{Z} := \{\hat{z}^k \,:\, k \in \mathcal{K}\}$ be the set of basic-feasible rays of $S$. Then (Q) is equivalent to the **Master Problem**

$$\begin{array}{rll} \min & \displaystyle\sum_{j\in\mathcal{J}} \left(c'\hat{x}^j\right)\lambda_j + \sum_{k\in\mathcal{K}} \left(c'\hat{z}^k\right)\mu_k & \\ & \displaystyle\sum_{j\in\mathcal{J}} \left(E\hat{x}^j\right)\lambda_j + \sum_{k\in\mathcal{K}} \left(E\hat{z}^k\right)\mu_k \quad = \quad h\,; & \\ & \displaystyle\sum_{j\in\mathcal{J}} \lambda_j \quad = \quad 1\,; & \\ & \lambda_j \geq 0\,,\ j \in \mathcal{J}\,, \quad \mu_k \geq 0\,,\ k \in \mathcal{K}\,. & \end{array} \tag{M}$$

*Proof.* Using the Representation Theorem, we just substitute the expression

$$\sum_{j\in\mathcal{J}} \lambda_j \hat{x}^j + \sum_{k\in\mathcal{K}} \mu_k \hat{z}^k$$

for $x$ in $c'x$ and in $Ex = h$ of (Q), and it is easy to see that (M) is equivalent to (Q). □

Decomposition is typically applied in a way such that the constraints defining (S) are somehow relatively "nice," and the constraints $Ex = h$ somehow are "complicating" the situation. For example, we may have a problem where the overall constraint matrix has the form depicted in Figure 7.1. In such a scenario, we would let

$$E := (\blacksquare \cdots \blacksquare)$$

and

$$A := \begin{pmatrix} \blacksquare & & \\ & \ddots & \\ & & \blacksquare \end{pmatrix}$$

### 7.1.2 Solution of the Master via the Simplex Algorithm

Next, we describe how to solve (M) using the Simplex Algorithm. Our viewpoint is that we cannot write out (M) explicitly; there are typically far too many variables. But we can reasonably maintain a basic solution of (M) , because the number of constraints is just one more than the number of constraints in $Ex = h$ .

The only part of the Simplex Algorithm that is sensitive to the total number of variables is the step in which we check whether there is a variable with a negative reduced cost. So rather than checking this directly, we will find an indirect way to carry it out.

Toward this end, we define dual variables $y$ and $\sigma$ for (M) .

$$\begin{array}{lrcll} \min & \displaystyle\sum_{j\in\mathcal{J}} \left(c'\hat{x}^j\right)\lambda_j + \sum_{k\in\mathcal{K}} \left(c'\hat{z}^k\right)\mu_k & & & \text{dual variables} \\ & \displaystyle\sum_{j\in\mathcal{J}} \left(E\hat{x}^j\right)\lambda_j + \sum_{k\in\mathcal{K}} \left(E\hat{z}^k\right)\mu_k & = & h\,; & y \\ & \displaystyle\sum_{j\in\mathcal{J}} \lambda_j & = & 1\,; & \sigma \\ & \lambda_j \geq 0\,,\ j\in\mathcal{J}\,, \quad \mu_k \geq 0\,,\ k\in\mathcal{K}\,. & & & \end{array} \tag{M}$$

While $\sigma$ is a scalar variable, $y$ is a vector with a component for each row of $E$ .

We will temporarily put aside how we calculate values for $y$ and $\sigma$ , but for now we suppose that we have a basic solution of (M) and an associated dual solution $\bar{y}$ and $\bar{\sigma}$ .

**Entering variable.** With respect to such a dual solution, the reduced cost of a variable $\lambda_j$ is

$$\left(c'\hat{x}^j\right) - \bar{y}'\left(E\hat{x}^j\right) - \bar{\sigma} \;=\; -\bar{\sigma} + \left(c' - \bar{y}'E\right)\hat{x}^j\,.$$

It is noteworthy that with the dual solution fixed (at $\bar{y}$ and $\bar{\sigma}$), the reduced cost of $\lambda_j$ is a constant ($-\bar{\sigma}$) plus a linear function of $\hat{x}^j$ . A variable $\lambda_j$ is eligible to enter the basis if its reduced cost is negative. So we *formulate* the following optimization problem:

$$\begin{array}{lrcl} -\bar{\sigma} + \min & \left(c' - \bar{y}'E\right)x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,. \end{array} \tag{SUB}$$

If the "subproblem" (SUB) has as optimal solution, then it has a basic optimal solution — that is, an $\hat{x}^j$ . In such a case, if the optimal objective value of (SUB) is negative, then the $\lambda_j$ corresponding to the optimal $\hat{x}^j$ is eligible to enter the current basis of (M) . On the other hand, if the optimal objective value of (SUB) is non-negative, then we have a proof that no non-basic $\lambda_j$ is eligible to enter the current basis of (M) .

If (SUB) is unbounded, then (SUB) has a basic feasible ray $\hat{z}^k$ having negative objective value. That is, $(c' - \bar{y}'E)\,\hat{z}^k < 0$ . Amazingly, the reduced cost of $\mu_k$ is precisely $\left(c'\hat{z}^k\right) - \bar{y}'\left(E\hat{z}^k\right) = (c' - \bar{y}'E)\,\hat{z}^k$ , so, in fact, $\mu_k$ is then eligible to enter the current basis of (M) .

**Leaving variable.** To determine the choice of leaving variable, let us suppose that $B$ is the basis matrix for (M) . Note that $B$ consists of at least one column of the form

$$\begin{pmatrix} E\hat{x}^j \\ 1 \end{pmatrix}$$

and columns of the form

$$\begin{pmatrix} E\hat{z}^k \\ 0 \end{pmatrix}.$$

With respect to the current basis, to carry out the *ratio test* of the Simplex Algorithm, we simply need

$$B^{-1}\begin{pmatrix} h \\ 1 \end{pmatrix}$$

and either

$$B^{-1}\begin{pmatrix} E\hat{x}^j \\ 1 \end{pmatrix}$$

if $\lambda_j$ is entering the basis, or

$$B^{-1}\begin{pmatrix} E\hat{z}^k \\ 0 \end{pmatrix}$$

if $\mu_k$ is entering the basis.

**Calculation of basic primal and dual solutions.** It is helpful to explain a bit about the calculation of basic primal and dual solutions. As we have said, $B$ consists of at least one column of the form

$$\begin{pmatrix} E\hat{x}^j \\ 1 \end{pmatrix}$$

and columns of the form

$$\begin{pmatrix} E\hat{z}^k \\ 0 \end{pmatrix}.$$

So organizing the basic variables $\lambda_j$ and $\mu_k$ into a vector $\zeta$ , with their order appropriately matched with the columns of $B$ , the vector $\bar{\zeta}$ of values of $\zeta$ is precisely the solution of

$$B\zeta = \begin{pmatrix} h \\ 1 \end{pmatrix}.$$

That is,

$$\bar{\zeta} = B^{-1} \begin{pmatrix} h \\ 1 \end{pmatrix} .$$

Finally, organizing the costs $c'\hat{x}^j$ and $c'\hat{z}^k$ of the basic variables $\lambda_j$ and $\mu_k$ into a vector $\xi$ , with their order appropriately matched with the columns of $B$ , the associated dual solution $(\bar{y}, \bar{\sigma})$ is precisely the solution of

$$(y', \sigma)B = \xi' .$$

That is,

$$(\bar{y}', \bar{\sigma}) = \xi' B^{-1} .$$

**Starting basis.** It is not obvious how to construct a feasible starting basis for (M) ; after all, we may not have at hand any basic feasible solutions and rays of (S) . Next, we give a simple recipe. We assume that the problem is first in a slightly different form, where the complicating constraints are inequalities:

$$\begin{array}{rrcl} \min & \tilde{c}'\tilde{x} & & \\ & \tilde{E}\tilde{x} & \leq & h\,; \\ & \tilde{A}\tilde{x} & = & b\,; \\ & \tilde{x} & \geq & \mathbf{0}\,, \end{array} \qquad (\tilde{\mathrm{Q}})$$

where $\tilde{x}$ is a vector of $n - m$ variables, $\tilde{E} \in \mathbb{R}^{m\times(n-m)}$ , $\tilde{A} \in \mathbb{R}^{p\times(n-m)}$ , and all of the other data has dimensions that conform.

Introducing $m$ slack variables for the $\tilde{E}\tilde{x} \leq h$ constraints, we have the equivalent problem

$$\begin{array}{rrcccl} \min & \tilde{c}'\tilde{x} & & & & \\ & \tilde{E}\tilde{x} & + & s & = & h\,; \\ & \tilde{A}\tilde{x} & & & = & b\,; \\ & \tilde{x} & , & s & \geq & \mathbf{0}\,. \end{array}$$

Now, it is convenient to define $x \in \mathbb{R}^n$ by

$$x_j := \begin{cases} \tilde{x}_j\,, & \text{for } j = 1, \ldots, n-m\,; \\ s_{j-(n-m)}\,, & \text{for } j = n-m+1, \ldots, n\,, \end{cases}$$

and to define $E \in \mathbb{R}^{m\times n}$ by

$$E_j := \begin{cases} \tilde{E}_j\,, & \text{for } j = 1, \ldots, n-m\,; \\ \boldsymbol{e_{j-(n-m)}}\,, & \text{for } j = n-m+1, \ldots, n\,, \end{cases}$$

and $A \in \mathbb{R}^{p\times n}$ by

$$E_j := \begin{cases} \tilde{A}_j\,, & \text{for } j = 1, \ldots, n-m\,; \\ \mathbf{0}\,, & \text{for } j = n-m+1, \ldots, n\,. \end{cases}$$

That is, $x = \begin{pmatrix} \tilde{x} \\ s \end{pmatrix}$ , $E = [\tilde{E}, \mathbf{I}_m]$ , $A = [\tilde{A}, \mathbf{0}]$ , so now our system has the familiar equivalent form

$$\begin{array}{rrcl} \min & c'x & & \\ & Ex & = & h\,; \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,. \end{array} \qquad (\mathrm{Q})$$

Now, we have everything organized to specify how to get an initial basis for (M). First, we take as $\hat{x}^1$ *any* basic feasible solution of (S) . Such a solution can be readily obtained using our usual (phase-one) methodology of the Simplex Algorithm. Next, we observe that for $k = 1, 2 \ldots, m$ , $\hat{z}^k := \boldsymbol{e_{n-(m+k)}}$ is a basic feasible ray of $S$ . The ray $\hat{z}^k$ corresponds to increasing the slack variable $s_k$ (arbitrarily); because, the slack variable $s_k = x_{n-m+k}$ does not truly enter into the $Ax = b$ constraints (i.e., the coefficients of $s_k$ is zero in those constraints), we do not have to alter the values of any other variables when $s_k$ is increased.

So, we have as an initial set of basic variables $\mu_1, \ldots, \mu_m$ (corresponding to $\hat{z}^1, \ldots, \hat{z}^m$) and $\lambda_1$ (corresponding to $x^1$). Notice that $E\hat{z}^k = \boldsymbol{e_k}$ , for $k = 1, 2 \ldots, m$ . Organizing our basic variables in the order $\mu_1, \mu_2, \ldots, \mu_m, \lambda_1$, we have the initial basis matrix

$$B = \left( \begin{array}{c|c} I_m & E\hat{x}^1 \\ \hline \mathbf{0}' & 1 \end{array} \right).$$

It is very easy to see that this is an invertible matrix.

Finally, it is very important to realize that we have given a recipe for finding an initial basic solution of (M) . [It happens that this basic solution is feasible precisely when $x^1$ satisfies the $Ex = h$ constraints]. But this solution may not be feasible. If the associated basic solution of (M) is not feasible (i.e., has any negative components), then we would introduce an artificial variable and do a phase-one procedure.

**A demonstration implementation.**


It is not completely trivial to write a small `MATLAB` code for the Decomposition Algorithm. First of all, we solve the subproblems (SUB) using functionality of `MATLAB`. With this, if a linear-optimization problem is unbounded, `MATLAB` does not give us access to the basic-feasible ray on which the objective is decreasing. Because of this, our `MATLAB` code quits if it encounters an unbounded subproblem.

Another point is that rather than carry out the simplex method at a detailed level on the Master Problem (M), we just accumulate all columns of (M) that we generate, and always solve linear-optimization problems, using functionality of `MATLAB`, with all of the columns generated thus far. In this way, we do not maintain bases ourselves, and we do not carry out the detailed pivots of the Simplex Algorithm. Note that the linear-optimization functionality of `MATLAB` does give us a dual solution, so we do not compute that ourselves.

```
% Decomp.m  // Jon Lee
%
% Apply Decomposition to
%   z = min c'x
%             Ex  + Is = h
%             Ax       = b
%               x , s >= 0
%
% NOTE: WE ONLY HANDLE PROBLEMS WITH BOUNDED SUBPROBLEM LPs.
%       IF WE ENCOUNTER AN UNBOUNDED SUBPROBLEM, THEN WE QUIT.

% generate a random example
n = 50 % number of x variables
m1 = 15 % number of 'complicating' equations = number of s variables
m2 = 10 % number of 'nice' equations

rng('default');
rng(1);  % set seed
E = rand(m1,n);  % random m1-by-n matrix
A = rand(m2,n);  % random m2-by-n matrix

w = rand(n,1);     % generate rhs's so problem is feasible
h = E*w + 0.01*rand(m1,1);
b = A*w;

c = rand(n,1);     % generate random objective

% first we will calculate the optimal value z of the original LP, just to
% compare later.
disp('Optimizing the original LP without decomposition');
%options = optimoptions(@linprog,'Algorithm','simplex');
%[x,z,exitflag,output,lambda]=linprog([c;zeros(m1,1)],[],[],...
%[horzcat(E,eye(m1)); horzcat(A,zeros(m2,m1))],[h; b],zeros(n+m1,1), ...
%[],[],options);
[x,z,exitflag,output,lambda]=linprog([c;zeros(m1,1)],[],[], ...
    [horzcat(E,eye(m1)); horzcat(A,zeros(m2,m1))],[h; b], ...
    zeros(n+m1,1),[],[]);
if (exitflag < 1)
     disp('fail 1: original LP without decomposition did not solve correctly');
     return;
end;
disp('Optimal value of the original LP:');
z
%disp('dual solution:');
%disp(- lambda.eqlin); % Need to flip the signs due to a Matlab convention

disp('Starting Decomposition Algorithm');

MAXIT = 50 % number of iterations

% Set up the initial master basis
% ... First, we need any extreme point of the subproblem
x1 = linprog(c',[],[],A,b,zeros(n,1),[]);
X = x1;      % start accumulating the extreme points
```

```
CX = c'*x1;  % and their costs
numx = 1;
%Next, we build out the initial master basis
B = [horzcat(eye(m1), E*x1) ; horzcat(zeros(1,m1), 1)];
numcol = m1+1;
% OK, now we have a basis matrix for the master.
% Let's see if this basis is feasible.
zeta = linsolve(B,[h ; 1])
% If it is not feasible, we need to adjoin an artificial column
if (min(zeta) < -0.00000001)
    disp('*** Phase-one needed');
    % adjoin an artificial column
    numcol = numcol + 1;
    artind = numcol;
    B(:,artind) = -B*ones(m1+1,1);
    xi = zeros(numcol,1);
    xi(artind) = 1;

    % OK, now we are going to solve a phase-one problem to drive the artificial
    % variable to zero. Of course, we need to do this with decomposition.
    k = 1; % iteration counter
    subval = -Inf;
    sigma = 0.0;
    while -sigma + subval < -0.00000001 && k <= MAXIT
      % Set the dual variables by solving the restricted master
      [zeta,masterval,exitflag,output,dualsol] = linprog(xi,[],[],B, ...
          [h ; 1],zeros(numcol,1),[]);
      masterval
      % Need to flip the signs due to a Matlab convention
        y = - dualsol.eqlin(1:m1);
        sigma = - dualsol.eqlin(m1+1);
      % OK, now we can set up and solve the phase-one subproblem
      [x,subval,exitflag] = linprog((-y'*E)',[],[],A,b,zeros(n,1),[]);
      if (exitflag < 1)
          disp('fail 2: phase-one subproblem did not solve correctly');
          return;
      end;
      % Append [E*x; 1] to B, and set up new restricted master
      numcol = numcol + 1;
      numx = numx + 1;
      X = [X x];
      CX = [CX ; c'*x];
      B = [B [E*x ; 1]];
      % Oh, we should build out the cost vector as well
      xi = [xi ; 0];
      k = k+1;
    end;

    % sanity check
    if (-sigma + subval < -0.00000001)
      disp('fail 3: we hit the max number of iterations for phase-one');
      return;
    end;

    % sanity check
    if (zeta(artind) > 0.00000001)
```

```
      disp('fail 4: we thought we finished phase-one');
      disp('   but it seems the aritifical variable is still positive');
      disp(zeta(artind));
      return;
    end;

% phase-one clean-up: peel off the last column --- which was not improving,
% and also delete the artifical column
B(:,numcol) =[];
numcol = numcol - 1;
X(:,numx) =[];
CX(numx,:) =[];
numx = numx - 1;
B(:,artind) = [];
numcol = numcol - 1;

end;

%
% Now time for phase-two
disp('*** Starting phase-two');

results1 = zeros(MAXIT,1);
results2 = zeros(MAXIT,1);

k = 0; % iteration counter
subval = -Inf;
sigma = 0.0;
while -sigma + subval < -0.00000001 && k <= MAXIT
  k = k + 1;
  % Set the dual variables by solving the restricted master
  obj = [zeros(m1,1) ; CX];
  [zeta,masterval,exitflag,output,dualsol] = linprog(obj,[],[],B, ...
      [h ; 1],zeros(numcol,1),[]);
  masterval
  % Need to flip the signs due to a Matlab convention
    y = - dualsol.eqlin(1:m1);
    sigma = - dualsol.eqlin(m1+1);
  results1(k)=k;
  results2(k)=masterval;
  % OK, now we can set up and solve the phase-two subproblem
  [x,subval,exitflag] = linprog((c'-y'*E)',[],[],A,b,zeros(n,1),[]);
  if (exitflag < 1)
      disp('fail 5: phase-two subproblem did not solve correctly');
      return;
  end;
  % Append [E*x; 1] to B, and set up new restricted master
  numcol = numcol + 1;
  numx = numx + 1;
  X = [X x];
  CX = [CX ; c'*x];
  B = [B [E*x ; 1]];
end;

% sanity check
if (-sigma + subval < -0.00000001)
```

```
  disp('fail 6: we hit the max number of iterations for phase-two');
  return;
end;

% Clean up from phase-two:
%
% Peel off the last column --- which was not improving
B(:,numcol) =[];
numcol = numcol -1;
X(:,numx) =[];
CX(numx,:) =[];
numx = numx - 1;

xdw = X*zeta(m1+1:numcol) % this is the solution in the original variables x

DWz = CX'*(zeta(m1+1:numcol))

z

if ((abs(DWz - z) < 0.01) && (abs(c'*xdw - DWz) < 0.00001) && ...
        (norm(b-A*xdw) < 0.00001) && (norm(min(h-E*xdw,0)) < 0.00001))
    disp('IT WORKED!!!'); % exactly what did we check?
end

clf; % clear figure
% plot the sequence of upper bounds
plot(results1(1:k),results2(1:k),'k--.','MarkerSize',15);
hold on;
% plot a horizontal line at height z
plot([1,k],[ z z ], 'r-','LineWidth',1.5)
axis tight;
xlabel('Iteration');
ylabel('Decomposition upper bound');
legend('Decomposition UB','z');

print -djpeg Decomp.jpeg;
```

In Figure 7.2, we see quite good behavior for ("phase-two" of) the Decomposition Algorithm.

**Convergence and lower bounds.** Practically speaking, the convergence behavior of the Decomposition Algorithm can suffer from a tailing-off effect. That is, while the sequence of objective values for successive iterates is non-increasing, at some point improvements can become quite small. It would be helpful to know when we already have a very good but possibly non-optimal solution. If we could rapidly get a good *lower* bound on $z$ , then we could stop the Decomposition when the its objective value is close to such a lower bound. Lower bounds on $z$ can be obtained from feasible solutions to the *dual* of (Q) . But there is another way, closely related to the dual of (Q), to rapidly get good lower bounds. We develop this in the next section.

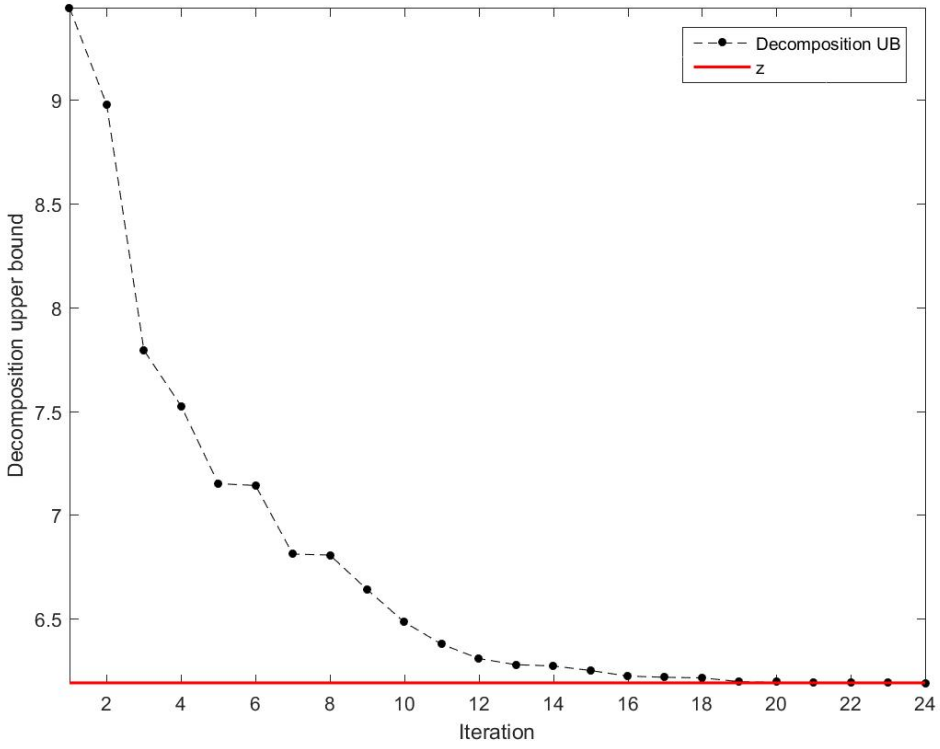



Figure 7.2: Example of upper bounds generated by Decomposition

## 7.2 Lagrangian Relaxation

Again, we consider

$$\begin{array}{rrcl} z := \min & c'x & & \\ & Ex & = & h\,; \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,, \end{array} \tag{Q}$$

but our focus now is on efficiently getting a good lower bound on $z$ , with again the view that we are able to quickly solve many linear-optimization problems having only the constraints: $Ax = b$ , $x \geq \mathbf{0}$ .

### 7.2.1 Lagrangian bounds

For any fixed choice of $\hat{y}$, consider the following "Lagrangian" optimization problem

$$\begin{array}{rrcl} v(\hat{y}) := \hat{y}'h + \min & (c' - \hat{y}'E)x & & \\ & Ax & = & b \\ & x & \geq & \mathbf{0}\,. \end{array} \tag{$L_{\hat{y}}$}$$

Note that the only variables in the minimization are $x$ — we consider $\hat{y}$ to be fixed.

**Theorem 7.3**
$v(\hat{y}) \leq z$ , for all $\hat{y}$ in the domain of $v$.

*Proof.* Let $x^*$ be an optimal solution for (Q). Clearly $x^*$ is feasible for ($L_{\hat{y}}$). Therefore

$$\begin{array}{rcl} v(\hat{y}) & \leq & \hat{y}'h + (c' - \hat{y}'E)x^* \\ & = & c'x^* + \hat{y}'(h - Ex^*) \\ & = & z\,. \end{array}$$

The last equation uses the fact that $x^*$ is optimal for (Q), so $z = c'x^*$ and $Ex^* = h$ . □

From what we learned in studying sensitivity analysis, it can be seen that $v$ is a concave (piecewise-linear) function on its domain (see Theorem 6.6). Because of this nice behavior, it is plausible that we could calculate the maximum of $v$ as a means of getting a good lower bound on $z$ . Before doing that, we examine the precise relationship between primal and dual solutions of (Q), minimizers of $v$ , and primal and dual solutions of the Lagrangian.

**Theorem 7.4**
Suppose that $x^*$ is optimal for (Q) , and suppose that $\hat{y}$ and $\hat{\pi}$ are optimal for the dual of (Q) . Then $x^*$ is optimal for $(L_{\hat{y}})$ , $\hat{\pi}$ is optimal for the dual of $(L_{\hat{y}})$ , $\hat{y}$ is a maximizer of $v$ , and the maximum value of $v$ is $z$ .

*Proof.* $x^*$ is clearly feasible for $(L_{\hat{y}})$ . Because $\hat{y}$ and $\hat{\pi}$ are feasible for the dual of (Q) , we have $\hat{\pi}'A + \hat{y}'E \leq c'$ , and so $\hat{\pi}$ is feasible for the dual of $(L_{\hat{y}})$ .

Using the Strong Duality Theorem for (Q) implies that $c'x^* = \hat{y}'h + \hat{\pi}'b$ . Using that $E\hat{x}^* = h$ (feasibility of $x^*$ in (Q)), we then have that $(c' - \hat{y}'E)\, x^* = \hat{\pi}'b$ . Finally, using the Weak Duality Theorem for $(L_{\hat{y}})$ , we have that $x^*$ is optimal for $(L_{\hat{y}})$ and $\hat{\pi}$ is optimal for the dual of $(L_{\hat{y}})$ .

Next,

$$\begin{aligned} z &\geq v(\hat{y}) \qquad \text{(by Theorem 7.3)} \\ &= \hat{y}'h + \left(c' - \hat{y}'E\right) x^* \qquad \text{(because } x^* \text{ is optimal for } (L_{\hat{y}})) \\ &= \hat{y}'h + c'x^* - \hat{y}'h \qquad \text{(because } Ex^* = h) \\ &= c'x^* \\ &= z\ . \end{aligned}$$

Therefore the first inequality is an equation, and so $\hat{y}$ is a maximizer of $v$ and the maximum value is $z$ . □

**Theorem 7.5**
Suppose that $\hat{y}$ is a maximizer of $v$ , and suppose that $\hat{\pi}$ is optimal for the dual of $(L_{\hat{y}})$ . Then $\hat{y}$ and $\hat{\pi}$ are optimal for the dual of (Q) , and the optimal value of (Q) is $v(\hat{y})$ .

*Proof.*

$$\begin{aligned} v(\hat{y}) &= \max_y \{v(y)\} \\ &= \max_y \left\{ y'h + \min_x \left\{ \left(c' - y'E\right) x \ :\ Ax = b\, , x \geq \mathbf{0} \right\} \right\} \\ &= \max_y \left\{ y'h + \max_\pi \left\{ \pi'b \ :\ \pi'A \leq c' - y'E \right\} \right\} \\ &= \max_{y,\pi} \left\{ y'h + \pi'b \ :\ y'E + \pi'A \leq c' \right\} \\ &= z\ . \end{aligned}$$

The third equation follows from taking the dual of the inner (minimization) problem. The last equation follows from seeing that the final maximization (over $y$ and $\pi$ simultaneously) is just the dual of (Q).

So, we have established that the optimal value $z$ of (Q) is $v(\hat{y})$ . Looking a bit more closely, we have established that $z = \hat{y}'h + \hat{\pi}'b$ , and because $\hat{\pi}'A \leq c' - \hat{y}'E$ , we have that $\hat{y}$ and $\hat{\pi}$ are optimal for the dual of (Q) . □

Note that the conclusion of Theorem 7.5 gives us an optimal $\hat{y}$ and $\hat{\pi}$ for the dual of (Q), but not and optimal $x^*$ for (Q) itself.

### 7.2.2 Solving the Lagrangian Dual

Theorem 7.3 gives us a simple way to calculate a lower bound on $z$, by solving a potentially much-easier linear-optimization problem. But the bound depends on the choice of $\hat{y} \in \mathbb{R}^m$. Can we find the best such $\hat{y}$? This would entail solving the so-called **Lagrangian Dual** problem of *maximizing* $v(y)$ over all $y \in \mathbb{R}^m$. It should seem that there is hope for doing this — because $v$ is a concave function. But $v$ is not a smooth function (it is piecewise linear), so we cannot rely on calculus-based techniques.

**Theorem 7.6**
Suppose that we fix $\hat{y}$, and solve for $v(\hat{y})$. Let $\hat{x}$ be the solution of $(L_{\hat{y}})$. Let $\hat{\gamma} := h - E\hat{x}$. Then

$$v(\tilde{y}) \leq v(\hat{y}) + (\tilde{y} - \hat{y})'\hat{\gamma} ,$$

for all $\tilde{y}$ in the domain of $v$.

*Proof.*

$$\begin{aligned} v(\hat{y}) + (\tilde{y} - \hat{y})'\hat{\gamma} &= \hat{y}'h + (c' - \hat{y}'E)\hat{x} + (\tilde{y} - \hat{y})'(h - E\hat{x}) \\ &= \tilde{y}'h + (c' - \tilde{y}'E)\hat{x} \\ &\geq v(\tilde{y}) . \end{aligned}$$

The last equation follows from the fact that $\hat{x}$ is feasible (but possible not optimal) for $(L_{\tilde{y}})$. □

**Subgradient.** What is $v(\hat{y})+(\tilde{y}-\hat{y})'\hat{\gamma}$? It is a linear estimation of $v(\tilde{y})$ starting from the actual value of $v$ at $\hat{y}$. The direction $\tilde{y} - \hat{y}$ is what we add to $\hat{y}$ to move to $\tilde{y}$. The choice of $\hat{\gamma} := h - E\hat{x}$ is made so that Theorem 7.6 holds. That is, $\hat{\gamma}$ is chosen in such a way that the linear estimation is always an upper bound on the value $v(\tilde{y})$ of the function, for all $\tilde{y}$ in the domain of $f$. The nice property of $\hat{\gamma}$ demonstrated with Theorem 7.6 has a name: we say that $\hat{\gamma} := h - E\hat{x}$ is a **subgradient** of (the concave function) $v$ at $\hat{y}$ (because it satisfies the inequality of Theorem 7.6).

**Subgradient Optimization.** Next, we describe a simple "Subgradient Optimization Algorithm" for solving the Lagrangian Dual. The general idea is to iteratively move in the direction of a subgradient.

**Subgradient Optimization Algorithm**

0. Start with any $\hat{y}^1 \in \mathbb{R}^m$. Let $k := 1$.
1. Solve $(L_{\hat{y}^k})$ to get $\hat{x}^k$.
2. Calculate the subgradient $\hat{\gamma}^k := h - E\hat{x}^k$.
3. Let $\hat{y}^{k+1} \leftarrow \hat{y}^k + \lambda_k \hat{\gamma}^k$.
4. Let $k \leftarrow k + 1$, and GOTO 1.

**Convergence.** We have neglected, thus far, to fully specify the Subgradient Optimization Algorithm. We can stop if, at some iteration $k$ , we have $\hat{\gamma}^k = \mathbf{0}$, because the algorithm will make no further progress if this happens, and indeed we will have found that $\hat{y}^k$ is a maximizer of $v$ . But this actually very unlikely to happen. In practice, we may stop if $k$ reaches some pre-specified iteration limit, or if after many iterations, $v$ is barely increasing.

We are interested in mathematically analyzing the convergence behavior of the algorithm, letting the algorithm iterate infinitely. We will see that the method converges (in a certain sense), if we take a sequence of $\lambda_k > 0$ such that $\sum_{k=1}^{\infty} \lambda_k^2 < +\infty$ and $\sum_{k=1}^{\infty} \lambda_k = +\infty$ . That is, "square summable, but not summable." For example, taking $\lambda_k := \alpha/(\beta + k)$ , with $\alpha > 0$ and $\beta \geq 0$ , we get a sequence of step sizes satisfying this property; in particular, for $\alpha = 1$ and $\beta = 0$ we have the harmonic series $\sum_{k=1}^{\infty} 1/k$ which satisfies $\ln(k+1) < \sum_{k=1}^{\infty} 1/k < \ln(k) + 1$ and $\sum_{k=1}^{\infty} 1/k^2 = \pi^2/6$ .

To prove convergence, we first must establish a key technical lemma.

**Lemma 7.7**
Let $y^*$ be any maximizer of $v$ . Suppose that $\lambda_k > 0$ , for all $k$ . Then

$$\|y^* - \hat{y}^{k+1}\|^2 \leq \|y^* - \hat{y}^1\|^2 - 2\sum_{i=1}^{k} \lambda_i \left(v(y^*) - v(\hat{y}^i)\right) + \sum_{i=1}^{k} \lambda_i^2 \|\hat{\gamma}^i\|^2 \, .$$

*Proof.* The proof is by induction on $k$ . The inequality is trivially true for the base case of $k = 0$ . So, consider general $k > 0$ .

$$\begin{aligned}
\|y^* - \hat{y}^{k+1}\|^2 &= \| \left(y^* - \hat{y}^k\right) - \left(\lambda_k \hat{\gamma}^k\right) \|^2 \\
&= \left(\left(y^* - \hat{y}^k\right) - \left(\lambda_k \hat{\gamma}^k\right)\right)' \left(\left(y^* - \hat{y}^k\right) - \left(\lambda_k \hat{\gamma}^k\right)\right) \\
&= \|y^* - \hat{y}^k\|^2 + \|\lambda_k \hat{\gamma}^k\|^2 - 2\lambda_k \left(y^* - \hat{y}^k\right)' \hat{\gamma}^k \\
&\leq \|y^* - \hat{y}^k\|^2 + \lambda_k^2 \|\hat{\gamma}^k\|^2 - 2\lambda_k \left(v(y^*) - v(\hat{y}^k)\right) \\
&\leq \left[\|y^* - \hat{y}^1\|^2 - 2\sum_{i=1}^{k-1} \lambda_i \left(v(y^*) - v(\hat{y}^i)\right) + \sum_{i=1}^{k-1} \lambda_i^2 \|\hat{\gamma}^i\|^2\right] \\
&\qquad + \lambda_k^2 \|\hat{\gamma}^k\|^2 \; - \; 2\lambda_k \left(v(y^*) - v(\hat{y}^k)\right) \, .
\end{aligned}$$

The penultimate inequality uses the assumption that $\lambda_k > 0$ and the subgradient inequality:

$$v(\tilde{y}) \; \leq \; v(\hat{y}^k) + (\tilde{y} - \hat{y}^k)' \hat{\gamma}^k \, ,$$

plugging in $y^*$ for $\tilde{y}$. The final inequality uses the inductive hypothesis. The final expression easily simplifies to the right-hand side of the inequality in the statement of the lemma. □

Now, let

$$v_k^* := \max_{i=1}^{k} \left\{ v(\hat{y}^i) \right\} \ , \text{ for } k = 1, 2, \ldots$$

That is, $v_k^*$ is the best value seen up through the $k$-th iteration.

**Theorem 7.8 ("Square summable, but not summable" convergence)**
Let $y^*$ be any maximizer of $v$ . Assume that we take a *basic* solution as the solution of each Lagrangian subproblem. Suppose that $\lambda_k > 0$ , for all $k$ . Suppose further that $\sum_{k=1}^{\infty} \lambda_k^2 < +\infty$ and $\sum_{k=1}^{\infty} \lambda_k = +\infty$ . Then $\lim_{k \to \infty} v_k^* = v(y^*)$ .

*Proof.* Because the left-hand side of the inequality in the statement of Lemma 7.7 is non-negative, we have

$$2 \sum_{i=1}^{k} \lambda_i \left( v(y^*) - v(\hat{y}^i) \right) \ \leq \ \|y^* - \hat{y}^1\|^2 + \sum_{i=1}^{k} \lambda_i^2 \|\hat{\gamma}^i\|^2 \ .$$

Because $v_k^* \geq v(\hat{y}^i)$ for all $i \leq k$ , we then have

$$2 \left( \sum_{i=1}^{k} \lambda_i \right) \left( v(y^*) - v_k^* \right) \ \leq \ \|y^* - \hat{y}^1\|^2 + \sum_{i=1}^{k} \lambda_i^2 \|\hat{\gamma}^i\|^2 \ ,$$

or

$$v(y^*) - v_k^* \ \leq \ \frac{\|y^* - \hat{y}^1\|^2 + \sum_{i=1}^{k} \lambda_i^2 \|\hat{\gamma}^i\|^2}{2 \sum_{i=1}^{k} \lambda_i} \ .$$

Next, we observe that $\|\hat{\gamma}^i\|^2$ is bounded by some constant $\Gamma$ , independent of $i$ , because our algorithm takes $\hat{\gamma} = h - E\hat{x}$ , where $\hat{x}$ is a *basic* solution of a Lagrangian subproblem. There are only a finite number of bases. Therefore, we can take

$$\Gamma = \max \left\{ \|h - E\hat{x}\|^2 \ : \ \hat{x} \text{ is a basic solution of } Ax = b \, , \, x \geq \mathbf{0} \right\} \ .$$

So, we have

$$v(y^*) - v_k^* \ \leq \ \frac{\|y^* - \hat{y}^1\|^2 + \Gamma \sum_{i=1}^{k} \lambda_i^2}{2 \sum_{i=1}^{k} \lambda_i} \ .$$

Now, we get our result by observing that $\|y^* - \hat{y}^1\|^2$ is a constant, $\sum_{i=1}^{k} \lambda_i^2$ is converging to a constant and $\sum_{i=1}^{k} \lambda_i$ goes to $+\infty$ (as $k$ increases without limit), and so the right-hand side of the final inequality converges to zero. The result follows. □

**A demonstration implementation.** It is very easy to write a small `MATLAB` code for Subgradient Optimization.

```
% SubgradientOpt.m // Jon Lee
%
% Apply Subgradient Optimization to
%   z = min c'x
%         Ex  = h
%         Ax  = b
%          x >= 0
%
% NOTE: WE ONLY HANDLE PROBLEMS WITH BOUNDED SUBPROBLEM LPs.
%       IF WE ENCOUNTER AN UNBOUNDED SUBPROBLEM, THEN WE QUIT.

% generate a random example
n = 50; % number of variables
m1 = 15; % number of equations to relax
m2 = 10; % number of equations to keep

rng('default');
rng(1);  % set seed
E = rand(m1,n);  % random m1-by-n matrix
A = rand(m2,n);  % random m2-by-n matrix

w = rand(n,1);     % generate rhs's so problem is feasible
h = E*w;
b = A*w;

c = rand(n,1);     % generate random objective
% first we will calculate the optimal value z of the original LP, just to
% compare later
disp('Optimizing the original LP without Lagrangian relaxation');
[x,z,exitflag] = linprog(c,[],[],[E; A],[h; b],zeros(n,1),[]);
if (exitflag < 1)
     disp('fail 1: original LP (without using LR) did not solve correctly');
     return;
end;

disp('Optimal value of the original LP:');
z

disp('Starting Subgradient Optimization');

MAXIT = 100; % number of iterations

results1 = zeros(MAXIT,1);
results2 = zeros(MAXIT,1);

k = 0; % iteration counter
y = zeros(m1,1); % initialize y as you like
g = zeros(m1,1); % initialize g arbitrarily --- it will essentially be ignored
stepsize = 0;    % just to have us really start at the initialized y
bestlb = -Inf;
while k < MAXIT
   k = k + 1; % increment the iteration counter
   y = y + stepsize*g; % take a step in the direction of the subgradient
   % solve the Lagrangian
```

```
   [x,subval,exitflag,output,dualsol] = linprog((c'-y'*E)',[],[],A,b,zeros(n,1),[]);
   v = y'*h + (c'-y'*E)*x;  % calculate the value of the Lagrangian
   disp(v);
   bestlb = max(bestlb,v);
   results1(k)=k;
   results2(k)=v;
   g = h - E*x; % calculate the subgradient
   stepsize = 1/k; % calculate the step size
end

bestlb

z

clf;  % clear figure
% plot the sequence of MAXIT lower bounds
plot(results1,results2,'k--.','MarkerSize',15);
hold on;
% plot a horizontal line at height z
plot([1,MAXIT],[ z z ], 'r-','LineWidth',2.5)
axis tight;
xlabel('Iteration');
ylabel('Lagrangian lower bound');
legend('Lagrangian LB','z','Location','SouthEast');

print -djpeg SubgradientOpt.jpeg;

% Next, if y really solves the Lagrangian dual, and pi is the
% optimal solution to the Lagrangian subproblem corresponding to y,
% then y and pi should be an optimal dual solution to the given problem.
%
% Let's see if y and pi are nearly dual feasible.

pi = - dualsol.eqlin;
Total_Dual_Infeasibility = norm(min(c' - y'*E - pi'*A,zeros(1,n)))
```

In Figure 7.3, we see quite good behavior for Subgradient Optimization.

**Practical steps.** Practically speaking, in order to get a $\hat{y}$ with a reasonably high value of $v(\hat{y})$ , it can be better to choose a sequence of $\lambda_k$ that depends on a "good guess" of the optimal value of $v(\hat{y})$, taking bigger steps when one is far away, and smaller steps when one is close. Then, the method is usually stopped after a predetermined number of iterations or after progress becomes very slow.

**Dual estimation.** From Theorem 7.5, we see that the Subgradient Optimization Method is a way to try and quickly find *estimates* of an optimal solution to the dual of (Q). But note that we give something up — we do not get an $x^*$ that solves (Q) from a $\hat{y}$ that maximizes $v$ and a $\hat{\pi}$ that is optimal for the dual of $(L_{\hat{y}})$ . There is no guarantee that a $\hat{x}$ that is optimal for $(L_{\hat{y}})$ will be feasible for (Q) . Moreover, we need for the Subgradient Optimization Method to have nearly converged to certify that $\hat{y}$ and $\hat{\pi}$ are nearly optimal for the dual of (Q).

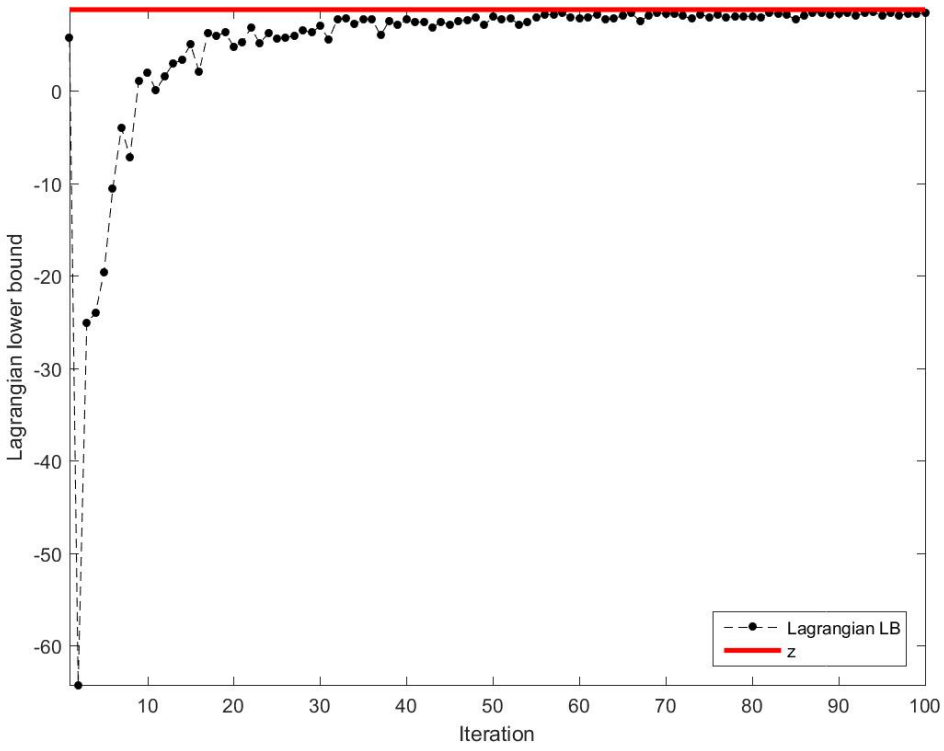



Figure 7.3: Example of lower bounds generated by Subgradient Optimization

## 7.3 The Cutting-Stock Problem

The cutting-stock problem is a nice concrete topic at this point. We will develop a technique for it, using column generation, but the context is different than for decomposition. Moreover, the topic is a nice segue into integer linear optimization — the topic of the next chapter.

The story is as follows. We have stock rolls of some type of paper of (integer) width $W$. But we encounter (integer) demand $d_i$ for rolls of (integer) width $w_i < W$ , for

$i = 1, 2, \ldots, m$ . The **cutting-stock problem** is to find a plan for satisfying demand, using as few stock rolls as possible.[8]

### 7.3.1 Formulation via cutting patterns

There are several different ways to formulate the cutting-stock problem mathematically. A particularly useful way is based on a consideration of the problem from the point of view of the worker who has to adjust the cutting machine. What she dearly hopes for is that a plan can be formulated that does not require that the machine be adjusted for (different cutting patterns) too many times. That is, she hopes that there are a relatively small number of ways that will be utilized for cutting a stock roll, and that these good ways can each be repeated many times.

With this idea in mind, we define a **cutting pattern** to be a solution of

$$\begin{array}{rcl} \sum_{i=1}^{m} w_i a_i & \leq & W \, ; \\ a_i & \geq & 0 \text{ integer}, i = 1, \ldots, m \, , \end{array}$$

where $a_i$ is the number of pieces of width $w_i$ that the pattern yields.

Conceptually, we could form a matrix $A$ with $m$ rows, and an enormous number of columns, where each column is a distinct pattern. Then, letting $x_j$ be the number of times that we use pattern $A_j$ , we can conceptually formulate the cutting-stock problem as

$$\begin{array}{rrcl} z := \min & \sum_j x_j & & \\ & \sum_j A_j x_j & \geq & d \, ; \\ & x_j & \geq & 0 \text{ integer}, \forall \, j \, . \end{array} \qquad \text{(CSP)}$$

### 7.3.2 Solution via continuous relaxation

Our approach to getting a good solution to (CSP) is to solve its continuous relaxation and then round. Toward this end, we subtract surplus variables and consider the linear-optimization problem

$$\begin{array}{rrcl} \underline{z} := \min & \sum_j x_j & & \\ & \sum_j A_j x_j & - t \; = & d \, ; \\ & x_j & \geq 0 \, , \; \forall \, j \, ; & \\ & t & \geq \mathbf{0} \, . & \end{array} \qquad \text{(CSP)}$$

We endeavor to compute a basic optimum $(\bar{x}, \bar{t})$ . Because of the nature of the formulation, we can see that $\lceil \bar{x} \rceil$ is feasible for (CSP). Moreover, we have produced a solution using $\mathbf{1}' \lceil \bar{x} \rceil$ stock rolls, and we can give an *a priori* bound on its quality. Specifically, as we will see in the next theorem, the solution that we obtain wastes at most $m-1$ stock rolls, in comparison with an optimal solution. Moreover, we have a practically-computable bound on the number of wasted rolls, which is no worse than the worst-case bound of $m - 1$. That is, our waste is at worst $\mathbf{1}' \lceil \bar{x} \rceil - \lceil \underline{z} \rceil$ .

**Theorem 7.9**

$$\lceil \underline{z} \rceil \; \leq \; z \; \leq \; \mathbf{1}' \lceil \bar{x} \rceil \; \leq \; \lceil \underline{z} \rceil + (m-1) \, .$$

*Proof.* Because (CSP) is a relaxation of (CSP) and because $z$ is an integer, we have $\lceil \underline{z} \rceil \leq z$. Because $\lceil \bar{x} \rceil$ is a feasible solution of (CSP), we have $z \leq \mathbf{1}'\lceil \bar{x} \rceil$. Now, $\mathbf{1}'\lceil \bar{x} \rceil = \sum_{i=1}^{m}(\bar{x}_{\beta_i} + f_i)$, with each $f_i < 1$. But $\sum_{i=1}^{m}(\bar{x}_{\beta_i} + f_i) = \mathbf{1}'\bar{x} + \sum_{i=1}^{m} f_i \leq \lceil \mathbf{1}'\bar{x} \rceil + \sum_{i=1}^{m} f_i$. Therefore, $\mathbf{1}'\lceil \bar{x} \rceil \leq \lceil \mathbf{1}'\bar{x} \rceil + \sum_{i=1}^{m} f_i$. Now the left-hand side of this last inequality is an integer, so we may round down the right-hand side. So we can conclude that $\mathbf{1}'\lceil \bar{x} \rceil \leq \lceil \underline{z} \rceil + (m-1)$. □

### 7.3.3 The knapsack subproblem

Toward describing how we can solve (CSP) by the Simplex Algorithm, we introduce a vector $y \in \mathbb{R}^m$ of dual variables.

$$\begin{array}{rrll} \underline{z} := \min & \sum_j x_j & & \text{dual variables} \\ & \sum_j A_j x_j & -t = d\,; & y \\ & x_j & \geq 0\,,\ \forall\, j\,; & \\ & t & \geq \mathbf{0}\,. & \end{array} \qquad \text{(CSP)}$$

We suppose that we have a feasible basis of (CSP) and that we have, at hand, the associated dual solution $\bar{y}$. For each $i$, $1 \leq i \leq m$, the reduced cost of $t_i$ is simply $\bar{y}_i$. Therefore, if $\bar{y}_i < 0$, then $t_i$ is eligible to enter the basis.

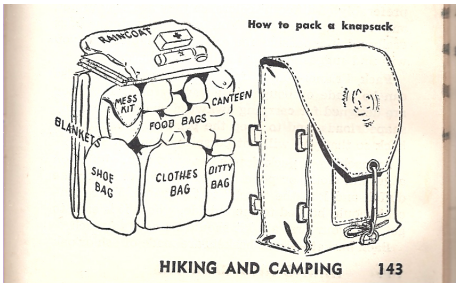
So, moving forward, we may assume that $\bar{y}_i \geq 0$ for all $i$. We now want to examine the reduced cost of an $x_j$ variable. The reduced cost is simply

$$1 - \bar{y}' A_j = 1 - \sum_{i=1}^{m} \bar{y}_i a_{ij}\,.$$

The variable $x_j$ is eligible to enter the basis then if $1 - \sum_{i=1}^{m} \bar{y}_i a_{ij} < 0$. Therefore, to check whether there is some column $x_j$ with negative reduced cost, we can solve the so-called **knapsack problem**

$$\begin{array}{rrll} \max & \sum_{i=1}^{m} \bar{y}_i a_i & & \\ & \sum_{i=1}^{m} w_i a_i & \leq & W\,; \\ & a_i & \geq & 0 \text{ integer}, i = 1, \ldots, m\,, \end{array}$$

and check whether the optimal value is greater than one. If it is, then the new variable that we associate with this solution pattern (i.e., column of the constraint matrix) is eligible to enter the basis.



Our algorithmic approach for the knapsack problem is via **recursive optimization** (known popularly as *dynamic programming*[9]). We will solve this problem for all positive integers up through $W$. That is, we will solve

$$\begin{array}{rrcl} f(s) := \max & \sum_{i=1}^m \bar{y}_i a_i & & \\ & \sum_{i=1}^m w_i a_i & \leq & s\ ; \\ & a_i & \geq & 0 \text{ integer}, i = 1, \ldots, m\ , \end{array}$$

starting with $f(s) = 0$, for $0 \leq s < \min_{i=1}^m \{w_i\}$, and proceeding from $s = \min_{i=1}^m \{w_i\} - 1$ by incrementing the argument of $f$ by 1 at each step. Then, we have the recursion

$$f(s) = \max_{i\,:\,w_i \leq s} \{\bar{y}_i + f(s - w_i)\}\ ,\ \text{for } s \geq \min_{i=1}^m \{w_i\}\ .$$

It is important to note that we can always calculate $f(s)$ provided that we have already calculated $f(s')$ for all $s' < s$. Why does this work? It follows from a very simple observation: If we have optimally filled a knapsack of capacity $s$ and we remove *any* item $i$, then what remains optimally fills a knapsack of capacity $s - w_i$. If there were a better way to fill the knapsack of capacity $s - w_i$, then we could take such a way, replace the item $i$, and we would have found a better way to fill a knapsack of capacity $s$. Of course, we do not know even a single item that we can be sure is in an optimally filled knapsack of capacity $s$, and this is why in the recursion, we maximize over all items that can fit in (i.e., $i\ :\ w_i \leq s$).

The recursion appears to calculate the value of $f(s)$, but it is not immediate how to recover optimal values of the $a_i$. Actually, this is rather easy.

**Recover the Solution of a Knapsack Problem**

0. Let $s := W$, and let $a_i := 0$, for $i = 1, \ldots, m$.
1. While $(s > 0)$
   (a) Find $\hat{\imath}\ :\ f(s) = \bar{y}_{\hat{\imath}} + f(s - w_{\hat{\imath}})$.
   (b) Let $a_{\hat{\imath}} := a_{\hat{\imath}} + 1$.
   (c) Let $s := s - w_{\hat{\imath}}$.
2. Return $a_i$, for $i = 1, \ldots, m$.

Note that in Step 1.a, there must be such an $\hat{\imath}$, by virtue of the recursive formula for calculating $f(s)$. In fact, if we like, we can save an appropriate $\hat{\imath}$ associated with each $s$ at the time that we calculate $f(s)$.

### 7.3.4 Applying the Simplex Algorithm

**An initial feasible basis.** It is easy to get an initial feasible basis. We just consider the $m$ patterns $A_i := \lfloor W/w_i \rfloor \boldsymbol{e_i}$, for $i = 1, 2, \ldots, m$. The values of the $m$ basic variables associated with the basis of these patterns are $\bar{x}_i = d_i / \lfloor W/w_i \rfloor$, which are clearly non-negative.

**Basic solutions: dual and primal.** At any iteration, the basis matrix $B$ has some columns corresponding to patterns and possibly other columns for $t_i$ variables. The column corresponding to $t_i$ is $-\boldsymbol{e_i}$ .

Organizing the basic variables $x_j$ and $t_i$ into a vector $\zeta$ , with their order appropriately matched with the columns of $B$ , the vector $\bar{\zeta}$ of values of $\zeta$ is precisely the solution of

$$B\zeta = d\,.$$

That is,

$$\bar{\zeta} = B^{-1}d\,.$$

The cost of an $x_j$ is 1, while the cost of a $t_i$ is 0. Organizing the costs of the basic variables into a vector $\xi$ , with their order appropriately matched with the columns of $B$ , the associated dual solution $\bar{y}$ is precisely the solution of

$$y'B = \xi'\,.$$

That is,

$$\bar{y}' = \xi' B^{-1}\,.$$

### 7.3.5 A demonstration implementation.

We can use AMPL, in a somewhat sophisticated manner, to implement our algorithm for the cutting-stock problem. As we did for the Decomposition Algorithm, rather than carry out the simplex method at a detailed level on (CSP), we just accumulate all columns of (CSP) that we generate, and always solve linear-optimization problems, using functionality of AMPL, with all of the columns generated thus far. In this way, we do not maintain bases ourselves, and we do not carry out the detailed pivots of the Simplex Algorithm. Note that the linear-optimization functionality of AMPL does give us a dual solution, so we do not compute that ourselves.

First, we specify our csp.mod :

---

```
# csp.mod // Jon Lee
#

param W > 0;                    # width of stock rolls
param m >0;                     # number of widths
set WIDTHS:={1..m};             # set of widths to be cut
param d{WIDTHS} > 0;            # number of each width to be cut
param w {WIDTHS} > 0;           # actual widths

param nPAT integer >= 0;        # number of patterns
set PATTERNS := 1..nPAT;        # set of patterns

param A {WIDTHS,PATTERNS} integer >= 0;

var X {PATTERNS} integer >= 0;
                                # rolls cut using each pattern
minimize RollsUsed:             # minimize rolls used
```

```
   sum {j in PATTERNS} X[j];

subj to FillDemand {i in WIDTHS}:
   sum {j in PATTERNS} A[i,j] * X[j]>= d[i];

# ---------------------------------------------
# KNAPSACK SUBPROBLEM FOR CUTTING STOCK
# ---------------------------------------------

param ybar {WIDTHS} default 0.0;

var a {WIDTHS} integer >= 0;

minimize Reduced_Cost:
   1 - sum {i in WIDTHS} ybar[i] * a[i];

subj to Width_Limit:
   sum {i in WIDTHS} w[i] * a[i] <= W;
```

For a sample data set, we can have `csp.dat` :

```
110
5
70 205
40 2321
55 143
25 1089
35 117
```

Finally, we can have `csp.run` :

```
# csp.run // Jon Lee
#

reset;

option solver cplex;

option solution_round 6;
option solver_msg 0;

model csp.mod;
#data csp.dat;
read W,m,{i in 1..m} (w[i],d[i]) < csp.dat;

problem Cutting_Opt: RollsUsed, FillDemand, X;
   option relax_integrality 1;
   option presolve 0;

problem Pattern_Gen: Reduced_Cost, Width_Limit, a;
   option relax_integrality 0;
```

```
   option presolve 1;

let nPAT := 0;

for {i in WIDTHS} {
   let nPAT := nPAT + 1;
   let A[i,nPAT] := floor (W/w[i]);
   let {i2 in WIDTHS: i2 <> i} A[i2,nPAT] := 0;
   };

repeat {
   solve Cutting_Opt > crap;
   let {i in WIDTHS} ybar[i] := FillDemand[i].dual;
   solve Pattern_Gen > crap;
   if Reduced_Cost < -0.00001 then {
      printf "\nImproving column generated. Reduced cost = %11.2e  ", Reduced_Cost;
      let nPAT := nPAT + 1;
      let {i in WIDTHS} A[i,nPAT] := a[i];
      }
   else break;
};

print " "; print "LP solved! Solution: "; print " ";
display X;
display RollsUsed;
printf "\nLower bound = %11i  (LP obj value rounded up) \n\n", ceil(RollsUsed);


let {j in PATTERNS} X[j] := ceil(X[j]); # round up

printf "\n\n\nSolution rounded up to integers: %3i rolls\n\n", sum {j in PATTERNS} X[j];

for {j in PATTERNS : X[j]>0} {
printf "CUT %4i OF:\n", X[j];
for {i in WIDTHS} {
   printf "%3i  ", i;
   printf "%4i", A[i,j];
   printf "\n";
   }
printf "\n\n";
};

# Try to use integer-linear optimization to get a better solution

option Cutting_Opt.relax_integrality 0;
option Cutting_Opt.presolve 10;
solve Cutting_Opt;

printf "\n\nBest integer solution considering all patterns generated:\n";
display RollsUsed; print " ";

for {j in PATTERNS : X[j]>0} {
printf "CUT %4i OF:\n", X[j];
for {i in WIDTHS} {
   printf "%3i  ", i;
   printf "%4i", A[i,j];
```

```
   printf "\n";
   }
printf "\n\n";
};
```

On the data set provided, we get the following output:

```
Improving column generated. Reduced cost =   -5.00e-01
Improving column generated. Reduced cost =   -2.50e-01
Improving column generated. Reduced cost =   -8.33e-02
Improving column generated. Reduced cost =   -5.56e-02
Improving column generated. Reduced cost =    5.55e-17
LP solved! Solution:

X [*] :=
1     0
2     0
3    71.5
4     0
5     0
6   205
7  1033.38
8    18.5385
9    49.2308
;

RollsUsed = 1377.65


Lower bound =        1378  (LP obj value rounded up)



Solution rounded up to integers: 1380 rolls

CUT   72 OF:
  1     0
  2     0
  3     2
  4     0
  5     0


CUT  205 OF:
  1     1
  2     1
  3     0
  4     0
  5     0
```

```
CUT 1034 OF:
  1     0
  2     2
  3     0
  4     1
  5     0


CUT   19 OF:
  1     0
  2     0
  3     0
  4     3
  5     1


CUT   50 OF:
  1     0
  2     1
  3     0
  4     0
  5     2


CPLEX 12.2.0.0:

Best integer solution considering all patterns generated:
RollsUsed = 1379


CUT   72 OF:
  1     0
  2     0
  3     2
  4     0
  5     0


CUT  205 OF:
  1     1
  2     1
  3     0
  4     0
  5     0


CUT 1034 OF:
  1     0
  2     2
  3     0
  4     1
```

```
  5     0


CUT   19 OF:
  1     0
  2     0
  3     0
  4     3
  5     1


CUT   49 OF:
  1     0
  2     1
  3     0
  4     0
  5     2
```

---

Our algorithm gives a lower bound of 1378 on the minimum number of stock rolls needed to cover demand, and it gives us an upper bound (feasible solution) of 1380. By solving a further integer-linear optimization problem to determine the best way to cover demand using all patterns generated in the course of our algorithm, we improve the upper bound to 1379. It remains unknown as to whether the optimal solution to this instance is 1378 or 1379.

## 7.4 Exercises

**Exercise 7.1 (Illustrate decomposition)**
Devise an original example of the decomposition method (see Section 7.1).

**Exercise 7.2 (Dual solutions)**
Refer to (Q) and (M) defined in the Decomposition Theorem (i.e., Corollary 7.2) What is the relationship between optimal *dual* solutions of (Q) and (M) ?

**Exercise 7.3 (Lagrangian value function)**
Using Theorem 6.6, prove that $v$ (from Section 7.2.1) is a concave piecewise-linear function on its domain.

**Exercise 7.4 (Play with subgradient optimization)**
Play with the MATLAB code for Subgradient Optimization. Try bigger examples. Try different ideas for the step size — be a real engineer and think 'out of the box' (you can use any information you like: e.g., the current subgradient $\hat{\gamma}^k$, the current function value $v(\hat{\gamma}^k)$, an estimate of the maximum value of $v$, etc.).

Also, Theorem 7.5, together with the fact that Subgradient Optimization converges in the limit to a a solution of the Lagrangian Dual together, tells us that the $\hat{y}$ and $\hat{\pi}$ produced should be optimal for the dual of (Q). Check this out: see to what extent $\hat{y}$ and $\hat{\pi}$ are optimal for the dual of (Q) (i.e., feasible and objective value near $z$).

**Exercise 7.5 (Cutting it closer to reality)**
Real cutting machines may have a limited number, say $K$ , of blades. This means that we can cut at most $K+1$ pieces for patterns that leave no scrap (i.e., $\sum_{i=1}^{m} w_i a_i = W \Rightarrow \sum_{i=1}^{m} a_i \leq K+1$) and at most $K$ pieces for patterns that leave scrap (i.e., $\sum_{i=1}^{m} w_i a_i < W \Rightarrow \sum_{i=1}^{m} a_i \leq K$). Describe how to modify our algorithm for the cutting-stock problem to account for this. Modify the `AMPL` code that I provided to try this out.

**Exercise 7.6 (Another kind of question)**
Print is dying, right? Why should we care about the cutting-stock problem?

# Chapter 8

# Integer-Linear Optimization

Our goals in this chapter are as follows:

- to develop some facility with modeling using integer variables;
- to learn how to recognize when we can expect solutions of linear-optimization problems to be integer automatically;
- to learn the framework that most solvers use to handle integer variables;
- to learn something about *solver-aware modeling* in the context of integer variables.

## 8.1 Integrality for Free

### 8.1.1 Some structured models

**Network-flow problem.** A finite **network** $G$ is described by a finite set of **nodes** $\mathcal{N}$ and a finite set $\mathcal{A}$ of **arcs**. Each arc $e$ has two key attributes, namely its **tail** $t(e) \in \mathcal{N}$ and its **head** $h(e) \in \mathcal{N}$ . We think of a (single) commodity as being allowed to "flow" along each arc, from its tail to its head. Indeed, we have "flow" variables

$$x_e := \text{ amount of flow on arc } e \text{ ,}$$

for $e \in \mathcal{A}$ . Formally, a **flow** $\hat{x}$ on $G$ is simply an assignment of *any* real numbers $\hat{x}_e$ to the variables $x_e$ , for $e \in \mathcal{A}$ . We assume that flow on arc $e$ should be non-negative and should not exceed

$$u_e := \text{ the flow upper bound on arc } e \text{ ,}$$

for $e \in \mathcal{A}$ . Associated with each arc $e$ is a cost

$$c_e := \text{ cost per-unit-flow on arc } e \ ,$$

for $e \in \mathcal{A}$ . The (total) **cost** of the flow $\hat{x}$ is defined to be

$$\sum_{e \in \mathcal{A}} c_e \hat{x}_e \ .$$

We assume that we have further data for the nodes. Namely,

$$b_v := \text{ the net supply at node } v \ ,$$

for $v \in \mathcal{N}$ . A flow is **conservative** if the net flow out of node $v$ , minus the net flow into node $v$ , is equal to the net supply at node $v$ , for all nodes $v \in \mathcal{N}$ .

The **(single-commodity min-cost) network-flow problem** is to find a minimum-cost conservative flow that is non-negative and respects the flow upper bounds on the arcs. We can formulate this as follows:

$$\begin{array}{lll} \min & \displaystyle\sum_{e \in \mathcal{A}} c_e x_e & \\ & \displaystyle\sum_{\substack{e \in \mathcal{A} : \\ t(e) = v}} x_e \ - \ \sum_{\substack{e \in \mathcal{A} : \\ h(e) = v}} x_e \ = \ b_v \ , & \forall \, v \in \mathcal{N} \ ; \\ & 0 \leq x_e \leq u_e \ , & \forall \, e \in \mathcal{A} \ . \end{array}$$

As we have stated this, it is just a structured linear-optimization problem. But there are many situations where the net supplies at the nodes and the flow capacities on the arcs are integer, and we wish to further constrain the flow variables to be integers.

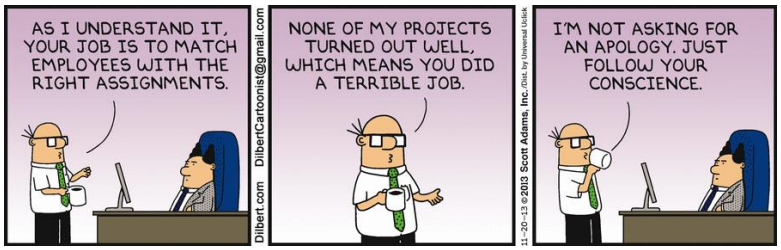
It is useful to think of the network-flow problem in matrix-vector language. We define the **network matrix** of $G$ to be a matrix $A$ having rows indexed from $\mathcal{N}$ , columns indexed from $\mathcal{A}$ , and entries

$$a_{ve} := \begin{cases} 1 \ , & \text{if } t(e) = v \ ; \\ -1 \ , & \text{if } h(e) = v; \\ 0 \ , & \text{otherwise,} \end{cases}$$

for $v \in \mathcal{N}$ , $e \in \mathcal{A}$ . With this notation, and organizing the $b_v$ in a column-vector indexed accordingly with the rows of $A$ , and organizing the $c_e$ , $x_e$ and $u_e$ as three column-vectors indexed accordingly with the columns of $A$ , we can rewrite the network-flow formulation as

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b \ ; \\ & x & \leq & u \ ; \\ & x & \geq & \mathbf{0} \ . \end{array}$$

**Assignment problem on a graph.**




A finite **bipartite graph** $G$ is described by two finite sets of **vertices** $V_1$ and $V_2$ , and a set $E$ of ordered pairs of **edges**, each one of which is of the form $(i, j)$ with $i \in V_1$ and $j \in V_2$ . A **perfect matching** $M$ of $G$ is a subset of $E$ such that at exactly one edge in $M$ meets each vertex of the graph. We assume that there are **edge weights**

$$c_{ij} := \text{ for } (i,j) \in E \ ,$$

and our goal is to find a perfect matching that has minimum (total) weight.

We can define

$$x_{ij} := \text{ indicator variable for choosing edge } (i,j) \text{ to be in } M \ ,$$

for $(i, j) \in E$ . Then we can model the problem of finding a perfect matching of $G$ having minimum weight via the formulation:

$$\begin{array}{lrcl} \min & \displaystyle\sum_{(i,j)\in E} c_{ij}x_{ij} & & \\ & \displaystyle\sum_{\substack{j\in V_2 \ : \\ (i,j)\in E}} x_{ij} & = & 1 \ , \quad \forall\, i \in V_1 \ ; \\ & \displaystyle\sum_{\substack{i\in V_1 \ : \\ (i,j)\in E}} x_{ij} & = & 1 \ , \quad \forall\, j \in V_2 \ ; \\ & x_{ij} \in \{0,1\} \ , & & \forall\, (i,j) \in E \ . \end{array}$$

It is useful to think of the assignment-problem formulation in matrix-vector language. We define the **vertex-edge incidence matrix of the bipartite graph** $G$ to be a matrix $A$ having rows indexed from $V_1 \cup V_2$ , columns indexed from $E$ , and entries

$$a_{v,(i,j)} := \begin{cases} 1 \ , & \text{if } v = i \text{ or } v = j \ ; \\ 0 \ , & \text{otherwise,} \end{cases}$$

for $v \in V_1 \cup V_2$ , $(i, j) \in E$ . With this notation, and organizing the $c_{ij}$ , $x_{ij}$ and as column-vectors indexed accordingly with the columns of $A$ , we can rewrite the assignment-problem formulation as

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & \mathbf{1} \ ; \\ & x & \in & \{0,1\}^E \ , \end{array}$$

**Staffing problem.** In this problem, we have discrete time periods numbered $1, 2, \ldots, m$, and we have

$$b_i := \text{ the minimum number of workers required at time period } i \,,$$

for each $i = 1, 2, \ldots, m$. Additionally, there is an allowable set of "shifts." An allowable shift is simply a collection of time periods that a worker is allowed to staff. It may well be that not all shifts are allowable. We suppose that the allowable shifts are numbered $1, 2, \ldots, n$, and we have

$$c_j := \text{ the per worker cost to staff shift } j \,,$$

for each $j = 1, 2, \ldots, n$. It is convenient to encode the shifts as a $0, 1$-valued matrix $A$, where

$$a_{ij} := \begin{cases} 1 \,, & \text{if shift } j \text{ contains time period } i \,; \\ 0 \,, & \text{otherwise,} \end{cases}$$

for $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$, Letting $x$ be an $n$-vector of variables, we can formulate the staffing problem as

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & \geq & b \,; \\ & x & \geq & \mathbf{0} \text{ integer.} \end{array}$$


As we have stated it, this so-called staffing problem is really a very general type of integer-linear-optimization problem because we have not restricted the form of $A$ beyond it being $0, 1$-valued. In some situations, however, it may may be reasonable to assume that shifts must consist of a consecutive set of time periods. In this case, the 1's in each column of $A$ occur consecutively, so we call $A$ a **consecutive-ones matrix**.

### 8.1.2 Unimodular basis matrices and total unimodularity


In this section we explore the essential properties of a constraint matrix so that basic solutions are guaranteed to be integer.

**Definition 8.1**
Let $A$ be an $m \times n$ real matrix. A basis matrix $A_\beta$ is **unimodular** if $\det(A_\beta) = \pm 1$ .

Checking whether a large unstructured matrix has all of its basis matrices unimodular is not a simple matter. Nonetheless, we will see that this property is very useful for guaranteeing *integer* optimal of linear-optimization problems, and certain *structured* constraint matrices have this property.

**Theorem 8.2**
If $A$ is integer, all basis matrices of $A$ are unimodular, and $b$ is integer, then every basic solution $\bar{x}$ of

$$\begin{aligned} Ax &= b\,; \\ x &\geq \mathbf{0} \end{aligned}$$

is integer.

*Proof.* By Cramer's rule, the basic variables take on the values

$$\bar{x}_{\beta_i} = \frac{\det(A_\beta(i))}{\det(A_\beta)}\,, \text{ for } i = 1, 2, \ldots, m\,,$$

where $A_\beta(i)$ is defined to be the matrix $A_\beta$ with its $i$-th column, $A_{\beta_i}$ , replaced by $b$ . Because we assume that $A$ and $b$ are integer, the numerator above is the determinant of an integer matrix, which is an integer. Next, the fact that $A$ has unimodular basis matrices tells us that the determinant of the invertible matrix $A_\beta$ is $\pm 1$ . That is, the denominator above is $\pm 1$ . So, we have an integer divided by $\pm 1$ , which results in an integer value for $\bar{x}_{\beta_i}$ . □

We note that Theorem 8.2 asserts that all basic solutions are integer, whether or not they are feasible. There is a converse to this theorem.

**Theorem 8.3**
Let $A$ be an integer matrix in $\mathbb{R}^{m\times n}$ . If the system

$$\begin{aligned} Ax &= b\,; \\ x &\geq \mathbf{0} \end{aligned}$$

has integer basic feasible solutions for every integer vector $b \in \mathbb{R}^m$ , then all basis matrices of $A$ are unimodular.

It is important to note that the hypothesis of Theorem 8.3 is weaker than the conclusion of Theorem 8.2. For Theorem 8.3, we only require integrality for basic *feasible* solutions.

*Proof. (Theorem 8.3).* Let $\beta$ be an arbitrary basis, choose an arbitrary $i$ ($1 \leq i \leq m$), and consider the associated basic solution when $b := \boldsymbol{e_i} + \Delta A_\beta \mathbf{1}$ . The basic solution

$\bar{x}$ has $\bar{x}_\beta$ equal to the $i$-th column of $A_\beta^{-1}$ plus $\Delta\mathbf{1}$ . Note that if we choose $\Delta$ to be an integer, then $b$ is integer. Furthermore, if we choose $\Delta$ to be sufficiently large, then $\bar{x}_\beta$ is non-negative. Therefore, we can chose $\Delta$ so that $b$ is integer and $\bar{x}$ is a basic *feasible* solution. Therefore, by our hypothesis, $\bar{x}$ is integer. So the $i$-th column of $A_\beta^{-1}$ plus $\Delta\mathbf{1}$ is an integer vector . But this clearly implies that the $i$-th column of $A_\beta^{-1}$ is an integer vector . Now, because $i$ was arbitrary, we conclude that $A_\beta^{-1}$ is an integer matrix . Of course $A_\beta$ is an integer matrix as well. Now, it is a trivial observation that an integer matrix has an integer determinant. Furthermore, the determinants of $A_\beta$ and $A_\beta^{-1}$ are reciprocals. Of course the only integers with integer reciprocal are $1$ and $-1$ . Therefore, the determinant of $A_\beta$ is $1$ or $-1$ . □

Before turning to specific structured linear-optimization problems, we introduce a stronger property than unimodularity of basis matrices. The main reason for introducing it is that for the structured linear-optimization problems that we will look at, the constraint matrices satisfy this stronger property, and the *inductive* proofs that we would deploy for proving the weaker property naturally prove the stronger property as well.

**Definition 8.4**
Let $A$ be an $m \times n$ real matrix. $A$ is **totally unimodular (TU)** if every square non-singular submatrix $B$ of $A$ has $\det(B) = \pm 1$ .

Obviously every entry of a TU matrix must be $0$ , $+1$ or $-1$ , because the determinant of a $1 \times 1$ matrix is just its single entry. It is quite easy to make an example of even a $2 \times 2$ non-TU matrix with all entries $0$ , $\pm 1$:

$$\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}.$$

It is trivial to see that if $A$ is TU, then every basis matrix of $A$ is unimodular. But note that even for integer $A$ , every basis matrix of $A$ could be unimodular, but $A$ need not be TU. For example,

$$\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$$

has only itself as a basis matrix, and its determinant is 1, but there is a $1 \times 1$ submatrix with determinant 2, so $A$ is not TU. Still, as the next result indicates, there is a way to get the TU property from unimodularity of basis matrices.

**Theorem 8.5**
If every basis matrix of $[A, \mathbf{I}_m]$ is unimodular, then $A$ is TU.

*Proof.* Let $B$ be an $r \times r$ invertible submatrix of $A$ , with $r < m$ . It is an easy matter to choose a $(m \times m)$ basis matrix $H$ of $[A, \mathbf{I}_m]$ that includes all $r$ columns of $A$ that include columns of $B$ , and then the $m - r$ identity columns that have their ones in rows other

than those used by $B$ . If we permute the rows of $A$ so that $B$ is within the first $r$ rows, then we can put the identity columns to the right, in their natural order, and the basis we construct is

$$H = \left(\begin{array}{c|c} B & \mathbf{0} \\ \hline \times & \mathbf{I}_{m-r} \end{array}\right).$$

Clearly $B$ and $H$ have the same determinant. Therefore, the fact that every basis matrix has determinant 1 or $-1$ implies that $B$ does as well. □

Next, we point out some simple transformations that preserve the TU property.

**Theorem 8.6**
If $A$ is TU, then all of the following leave $A$ TU.

(i) multiplying any rows or columns of $A$ by $-1$ ;

(ii) duplicating any rows or columns of $A$ ;

(iii) appending standard-unit columns (that is, all entries equal to $0$ except a single entry of $1$) ;

(iv) taking the transpose of $A$ .

We leave the simple proof to the reader.

**Remark 8.7**
**Relationship with transformations of linear-optimization problems.** The significance of Theorem 8.6 can be understood via the following observations.

- $(i)$ allows for reversing the sense of an inequality (i.e., switching between "$\leq$" and "$\geq$") or variable (i.e., switching between non-negative and non-positive) in a linear-optimization problem with constraint matrix $A$ .

- $(ii)$ together with $(i)$ allows for replacing an equation with a pair of oppositely senses inequalities and for replacing a sign-unrestricted variable with the difference of a pair of non-negative variables.

- $(iii)$ allows for adding a non-negative slack variable for a "$\leq$" inequality, to transform it into an equation. Combining $(iii)$ with $(i)$ , we can similarly subtract a non-negative surplus variable for a "$\geq$" inequality, to transform it into an equation.

- $(iv)$ allows for taking the dual of a linear-optimization problem with constraint matrix $A$ .

### 8.1.3 Consequences of total unimodularity

**Network flow.**

**Theorem 8.8**
If $A$ is a network matrix, then $A$ is TU.

*Proof.* A network matrix is simply a $0$ , $\pm 1$-valued matrix with exactly one $+1$ and one $-1$ in each column.

Let $B$ be an $r \times r$ invertible submatrix of the network matrix $A$ . We will demonstrate that $\det(B) = \pm 1$ , by induction on $r$ . For the base case, $r = 1$ , the invertible submatrices have a single entry which is $\pm 1$ , which of course has determinant $\pm 1$ . Now suppose that $r > 1$ , and we inductively assume that all $(r-1) \times (r-1)$ invertible submatrices of $A$ have determinant $\pm 1$ .

Because we assume that $B$ is invertible, it cannot have a column that is a zero-vector.

Moreover, it cannot be that every column of $B$ has exactly one $+1$ and one $-1$ . Because, by simply adding up all the rows of $B$ , we have a non-trivial linear combination of the rows of $B$ which yields the zero vector. Therefore, $B$ is not invertible in this case.

So, we only need to consider the situation in which $B$ has a column with a single non-zero $\pm 1$ . By expanding the determinant along such a column, we see that, up to a sign, the determinant of $B$ is the same as the determinant of an $(r-1)\times(r-1)$ invertible submatrix of $A$ . By the inductive hypothesis, this is $\pm 1$ . □

**Corollary 8.9**
The (single-commodity min-cost) network-flow formulation

$$\begin{aligned} \min \quad & \sum_{e \in \mathcal{A}} c_e x_e \\ & \sum_{\substack{e \in \mathcal{A} : \\ t(e) = v}} x_e \; - \sum_{\substack{e \in \mathcal{A} : \\ h(e) = v}} x_e \;\; = \;\; b_v \,, \quad \forall\, v \in \mathcal{N} \,; \\ & 0 \le x_e \le u_e \,, \qquad\qquad\qquad\qquad \forall\, e \in \mathcal{A} \,. \end{aligned}$$

has an integer optimal solution if it has an optimal solution, each $b_v$ is an integer, and each $u_e$ is an integer or is infinite.

*Proof.* Recall that we can rewrite the network-flow formulation as

$$\begin{aligned} \min \quad & c'x \\ & Ax \;=\; b \,; \\ & x \;\le\; u \,; \\ & x \;\ge\; \mathbf{0} \,, \end{aligned}$$

where $A$ is a network matrix. For the purpose of proving the theorem, we may as well assume that the linear-optimization problem has an optimal solution. Next, we transform the formulation into standard form:

$$\begin{array}{lrcccl} \min & c'x & & & & \\ & Ax & & & = & b\,; \\ & x & + & s & = & u\,; \\ & x & , & s & \geq & \mathbf{0}\,. \end{array}$$

The constraint matrix has the form $\left(\begin{array}{c|c} A & \mathbf{0} \\ \hline I & I \end{array}\right)$. This matrix is TU, by virtue of the fact that A is TU, and that it arises from A using operations that preserve the TU property. Finally, we delete any redundant equations from this system of equations, and we delete any rows that have infinite right-hand side $u_e$ . The resulting constraint matrix is TU, and the right-hand side is integer, so an optimal basic solution exists and will be integer. □

**Assignments.**

**Theorem 8.10**
If $A$ is the vertex-edge incidence matrix of a bipartite graph, then $A$ is TU.

*Proof.* The constraint matrix $A$ for the formulation has its rows indexed by the vertices of $G$ . With each edge having exactly one vertex in $V_1$ and exactly one vertex in $V_2$ , the constraint matrix has the property that for each column, the only non-zeros are a single $1$ in a row indexed from $V_1$ and a single $1$ in a row indexed from $V_2$ .

Certainly multiplying any rows (or columns) of a matrix does not bear upon whether or not it is TU. It is easy to see that by multiplying the rows of $A$ indexed from $V_1$ , we obtain a network matrix, thus by Theorem 8.8, the result follows. □

**Corollary 8.11**
The continuous relaxation of the following formulation for finding a minimum-weight perfect matching of the bipartite graph $G$ has an $0, 1$-valued solution whenever it is feasible.

$$\begin{array}{lrcll} \min & \displaystyle\sum_{(i,j)\in E} c_{ij}x_{ij} & & & \\ & \displaystyle\sum_{\substack{j\in V_2\,:\\(i,j)\in E}} x_{ij} & = & 1\,, & \forall\, i \in V_1\,; \\ & \displaystyle\sum_{\substack{i\in V_1\,:\\(i,j)\in E}} x_{ij} & = & 1\,, & \forall\, j \in V_2\,; \\ & x_{ij} \geq 0\,, & & & \forall\,(i,j) \in E\,. \end{array}$$

*Proof.* After deleting any redundant equations, the resulting formulation as a TU constraint matrix and integer right-hand side. Therefore, its basic solutions are all integer. The constraints imply that no variable can be greater than 1, therefore the optimal value is not unbounded, and the only integer solutions have all $x_{ij} \in \{0, 1\}$ . The result follows. □

A **matching** $M$ of $G$ is a subset of $E$ such that *no more that one edge* in $M$ meets each vertex of the graph. An interesting variation on the problem of finding a perfect matching of $G$ having minimum weight, is to find a maximum-cardinality matching of $G$ . This problem is always feasible, because $M := \emptyset$ is always a matching.

How big can a matching of a finite graph $G$ be? A **vertex cover** of $G$ is a set $W$ of vertices that touches all of the edges of $G$ . Notice that if $M$ is a matching and $W$ is a vertex cover, then $|M| \leq |W|$ , because each element of $W$ touches at most one element of $M$ . Can we always find a matching $M$ and a vertex cover $W$ so that $|M| = |W|$ ? The next result of the mathematician König[10] tells us that the answer is 'yes' when $G$ is bipartite.

**Corollary 8.12 (König's Theorem)**
If $G$ is a finite bipartite graph, then the maximum cardinality of a matching of $G$ is equal to the minimum cardinality of a vertex cover of $G$ .

*Proof.* We can formulate the problem of finding the maximum cardinality of a matching of $G$ as follows:

$$\begin{aligned}
\max \quad & \sum_{(i,j)\in E} x_{ij} \\
& \sum_{\substack{j\in V_2 \,:\\ (i,j)\in E}} x_{ij} \;\leq\; 1\,, \quad \forall\, i \in V_1\,; \\
& \sum_{\substack{i\in V_1 \,:\\ (i,j)\in E}} x_{ij} \;\leq\; 1\,, \quad \forall\, j \in V_2\,; \\
& \qquad x_{ij} \;\geq\; 0 \quad \text{integer},\ \forall\, (i,j) \in E\,.
\end{aligned}$$

It is easy to see that we can relax integrality, and the optimal value will be unchanged, because $A$ is TU, and the constraint matrix will remain TU after introducing slack variables. The dual of the linear-optimization problem is

$$\begin{aligned}
\min \quad & \sum_{v\in V} y_v \\
& y_i + y_j \;\geq\; 1\,, \quad \forall\, (i,j) \in E\,; \\
& \qquad y_v \;\geq\; 0 \quad \text{integer},\ \forall\, v \in V\,.
\end{aligned}$$

It is easy to see that after putting this into standard form via the subtraction of surplus variables, the constraint matrix has the form $[A', -I]$ , where $A$ is the vertex-edge incidence matrix of $G$ . This matrix is TU, therefore an optimal integer solution exists.

Next, we observe that because of the minimization objective and the form of the constraints, an *optimal* integer solution will be $0,1$-valued; just observe that if $\bar{y}$ is an integer feasible solution and $\bar{y}_v > 1$ , for some $v$ , then decreasing $\bar{y}_v$ by 1 (holding the other components of $\bar{y}$ constant, produces another integer feasible solution with a lesser objective value. This implies that every integer feasible solution $\bar{y}$ with any $\bar{y}_v > 1$ is not optimal.

Next, let $\bar{y}$ be an optimal $0,1$-valued solution. We observe that

$$W := \{v \in V \ : \ \bar{y}_v = 1\} \ ,$$

and that $|W| = \sum_{v \in V} y_v$ . The result now follows from the strong duality theorem. □

**Staffing.**

**Theorem 8.13**
If $A$ is a consecutive-ones matrix, then $A$ is TU.

*Proof.* Let $B$ be an $r \times r$ invertible submatrix of a consecutive-ones matrix $A$ . We will demonstrate that $\det(B) = \pm 1$ , by induction on $r$ . We take care that we preserve the ordering of the rows of $A$ in $B$ . In this way, $B$ is also a consecutive-ones matrix. Note that only the sign of the determinant of $B$ depends on the ordering of its rows (and columns).

For the base case, $r = 1$ , the invertible submatrix $B$ has a single entry which is $1$ , which of course has determinant $1$ . Now suppose that $r > 1$ , and we inductively assume that all $(r-1) \times (r-1)$ invertible submatrices of *all* consecutive-ones matrices have determinant $\pm 1$ .

Next, we will reorder the columns of $B$ so that all columns with a $1$ in the first row come before all columns with a $0$ in the first row. Note that there must be a column with a $1$ in the first row, otherwise $B$ would not be invertible. Next, we further reorder the columns, so that among all columns with a $1$ in the first row, a column of that type with the fewest number of 1s is first.

Our matrix $B$ now has this form

$$\left(\begin{array}{c|c|c} 1 & 1\cdots 1 & 0\cdots 0 \\ \hline 1 & 1\cdots 1 & \\ \vdots & \vdots\ \ \vdots & \\ 1 & 1\cdots 1 & \\ \hline 0 & & \\ \vdots & F & G \\ 0 & & \end{array}\right),$$

where $F$ and $G$ are the submatrices indicated. Note that $F$ and $G$ are each consecutive-ones matrices.


Next, we subtract the top row from all other rows that have a $1$ in the first column. Such row operations do not change the determinant of $B$ , and we get a matrix of the form

$$\left(\begin{array}{c|c|c} 1 & 1\cdots 1 & 0\cdots 0 \\ \hline 0 & 0\cdots 0 & \\ \vdots & \vdots\ \ \vdots & \\ 0 & 0\cdots 0 & \\ \hline 0 & & \\ \vdots & F & G \\ 0 & & \end{array}\right).$$

Note that this resulting matrix need *not* be a consecutive-ones matrix — but that is not needed. By expanding the determinant of this latter matrix along the first column, we see that the determinant of this matrix is the same as that of the matrix obtained by striking out its first row and column,

$$\left(\begin{array}{c|c} 0\cdots 0 & \\ \vdots\ \ \vdots & \\ 0\cdots 0 & G \\ \hline F & \end{array}\right).$$

But this matrix is an $(r-1)\times(r-1)$ invertible consecutive-ones matrix (note that it is not necessarily a submatrix of $A$). So, by the inductive hypothesis, its determinant is $\pm 1$ .

□

**Corollary 8.14**
Let $A$ be a shift matrix such that each shift is a contiguous set of time periods, let $c$ be a vector of non-negative costs, and let $b$ be a vector of non-negative integer demands for workers in the time periods. Then there is an optimal solution $\bar{x}$ of the continuous relaxation

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & \geq & b\,; \\ & x & \geq & \mathbf{0} \end{array}$$

of the staffing formulation that has $\bar{x}$ integer, whenever the relaxation is feasible.

*Proof.* $A$ is a consecutive-ones matrix when each shift is a contiguous set of time periods. Therefore $A$ is TU. After subtracting surplus variables to put the problem into standard form, the constraint matrix takes the form $[A, -I]$, which is also $TU$. The result follows. □

## 8.2 Modeling Techniques

### 8.2.1 Disjunctions

**Example 8.15**
Suppose that we have a single variable $x \in \mathbb{R}$ , and we want to model the disjunction

$$-12 \leq x \leq 2 \text{ or } 5 \leq x \leq 20 .$$

By introducing a binary variable $y \in \{0, 1\}$ , we can model the disjunction as

$$\begin{aligned} x &\leq 2 + M_1 y , \\ x + M_2(1 - y) &\geq 5 , \end{aligned}$$

where the constant scalars $M_1$ and $M_2$ (so-called **big M**'s) are chosen to be appropriately large. A little analysis tell us how large. Considering our assumption that $x$ could be as large as $20$ , we see that $M_1$ should be at least $18$ . Considering our assumption that $x$ could be as small as $-12$ , we see that $M_2$ should be at least $17$ . In fact, we should choose these constants to be as small as possible so as make the feasible region with $y \in \{0, 1\}$ relaxed to $0 \leq y \leq 1$ as small as possible. So, the best model for us is:

$$\begin{aligned} x &\leq 2 + 18y , \\ x + 17(1 - y) &\geq 5 . \end{aligned}$$

It is interesting to see a two-dimensional graph of this in $x - y$ space; see Figures 8.1 and 8.2.

### 8.2.2 Forcing constraints

The **uncapacitated facility-location problem** involves $n$ customers, numbered $1, 2, \ldots, n$ and $m$ facilities, numbered $1, 2 \ldots, m$ . Associated with each facility, we have

$$f_i := \text{ fixed cost for operating facility } i ,$$

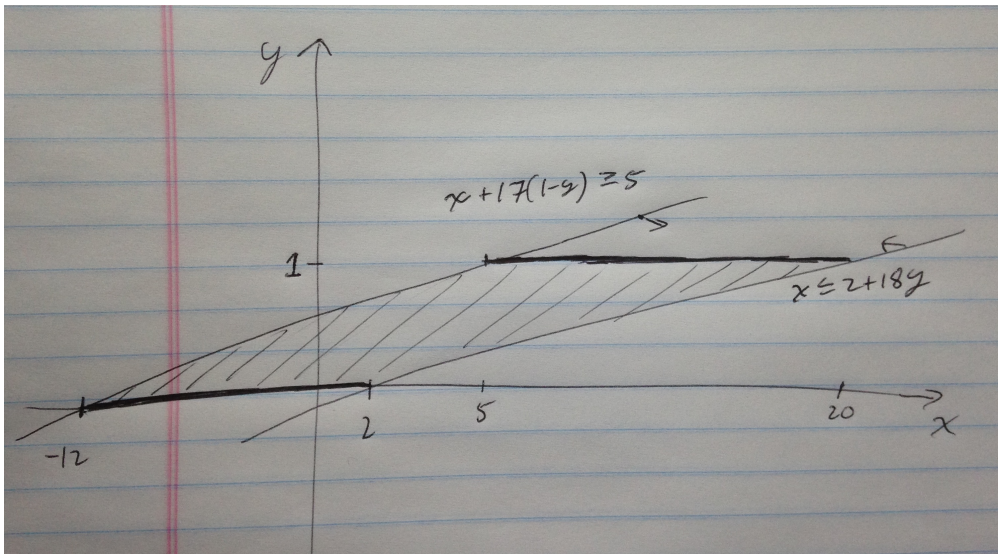



Figure 8.1: Optimal choice of "big M's"

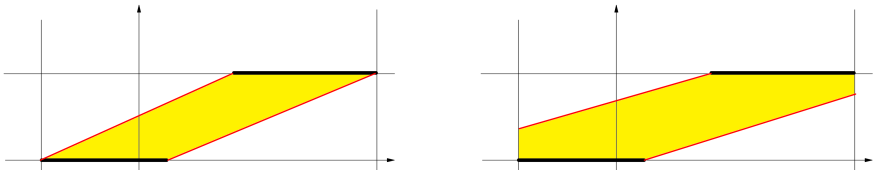


Figure 8.2: Comparing optimal vs non-optimal "big M's"

for $i = 1, \ldots, m$ . Associated with each customer/facility pair, we have

$$c_{ij} := \text{ cost for satisfying all of customer } j\text{'s demand from facility } i \text{ ,}$$

for $i = 1, \ldots, m$ , $j = 1, \ldots, n$ . The goal is to determine a set of facilities to operate and an allocation of each customers demand across operation facilities, so as to minimize the total cost. The problem is "uncapacitated" in the sense that each facility has no limit on its ability to satisfy demand from even all customers.

We formulate this optimization problem with

$$y_i := \text{ indicator variable for operating facility } i \text{ ,}$$

for $i = 1, \ldots, m$ , and

$$x_{ij} := \text{ fraction of customer } j \text{ demand satisfied by facility } i \text{ ,}$$

for $i = 1, \ldots, m$ , $j = 1, \ldots, n$ .

Our formulation is as follows:

$$\begin{array}{lrcll} \min & \sum_{i=1}^m f_i y_i \;+\; \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} & & & \\ & \sum_{i=1}^m x_{ij} & = & 1, & \text{for } j = 1, \ldots, n\,; \\ -y_i & + x_{ij} & \leq & 0, & \begin{array}{l}\text{for } i = 1, \ldots, m\,, \\ \text{for } j = 1, \ldots, n\,;\end{array} \\ y_i \in \{0, 1\}, & & & & \text{for } i = 1, \ldots, m\,; \\ & x_{ij} \geq 0\,, & & & \begin{array}{l}\text{for } i = 1, \ldots, m\,, \\ \text{for } j = 1, \ldots, n\,.\end{array} \end{array}$$

All of these constraints are self-explanatory except for the $mn$ constraints:

$$-y_i + x_{ij} \leq 0 \text{ for } i = 1, \ldots, m\,, \text{ for } j = 1, \ldots, n\,. \tag{S}$$

These constraints simply enforce that for any feasible solution $\hat{x}, \hat{y}$, we have that $\hat{y}_i = 1$ whenever $\hat{x}_{ij} > 0$. It is an interesting point that this could also be enforced via the $m$ constraints:

$$-ny_i + \sum_{j=1}^n x_{ij} \leq 0\,, \text{ for } i = 1, \ldots, m\,. \tag{W}$$

One can view the coefficient $-n$ of $y_i$ as a "big M", rendering the constrain vacuous when $y_i = 1$.

Despite the apparent parsimony of the latter formulation, it turns out that the original formulation is preferred.

### 8.2.3 Piecewise-linear univariate functions

Of course many useful functions are non-linear. Integer-linear optimization affords a good way to approximate well-behaved univariate non-linear functions. Suppose that $f : \mathbb{R} \to \mathbb{R}$ has domain the interval $[l, u]$, with $l < u$. For $n \geq 2$, we choose say $n$ **breakpoints** $l = \xi^1 < \xi^2 < \cdots < \xi^{n-1} < \xi^n = u$. Then, we approximate $f$ linearly between *adjacent* pairs of extreme points. That is, we approximate $f$ by

$$\hat{f}(x) := \sum_{j=1}^n \lambda_j f(\xi^j)\,,$$

where we require that

$$\begin{array}{rcll} \sum_{j=1}^n \lambda_j & = & 1\,; & \\ \lambda_j & \geq & 0\,, \text{ for } j = 1, \ldots, n\,, & \end{array}$$

and **the adjacency condition**:

$\lambda_j$ and $\lambda_{j+1}$ may be positive for only one value of $j$.

This adjacency condition means that we "activate" the interval $[\xi^j, \xi^{j+1}]$ for approximating $f(x)$ . That is, we will approximate $f(x)$ by

$$\lambda_j f(\xi^j) + \lambda_{j+1} f(\xi^{j+1}) \,,$$

with

$$\begin{aligned} \lambda_j + \lambda_{j+1} &= 1 \,; \\ \lambda_j \,, \lambda_{j+1} &\geq 0 \,. \end{aligned}$$

We can enforce the adjacency condition using $0, 1$-variables. Let

$$y_j := \begin{cases} 1, & \text{if the interval } [\xi^j, \xi^{j+1}] \text{ is activated;} \\ 0, & \text{otherwise,} \end{cases}$$

for $j = 1, 2, \ldots, n-1$ .

The situation is depicted in Figure 8.3, where the red curve graphs the non-linear function $f$.

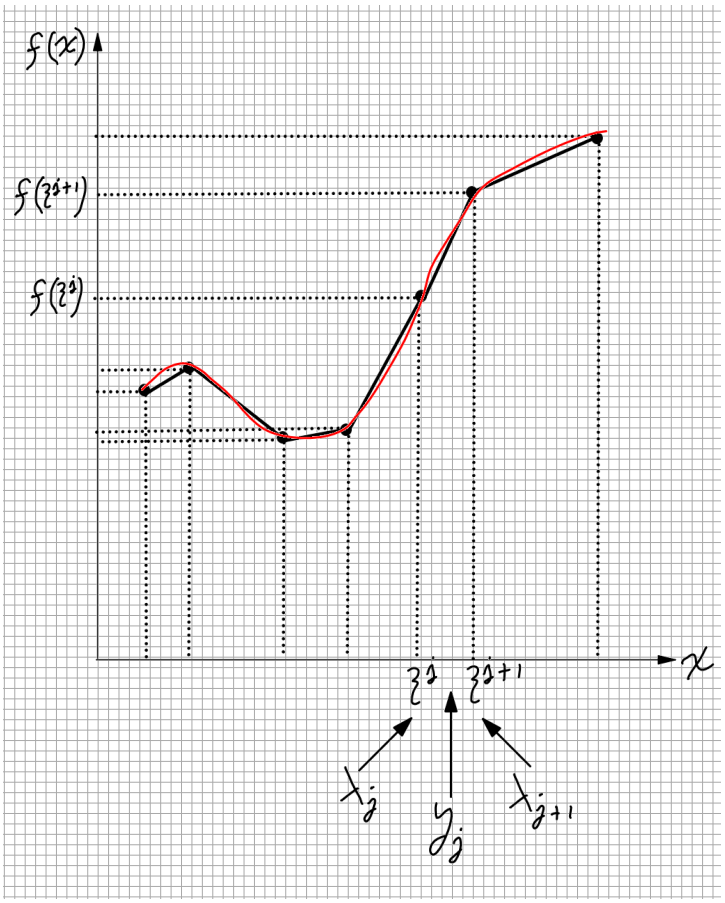



Figure 8.3: Piecewise-linear approximation

We only want to allow one of the $n-1$ intervals to be activated, so we use the constraints

$$\sum_{j=1}^{n-1} y_j \;=\; 1 \;.$$

We only want to allow $\lambda_1 > 0$ if the first interval $[\xi^1, \xi^2]$ is activated. We only want to allow $\lambda_n > 0$ if the last interval $[\xi^{n-1}, \xi^n]$ is activated. For an internal breakpoint $\xi^j$ , $1 < j < n$ , we only want to allow $\lambda_j > 0$ if either $[\xi^{j-1}, \xi^j]$ *or* $[\xi^j, \xi^{j+1}]$ is activated. We can accomplish these forcings with the constraints

$$\begin{aligned}
\lambda_1 &\leq y_1 \;; \\
\lambda_j &\leq y_{j-1} + y_j \;, \text{ for } j = 2, \ldots, n-1 \;; \\
\lambda_n &\leq y_{n-1} \;.
\end{aligned}$$

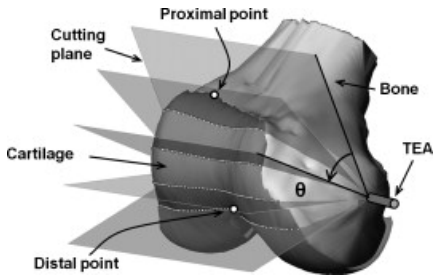
Notice how if $y_k$ is 1 , for some $k$ ($1 \leq k \leq n$), and necessarily all of the other $y_j$ are 0 , then only $\lambda_k$ and $\lambda_{k+1}$ can be positive.

How do we actually use this? If we have a model involving such a non-linear $f(x)$, then wherever we have $f(x)$ in the model, we simply substitute $\sum_{j=1}^{n} \lambda_j f(\xi^j)$ , and we incorporate the further constraints:

$$\begin{aligned}
\sum_{j=1}^{n} \lambda_j &= 1 \;; \\
\sum_{j=1}^{n-1} y_j &= 1 \;; \\
\lambda_1 &\leq y_1 \;; \\
\lambda_j &\leq y_{j-1} + y_j \;, \text{ for } j = 2, \ldots, n-1 \;; \\
\lambda_n &\leq y_{n-1} \;; \\
\lambda_j &\geq 0 \;, \text{ for } j = 1, \ldots, n \;; \\
y_j &\in 0 \;, \text{ for } j = 1, \ldots, n-1 \;.
\end{aligned}$$

Of course a very non-linear $f(x)$ will demand an $\hat{f}(x) := \sum_{j=1}^{n} \lambda_j f(\xi^j)$ with a high value for $n$ , so as to get an accurate approximation. And higher values for $n$ imply more binary variables $y_j$ , which come at a high computational cost.

## 8.3 Cutting Planes



### 8.3.1 Pure-integer problems

Before we get started, it is useful to look at a very simple two-variable problem, to set the stage for what the difficulty can be for handling integer variables. In Section 7.3, we saw a problem class (i.e., cutting-stock problems) for which it was sufficient to solve a continuous relaxation of an integer-linear optimization problem and then round the variables. It is instructive to see a simple example where naïve rounding does not suffice.

**Example 8.16 (Rounding can be useless)**
Choose $p_1 < p_2$ to be a pair of large, relatively-prime, positive integers. Consider the integer standard-form problem

$$\begin{array}{lrcrcl} \min & x_1 & & & & \\ & p_2 x_1 & - & p_1 x_2 & = & p_1 - p_2 \, ; \\ & x_1 & , & x_2 & \geq & 0 \, , \text{ integer.} \end{array}$$

By the equation in the formulation, every feasible solution $(x_1, x_2)$ satisfies $(x_1+1)/(x_2+1) = p_1/p_2$ . But, because $p_1$ and $p_2$ are relatively-prime, there cannot be a feasible solution with $x_1$ smaller than $p_1 - 1$. So the optimal solution is $(x_1, x_2) = (p_1 - 1, p_2 - 1)$ . But it is very easy to see that the optimal solution to the continuous relaxation is $(x_1, x_2) = (0, -1 + p_2/p_1)$ . In such a situation, both variables differ substantially from their optimal values, and naïve rounding is completely useless. In Figure 8.4, we can see the situation for $p_1 = 11$ , $p_2 = 15$ . The suspicious reader can check that the 'near misses' are legitimate (e.g., $(x_1, x_2) = (7, 10)$ has $(x_1 + 1)/(x_2 + 1) = 8/11$ which differs from $p_1/p_2 = 11/15$ by $1/165$).

**Chvátal-Gomory cuts.** Here, it is natural to consider a system of $m$ linear $\leq$-inequalities in $n$ non-negative variables,

$$\begin{array}{rcl} Ax & \leq & b \, ; \\ x & \geq & \mathbf{0} \, , \text{ integer.} \end{array}$$

We can choose any non-negative $u \in \mathbb{R}^m$ , and we see that

$$u \geq \mathbf{0} \text{ and } Ax \leq b \implies (u'A)x \leq u'b \, .$$

Next, we observe that

$$x \geq \mathbf{0} \implies \lfloor u'A \rfloor x \leq u'b \, .$$

Note that this inequality is valid for all solutions of

$$\begin{array}{rcl} Ax & \leq & b \, ; \\ x & \geq & \mathbf{0} \, , \end{array}$$

*integer or not*. Finally, we use the integrality of $x$ . We see that

$$x \text{ integer} \implies \lfloor u'A \rfloor x \leq \lfloor u'b \rfloor \, .$$

This last inequality is called a **Chvátal-Gomory cut**.

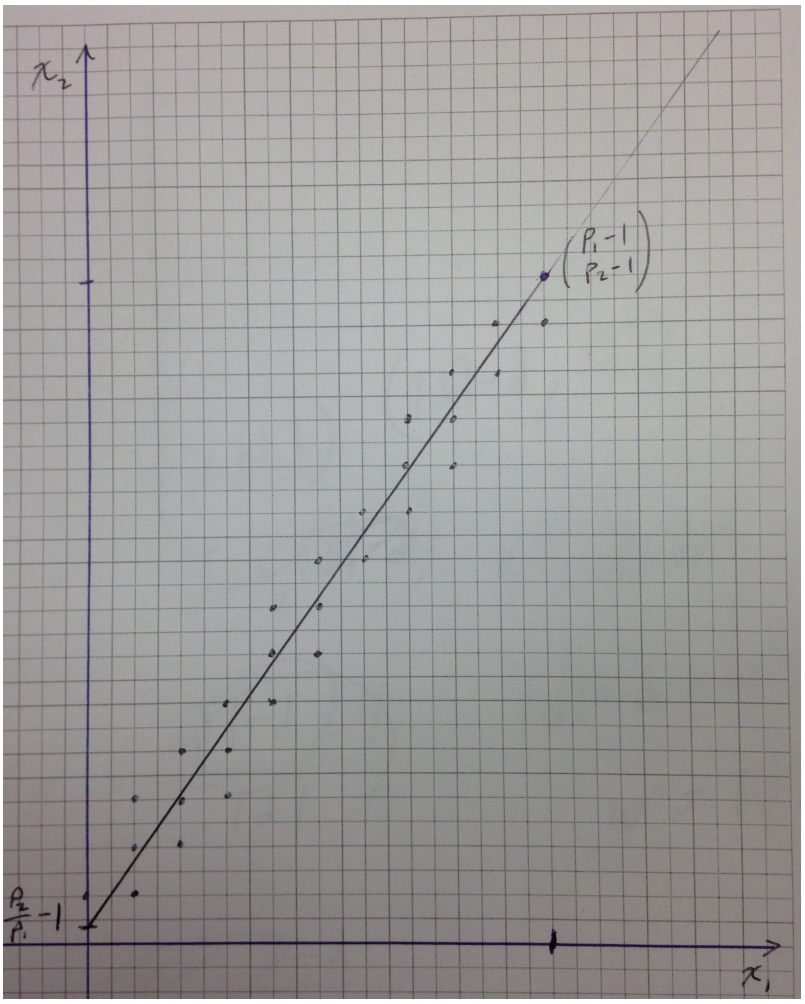



Figure 8.4: Bad example: $p_1 = 11$ , $p_2 = 15$

**Example 8.17**
Consider the system

$$\begin{aligned} x_1 + 3x_2 &\leq 3\,; \\ 2x_1 + x_2 &\leq 2\,; \\ x_1\,,\ x_2 &\geq 0\,, \text{ integer.} \end{aligned}$$

With $u' = (1/5, 2/5)$ , it is easy to check that the corresponding Chvátal-Gomory cut is $x_1 + x_2 \leq 1$ . In fact, with this cut, all of the extreme points of the continuous relaxation of the new system are integer; see Figure 8.5.

Of course, it is by no means clear how to choose $u \in \mathbb{R}^m$ , and this is critical for getting useful inequalities. We should also bear in mind that there are examples for which Chvátal-Gomory are rather ineffectual. Trying to apply such cuts to Example 8.16 reveals that *infeasible* integer points can "guard" Chvátal-Gomory cuts from getting close to any feasible integer points.

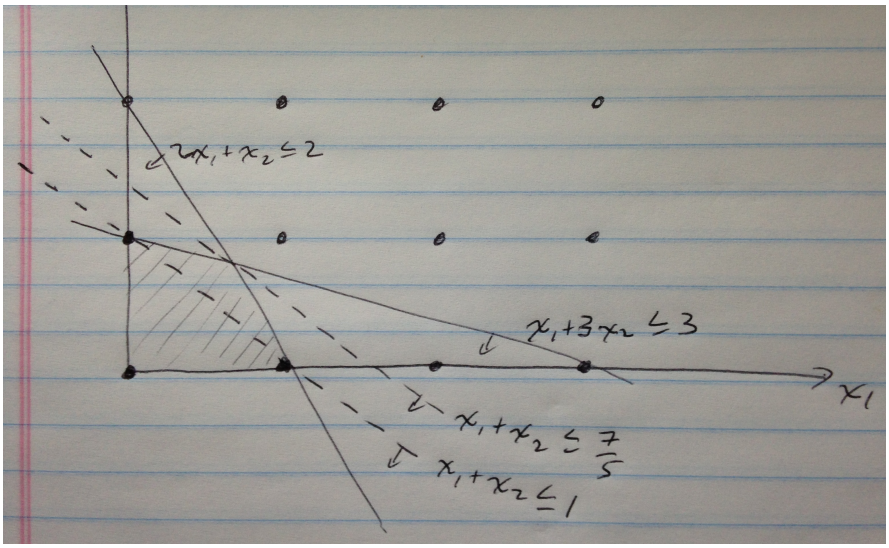



Figure 8.5: A Chvátal-Gomory cut

**Gomory cuts.** One idea for choosing useful cuts using the Chvátal-Gomory idea is based on basic solutions. Consider, now, the standard-form problem with integer constraints on all variables:

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,, \text{ integer.} \end{array}$$

Suppose that we have a basic feasible solution

$$\begin{array}{rcl} \bar{x}_\eta & := & \mathbf{0}\,; \\ \bar{x}_\beta & := & A_\beta^{-1} b\,. \end{array}$$

to the continuous relaxation. Associated with this choice of basis is an equivalent way to express $Ax = b$, namely

$$x_\beta + A_\beta^{-1} A_\eta x_\eta = A_\beta^{-1} b\,.$$

Let $\bar{b} := \bar{x}_\beta = A_\beta^{-1} b$, and let $\bar{A}_j := A_\beta^{-1} A_j$. So, we can view this equation system as

$$x_\beta + \bar{A}_\eta x_\eta = \bar{b}\,.$$

Taking a single one of these $m$ equations, we have the **source equation**

$$x_{\beta_i} + \sum_{j=1}^{n-m} \bar{a}_{i,\eta_j} x_{\eta_j} = \bar{b}_i\,.$$

Using the restriction $x_\eta \geq \mathbf{0}$, we see that

$$x_{\beta_i} + \sum_{j=1}^{n-m} \lfloor \bar{a}_{i,\eta_j} \rfloor x_{\eta_j} \leq \bar{b}_i\,.$$

Next, using the restriction that $x_{\beta_i}$ and $x_\eta$ are to be integer, we get the **Gomory cut**

$$x_{\beta_i} + \sum_{j=1}^{n-m} \lfloor \bar{a}_{i,\eta_j} \rfloor x_{\eta_j} \leq \lfloor \bar{b}_i \rfloor \ . \tag{Gom}$$

Notice how this inequality is violated by the basic solution $\bar{x}$ if we choose $i$ so that $\bar{x}_{\beta_i} = \bar{b}_i$ is not an integer. This suggests an iterative algorithm based on alternately (i) solving a continuous relaxation to get a basic optimal solution $\bar{x}$ , and (ii) appending a Gomory cut that is violated by $\bar{x}$ .

Finally, it can be useful to rewrite the Gomory cut in the space of just the non-basic variables:

$$\sum_{j=1}^{n-m} \left( \bar{a}_{i,\eta_j} - \lfloor \bar{a}_{i,\eta_j} \rfloor \right) x_{\eta_j} \geq \bar{b}_i - \lfloor \bar{b}_i \rfloor \ .$$

**Relationship between Gomory cuts and Chvátal-Gomory cuts.** If the data $\tilde{A} \in \mathbb{R}^{m \times (n-m)}$ and $b$ are integer, and we start with the inequality system

$$\begin{array}{rcl} \tilde{A}\tilde{x} & \leq & b \, ; \\ \tilde{x} & \geq & \mathbf{0} \, , \end{array}$$

we can actually see that in a certain way each Gomory cut is a Chvátal-Gomory cut. Introducing $m$ slack variables, we have the equivalent system

$$\begin{array}{rcccl} \tilde{A}\tilde{x} & + & s & = & b \, ; \\ \tilde{x} & , & s & \geq & \mathbf{0} \, . \end{array}$$

It is convenient to define $x \in \mathbb{R}^n$ by

$$x_j := \begin{cases} \tilde{x}_j \, , & \text{for } j = 1, \ldots, n-m \, ; \\ s_{j-(n-m)} \, , & \text{for } j = n-m+1, \ldots, n \, , \end{cases}$$

and to define $A \in \mathbb{R}^{m \times n}$ by

$$A_j := \begin{cases} \tilde{A}_j \, , & \text{for } j = 1, \ldots, n-m \, ; \\ \boldsymbol{e_{j-(n-m)}} \, , & \text{for } j = n-m+1, \ldots, n \, . \end{cases}$$

That is, $x = \begin{pmatrix} \tilde{x} \\ s \end{pmatrix}$ , and $A = [\tilde{A}, \mathbf{I}_m]$ , so now our system has the simple equivalent form

$$\begin{array}{rcl} Ax & = & b \, ; \\ x & \geq & \mathbf{0} \, . \end{array}$$

We can view the source equation

$$x_{\beta_i} + \sum_{j=1}^{n-m} \bar{a}_{i,\eta_j} x_{\eta_j} = \bar{b}_i$$

as a linear combination of the $m$ equations of the system $\tilde{A}\tilde{x} + s = b$ . Specifically, letting $\Lambda'$ be the $i$-th row of $A_\beta^{-1}$ , we have that the source equation is precisely

$$(\Lambda' \tilde{A})\tilde{x} + \Lambda' s = \Lambda b \, ,$$

and we can express the associated Gomory cut as

$$\lfloor \Lambda' \tilde{A} \rfloor \tilde{x} + \lfloor \Lambda' \rfloor s \leq \lfloor \Lambda' b \rfloor \ .$$

Now, let $u := \Lambda - \lfloor \Lambda \rfloor \geq \mathbf{0}$ , and consider the associated Chvátal-Gomory cut

$$\lfloor u' \tilde{A} \rfloor \tilde{x} \leq \lfloor u' b \rfloor \ ,$$

with respect to the system $\tilde{A}\tilde{x} \leq b$ , $\tilde{x} \geq \mathbf{0}$ integer. Substituting in $u := \Lambda - \lfloor \Lambda \rfloor$ , we get

$$\begin{array}{rcl} \lfloor (\Lambda - \lfloor \Lambda \rfloor)' \tilde{A} \rfloor \tilde{x} & \leq & \lfloor (\Lambda - \lfloor \Lambda \rfloor)' b \rfloor \ ; \\ \lfloor \Lambda' \tilde{A} - \lfloor \Lambda' \rfloor \tilde{A} \rfloor \tilde{x} & \leq & \lfloor \Lambda' b - \lfloor \Lambda' \rfloor b \rfloor \ . \end{array}$$

Using the integrality of the data $\tilde{A}$ and $b$ , we get

$$(\lfloor \Lambda' \tilde{A} \rfloor - \lfloor \Lambda' \rfloor \tilde{A}) \tilde{x} \leq \lfloor \Lambda' b \rfloor - \lfloor \Lambda' \rfloor b \ ,$$

which is equivalent to

$$\begin{array}{rcl} \lfloor \Lambda' \tilde{A} \rfloor \tilde{x} + \lfloor \Lambda' \rfloor (b - \tilde{A}\tilde{x}) & \leq & \lfloor \Lambda' b \rfloor \ ; \\ \lfloor \Lambda' \tilde{A} \rfloor \tilde{x} + \lfloor \Lambda' \rfloor s & \leq & \lfloor \Lambda' b \rfloor \ , \end{array}$$

which is precisely the Gomory cut that we were aiming for.

### 8.3.2 Mixed-integer problems

**Mixed-integer cuts.** Consider the *two-variable* system (I):

$$\begin{array}{rcl} w^I & \leq & d + w^C \ ; \\ w^I & & \text{integer;} \\ w^C & \geq & 0 \ , \end{array}$$

where the scalar $d$ is *not* an integer. Let $f := d - \lfloor d \rfloor$ . With a simple picture, it is easy to verify that the **basic mixed-integer inequality**

$$w^I \leq \lfloor d \rfloor + \frac{w^C}{1 - f} \qquad \text{(BMI)}$$

is satisfied by all solutions of (I) and that it cuts off part of the *continuous relaxation* of (I); see Figure 8.6. The line describing the inequality should pass through the points $(w^C, w^I) = (0, \lfloor d \rfloor)$ and $(w^C, w^I) = (\lceil d \rceil - d, \lceil d \rceil)$ . So the $w^I$-intercept is at $\lfloor d \rfloor$ and the slope is

$$\frac{\lceil d \rceil - \lfloor d \rfloor}{\lceil d \rceil - d} = \frac{1}{1 - f} \ .$$

This is enough to confirm that the (BMI) inequality is valid.

Next, we consider an apparently more complicated system (II):

$$\begin{array}{rcl} \displaystyle\sum_{j \in \mathcal{J}} \alpha_j w_j^I & \leq & d + w^C \ ; \\ w_j^I & \geq & 0 \text{ and integer, for } j \in \mathcal{J} \ ; \\ w^C & \geq & 0 \ , \end{array}$$

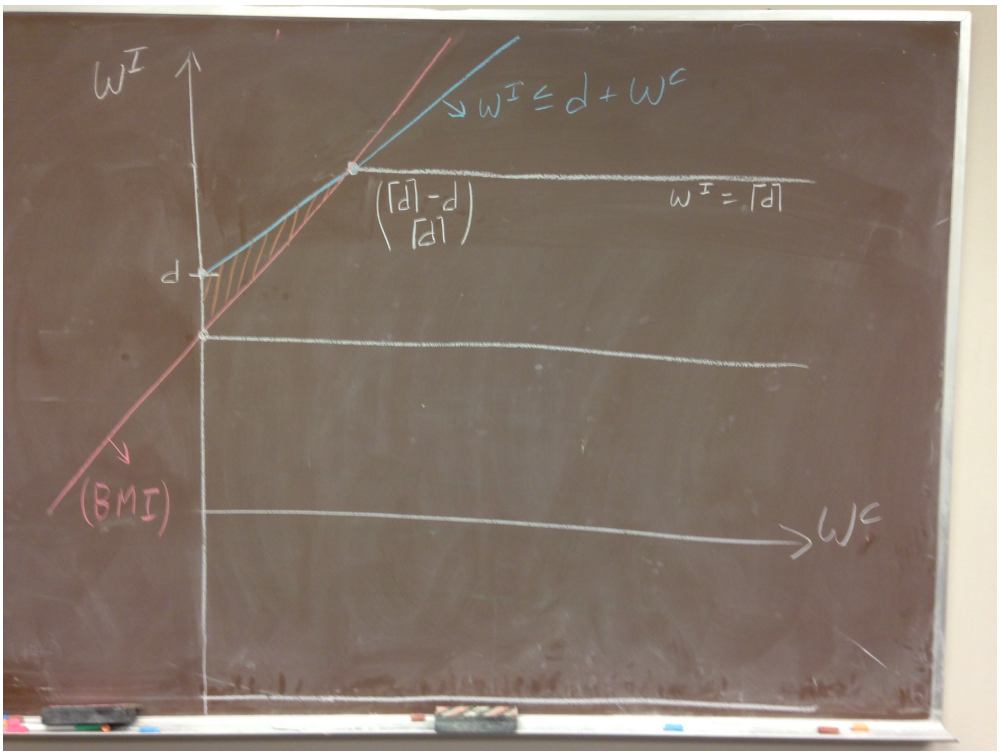



Figure 8.6: The basic mixed-integer inequality

where the scalar $d$ is *not* an integer. As before, let $f := d - \lfloor d \rfloor$ , and also let $f_j := \alpha_j - \lfloor \alpha_j \rfloor$ , for $j \in \mathcal{J}$ .

So now we have many integer variables, but they are constrained to be non-negative.

**Theorem 8.18**
The **mixed-integer rounding inequality**

$$\sum_{\substack{j \in \mathcal{J}: \\ f_j \le f}} \lfloor \alpha_j \rfloor w_j^I + \sum_{\substack{j \in \mathcal{J}: \\ f_j > f}} \left( \lfloor \alpha_j \rfloor + \frac{f_j - f}{1 - f} \right) w_j^I \le \lfloor d \rfloor + \frac{w^C}{1 - f} \qquad \text{(MIR)}$$

is satisfied by all solutions of (II).

*Proof.* Let $S_l$ , $S_h$ be a partition of $\mathcal{J}$ with $S_l$ containing $\{j \in \mathcal{J} \ : \ f_j = 0\}$ . We rewrite the key inequality of (II) as

$$\sum_{j \in S_l} \alpha_j w_j^I + \sum_{j \in S_h} \alpha_j w_j^I \le d + w^C \ .$$

Next, we (possibly) weaken the inequality by replacing $\alpha_j$ with $\lfloor \alpha_j \rfloor$ for $j \in S_l$ . Fur-

thermore, we replace $\alpha_j$ with the equivalent expression $\lceil \alpha_j \rceil - 1 + f_j$ for $j \in S_h$ (note that this substitution would not be valid except for the fact that $f_j > 0$ for $j \in S_h$ ).

So, we have the valid inequality

$$\sum_{j \in S_l} \lfloor \alpha_j \rfloor w_j^I + \sum_{j \in S_h} \left( \lceil \alpha_j \rceil - 1 + f_j \right) w_j^I \leq d + w^C \ ,$$

which we rearrange as

$$\underbrace{\left( \sum_{j \in S_l} \lfloor \alpha_j \rfloor w_j^I + \sum_{j \in S_h} \lceil \alpha_j \rceil w_j^I \right)}_{\text{integer}} \leq d + \underbrace{\left( w^C + \sum_{j \in S_h} \left(1 - f_j\right) w_j^I \right)}_{\geq 0} \ .$$

We have annotated some expressions to indicate how the (BMI) inequality can now be applied. It follows that the following inequality is valid:

$$\sum_{j \in S_l} \lfloor \alpha_j \rfloor w_j^I + \sum_{j \in S_h} \lceil \alpha_j \rceil w_j^I \leq \lfloor d \rfloor + \frac{w^C + \sum_{j \in S_h} \left(1 - f_j\right) w_j^I}{1 - f} \ .$$

Reorganizing the inequality, we have:

$$\sum_{j \in S_l} \lfloor \alpha_j \rfloor w_j^I + \sum_{j \in S_h} \left( \lceil \alpha_j \rceil + \frac{f_j - 1}{1 - f} \right) w_j^I \leq \lfloor d \rfloor + \frac{w^C}{1 - f} \ ,$$

which can be re-expressed as

$$\sum_{j \in S_l} \lfloor \alpha_j \rfloor w_j^I + \sum_{j \in S_h} \left( \lfloor \alpha_j \rfloor + \frac{f_j - f}{1 - f} \right) w_j^I \leq \lfloor d \rfloor + \frac{w^C}{1 - f} \ ,$$

Finally, we observe that because the $w_j^I$ are non-negatively constrained, and because we have a choice about how to partition $\mathcal{J}$ into $S_l$ and $S_h$ , as long as we have $S_l$ containing the $j$ such that $f_j = 0$ , it is clear that the partition yielding the strongest such inequality has $S_l = \{ j \in \mathcal{J} \ : \ f_j \leq f \}$ and $S_h = \{ j \in \mathcal{J} \ : \ f_j > f \}$ . This is exactly the form of (MIR). □

Now, consider a standard-form problem for which *some* of the variables are constrained to be integer. Consider a basic solution for which an integer-constrained variable $x_{\beta_i}$ takes on a non-integer value $\bar{b}_i$ . Expressing $x_{\beta_i}$ in terms of the non-basic variables, we have

$$x_{\beta_i} + \sum_{j \in \eta^I} \bar{a}_{i,j} x_j + \sum_{j \in \eta^C} \bar{a}_{i,j} x_j = \bar{b}_i \ ,$$

where the non-basic index set $\eta$ breaks up into the set $\eta^I$ of indices of integer variables that are non-basic and the set $\eta^C$ of indices of continuous variables that are non-basic. Now, let $f := \bar{b}_i - \lfloor \bar{b}_i \rfloor$ and $f_j := \bar{a}_{i,j} - \lfloor \bar{a}_{i,j} \rfloor$ , for non-basic $j$ .

Dropping terms $\bar{a}_{i,j}x_j$ when $j \in \eta^C$ and $\bar{a}_{i,j} > 0$ and moving terms $\bar{a}_{i,j}x_j$ to the right-hand side when $j \in \eta^C$ and $\bar{a}_{i,j} < 0$ , we get the valid inequality

$$\underbrace{x_{\beta_i} + \sum_{j \in \eta^I} \bar{a}_{i,j} x_j}_{\sum_{j \in \mathcal{J}} \alpha_j w^I_j} \leq \bar{b}_i + \underbrace{\sum_{\substack{j \in \eta^C : \\ \bar{a}_{i,j} < 0}} (-\bar{a}_{ij})\, x_j}_{w^C} ,$$

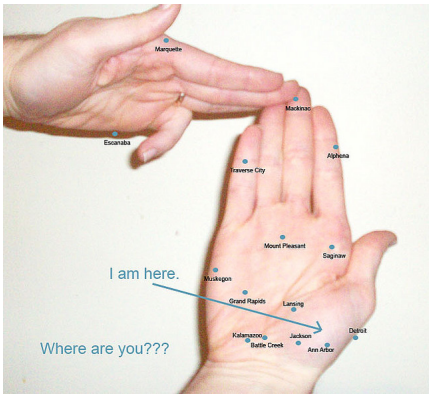
Finally, applying the (MIR) inequality as indicated above, where here $f := \bar{b}_i - \lfloor \bar{b}_i \rfloor$ and $f_j := \bar{a}_{i,j} - \lfloor \bar{a}_{i,j} \rfloor$ , we are led to the **Gomory mixed-integer inequality**

$$x_{\beta_i} + \sum_{\substack{j \in \eta^I : \\ f_j \leq f}} \lfloor \bar{a}_{i,j} \rfloor x_j + \sum_{\substack{j \in \eta^I : \\ f_j > f}} \left( \lfloor \bar{a}_{i,j} \rfloor + \frac{f_j - f}{1 - f} \right) x_j \leq \lfloor \bar{b}_i \rfloor + \frac{\sum_{\substack{j \in \eta^C : \\ \bar{a}_{i,j} < 0}} (-\bar{a}_{i,j})\, x_j}{1 - f} , \qquad \text{(GMI)}$$

which we have now shown to be valid for a standard-form problem in which some of the variables are constrained to be integer.

Notice how the (GMI) inequality is violated by the basic solution $\bar{x}$ if we choose $i$ so that $\bar{x}_{\beta_i} = \bar{b}_i$ is not an integer. This suggests an iterative algorithm based on alternately (i) solving a continuous relaxation to get a basic optimal solution $\bar{x}$ , and (ii) appending a (GMI) inequality that is violated by $\bar{x}$ .

## 8.4 Branch-and-Bound




In this section, we look at a rudimentary framework called **Branch-and-Bound**, which aims at finding an optimal solution of a linear-optimization problem having some integer variables. We consider a problem of the form

$$\begin{array}{lllll} z := \max & y'b & & & \\ & y'A & \leq & c' \,; & \\ & y & \in & \mathbb{R}^m \,; & \\ & y_i & & \text{integer, for } i \in \mathcal{I} \,. & \end{array} \qquad (\mathrm{D}_{\mathcal{I}})$$

The set $\mathcal{I} \subset \{1, 2, \ldots, m\}$ allows for a given subset of the variables to be constrained to be integer. This linear-optimization problem has a non-standard form as a point

of departure, but it is convenient that the dual of the continuous relaxation has the standard form

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,. \end{array} \qquad \text{(P)}$$

We assume that (P) has a feasible solution. Hence, even the continuous relaxation (D) of ($D_{\mathcal{I}}$) is not unbounded.

Our algorithm maintains a list $\mathcal{L}$ of optimization problems that all have the general form of ($D_{\mathcal{I}}$). Keep in mind that problems on the list have integer variables. We maintain a **lower bound** LB, satisfying LB $\leq z$ . Put simply, LB is the objective value of the best (objective maximizing) feasible solution $\tilde{y}_{\text{LB}}$ of ($D_{\mathcal{I}}$) that we have seen so far; that is, LB is the maximum objective value attained, over all 'solved' problems. Initially, we set LB $= -\infty$ , and we update it in an increasing fashion.

The algorithm maintains an important property:

*Every feasible solution of the original problem ($D_{\mathcal{I}}$) with greater objective value than LB is feasible for a problem on the list.*

We stop when the list is empty, and because of the property that we maintain, we correctly conclude that the optimal value of ($D_{\mathcal{I}}$) is LB when we do stop.

At a general step of the algorithm, we select and remove a problem ($\tilde{D}_{\mathcal{I}}$) on the list, and we solve its continuous relaxation ($\tilde{D}$). If this continuous relaxation is infeasible, then we do nothing further with this problem. Otherwise, we let $\bar{y}$ be its optimal solution, and we proceed as follows.

- If $\bar{y}'b \leq$ LB , then no feasible solution to the selected problem can have objective value greater than LB, so we are done processing this selected problem.
- If $\bar{y}_i$ is integer for all $i \in \mathcal{I}$ , then we have solved the selected problem. In this case, if $\bar{y}'b >$ LB , then we
  - reset LB to $\bar{y}'b$ ;
  - reset $\bar{y}_{\text{LB}}$ to $\bar{y}$ .
- Finally, if $\bar{y}'b >$ LB and $\bar{y}_i$ is not integer for all $i \in \mathcal{I}$ , then (it is possible that this selected problem has a feasible solution that is better than $\bar{y}_{\text{LB}}$ , so) we
  - select some $i \in \mathcal{I}$ such that $\bar{y}_i$ is not integer;
  - place two new "child" problems on the list, one with the constraint $y_i \leq \lfloor \bar{y}_i \rfloor$ appended (the so-called **down branch**), and the other with the constraint $y_i \geq \lceil \bar{y}_i \rceil$ appended (the so-called **up branch**).

  (observe that every feasible solution to a parent is feasible for one of its children, if it has children.)

The following result is evident.

**Theorem 8.19**
Suppose that the original (P) is feasible. Then at termination of Branch-and-Bound, we have LB= $-\infty$ if ($D_{\mathcal{I}}$) is infeasible or with $\bar{y}_{\text{LB}}$ being an optimal solution of ($D_{\mathcal{I}}$).

**Finite termination.** If the *feasible region* of the continuous relaxation (D) of ($D_{\mathcal{I}}$) is a bounded set, then we can guarantee finite termination. If we do not want to make such an assumption, then if we assume that the data for the formulation is rational, it is possible to bound the region that needs to be searched, and we can again assure finite termination.

**Solving continuous relaxations.** Some remarks are in order regarding the solution of continuous relaxations. Conceptually, we apply the Simplex Algorithm to the dual ($\tilde{P}$) of the continuous relaxation ($\tilde{D}$) of a problem ($\tilde{D}_{\mathcal{I}}$) selected and removed from the list. At the outset, for an optimal basis $\beta$ of ($\tilde{P}$), the optimal dual solution is given by $\bar{y}' := c'_\beta A_\beta^{-1}$ . If $i \in \mathcal{I}$ is chosen, such that $\bar{y}_i$ is not an integer, then we replace the selected problem ($\tilde{D}_{\mathcal{I}}$) with one child having the additional constraint $y_i \leq \lfloor \bar{y}_i \rfloor$ (the down branch) and another with the constraint $y_i \geq \lceil \bar{y}_i \rceil$ appended (the up branch).

Adding a constraint to ($\tilde{D}$) adds a variable to the standard-form problem ($\tilde{P}$). So, a basis for ($\tilde{P}$) remains feasible after we introduce such a variable.

- **The down branch:** The constraint $y_i \leq \lfloor \bar{y}_i \rfloor$ , dualizes to a new variable $x_{\text{down}}$ in ($\tilde{P}$). The variable $x_{\text{down}}$ has a new column $A_{\text{down}} := \boldsymbol{e_i}$ and a cost coefficient of $c_{\text{down}} := \lfloor \bar{y}_i \rfloor$ . Notice that the fact that $\bar{y}_i$ is not an integer (and hence $\bar{y}$ violates $y_i \leq \lfloor \bar{y}_i \rfloor$) translates into the fact that the reduced cost $\bar{c}_{\text{down}}$ of $x_{\text{down}}$ is $\bar{c}_{\text{down}} = c_{\text{down}} - \bar{y}' A_{\text{down}} = \lfloor \bar{y}_i \rfloor - \bar{y}_i < 0$ , so $x_{\text{down}}$ is eligible to enter the basis.

- **The up branch:** Similarly, the constraint $y_i \geq \lceil \bar{y}_i \rceil$ , or equivalently $-y_i \leq -\lceil \bar{y}_i \rceil$ , dualizes to a new variable $x_{\text{up}}$ in ($\tilde{P}$). The variable $x_{\text{up}}$ has a new column $A_{\text{up}} := -\boldsymbol{e_i}$ and a cost coefficient of $c_{\text{up}} := -\lceil \bar{y}_i \rceil$ . Notice that the fact that $\bar{y}_i$ is not an integer (and hence $\bar{y}$ violates $y_i \geq \lceil \bar{y}_i \rceil$) translates into the fact that the reduced cost $\bar{c}_{\text{up}}$ of $x_{\text{up}}$ is $\bar{c}_{\text{up}} = c_{\text{up}} - \bar{y}' A_{\text{up}} = -\lceil \bar{y}_i \rceil + \bar{y}_i < 0$ , so $x_{\text{up}}$ is eligible to enter the basis.

In either case, provided that we have kept the optimal basis for the ($\tilde{P}$) associated with a problem ($\tilde{D}_{\mathcal{I}}$), the Simplex Algorithm picks up on the ($\tilde{\tilde{P}}$) associated with a child ($\tilde{\tilde{D}}_{\mathcal{I}}$) of that problem , with the new variable of the child's ($\tilde{\tilde{P}}$) entering the basis.

Notice that the ($\tilde{P}$) associated with a problem ($\tilde{D}_{\mathcal{I}}$) on the list could be unbounded. But this just implies that the problem ($\tilde{D}_{\mathcal{I}}$) is infeasible.

**Partially solving continuous relaxations.** Notice that as the Simplex Algorithm is applied to the ($\tilde{P}$) associated with any problem ($\tilde{D}_{\mathcal{I}}$) from the list, we generate a sequence of non-increasing objective values, each one of which is an upperbound on the optimal objective value of ($\tilde{D}_{\mathcal{I}}$). That is, for any such ($\tilde{P}$), we start with the upperbound value

of its parent, and then we gradually decrease it, step-by-step of the Simplex Algorithm. At any point in this process, if the objective value of the Simplex Algorithm falls at or below the current LB, we can immediately terminate the Simplex Algorithm on such a ($\tilde{\text{P}}$) — its optimal objective value will be no greater than LB — and conclude that the optimal objective value of ($\tilde{\text{D}}_{\mathcal{I}}$) is no greater than LB.

**A global upper bound.** As the algorithm progresses, if we let $\text{UB}_{\text{better}}$ be the maximum, over all problems on the list, of the objective value of the continuous relaxations, then any feasible solution $\hat{y}$ with objective value greater than that LB satisfies $\hat{y}'b \leq \text{UB}_{\text{better}}$ . Of course, it may be that no optimal solution is feasible to any problem on the list — for example if it happens that LB $= z$ . But we can see that

$$z \leq \text{UB} := \max\left\{\text{UB}_{\text{better}}, LB\right\} \ .$$

It may be useful to have UB at hand, because we can always stop the computation early, say when UB $-$ LB $< \tau$ , returning the feasible solution $\bar{y}_{\text{LB}}$ , with the knowledge that $z - \bar{y}'_{\text{LB}}b \leq \tau$ . But notice that we do not readily have the objective value of the continuous relaxation for problems on the list — we only solve the continuous relaxation for such a problem after it is selected (for processing). But, for every problem on the list, we can simply keep track of the optimal objective value of its parent's continuous relaxation, and use that instead. Alternatively, we can re-organize our computations a bit, solving continuous relaxations of subproblems *before* we put them on the list.

**Selecting a subproblem from the list.** Which subproblem from the list should we process next?

- A strategy of *last-in/first-out*, known as **diving**, often results in good increases in LB. To completely specify such a strategy, one would have to decide which of the two children of a subproblem is put on the list last (i.e., the down branch or the up branch). A good choice can affect the performance of this rule, and such a good choice depends on the type of model being solved.
- A strategy of *first-in/first-out* is *very bad*. It can easily result in an explosion in the size of the list of subproblems.
- A strategy of choosing a subproblem to branch on having objective value for its continuous relaxation equal to UB, known as **best bound**, is a sound strategy for seeking a decrease in UB. If such a rule is desired, then it is best to solve continuous relaxations of subproblems *before* we put them on the list.

A hybrid strategy, doing mostly diving at the start (to get a reasonable value of LB) and shifting more and more to best bound (to work on proving that LB is at or near the optimal value) has rather robust performance.

**Selecting a branching variable.** Probably very many times, we will need to choose an $i \in \mathcal{I}$ for which $\bar{y}_i$ is fractional, in order to branch and create the child subproblems. Which such $i$ should we choose? Naïve rules such as choosing randomly or the so-called

**most fractional** rule of choosing an $i$ that maximizes $\min\{\bar{y}_i - \lfloor \bar{y}_i \rfloor \ , \ \lceil \bar{y}_i \rceil - \bar{y}_i\}$ seem to have rather poor performance. Better rules are based on estimates of how the objective value of the children will change relative to the parent.

**Using dual variables to bound the "other side" of an inequality.**

**Theorem 8.20**
Let LB be the objective value of any feasible solution of ($D_{\mathcal{I}}$). Let $\bar{x}$ be an optimal solution of (P), and assume that $\bar{x}_j > 0$ for some $j$ . Then

$$c_j + \frac{\text{LB} - c'\bar{x}}{\bar{x}_j} \leq y' A_j$$

is satisfied by every optimal solution of ($D_{\mathcal{I}}$).

*Proof.* We consider a parametric version of ($D_{\mathcal{I}}$). For $\Delta_j \in \mathbb{R}$, consider

$$\begin{array}{lll} z(\Delta_j) := \max & y'b & \\ & y'A \leq & c' + \Delta_j \boldsymbol{e_j'} \; ; \\ & y \in & \mathbb{R}^m \; ; \\ & y_i & \text{integer, for } i \in \mathcal{I} \; . \end{array} \qquad (D_{\mathcal{I}}(\Delta_j))$$

Let $z_R(\Delta_j)$ be defined the same way as $z(\Delta_j)$, but with integrality relaxed. Using ideas from Chapters 6 and 7, we can see that $z_R$ is a concave (piecewise-linear) function on its domain, and $\bar{x}_j$ is a subgradient of $z_R$ at $\Delta_j = 0$ . It follows that

$$z(\Delta_j) \leq z_R(\Delta_j) \leq z_R(0) + \Delta_j \bar{x}_j = c'\bar{x} + \Delta_j \bar{x}_j \; .$$

So, we can observe that for

$$\Delta_j < \frac{\text{LB} - c'\bar{x}}{\bar{x}_j} \; ,$$

we will have $z(\Delta_j) < \text{LB}$. Therefore, every $\hat{y}$ that is feasible for ($D_{\mathcal{I}}(\Delta_j)$) with $\Delta_j < (\text{LB} - c'\bar{x})/\bar{x}_j$ will have $\hat{y}'b < \text{LB}$ . So such a $\hat{y}$ cannot be optimal for ($D_{\mathcal{I}}$). □

It is interesting to consider two special cases of Theorem 8.20:

**Corollary 8.21 (Variable fixing)**
Let LB be the objective value of any feasible solution of ($D_{\mathcal{I}}$). Let $\bar{x}$ be an optimal solution of (P). Assume that $\bar{x}_j > 0$ is the optimal dual variable for a constraint of the form: $y_k \leq 1$ (or $-y_k \leq 0$) . If $c'\bar{x} - \text{LB} < \bar{x}_j$, then $y_k = 1$ (respectively, $y_k = 0$) is satisfied by every optimal solution of ($D_{\mathcal{I}}$).

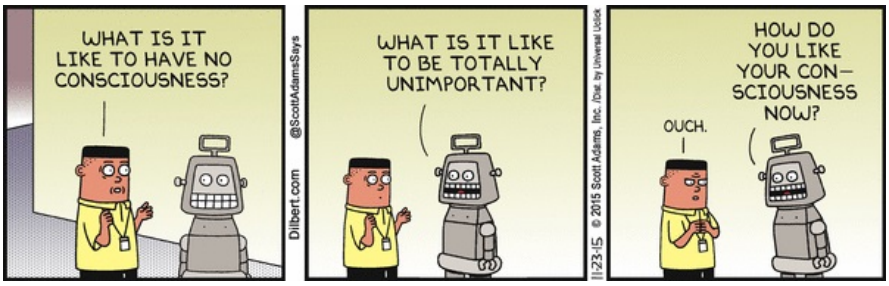
Because of Theorem 6.4, this is known as **reduced-cost fixing**.

**Branch-and-Cut.** State-of-the-art algorithms for (mixed-)integer linear optimization combine cuts with Branch-and-Bound. There are a lot of software design and tuning issues that make this work successfully.

## 8.5 Exercises

**Exercise 8.1 (Task scheduling, continued)**
Consider again the "task scheduling" Exercise 2.5. Take the dual of the linear-optimization problem that you formulated. Explain how this dual can be interpreted as a kind of network problem. Using `AMPL`, solve the dual of the example that you created for Exercise 2.5 and interpret the solution.

**Exercise 8.2 (Pivoting and total unimodularity)**




A **pivot** in an $m \times n$ matrix $A$ means choosing a row $i$ and column $j$ with $a_{ij} \neq 0$, subtracting $\frac{a_{kj}}{a_{ij}}$ times row $i$ from all other rows $k(\neq i)$, and then dividing row $i$ by $a_{ij}$. Note that after the pivot, column $j$ becomes the $i$-th standard-unit column. Prove that if $A$ is TU, then it is TU after a pivot.

**Exercise 8.3 (Comparing formulations for a toy problem)**
Consider the systems:

$$\begin{aligned} S_1 : \quad 2x_1 + 2x_2 + x_3 + x_4 &\leq 2\,; \\ x_j &\leq 1\,; \\ -x_j &\leq 0\,. \end{aligned}$$

$$\begin{aligned} S_2 : \quad x_1 + x_2 + x_3 &\leq 1\,; \\ x_1 + x_2 + x_4 &\leq 1\,; \\ -x_j &\leq 0\,. \end{aligned}$$

$$\begin{aligned} S_3 : \quad x_1 + x_2 &\leq 1\,; \\ x_1 + x_3 &\leq 1\,; \\ x_1 + x_4 &\leq 1\,; \\ x_2 + x_3 &\leq 1\,; \\ x_2 + x_4 &\leq 1\,; \\ -x_j &\leq 0\,. \end{aligned}$$

Notice that each system has precisely the same set of integer solutions. Compare the feasible regions $S_i$ of the continuous relaxations, for each pair of these systems. Specifically, for each choice of pair $i \neq j$, demonstrate whether or not the solution set of $S_i$

is contained in the solution set of $S_j$ . HINT: To prove that the solution set of $S_i$ is contained in the solution set of $S_j$ , it suffices to demonstrate that every inequality of $S_j$ is a non-negative linear combination of the inequalities of $S_i$ . To prove that the solution set of $S_i$ is *not* contained in the solution set of $S_j$ , it suffices to give a solution of $S_i$ that is not a solution of $S_j$ .

**Exercise 8.4 (Comparing facility-location formulations)**
We have seen two formulations of the forcing constraints for the uncapacitated facility-location problem. We have a choice of the $mn$ constraints: $-y_i + x_{ij} \leq 0$ , for $i = 1, \ldots, m$ and $j = 1, \ldots, n$ , or the $m$ constraints: $-ny_i + \sum_{j=1}^{n} x_{ij} \leq 0$ , for $i = 1, \ldots, m$ . Which formulation is stronger? That is, compare (both computationally *and analytically*) the strength of the two associated continuous relaxations (i.e., when we relax $y_i \in \{0, 1\}$ to $0 \leq y_i \leq 1$ , for $i = 1, \ldots, m$). In Appendix A.3, there is `AMPL` code for trying computational experiments.

**Exercise 8.5 (Comparing piecewise-linear formulations)**
We have seen that the adjacency condition for piecewise-linear univariate functions can be modeled by

$$\begin{aligned} \lambda_1 &\leq y_1 \ ; \\ \lambda_j &\leq y_{j-1} + y_j \ , \text{ for } j = 2, \ldots, n-1 \ ; \\ \lambda_n &\leq y_{n-1} \ . \end{aligned}$$

An alternative formulation is

$$\begin{aligned} &\sum_{i=1}^{j} y_i \leq \sum_{i=1}^{j+1} \lambda_i \ , \text{ for } j = 1, \ldots, n-2 \ ; \\ &\sum_{i=j}^{n-1} y_i \leq \sum_{i=j}^{n} \lambda_i \ , \text{ for } j = 2, \ldots, n-1 \ . \end{aligned}$$

Explain why this alternative formulation is valid, and compare its strength to the original formulation, when we relax $y_i \in \{0, 1\}$ to $0 \leq y_i \leq 1$ , for $i = 1, \ldots, n-1$ . (Note that for both formulations, we require $\lambda_i \geq 0$, for $i = 1, \ldots, n$ , $\sum_{i=1}^{n} \lambda_i = 1$ , and $\sum_{i=1}^{n-1} y_i = 1$).

**Exercise 8.6 (Variable fixing)**
Prove Corollary 8.21.

**Exercise 8.7 (Gomory cuts)**
Consider the standard form problem having

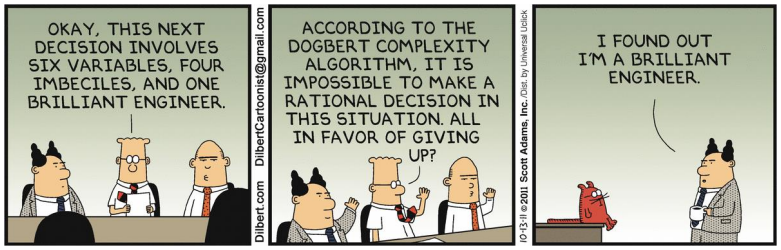
$$A := \begin{pmatrix} 7 & 1 & 1 & 0 & 0 \\ -1 & 3 & 0 & 1 & 0 \\ -8 & -9 & 0 & 0 & 1 \end{pmatrix} , \ b := \begin{pmatrix} 28 \\ 7 \\ -32 \end{pmatrix} , \ c := (-2, -1, 0, 0, 0)' \ .$$

An optimal basis is $\beta := (1, 2, 5)$ , $\eta := (3, 4)$ . Suppose that all of the variables are constrained to be integer. Derive all (Gom) cuts for this basis and compare their strength

on the continuous relaxation of the feasible region. HINT: View everything in the space of the variables $x_\eta$ .

Can you get a stronger (GMI) cut (with respect to this basis) by ignoring the integrality on some variables?

**Exercise 8.8 (Make amends)**
Find an *interesting applied problem*, model it as a pure- or mixed- *integer* linear-optimization problem, and test your model with `AMPL`.




Credit will be given for *deft* modeling, *sophisticated* use of `AMPL`, testing on meaningfully-*large* instances, and *insightful* analysis. Try to play with `CPLEX` *integer* solver options (they can be set through `AMPL`) to get better behavior of the solver.

Your grade on this problem will **replace your grades** on up to 6 homework problems (i.e., up to 6 homework problems on which you have lower grades than you get on this one). *I will not consider any re-grades on this one!* If you already have all or mostly A's (or not), do a good job on this one because you want to impress me, and because you are ambitious, and because this problem is what we have been working towards all during the course, and because you should always finish strong.

# Take rest

# Appendices

## A.1 LaTeX template

# LaTeX TEMPLATE

Your actual name (youremail@umich.edu), Another actual name (anotheremail@umich.edu)

This template can serve as a starting point for learning LaTeX. You may download MiKTeX from `miktex.org` to get started. Look at the source file for this document (in Section 5) to see how to get all of the effects demonstrated.

## 1. This is the first section where we make some lists

It is easy to make enumerated lists:

(1) This is the first item
(2) Here is the second

And even enumerated sublists:

(1) This is the first item
(2) Here is the second with a sublist
    (a) first sublist item
    (b) and here is the second

## 2. Here is a second section where we typeset some math

You can typeset math inline, like $\sum_{j=1}^{n} a_{ij}x_j$, by just enclosing the math in dollar signs. But if you want to *display* the math, then you do it like this:

$$\sum_{j=1}^{n} a_{ij}x_j \ \forall \ i = 1, \ldots, m.$$

And here is a matrix:

$$\begin{pmatrix} 1 & \pi & 2 & \frac{1}{2} & \nu \\ 6.2 & r & 2 & 4 & 5 \\ |y'| & \mathcal{R} & \mathbb{R} & \underline{\mathrm{r}} & \hat{R} \end{pmatrix}$$

Here is an equation array, with the equal signs nicely aligned:

$$\sum_{j=1}^{n} x_j = 5 \tag{2.1}$$

$$\sum_{j=1}^{n} y_j = 7 \tag{2.2}$$

$$\sum_{j \in S} x_j = 29 \tag{2.3}$$

The equations are automatically numbered, like $x.y$, where $x$ is the section number and $y$ is the $y$-th equation in section $x$. By tagging the equations with labels, we can refer to them later, like (2.3) and (2.1).

**Theorem 2.1.** *This is my favorite theorem.*

*Proof.* Unfortunately, the space here does not allow for including my ingenious proof of Theorem 2.1. □

## 3. Here is how I typset a standard-form linear-optimization problem

$$\begin{array}{lrcl} \min & c'x & & \\ & Ax & = & b\,; \\ & x & \geq & \mathbf{0}\,. \end{array} \tag{P}$$

Notice that in this example, there are 4 columns separated by 3 &'s. The 'rrcl' organizes justification within a column. Of course, one can make more columns.

---

*Date*: December 20, 2016.

## 4. Graphics

This is how to include and refer to Figure 1 with pdfLaTeX.

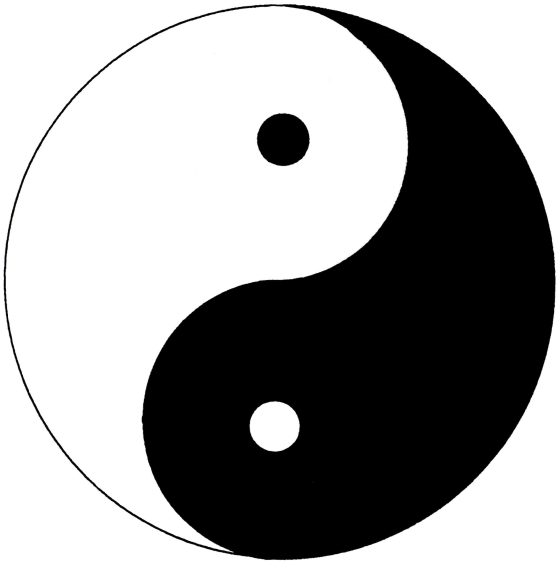


Figure 1. Another duality

## 5. The LaTeX commands to produce this document

Look at the LaTeX commands in this section to see how each of the elements of this document was produced. Also, this section serves to show how text files (e.g., programs) can be included in a LaTeX document verbatim.

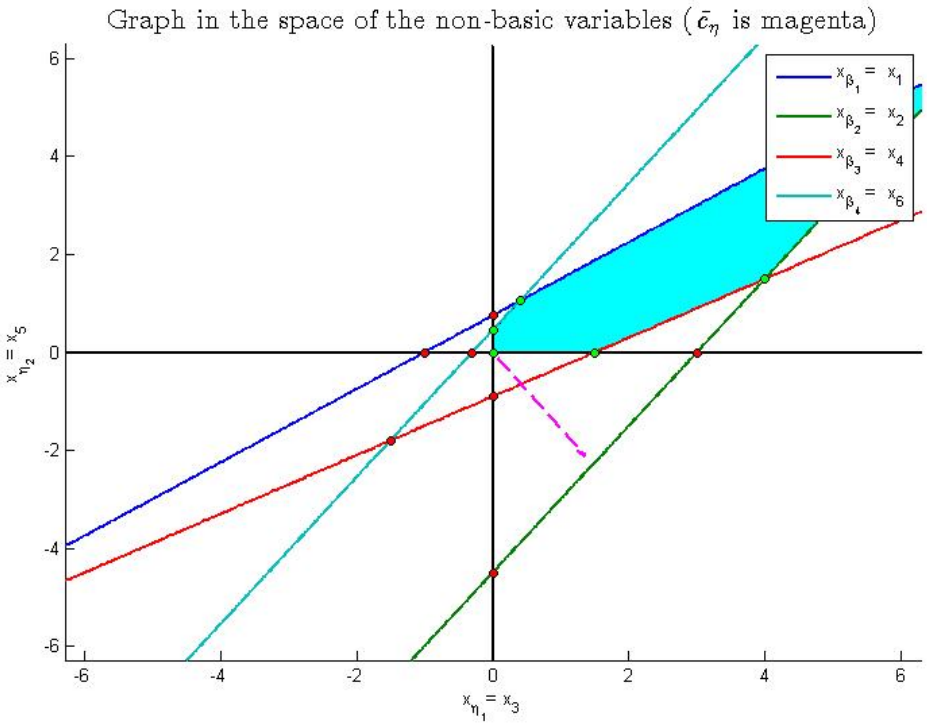
```
%  LaTeX_Template.tex // J. Lee
%
% ------------------------------------------------------------
% AMS-LaTeX ***********************************************
% **** --------------------------------------------------------
\documentclass{amsart}
\usepackage{graphicx,amsmath,amsthm}
\usepackage{hyperref}
\usepackage{verbatim}
\usepackage[a4paper,text={16.5cm,25.2cm},centering]{geometry}
% ------------------------------------------------------------
\vfuzz2pt % Don't report over-full v-boxes if over-edge is small
\hfuzz2pt % Don't report over-full h-boxes if over-edge is small
% THEOREMS -----------------------------------------------------
\newtheorem{thm}{Theorem}[section]
\newtheorem{cor}[thm]{Corollary}
\newtheorem{lem}[thm]{Lemma}
\newtheorem{prop}[thm]{Proposition}
\theoremstyle{definition}
\newtheorem{defn}[thm]{Definition}
\theoremstyle{remark}
\newtheorem{rem}[thm]{Remark}
\numberwithin{equation}{section}
% MATH ---------------------------------------------------------
\newcommand{\Real}{\mathbb R}
\newcommand{\eps}{\varepsilon}
\newcommand{\To}{\longrightarrow}
\newcommand{\BX}{\mathbf{B}(X)}
\newcommand{\A}{\mathcal{A}}
% --------------------------------------------------------------
\begin{document}
```

```
\title{\LaTeX~ Template}

\date{\today}

\maketitle

 \href{mailto:youremail@umich.edu,anotheremail@umich.edu}
{Your actual name (youremail@umich.edu),
Another actual name (anotheremail@umich.edu)}

%
%\medskip
%
%(this identifies your work and it \emph{greatly} help's me in returning homework to you by email
%---- just plug in the appropriate replacements in the \LaTeX~ source; then when I click on the
%hyperlink above, my email program opens up starting a message to you)

\bigskip

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

This template can serve as a starting point for learning \LaTeX. You may download MiKTeX from
{\tt miktex.org}
to get started. Look at the source file for this
document (in Section \ref{sec:appendix})
to see how to get all of the effects demonstrated.

\section{This is the first section where we make some lists}

It is easy to make enumerated lists:
\begin{enumerate}
\item This is the first item
\item Here is the second
\end{enumerate}

And even enumerated sublists:
\begin{enumerate}
\item This is the first item
\item Here is the second with a sublist
\begin{enumerate}
\item first sublist item
\item and here is the second
\end{enumerate}
\end{enumerate}

\section{Here is a second section where we typeset some math}

You can typeset math inline, like $\sum_{j=1}^n a_{ij} x_j$, by just enclosing the math in dollar signs.

But if you want to \emph{display} the math, then you do it like this:

\[
\sum_{j=1}^n a_{ij} x_j~ \forall~ i=1,\ldots,m.
\]

And here is a matrix:
\[
\left(
  \begin{array}{ccccc}
    1 & \pi & 2& \frac{1}{2} & \nu \\
    6.2 & r & 2 & 4 & 5 \\
    |y'| & \mathcal{R} & \mathbb{R} & \underbar{r} & \hat{R} \\
  \end{array}
\right)
\]
```

```
Here is an equation array, with the equal signs nicely aligned:
\begin{eqnarray}
  \sum_{j=1}^n x_j &=& 5 \label{E1} \\
    \sum_{j=1}^n y_j &=& 7 \label{E7} \\
  \sum_{j\in S} x_j &=& 29 \label{E4}
\end{eqnarray}

The equations are automatically numbered, like $x.y$, where
$x$ is the section number and $y$ is the $y$-th equation in section $x$.
By tagging the equations
with labels, we can refer to them later, like (\ref{E4}) and (\ref{E1}).

\begin{thm}\label{Favorite}
This is my favorite theorem.
\end{thm}
\begin{proof}
Unfortunately, the space here does not allow for including my ingenious proof
of Theorem \ref{Favorite}.
\end{proof}

\section{Here is how I typset a standard-form linear-optimization problem}

\[
\tag{P}
\begin{array}{rrcl}
 \min & c'x  &      &   \\
      &  Ax  &   =  & b~; \\
      &   x  & \geq & \mathbf{0}~.
\end{array}
\]

Notice that in this example, there are 4 columns separated by 3 \&'s.
The 'rrcl' organizes justification within a column.
Of course, one can make more columns.

\section{Graphics}

This is how to include and refer to Figure \ref{nameoffigure} with pdfLaTeX.

\begin{figure}[h!!]
\includegraphics[width=0.5\textwidth]{yinyang.jpg}
\caption{Another duality}\label{nameoffigure}
\end{figure}

\section{The \LaTeX~ commands to produce this document}
\label{sec:appendix}

Look at the \LaTeX~ commands in this section to see how each of the elements
of this document was produced. Also, this section serves to show
how text files (e.g., programs) can be included in a \LaTeX~ document verbatim.

\bigskip

\hrule

\small
\verbatiminput{LaTeX_Template.tex}
\normalsize

\hrule

\bigskip

% ---------------------------------------------------------------
```

```
\end{document}
% ---------------------------------------------------------------
```

## A.2 MATLAB for deconstructing the Simplex Algorithm




- `pivot_setup.m` : Script sets up all of the data structures and executes the script `pivot_input.m` (or a user-created `.m` file — file name input at `MATLAB` command prompt) to read in the input. Must execute `pivot_algebra.m` after setup.
- `pivot_input.m` : Script holds the *default* input which (or alternative user input file) is executed by the script `pivot_setup.m`. Do not run `pivot_input.m` from a `MATLAB` command prompt.
- `pivot_algebra.m` : Script to carry out the linear algebra after setup and after a swap.
- `pivot_direction.m` : Function `pivot_direction(j)` displays the basic direction associated with the current basis and increasing the $j$-th non-basic variable (up from zero). Should only be used after the script `pivot_algebra.m` is executed.
- `pivot_ratios.m` : Used to determine a leaving variable to maintain feasibility when the $j$-th non-basic variable is increased (from zero). Function `pivot_ratios(j)` calculates the ratios associated with the current basis and increasing the $j$-th non-basic variable (up from zero). Should only be used after the script `pivot_algebra.m` is executed.
- `pivot_swap.m`: Function `pivot_swap(j,i)` exchanges the $j$-th non-basic variable with the $i$-th basic variable. Must execute the script `pivot_algebra.m` after a swap.

- `pivot_plot.m` : If $n - m = 2$ , script produces a plot in the space of the (two) non-basic variables. Feasible basic solutions map to green points and infeasible ones map to red points. If the feasible region is bounded and there are at least three extreme points, then the feasible region is shaded with cyan. If there are at least three extreme points and there is no basic feasible ray relative to the current basis, then the convex hull of the basic feasible solutions is still shaded with cyan.

---

```
% pivot_setup.m // Jon Lee
% set up a pivoting example

clear all;

global A b c m n beta eta ratios

try
    reply = input('NAME of NAME.m file holding input? [pivot_input]: ', 's');
    if isempty(reply)
       reply = 'pivot_input';
       end
catch
    reply = 'pivot_input'; % need catch for execution on MathWorks Cloud
end
eval(reply);

if (size(b) ~= m)
    display('size(b) does not match number of rows of A')
    return
end
if (size(c) ~= n)
    display('size(c) does not match number of columns of A')
    return
end
if(size(setdiff(beta,1:n)) > 0)
    display('beta has elements not in 1,2,...,n')
    return
end
if(size(setdiff(eta,1:n)) > 0)
    display('eta has elements not in 1,2,...,n')
    return
end
if (size(beta) ~= m)
    display('size(beta) does not match number of rows of A')
    return
end
if (size(eta) ~= n-m)
    display('size(eta) does not match number of cols minus number of rows of A')
    return
end

display('Available quantites: A, b, c, m, n, beta, eta');
display('Seems like a good time to run pivot_algebra');
```

```
% pivot_input.m // Jon Lee
% data for pivoting example

A = [1 2 1 0 0 0; 3 1 0 1 0 0; 1.5 1.5 0 0 1 0; 0 1 0 0 0 1];
A  =sym(A);
c = [6 7 -2 0 4 4.5]';
c = sym(c);
b = [7 9 6 3.3]';
b = sym(b);
beta = [1,2,4,6];
[m,n] = size(A);
eta = setdiff(1:n,beta); % lazy eta initialization
```

```
% pivot_algebra.m // Jon Lee
% do the linear algebra

global A_beta Abar_eta c_beta ybar xbar_beta xbar c_eta cbar_eta

A_beta = A(:,beta);
% if (rank(A_beta,0.0001) < size(A_beta,1))
%     display('error: A_beta not invertible')
%     return;
% end;
A_eta = A(:,eta);
c_beta = c(beta);
c_eta = c(eta);
ybar = linsolve(A_beta',c_beta);
xbar = sym(zeros(n,1));
xbar_beta = linsolve(A_beta,b);
xbar(beta) = xbar_beta;
xbar_eta = sym(zeros(n-m,1));
xbar(eta) =  xbar_eta;

Abar_eta = linsolve(A_beta,A_eta);

% if (norm(A * xbar - A_beta * xbar_beta + A_eta * xbar_eta) > 0.00001)
%     display('error: something is very wrong')
%     return
% end

cbar_eta = (c(eta)' - ybar'*A(:,eta));
obj = c'*xbar;

display('Available display quantites:');
display('   A, b, c, m, n, beta, eta,');
display('   A_beta, A_eta, c_beta, c_eta,');
display('   xbar_beta, xbar, obj, ybar, Abar_eta, cbar_eta,');
display('   pivot_ratios(j), pivot_direction(j), pivot_plot');
display('Available functionality:');
display('   pivot_swap(j,i)');
```

```
% pivot_direction.m // Jon Lee
% view the direction zbar corresponding to increasing x_eta(j).
% syntax is pivot_direction(j)

function [zbar] = pivot_direction(j)
global beta eta m n Abar_eta
if (j > n-m)
    display('error: not so many directions')
    return
end
zbar = sym(zeros(n,1));
zbar(beta) = -Abar_eta(:,j);
zbar(eta(j))=1;
zbar;
end
```

---

```
% pivot_ratios.m // Jon Lee
% calculate ratios for determing a variable to leave the basis.
% syntax is ratios(j), where we are letting eta_j enter the basis

function ratios(j)
global xbar_beta Abar_eta m n ratios;
if (j>n-m)
    display('error: j out of range.')
else
ratios = xbar_beta ./ Abar_eta(:,j)
end
```

---

```
% pivot_swap.m // Jon Lee
% swap eta_j with beta_i
% syntax is pivot_swap(j,i)

function swap(j,i)
global beta eta m n;
if ((i>m) | (j>n-m))
    display('error: i or j out of range. swap not accepted');
else
savebetai = beta(i);
beta(i) = eta(j);
eta(j) = savebetai;
display('swap accepted --- new partition:')
display(beta);
display(eta);
display('*** MUST APPLY pivot_algebra! ***')
end
```

---

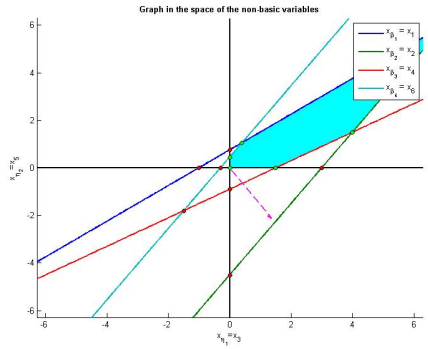
```
% pivot_plot.m // Jon Lee
% Make a 2-D plot in the space of the nonbasic variables (if n-m=2)

warning('off','all')
```

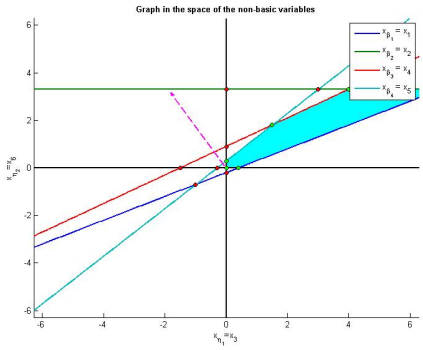
```
if (n-m ~= 2)
    display('error: cannot plot unless n-m = 2')
    return
end

clf;
hold on

syms x_eta1 x_eta2;

% a bit annoying but apparently necessary to symbolically perturb (below)
% in case a line is parallel to an axis.
x_eta1_axis = ezplot('0*x_eta1+x_eta2');
x_eta2_axis = ezplot('x_eta1+0*x_eta2');

set(x_eta1_axis,'LineStyle','-','Color',[0 0 0],'LineWidth',2);
set(x_eta2_axis,'LineStyle','-','Color',[0 0 0],'LineWidth',2);

h = Abar_eta*[x_eta1 ; x_eta2] - xbar_beta;
h = [h ; 'x_eta1' ; 'x_eta2'];

jcmap = colormap(lines);
for i = 1:m
    % a bit annoying but apparently necessary to symbolically perturb (below)
    % in case a line is parallel to an axis.
    p(i) = ezplot(strcat(char(h(i)),'+0*x_eta1+0*x_eta2'));
    set(p(i),'Color',jcmap(i,:),'LineWidth',2);
    name{i} = ['x_{\beta_{',num2str(i),'}} =  x_{' int2str(beta(i)),'}'];
end

title('Graph in the space of the nonbasic variables ($\bar{c}_\eta$ is magenta)',...
    'interpreter','Latex','FontSize',14);
legend(p, name);
xlabel(strcat('x_{\eta_1}= x_{',int2str(eta(1)),'}'));
ylabel(strcat('x_{\eta_2}= x_{',int2str(eta(2)),'}'));

if (xbar_beta >= 0)
    plot([0],[0],'Marker','o','MarkerFaceColor','green','MarkerEdgeColor',...
        'black','MarkerSize',5);
    polygonlist=[0,0];
    else
    plot([0],[0],'Marker','o','MarkerFaceColor','red','MarkerEdgeColor',...
        'black','MarkerSize',5);
    polygonlist=[ , ];
end

for i = 1:m+1
    for j = i+1:m+2
        [M, d] = equationsToMatrix([h(i) == '0*x_eta1+0*x_eta2', ...
            h(j) == '0*x_eta1+0*x_eta2'], [x_eta1, x_eta2]);
        if (rank(M) == 2)
            intpoint=linsolve(M,d);
            [warnmsg, msgid] = lastwarn;
            if ~strcmp(msgid,'symbolic:sym:mldivide:warnmsg2')
                if ( [linsolve(A_beta, b-A_eta*intpoint) ; intpoint] >=0 )
```

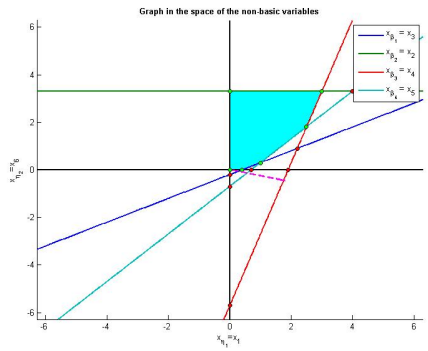
```
                    plot([intpoint(1)],[intpoint(2)],'Marker','o', ...
                        'MarkerFaceColor','green','MarkerEdgeColor','black', ...
                        'MarkerSize',5);
                    polygonlist =[polygonlist;  intpoint'];
                else
                    plot([intpoint(1)],[intpoint(2)],'Marker','o', ...
                        'MarkerFaceColor','red','MarkerEdgeColor','black', ...
                        'MarkerSize',5);
                end
            end
        end
    end
end

if ( size(polygonlist,1) > 2 && ...
        any(pivot_direction(1) < 0) && any(pivot_direction(2) < 0) )
    polyx=polygonlist(:,1);
    polyy=polygonlist(:,2);
    K = convhull(double(polyx),double(polyy));
    hull = fill(polyx(K),polyy(K),'c');
    uistack(hull, 'bottom');
end
quiver([0],[0],cbar_eta(1),cbar_eta(2),'LineWidth',2,'MarkerSize',2, ...
    'Color','magenta','LineStyle','--');
```

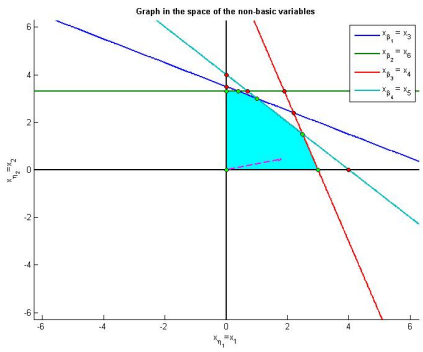
Sample sequence of commands:

```
>> pivot_setup
>> pivot_algebra
>> pivot_plot
>> pivot_ratios(2)
>> pivot_swap(2,4)
>> pivot_algebra
>> pivot_ratios(1)
>> pivot_swap(1,1)
>> pivot_algebra
>> pivot_ratios(2)
>> pivot_swap(2,2)
>> pivot_algebra
```

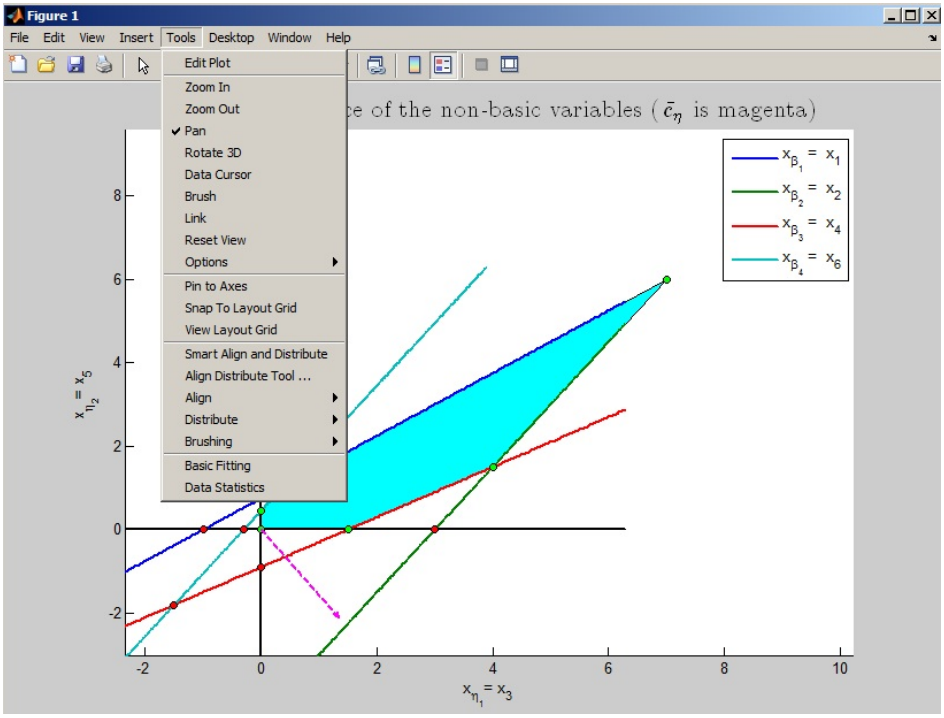


Note that the MATLAB GUI allows to pan beyond the initial display range:



## A.3 AMPL for uncapacitated facility-location problem

AMPL code for computationally comparing formulations of the uncapacitated facility-location problem:

```
# uflstrong.mod // Jon Lee
#

param M integer > 1;  # NUMBER OF FACILITIES
param N integer > 1;  # NUMBER OF CUSTOMERS

set FACILITIES := {1..M};
set CUSTOMERS := {1..N};

set LINKS := FACILITIES cross CUSTOMERS;

param facility_cost {FACILITIES} :=  round(Uniform(1,100));
param shipping_cost {LINKS} :=  round(Uniform(1,100));

var W {(i,j) in LINKS} >= 0;
var Y {i in FACILITIES} binary;

minimize Total_Cost:
   sum {(i,j) in LINKS} shipping_cost[i,j] * W[i,j]
     + sum {i in FACILITIES} facility_cost[i] * Y[i];

subject to Only_Ship_If_Open {(i,j) in LINKS}:
   W[i,j] <= Y[i];

subject to Satisfy_Demand {j in CUSTOMERS}:
   sum {(i,j) in LINKS} W[i,j] = 1;
```

```
# uflstrong.run // Jon Lee
#

reset;
option randseed 412837;
option display_1col 0;
option display_transpose -20;
option show_stats 1;
option solver cplex;
model uflstrong.mod;
data uflstrong.dat;
solve;
display W;
display Y;
display Total_Cost;
```

```
# uflstrong.dat // Jon Lee
#
```

```
param M := 10;

param N := 25;
```

---

```
# uflweak.mod // Jon Lee
#

param M integer > 1;  # NUMBER OF FACILITIES
param N integer > 1;  # NUMBER OF CUSTOMERS

set FACILITIES := {1..M};
set CUSTOMERS := {1..N};

set LINKS := FACILITIES cross CUSTOMERS;

param facility_cost {FACILITIES} :=  round(Uniform(1,100));
param shipping_cost {LINKS} :=  round(Uniform(1,100));

var W {(i,j) in LINKS} >= 0;
var Y {i in FACILITIES} binary;

minimize Total_Cost:
   sum {(i,j) in LINKS} shipping_cost[i,j] * W[i,j]
     + sum {i in FACILITIES} facility_cost[i] * Y[i];

subject to Only_Ship_If_Open {i in FACILITIES}:
   sum {(i,j) in LINKS} W[i,j] <= N * Y[i];

subject to Satisfy_Demand {j in CUSTOMERS}:
   sum {(i,j) in LINKS} W[i,j] = 1;
```

---

```
# uflweak.run // Jon Lee
#

reset;
option randseed 412837;
option display_1col 0;
option display_transpose -20;
option show_stats 1;
option solver cplex;
model uflweak.mod;
data uflweak.dat;
solve;
display W;
display Y;
display Total_Cost;
```

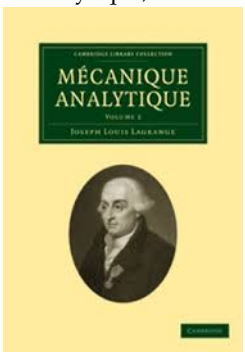
---

```
# uflweak.dat // Jon Lee
```

```
#
param M := 10;  # NUMBER OF FACILITIES
param N := 25;  # NUMBER OF CUSTOMERS
```

# End Notes

[1] "The reader will find no figures in this work. The methods which I set forth do not require either constructions or geometrical or mechanical reasonings: but only algebraic operations, subject to a regular and uniform rule of procedure." — **Joseph-Louis Lagrange**, Preface to "Mécanique Analytique," 1815.




[2] "Il est facile de voir que...", "il est facile de conclure que...", etc. — **Pierre-Simon Laplace**, frequently in "Traité de Mécanique Céleste."

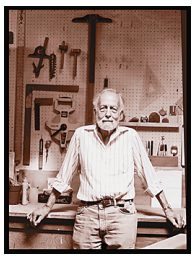

[3] "One would be able to draw thence well some corollaries that I omit for fear of boring you." — **Gabriel Cramer**, Letter to Nicolas Bernoulli, 21 May 1728. Translated from "Die Werke von Jakob Bernoulli," by R.J. Pulskamp.

[4] "Two months after I made up the example, I lost the mental picture which produced it. I really regret this, because a lot of people have asked me your question, and I can't answer." — **Alan J. Hoffman**, private communication with J. Lee, August, 1994.

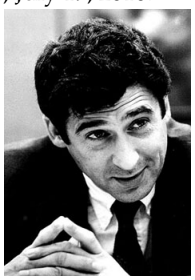
[5] "Fourier hat sich selbst vielfach um Ungleichungen bemüht, aber ohne erheblichen Erfolg." — **Gyula Farkas**, "Über die Theorie der Einfachen Ungleichungen," *Journal für die Reine und Angewandte Mathematik*, vol. 124:1–27.

[6] "The particular geometry used in my thesis was in the dimension of the columns instead of the rows. This column geometry gave me the insight that made me believe the Simplex Method would be a very efficient solution technique for solving linear programs. This I proposed in the summer of 1947 and by good luck it worked!" — **George B. Dantzig**, "Reminiscences about the origins of linear programming," *Operations Research Letters* vol. 1 (1981/82), no. 2, 43–48.

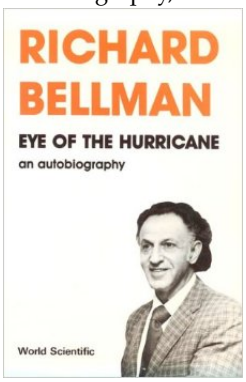
[7] "George would often call me in and talk about something on his mind. One day in around 1959, he told me about a couple of problem areas: something that Ray Fulkerson worked on, something else whose details I forget. In both cases, he was using a linear programming model and the simplex method on a problem that had a tremendous amount of data. Dantzig in one case, Fulkerson in another, had devised an ad hoc method of creating the data at the moment it was needed to fit into the problem. I reflected on this problem for quite awhile. And then it suddenly occurred to me that they were all doing the same thing! They were essentially solving a linear programming problem whose data - whose columns - being an important part of the data, were too many to write down. But you could devise a procedure for creating one when you needed it, and creating one that the simplex method would choose to work with at that moment. Call it the column-generation method. The immediate, lovely looking application was to the linear programming problem, in which you have a number of linear programming problems connected only by a small number of constraints. That fit in beautifully with the pattern. It was a way of decomposing such a problem. So we referred to it as the decomposition algorithm. And that rapidly became very famous." — **Philip Wolfe**, interviewed by Irv Lustig ∼2003.



[8] "So they have this assortment of widths and quantities, which they are somehow supposed to make out of all these ten-foot rolls. So that was called the cutting stock problem in the case of paper. So Paul [Gilmore] and I got interested in that. We struck out (failed) first on some sort of a steel cutting problem, but we seemed to have some grip on the paper thing, and we used to visit the paper mills to see what they actually did. And I can tell you, paper mills are so impressive. I mean they throw a lot of junk in at one end, like tree trunks or something that's wood, and out the other end comes – swisssssh – paper! It's one damn long machine, like a hundred yards long. They smell a lot, too. We were quite successful. They didn't have computers; believe me, no computer in the place. So we helped the salesman to sell them the first computer." — **Ralph E. Gomory**, interviewed by William Thomas, New York City, July 19, 2010.



[9] "I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word 'programming'. I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying I thought, lets kill two birds with one stone. Lets take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is its impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities." — **Richard E. Bellman**, "Eye of the Hurricane: An Autobiography," 1984.




[10] "Vielleicht noch mehr als der Berührung der Menschheit mit der Natur verdankt die Graphentheorie der Berührung der Menschen untereinander." — **Dénes König**, "Theorie Der Endlichen Und Unendlichen Graphen," 1936.

# The Afterward

# Bibliography

[1] Stephen P. Bradley, Arnoldo C. Hax, and Thomas L. Magnanti. *Applied mathematical programming*. Addison Wesley, 1977. Available at `http://web.mit.edu/15.053/www/`.

[2] Jon Lee. *A first course in combinatorial optimization*. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, 2004.

[3] Katta G. Murty. *Linear and combinatorial programming*. John Wiley & Sons Inc., New York, 1976.

# Index of definitions