
A First Course in Linear Optimization

— a dynamic book —

by
Jon Lee

Fourth Edition (Version 4.08)



REEX PRESS

Jon Lee

2013–2024



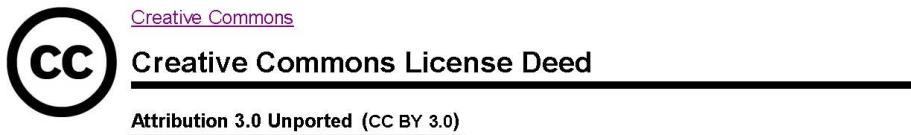
This work is licensed under the

**Creative Commons Attribution 3.0 Unported License
(CC BY 3.0) **

To view a copy of this license, visit

<http://creativecommons.org/licenses/by/3.0/>

where you will see the summary information below and can click through to the full license information.



This is a human-readable summary of the [Legal Code \(the full license\)](#).
[Disclaimer](#)

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

to make commercial use of the work



Under the following conditions:



Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

With the understanding that:

Waiver — Any of the above conditions can be [waived](#) if you get permission from the copyright holder.

Public Domain — Where the work or any of its elements is in the [public domain](#) under applicable law, that status is in no way affected by the license.

Other Rights — In no way are any of the following rights affected by the license:

- Your fair dealing or [fair use](#) rights, or other applicable copyright exceptions and limitations;
- The author's [moral](#) rights;
- Rights other persons may have either in the work itself or in how the work is used, such as [publicity](#) or privacy rights.

- **Notice** — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

Go Forward

This is a book on linear optimization, written in L^AT_EX. I use it for graduate-level courses 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 L^AT_EX source might be available) and redistributed as anyone pleases, subject to the terms of the [Creative Commons Attribution 3.0 Unported License \(CC BY 3.0\) !\[\]\(35e4f762fc1cfea5610d92e2d225d5b4_img.jpg\)](#). 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, Python/Gurobi^{*}).

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., even in describing branch-and-bound and cutting-plane algorithms), 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!

Conventional illustrations are woefully few. Though if Lagrange could not be bothered¹, who am I to aim higher? Still, I am gradually improving this aspect, and many of the algorithms and concepts are *illustrated and verified* in the modern way, with computer code.²

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 [5]. The core of the present book is a treatment of much of that material, with additional material on integer-linear optimization, much of which I originally learned from George Nemhauser and Les. But there is new material too; in particular, a "deconstructed post-modern" version of Gomory pure and mixed-integer cuts. 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)

*New in the 4th Edition! But thanks for the *very* fond memories AMPL.

Serious Acknowledgments

Throw me some serious funding for this project, and I will acknowledge you — seriously!

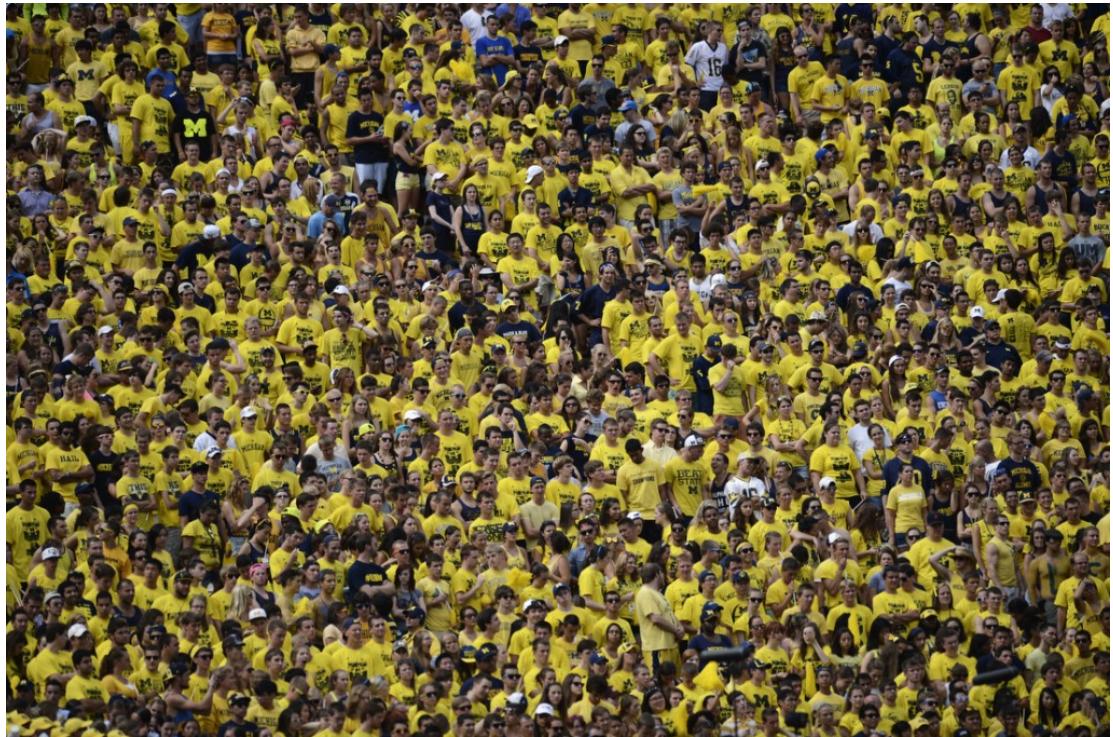
Many of the pictures in this book were found floating around on the web. I am making "*fair use*" of them as they float through this document. Of course, I gratefully acknowledge those who own them.

Hearty thanks to many students and to Professors Marina Epelman, Marcia Fampa, and Siqian Shen for pointing out typos in earlier versions.



Dedication

For students (even **Ohio** students). Not for publishers — maybe next time.



The Nitty Gritty



You can always get the released edition of this book (in .pdf format) from my [web page](#) or [github](#) and maybe the materials to produce them ([L^AT_EX](#) source, etc.) from me. I make significant use of software. Everything seems to work with:

```
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
  (via Anaconda distribution)
Jupyter Notebook server 6.5.6 (via Anaconda distribution)
Gurobi Optimizer version 11.0.2 build v11.0.2rc0 (win64 - Windows 10.0 (19045.2))
WinEdt 10.3
MiKATEX 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 [L^AT_EX](#) 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 WinEdt does endeavor to keep the packages up to date).

Contents

1	Let's Get Started	1
1.1	Linear Optimization and Standard Form	1
1.2	A Standard-Form Problem and its Dual	2
1.3	Linear-Algebra Review	3
1.4	Exercises	7
2	Modeling	11
2.1	A Production Problem	11
2.2	Norm Minimization	12
2.3	Network Flow	13
2.4	Modeling in Software	15
2.5	Exercises	18
3	Algebra Versus Geometry	21
3.1	Basic Feasible Solutions and Extreme Points	21
3.2	Basic Feasible Directions	25
3.3	Basic Feasible Rays and Extreme Rays	27
3.4	Exercises	28
4	The Simplex Algorithm	29
4.1	A Sufficient Optimality Criterion	29
4.2	The Simplex Algorithm with No Worries	32
4.3	Anticycling	36
4.4	Obtaining a Basic Feasible Solution	38
4.4.1	Ignoring degeneracy	39
4.4.2	Not ignoring degeneracy	41
4.5	The Simplex Algorithm	42
4.6	Exercises	43
5	Duality	47
5.1	The Strong Duality Theorem	48
5.2	Complementary Slackness	48
5.3	Duality for General Linear-Optimization Problems	50
5.4	Theorems of the Alternative	52
5.5	Exercises	55

6 Sensitivity Analysis	59
6.1 Right-Hand Side Changes	59
6.1.1 Local analysis	60
6.1.2 Global analysis	60
6.1.3 A brief detour: the column geometry for the Simplex Algorithm .	62
6.2 Objective Changes	63
6.2.1 Local analysis	64
6.2.2 Global analysis	64
6.3 Exercises	65
7 Large-Scale Linear Optimization	67
7.1 Decomposition	67
7.1.1 Reformulation via the representation theorem	68
7.1.2 Solution of the Main Problem (M) via the Simplex Algorithm .	70
7.2 Lagrangian Relaxation	75
7.2.1 Lagrangian bounds	75
7.2.2 Solving the Lagrangian Dual	77
7.3 The Cutting-Stock Problem	82
7.3.1 Formulation via cutting patterns	83
7.3.2 Solution via continuous relaxation	84
7.3.3 The knapsack subproblem	85
7.3.4 Applying the Simplex Algorithm	87
7.3.5 A demonstration implementation	87
7.4 Exercises	90
8 Integer-Linear Optimization	93
8.1 Integrality for Free	93
8.1.1 Some structured models	93
8.1.2 Unimodular basis matrices and total unimodularity	96
8.1.3 Consequences of total unimodularity	99
8.2 Modeling Techniques	104
8.2.1 Disjunctions	104
8.2.2 Forcing constraints	105
8.2.3 Piecewise-linear univariate functions	107
8.3 A Prelude to Algorithms	109
8.4 Branch-and-Bound	110
8.5 Cutting Planes	114
8.5.1 Pure	114
8.5.2 Mixed	120
8.5.3 Finite termination	124
8.5.4 Branch-and-Cut	124
8.6 Exercises	124

Appendices	129
A.1 <code>LATEX template</code>	131
A.2 <code>MatrixLP.ipynb</code>	137
A.3 <code>Production.ipynb</code>	143
A.4 <code>Multi-commodityFlow.ipynb</code>	149
A.5 <code>pivot_example.ipynb</code>	159
A.6 <code>pivot_tools.ipynb</code>	171
A.7 <code>Circle.ipynb</code>	181
A.8 <code>Decomp.ipynb</code>	185
A.9 <code>SubgradProj.ipynb</code>	239
A.10 <code>CSP.ipynb</code>	247
A.11 <code>UFL.ipynb</code>	261
A.12 <code>pure_gomory_example_1.ipynb</code>	275
A.13 <code>pure_gomory_example_2.ipynb</code>	285
End Notes	311
Bibliography	317
Indexes	319
Index of definitions	319
Index of Jupyter notebooks	321

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.

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**

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

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 is 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 non-positive variable x_j with a non-negative variables x_j^- , substituting $-x_j^-$ for x_j . Additionally, 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 system of equations has 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

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

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

$$\max \quad y'b \\ y'A \leq c'. \quad (\text{D})$$

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 1.1 (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. \square

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 1×1 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

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

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 $x^1, x^2, \dots, x^p \in \mathbb{R}^n$, and scalars $\lambda_1, \lambda_2, \dots, \lambda_p$, $\sum_{i=1}^p \lambda_i x^i$ is a **linear combination** of x^1, x^2, \dots, x^p . The linear combination is **trivial** if all $\lambda_i = 0$. The vectors $x^1, x^2, \dots, x^p \in \mathbb{R}^n$ are **linearly independent** if the only representation of the zero vector in \mathbb{R}^n as a linear combination of x^1, x^2, \dots, x^p is trivial. The set of all linear combinations of x^1, x^2, \dots, x^p is the vector-space **span** of $\{x^1, x^2, \dots, x^p\}$. 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 $\{x^1, x^2, \dots, x^p\}$, then there is a subset of $\{x^1, x^2, \dots, x^p\}$ that is a basis for V . It is not hard to prove the following very useful result.

Theorem 1.2 (Greedy Basis Extension Theorem)

Let V be the vector-space span of $\{x^1, x^2, \dots, x^p\}$. Then every linearly-independent subset of $\{x^1, x^2, \dots, x^p\}$ can be extended to a basis for V using vectors from x^1, x^2, \dots, 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 $\text{n.s.}(A) := \{x \in \mathbb{R}^n : Ax = \mathbf{0}\}$.

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. add a scalar multiple of a row to another row;
2. interchange a pair of rows;
3. multiply a row by a non-zero scalar;
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*: $1, 2, \dots, n$. So we have

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

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

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

This matrix is really equivalent to A , because we regard its columns as labeled by $\sigma_1, \sigma_2, \dots, \sigma_n$ rather than $1, 2, \dots, n$. 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} .

The **Sherman-Morrison formula** is a useful way to relate the inverse of a matrix to the inverse of a rank-1 change to the matrix:

$$(B + uv')^{-1} = B^{-1} - \frac{B^{-1}uv'B^{-1}}{1 + v'B^{-1}u},$$

where the $r \times r$ matrix B is invertible, $u, v \in \mathbb{R}^r$, and it must be assumed that $1 + v'B^{-1}u \neq 0$ for otherwise $B + uv'$ would not be invertible.

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**.³

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 $\det(B)$.

The determinant is well behaved under elementary row operations. Specifically, the first elementary row operation (i.e., 1. add a scalar multiple of a row to another row) does not alter the determinant of a square matrix. The second elementary row operation (i.e., 2. interchange a pair of rows) only changes the sign of the determinant; likewise for interchanging columns. For the third elementary row operation, multiply a row by a non-zero scalar, the determinant is multiplied by the scalar.

Considering these facts further, we can calculate the determinant $\det(B)$ efficiently. 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:

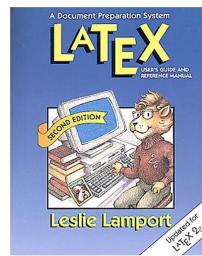
$$\bar{x}_j = \frac{\det(B(j))}{\det(B)}, \text{ for } j = 1, 2, \dots, r,$$

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.⁴

1.4 Exercises

Exercise 1.0 (Learn L^AT_EX)

Learn to use L^AT_EX for writing *all* of your homework solutions. Personally, I use MiK^TE_X, which is an implementation of L^AT_EX for Windows. Specifically, within MiK^TE_X, I am using pdfL^AT_EX (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 L^AT_EX friendly. A good book on L^AT_EX is



In Appendix 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; $n + 1$ nonnegative variable)

1. 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 $m + 1$ inequalities; rather than the more obvious $2m$ inequalities: $Ax \leq b$, $Ax \geq b$).
2. For a linear-optimization problem, devise a method for replacing n unrestricted variables with only $n + 1$ nonnegative variables (rather than a total of $2n$ nonnegative variables, as is suggested in the method in Section 1.1).

Exercise 1.5 (Weak duality for another form)

Give and prove a Weak Duality Theorem for

$$\begin{aligned} \max \quad & c'x \\ Ax \leq & b; \\ x \geq & \mathbf{0}. \end{aligned} \tag{P'}$$

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

$$\begin{aligned} \min \quad & c'x + f'w \\ Ax + Bw \leq & b; \\ Dx = & g; \\ x \geq & \mathbf{0}; \quad w \leq \mathbf{0}. \end{aligned} \tag{P'}$$

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

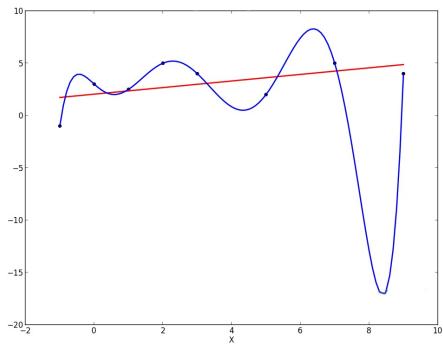
Exercise 1.7 (Weak duality for a complicated matrix form — with Python/Gurobi)

Python is an interpreted, general-purpose programming language. Anaconda is a free and open-source distribution of Python (and R). Via the Anaconda distribution, one also gets Jupyter Notebook, which is a convenient way to experiment with Python. Gurobi is a state-of-the art commercial linear and integer linear optimization software, with free temporary licensing for students. Gurobi can be easily accessed with `gurobipy`, a Python module. The Jupyter notebook `MatrixLP.ipynb` (see Appendix A.2) sets up

and solves an instance of (P') from Exercise 1.6. *Read the code and run it, 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 Python/Gurobi to solve that problem and its dual. Make sure that you get the same optimal value for all of these problems.

Chapter 2

Modeling



Our goals in this chapter are as follows:

- Learn some basic linear-optimization modeling techniques.
- Learn how to use Python as an LP modeling language in connection with Gurobi as an LP solver.

2.1 A Production Problem



We suppose that a company has m resources, available in quantities $b_i, i = 1, 2, \dots, m$, and n production activities, with per-unit profits $c_j, j = 1, 2, \dots, n$. 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

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

where $b := (b_1, b_2, \dots, b_m)'$, $c := (c_1, c_2, \dots, c_n)'$, $A \in \mathbb{R}^{m \times n}$ 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 $i = 1, 2, \dots, m$. 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 these prices. That is, we require that $\sum_{i=1}^m y_i a_{ij} \geq c_j$, for $j = 1, 2, \dots, n$. 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 $i = 1, 2, \dots, m$. All in all, we have the linear-optimization problem

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

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 **∞ -norm** (or **max-norm**) of x is defined as

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

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

$$\begin{aligned} \min \quad & t \\ t - x_i & \geq 0, \quad i = 1, 2, \dots, n; \\ t + x_i & \geq 0, \quad i = 1, 2, \dots, n; \\ Ax & = b, \end{aligned}$$

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

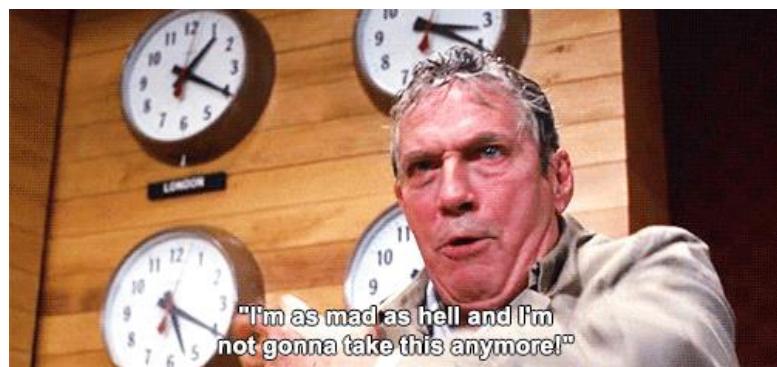
$$\|x\|_1 := \sum_{j=1}^n |x_j|.$$

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:

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

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 $j = 1, 2, \dots, n$ (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 $K \geq 1$ commodities as being allowed to "flow" along each arc, from its tail to its head. Indeed, we have "flow" variables

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

for $e \in \mathcal{A}$, and $k = 1, 2, \dots, K$. Formally, a **flow** \hat{x} on G is simply an assignment of *any* real numbers \hat{x}_e^k to the variables x_e^k , for $e \in \mathcal{A}$, and $k = 1, 2, \dots, K$. We assume that the total flow on arc e should not exceed

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

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

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

for $e \in \mathcal{A}$, and $k = 1, 2, \dots, K$. The (total) **cost** of the flow \hat{x} is defined to be

$$\sum_{k=1}^K \sum_{e \in \mathcal{A}} c_e^k x_e^k.$$

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

$$b_v^k := \text{the net supply of commodity } k \text{ at node } v,$$

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 $v \in \mathcal{N}$, and all commodities $k = 1, 2, \dots, K$.

The **multi-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:

$$\begin{aligned} \min & \quad \sum_{k=1}^K \sum_{e \in \mathcal{A}} c_e^k x_e^k \\ & \sum_{\substack{e \in \mathcal{A} : \\ t(e)=v}} x_e^k - \sum_{\substack{e \in \mathcal{A} : \\ h(e)=v}} x_e^k = b_v^k, \quad \forall v \in \mathcal{N}, k = 1, 2, \dots, K; \\ & \sum_{k=1}^K x_e^k \leq u_e, \quad \forall e \in \mathcal{A}; \\ & x_e^k \geq 0, \quad \forall e \in \mathcal{A}, k = 1, 2, \dots, K. \end{aligned}$$

2.4 Modeling in Software



Optimization modeling languages facilitate rapid development of mathematical optimization models, instantiation with data, and the subsequent solution by solvers. Well-known examples of optimization modeling languages are [AMPL](#) and [GAMS](#). Another is [Pyomo](#), which is a Python package. All of these are means to set up structured LP models, instantiate them with data, pass to an LP solver, and recover the solutions (with the opportunity to manipulate them) back in their environments. All have the ability to iterate, solving sequences of LPs, dynamically setting them up at each iteration. In fact, we will not use any of them here, but will instead work directly in Python, making direct calls to [Gurobi](#), a state-of-the-art LP solver*. A strong advantage of working in Python is that it is a well-supported programming language with lots of useful add-on packages.

In Exercise 1.7, we saw how to set up "matrix-style" optimization models, instantiate them with data, and solve them. For models that relate to applications, it is often more natural and convenient to specify models in a way that does not obscure the problem being solved and is close to the way that we would naturally write the model mathematically. We will do this in Python, making direct calls to Gurobi. As a first step in this direction, we consider the Production problem of Section 2.1. For this problem, we specify the model in Python/Gurobi as follows.

First, it is convenient to number the resources as $M := \{0, 1, \dots, m - 1\}$ and the variables as $N := \{0, 1, \dots, n - 1\}$. We do this in Python via:

```
M=list(range(0,m))
N=list(range(0,n))
```

We instantiate a Gurobi Model object via

```
model = gp.Model()
```

Note that `model` is the name that we have given Gurobi Model object in Python.

We create (continuous non-negative) variables $x[j]$, for $j \in N$, attached to `model`, via:

```
x = model.addMVar(n)
```

*Another is [CPLEX](#)

These variable names $x[j]$ are accessible to us in Python and are not used internally by Gurobi. Note that the Gurobi is that variables are continuous and non-negative (some other types are available, but those types must be explicitly declared).

We define and attach our objective function, `revenueobjective`, to model via:

```
revenueobjective = model.setObjective(sum(c[j]*x[j] for j in N), GRB.MAXIMIZE)
```

This objective name is accessible to us in Python and is not used internally by Gurobi.

Finally, we define our resource constraints and attach them to `model` via:

```
for i in M:
    model.addConstr(sum(A[i,j]*x[j] for j in N) <= b[i], name='r'+str(i))
```

Note that we have created names, r_j , for $j \in N$, for the constraints *inside* Gurobi. This enables us to easily retrieve constraint "attributes" from Gurobi. A Jupyter notebook giving the full Python/Gurobi implementation is [Production.ipynb](#) (see Appendix A.3).

Next, we consider the Network Flow problem of Section 2.3. The model is specified as:

```
x = model.addVars(ArcsCrossCommods)
model.setObjective(sum(sum(CapacityCosts[i,j][k]*x[(i,j),k] for (i,j) in Arcs)
    for k in Commoditys), GRB.MINIMIZE)
model.addConstrs(sum(x[(i,j),k] for k in Commoditys) <= CapacityCosts[i,j][0]
    for (i,j) in Arcs)
model.addConstrs(
    (sum(x[(i, j),k] for j in Nodes if (i, j) in Arcs) - sum(x[(j, i),k]
        for j in Nodes if (j,i) in Arcs)
    == Supplies[i][k-1] for i in Nodes for k in Commoditys))
```

A Python/Gurobi implementation is in the Jupyter notebook [Multi-commodityFlow.ipynb](#) (see Appendix A.4).

Example 2.1

Figure 2.1 depicts an 8-node network for a $K = 2$ commodity example. Each arc e is labeled $[u_e, c_e^1, c_e^2]$. Figures 2.2 and 2.3 depict the node supply data and the optimal solutions. Figures 2.2 corresponds to commodity 1 and Figure 2.3 corresponds to commodity 2. Node v is labeled $v : b_v^k$ and arc e is labeled with the optimal value of x_e^k .

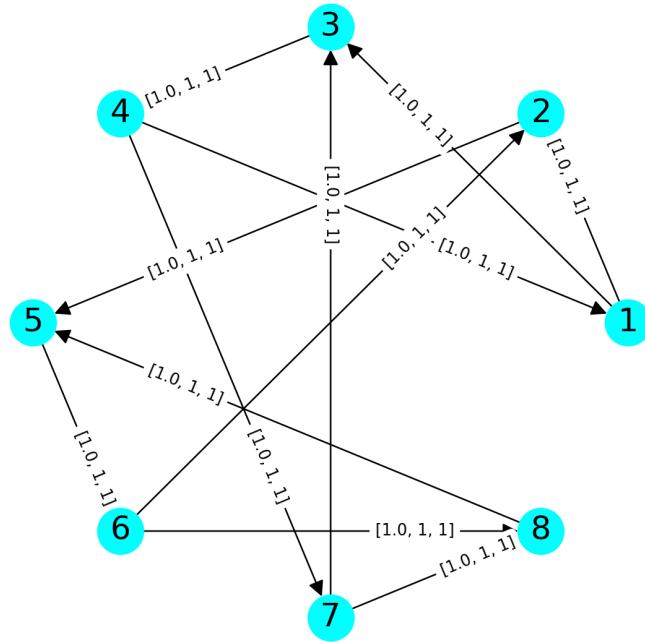


Figure 2.1: Arc data

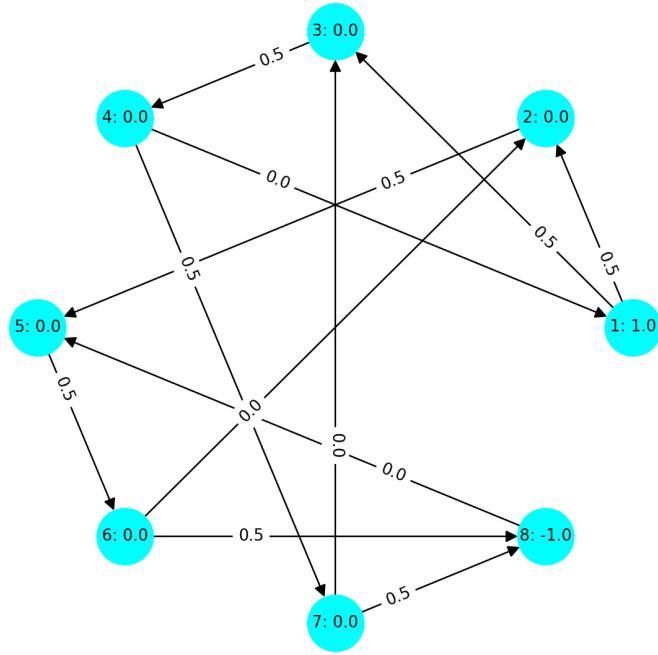


Figure 2.2: Commodity 1: supplies and flows

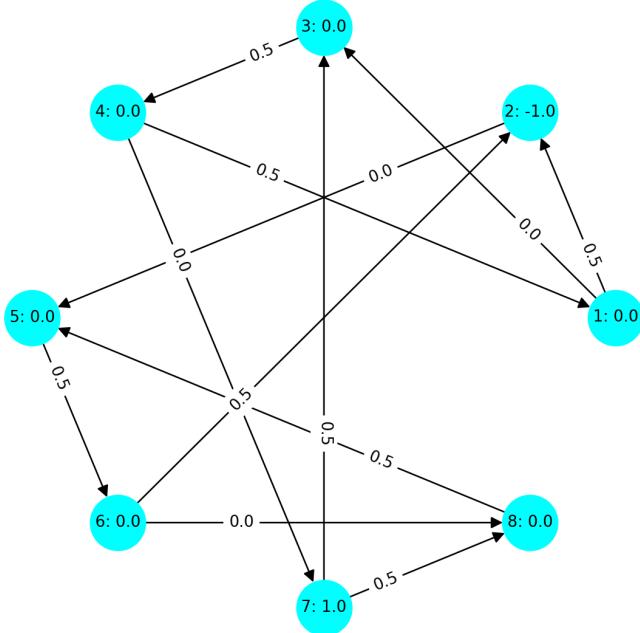


Figure 2.3: Commodity 2: supplies and flows

2.5 Exercises

Exercise 2.1 (Dual in Python/Gurobi)

Without changing the data specification in [Production.ipynb](#) (see Appendix A.3), use Python/Gurobi to solve the *dual* of the Production Problem example, as described in Section 2.1. You will need to modify the model in [Production.ipynb](#) appropriately.

Exercise 2.2 (Sparse solution for linear equations)

In some application areas, it is interesting to find a "sparse solution" — that is, one with few non-zeros — to a system of equations $Ax = b$, on say the domain $-1 \leq x_j \leq +1$, for $j = 1, 2, \dots, n$.

It is empirically well known that a 1-norm minimizing solution is a good heuristic for finding a sparse solution. The moral justification of this is as follows. We define the function indicator function $I_{\neq 0} : \mathbb{R} \mapsto \mathbb{R}$ by

$$I_{\neq 0}(w) := \begin{cases} 1, & w \neq 0; \\ 0, & w = 0. \end{cases}$$

It is easy to see (make a graph) that $f(w) := |w|$ is the "best convex function*" under-estimator of $I_{\neq 0}$ on the domain $[-1, 1]$. So we can hope that minimizing $\sum_{j=1}^n |x_j|$ comes close to minimizing $\sum_{j=1}^n I_{\neq 0}(x_j)$.

Using Python/Gurobi try this idea out on several large examples, using 1-norm minimization as a heuristic for finding a sparse solution.

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. Next, choose a random $\tilde{z} \in \mathbb{R}^{\frac{m}{2}}$ satisfying $-1 \leq \tilde{z}_j \leq +1$, for $j = 1, 2, \dots, m/2$, and $\tilde{z}_j = 0$ for $j = m/2 + 1, \dots, n$. Now let $b := A\tilde{z}$. In this way, you will know that there is a solution (i.e., \tilde{z}) with only $m/2$ non-zeros (which is already pretty sparse).

*See page 61 for the definition of a convex function

Your 1-norm minimizing solution might in fact recover this solution (⊖), or it may be sparser (⊖⊖), or perhaps less sparse (⊖).

Exercise 2.3 (Bloody network)

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 allocates 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 [Multi-commodityFlow.ipynb](#) (see Appendix A.4) with a single commodity only (that is, $K := 1$), 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 $1, 2, \dots, m$ with availabilities b_i , measured in kilograms, and per-kilogram costs c_i . We are given a set of products $1, 2, \dots, n$ with minimum production requirements d_j , measured in kilograms, and per-kilogram revenues e_j . It is required that product j have at least a *fraction* (by weight) of l_{ij} of ingredient i and at most a *fraction* (by weight) 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 Python/Gurobi, 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 $1, 2, \dots, n$ 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 $d_0 := 0$. Any number of tasks can be carried out simultaneously, except that there are precedences between tasks. Specifically, Ψ_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 Python/Gurobi, 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."

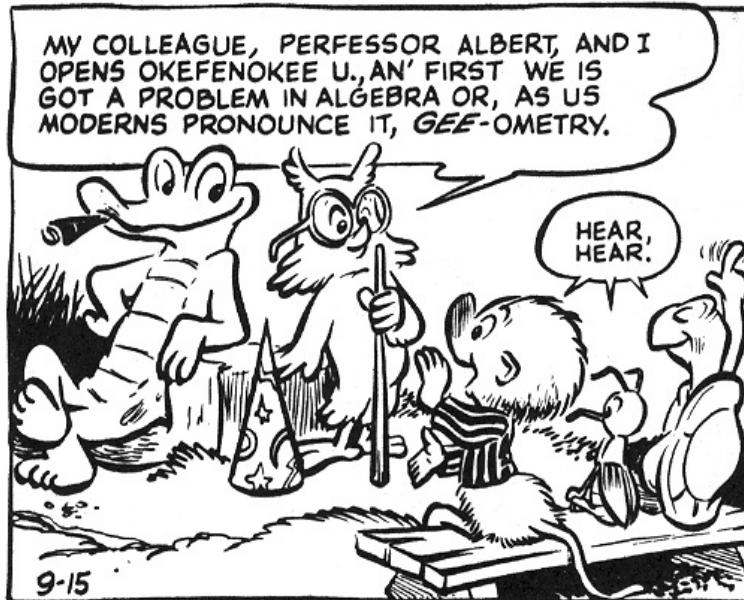
A company wants to maximize their cash holdings at the end of T time periods. They have an external inflow of p_t dollars at the *start* of time period t , for $t = 1, 2, \dots, T$. At the start of each time period, available cash can be allocated to any of K different investment vehicles (in any available 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 $v_{t,t}^k, v_{t,t+1}^k, \dots, v_{t,T}^k$, per dollar invested. It should be assumed that money obtained from cashing out the investment is incorporated into these parameters. For example, $(v_{4,4}^9, v_{4,5}^9, v_{4,6}^9, v_{4,7}^9, v_{4,8}^9, v_{4,9}^9, v_{4,10}^9, v_{4,11}^9, v_{4,12}^9) = (0.1, 0.1, 0.1, 1.1, 0, 0, 0, 0, 0)$ can be interpreted as 1 dollar invested in investment vehicle #9 at the start of time period 4 yields 0.1 dollars of income for times periods 4–7, and with the original dollar returned in time period 7, and no returns at all in the remaining time periods 8–12.

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 in any time period earns interest of q percent.

Formulate the problem, mathematically, as a linear-optimization problem. Then, model the problem with Python/Gurobi, 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

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

3.1 Basic Feasible Solutions and Extreme Points

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

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

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

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

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_β . Of course this makes sense, because the columns of A_β 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 $\bar{x}_\beta = A_\beta^{-1}b \geq \mathbf{0}$. Additionally, for a basic partition β, η , we call β a **feasible basis** if the associated basic solution is feasible.

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

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

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

$$\begin{aligned}(A_\beta^{-1}A_\eta)x_\eta &\leq A_\beta^{-1}b; \\ x_\eta &\geq \mathbf{0}.\end{aligned}$$

Notice how we can view the x_β variables as slack variables.

In the following example, because it is convenient in Python, we use "zero indexing". In particular, we use indices $\{0, 1, \dots, n-1\}$ for the variables x_j and the columns A_j , and for the basic partition we label $\beta := (\beta_0, \beta_1, \dots, \beta_{m-1})$ and $\eta := (\eta_0, \eta_1, \dots, \eta_{n-m-1})$.

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:

$$\begin{aligned}A &:= \begin{pmatrix} 1 & 2 & 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 & 0 \\ 3/2 & 3/2 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}, \\ b &:= (7, 9, 6, 33/10)', \\ \beta &:= (\beta_0, \beta_1, \beta_2, \beta_3) = (0, 1, 3, 5), \\ \eta &:= (\eta_0, \eta_1) = (2, 4).\end{aligned}$$

Then

$$\begin{aligned} A_\beta &= [A_{\beta_0}, A_{\beta_1}, A_{\beta_2}, A_{\beta_3}] = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 3 & 1 & 1 & 0 \\ 3/2 & 3/2 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \\ A_\eta &= [A_{\eta_0}, A_{\eta_1}] = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \\ x_\beta &= (x_0, x_1, x_3, x_5)' \\ x_\eta &:= (x_2, x_4)' . \end{aligned}$$

We can calculate

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

We have plotted this in Figure 3.1. The plot has $x_{\eta_0} = x_2$ as the abscissa, and $x_{\eta_1} = x_4$ as the ordinate. In the plot, besides the non-negativity of the variables x_2 and x_4 , the four inequalities of $(A_\beta^{-1} A_\eta) x_\eta \leq A_\beta^{-1} b$ are labeled with their slack variables — these are the *basic variables* x_0, x_1, x_3, x_5 . The correct matching of the basic variables to the inequalities of $(A_\beta^{-1} A_\eta) x_\eta \leq A_\beta^{-1} b$ is simply achieved by seeing that the i -th inequality has slack variable x_{β_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

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

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

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

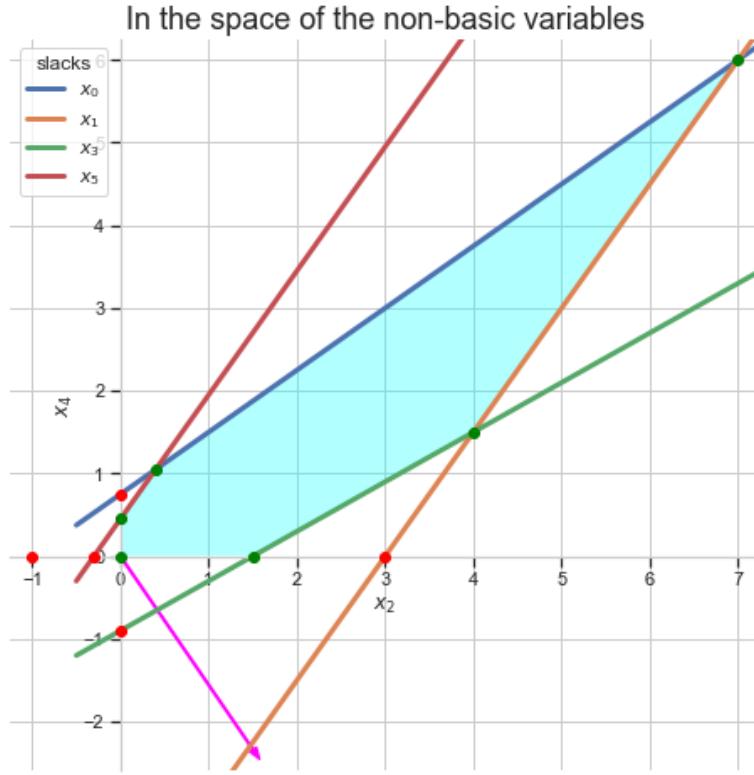


Figure 3.1: Feasible region projected into the space of non-basic variables

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 $\mathbf{0} = \bar{x}_\eta = \lambda x_\eta^1 + (1 - \lambda)x_\eta^2$ and $0 < \lambda < 1$ implies that $x_\eta^1 = x_\eta^2 = \mathbf{0}$. But then $A_\beta x_\beta^i = b$ implies that $x_\beta^i = A_\beta^{-1}b = \bar{x}_\beta$, 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 . \square

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

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

That is, ρ is the list of indices of the positive components of \hat{x} . Also, we let

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

That is, ζ is the list of indices of the zero components of \hat{x} . Together, ρ and ζ partition $\{1, 2, \dots, n\}$.

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

The first thing that we will establish is that the columns of A_ρ 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_ρ 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 ϵ chosen to be sufficiently small so that x^1 and x^2 are non-negative. Because z is only non-zero on the ρ coordinates (where \hat{x} is positive), we can choose an appropriate ϵ . Notice that $x^1 \neq x^2$, because z_ρ 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_ρ 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 (via Theorem 1.2) $m - |\rho|$ columns of A_ζ to append to A_ρ in such a way as to form a matrix A_β having m linearly-independent columns — we note that such a choice is not necessarily unique. As usual, we let η be a list of the $n - m$ indices not in β . 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}_η is a subvector of $\hat{x}_\zeta = \mathbf{0}$. Therefore, $\hat{x} = \bar{x}$. That is, \hat{x} is a basic solution of (P). \square



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 $\hat{z} \in \mathbb{R}^n$, $\hat{z} \neq \mathbf{0}$, 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 β, η and a choice of non-basic index η_j via

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

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

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

So

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

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_{η_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 β, η , 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 η_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 β, η . 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 $\epsilon \in \mathbb{R}$.

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

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

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

$$\bar{x}_{\beta_i} - \epsilon \bar{a}_{i,\eta_j} \geq 0,$$

for $i = 1, 2, \dots, m$. If $\bar{a}_{i,\eta_j} \leq 0$, for some i , then this imposes no restriction at all on ϵ . 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

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

Equivalently, we simply need that

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

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 β, η . Consider choosing a non-basic index η_j . Then the associated basic direction \bar{z} is a feasible direction relative to \bar{x} if and only if

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

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-form 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 β, η and a choice of non-basic index η_j via

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

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 β, η 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)

Using the Jupyter notebook `pivot_tools.ipynb` (see Appendix A.6), make a small example, say with six variables and four equations, to fully illustrate the concepts in this chapter. The Jupyter notebook `pivot_example.ipynb` (see Appendix A.5) shows how to start to work with `pivot_tools.ipynb`.

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 β, η 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 β , a dual solution

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

of

$$\max_{y' A} \begin{array}{l} y'b \\ y' A \end{array} \leq c'. \quad (\text{D})$$

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 β are "active".

Let us assume that \bar{y} is feasible for (D). Now, let β_ℓ be a basic index, and let $\bar{w} := H_\ell$ be row ℓ of $H := A_\beta^{-1}$. Consider $\tilde{y} := \bar{y} - \lambda \bar{w}'$, and explain (with algebraic justification) what is happening to the "activity" of each constraint, indexed by β_i ($i = 1, 2, \dots, m$) and η_j ($j = 1, 2, \dots, n-m$) of (D), as λ increases up from zero. HINT: Think about the cases of (i) β_ℓ , (ii) $\beta_i, i \neq \ell$, and (iii) η_j .

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

The **dual solution** of (D) associated with basis β is

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

Lemma 4.1

If β is a basis, then the primal basic solution \bar{x} (feasible or not) and the dual solution \bar{y} (feasible or not) associated with β 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 $\bar{y}' b = (c'_\beta A_\beta^{-1}) b = c'_\beta A_\beta^{-1} b$. \square

We call β a **dual-feasible basis** if the dual solution associated with the basis β is feasible. The vector of **reduced costs** associated with basis β is

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

Lemma 4.2

The dual solution of (D) associated with basis β 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

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

So we have

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

or, equivalently,

$$\bar{y}' A \leq c'.$$

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

Theorem 4.3 (Weak Optimal Basis Theorem)

If β is a feasible basis and a dual-feasible basis, then the primal solution \bar{x} and the dual solution \bar{y} associated with β 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 β . If these solutions \bar{x} and \bar{y} are feasible for (P) and (D), respectively, then by weak duality these solutions are optimal. \square

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

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

Next, multiplying the equations on the left by A_β^{-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_β 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_η . 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 4.4

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

$$\begin{array}{ll} \min & \bar{c}'_\eta x_\eta \\ \text{s.t.} & \bar{A}_\eta x_\eta \leq \bar{x}_\beta \\ & x_\eta \geq \mathbf{0}. \end{array}$$

This plot demonstrates the optimality of $\beta := (2, 5, 3, 4)$ ($\eta := (0, 1)$). 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 $\bar{c}_\eta \geq 0$; we can also think of this as \bar{c}_η having a non-negative dot product with each of the standard unit vectors, hence neither direction is improving.

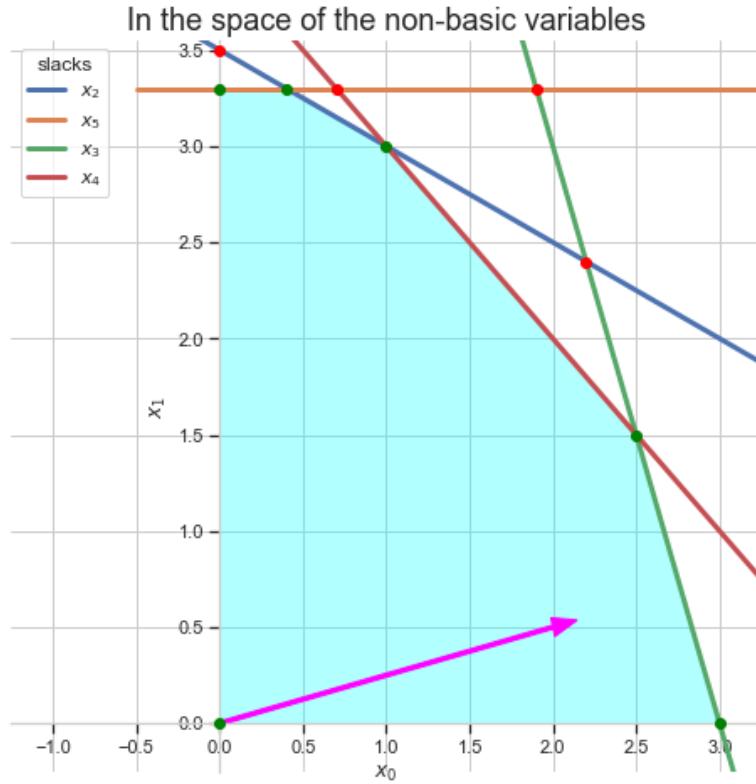


Figure 4.1: Sufficient optimality criterion

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}_η ; namely,

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

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

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

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

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}_{η_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_{η_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

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

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

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

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

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

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

Non-degeneracy. There is a significant issue in even carrying out one iteration of this algorithm. If $\bar{x}_{\beta_i} = 0$ 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. The problem (P) satisfies the **non-degeneracy hypothesis** if for every feasible basis β , we have $\bar{x}_{\beta_i} > 0$ for $i = 1, 2, \dots, m$. 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_{η_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

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

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_{η_j} .

Let

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

and

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

Lemma 4.5

$A_{\tilde{\beta}}$ is invertible.

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

$$\begin{aligned} A_{\beta}^{-1} A_{\tilde{\beta}} &= A_{\beta}^{-1} [A_{\beta_1}, A_{\beta_2}, \dots, A_{\beta_{i^*-1}}, A_{\eta_j}, A_{\beta_{i^*+1}}, \dots, A_{\beta_m}] \\ &= [\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{i^*-1}, \bar{A}_{\eta_j}, \mathbf{e}_{i^*+1}, \dots, \mathbf{e}_m]. \end{aligned}$$

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

Lemma 4.6

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

Proof. $(\bar{x} + \bar{\lambda}\bar{z})_j = 0$, for $j \in \tilde{\eta}$. 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 4.7

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

Passing from the partition β, η 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:

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

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

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

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

replace β with

$$(\beta_1, \beta_2, \dots, \beta_{i^*-1}, \underline{\underline{\beta_{i^*}}}, \beta_{i^*+1}, \dots, \beta_m)$$

and η with

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

5. GOTO 1.

Example 4.8

This is a continuation of Example 3.1 / Example 4.4. In Figure 4.2, we have depicted the solution one step after the initial solution depicted in Figure 3.1. The result of the next pivot is depicted in Figure 4.3. Finally, in one more pivot, we reach the optimum depicted in Figure 4.1.

Theorem 4.9

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. \square


Remark 4.10

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).

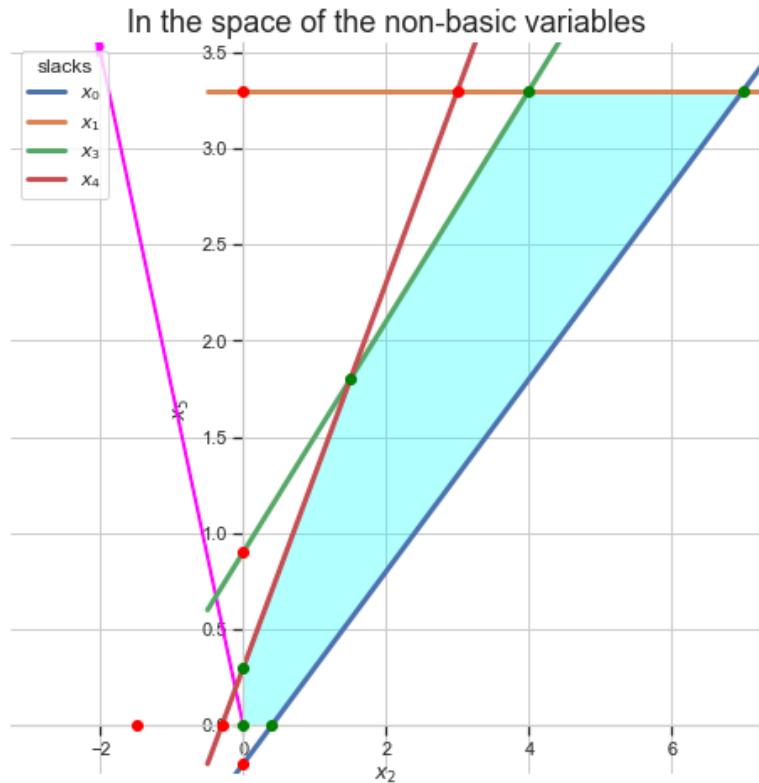


Figure 4.2: After one pivot

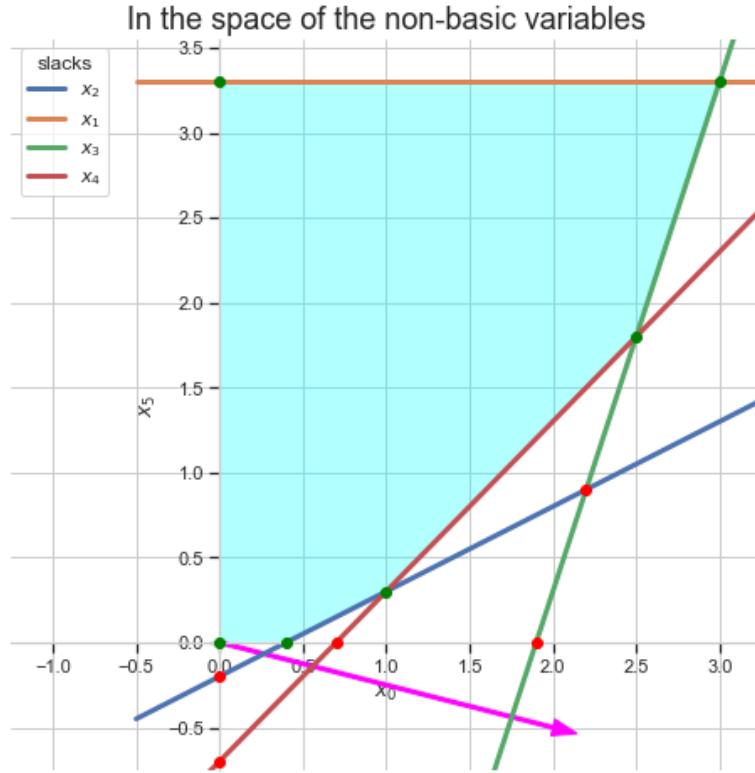


Figure 4.3: After two pivots

4.3 Anticycling



Without the non-degeneracy hypothesis, the Worry-Free Simplex Algorithm could go through a sequence of bases, with all basic solutions in the sequence having the same objective value, returning to the first basis in the sequence; see Exercise 4.4. This phenomenon is known as **cycling**. With the specter of cycling, the Worry-Free Simplex Algorithm could fail to terminate with some data.

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

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

where

- $b_\epsilon(B) := b + B\vec{\epsilon}$ and $\vec{\epsilon} := (\epsilon, \epsilon^2, \dots, \epsilon^m)'$ (these are *exponents* not superscripts);
- the scalar ϵ is an arbitrarily small indeterminant; ϵ 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 ϵ);
- the variables x_j are polynomials in ϵ with real coefficients;
- the ordering of polynomials used to interpret the inequality \geq_ϵ 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 ϵ , 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 ϵ 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_ϵ " (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 β , the associated basic solution \bar{x}^ϵ has $\bar{x}_\beta^\epsilon := A_\beta^{-1}(b + B\vec{\epsilon}) = \bar{x}_\beta + A_\beta^{-1}B\vec{\epsilon}$. It is evident that $\bar{x}_{\beta_i}^\epsilon$ is a polynomial, of degree at most m , in ϵ , for each $i = 1, \dots, m$. Because the ordering $<_\epsilon$ refines the ordering $<$, we have that $\bar{x}_\beta^\epsilon \geq_\epsilon \mathbf{0}_\epsilon$ implies that $\bar{x}_\beta \geq \mathbf{0}$. That is, any basic feasible partition for $(P_\epsilon(B))$ is a basic feasible partition for (P) . This implies that applying the Worry-Free Simplex Algorithm to $(P_\epsilon(B))$, using the ratio test to enforce feasibility of \bar{x} in $(P_\epsilon(B))$ at each iteration, implies that each associated \bar{x}_β is feasible for (P) . That is, the choice of a leaving variable dictated by the ratio test when we work with $(P_\epsilon(B))$ is valid if we instead do the ratio test working with (P) .

The objective value associated with \bar{x}^ϵ 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 ϵ . Therefore, we can order basic solutions for $(P_\epsilon(B))$ 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}^ϵ is optimal for $(P_\epsilon(B))$, then the \bar{x} associated with the same basis is optimal for (P) .

Lemma 4.11

The ϵ -perturbed problem $(P_\epsilon(B))$ satisfies the non-degeneracy hypothesis.

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

Theorem 4.12

Let β^0 be a basis that is feasible for (P) . Then the Worry-Free Simplex Algorithm applied to $(P_\epsilon(A_{\beta^0}))$, starting from the basis β^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 $\bar{x}_{\beta^0}^\epsilon := A_{\beta^0}^{-1}(b + A_{\beta^0}\vec{\epsilon}) = A_{\beta^0}^{-1}b + \vec{\epsilon}$. Now because \bar{x} is feasible for (P) , we have $A_{\beta^0}^{-1}b \geq \mathbf{0}$. Then, the ordering $<_\epsilon$ implies that $\bar{x}_{\beta^0}^\epsilon = A_{\beta^0}^{-1}b + \vec{\epsilon} \geq_\epsilon \mathbf{0}$. Therefore, the basis β^0 is feasible for $(P_\epsilon(A_{\beta^0}))$, and the Worry-Free Simplex Algorithm can indeed be started for $(P_\epsilon(A_{\beta^0}))$ on β^0 .

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

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

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



(Pivot.mp4)

Figure 4.4: 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](#).

4.4.1 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

$$\tilde{\beta} = (\tilde{\beta}_1, \tilde{\beta}_2, \dots, \tilde{\beta}_m),$$

while our non-basic indices are extended to

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

This variable x_{n+1} is termed an **artificial variable**. The column for the constraint matrix associated with x_{n+1} 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 the artificial variable x_{n+1} . That is, we consider the so-called **phase-one problem**

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

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



It is evident that any feasible solution \hat{x} of (Φ) with $\hat{x}_{n+1} = 0$ is feasible for (P) . Moreover, if the minimum objective value of (Φ) 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 (Φ) . We will soon see that we can easily find a basic feasible solution of (Φ) .

Finding a basic feasible solution of (Φ) . 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

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

and our new non-basic indices are

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

Lemma 4.13

The basic solution of (Φ) associated with the basic partition β, η is feasible for (Φ) .

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

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

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 β, η . 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} . \square

The end game for (Φ) . If (P) is feasible, then at the very last iteration of the Worry-Free Simplex Algorithm on (Φ) , 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^* so that $\beta_{i^*} = n + 1$; 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 ϵ -perturbed version

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

where $b_\epsilon(B) := b + \vec{\epsilon}$, and $\vec{\epsilon} := (\epsilon, \epsilon^2, \dots, \epsilon^m)'$. Then we choose i^* so that $\bar{x}_{\tilde{\beta}_{i^*}}^\epsilon$ is most negative with respect to the ordering $<_\epsilon$, and exchange $\tilde{\beta}_{i^*}$ with $\tilde{\eta}_{n-m+1} = n+1$ as before. Then, as in Lemma 4.13, the resulting basis is feasible for (Φ_ϵ) .



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 ϵ . That is, the constant term will become 0. At this point, although x_{n+1} may not be eligible to leave the basis for (Φ_ϵ) , it *will* be eligible to leave for (Φ) . So, at this point we let x_{n+1} leave the basis, and we terminate the solution process for (Φ_ϵ) , having found a feasible basis for (P) . In fact, we have just constructively proved the following result.

Theorem 4.14

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 (Φ_ϵ) when we apply the "early arrival" idea, the resulting basis may not be feasible for (P_ϵ) . So we will have to re-perturb (P) .



"Be patient". Perhaps a more elegant way to handle the situation is to fully solve (Φ_ϵ) . In doing so, if (P) has a feasible solution, then the minimum objective value of (Φ_ϵ) 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 (Φ_ϵ) is *positive*. Because x_{n+1} legally left the basis for (Φ_ϵ) at the final iteration, the resulting basis is feasible for (P_ϵ) . So we do not re-perturb (P) , and we simply revert to solving (P_ϵ) from the final basis of (Φ_ϵ) .

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. I would like to admit that the plastic Simplex derailleurs which I have owned and worked on have been a joy both to use and to set up. From time to time I have read remarks by bikers about these derailleurs. Usually such remarks are based on inadequate knowledge of how to take care of the things.

The cable on the Simplex front changer is attached to a pushrod which is moved by a lever. The lever is moved by a cable which is mounted in the housing. 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 housing. Put some oil inside the housing and to lubricate the pushed. I don't think it matters much whether one uses grease or oil on the push-

rod. I use whatever may be handy and they seem to work equally well, although I suspect grease and do a slightly better job. I have had to do this after many hours of riding in the rain, sometimes all the lubricant will wash out no matter what one uses. Then I sometimes find myself in a bind because the outer travel is so small that I am forced to sit chattering 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 moving the cage to 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 to the Campagnolo front changer.

To adjust the inner travel, you put the chain on the largest (innermost) rear sprocket and the smallest front sprocket. Loosen the hex head screws on the cage. Set the cage so that it just misses getting rubbed by the chain and tightens down the screw. Now, you will have to have a spoke 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 so that it just misses the chain.

It does not matter what kind of wrench you use to tighten down the setscrew. I guess a 4-inch adjustable is the best. There is a 12-inch adjustable which have a groove in it will double as a spoke wrench for off-size spoke nipples, but I generally use whatever wrench I can grab. You will notice that the "for-the-god" parts are hanging around the shop (you can spot them by their expensive equipment and their inability to hold a job). I have a 12-inch adjustable. 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 apparently mechanic must have one of these in their shop. He can be considered to have the feel off it. (He or she must also break off two stem expander bolts, strip three freewheels, install a new rear wheel, and turn the bracket spindle backwards, put in two sets of Sturmey-Archer high-gear pawls upside down, destroy the seat tube decals and the head tube decal, and turn the rear wheels to the wrong side before graduating to journeyman mechanic.) If the setscrew is broken off but the cage is adjusted correctly, you can still ride it. 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 my skill for it 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 move the cage. This is normal. This is play in the pushrod: nothing to worry about. Some people mistake this play for a loose setscrew so they tighten the setscrew.

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

Bike World Magazine



"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 (see the "Early arrival" paragraph in Section 4.4.2);
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 ϵ -perturbed problems. Therefore, in practice, no computer code that is routinely applied to large instances worries about the *potential* for cycling associated with the very-real possibility of degeneracy.

4.6 Exercises

Exercise 4.1 (Carry out the Simplex Algorithm)

`pivot_tools.ipynb` (see Appendix A.6) implements the primitive steps of the simplex algorithm. *Using these primitives only*, write a Python function to carry out the simplex algorithm. Initialize your data as is done in `pivot_example.ipynb` (see Appendix A.5). 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 illustrate the different possibilities for (P) (i.e., infeasible, optimal, unbounded).

Exercise 4.2 (Dual change — first do Exercise 3.4)

Let β, η 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 η_j enter the basis and β_ℓ 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{y} be the dual solution associated with the basic partition $\tilde{\beta}, \tilde{\eta}$, and let H_ℓ be row ℓ of $H := A_\beta^{-1}$. Prove that

$$\tilde{y} = \bar{y} + \frac{\bar{c}_{\eta_j}}{\bar{a}_{\ell, \eta_j}} H'_\ell.$$

HINT: Use the Sherman-Morrison formula; see Section 1.3.

Exercise 4.3 (Traditional phase one)

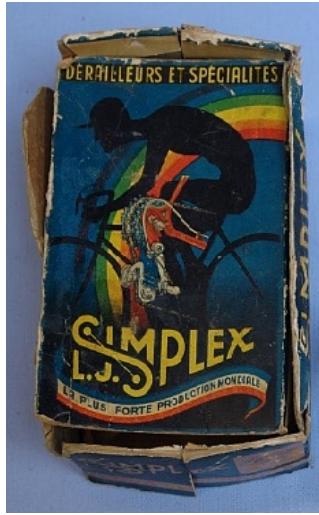
Instead of organizing the phase-one problem as (Φ) , we could first scale rows of $Ax = b$ as necessary so as to achieve $b \geq 0$. Then we can formulate the "traditional phase-one problem"

$$\begin{array}{lll} \min & \sum_{j=1}^m x_{n+j} \\ Ax & + & \sum_{j=1}^m e_j x_{n+j} = b; \\ x \geq \mathbf{0} & , & x_{n+1}, \dots, x_{n+m} \geq 0. \end{array} \quad (\hat{\Phi})$$

Here we have m artificial variables: $x_{n+1}, x_{n+2}, \dots, x_{n+m}$. It is easy to see that (i) $\beta := \{n+1, n+2, \dots, n+m\}$ is feasible for $(\hat{\Phi})$, and (ii) the optimal value of $(\hat{\Phi})$ is zero if and only if (P) has a feasible solution.

It may be that the optimal value of $(\hat{\Phi})$ is zero, but the optimal basis discovered by the Simplex Algorithm applied to $(\hat{\Phi})$ contains some of the indices $\{n+1, n+2, \dots, n+m\}$ of artificial variables. Describe how we can take such an optimal basis and pass to a different optimal basis that uses none of $\{n+1, n+2, \dots, n+m\}$ (and is thus a feasible basis for (P)). HINT: Use Theorem 1.2.

Exercise 4.4 (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 γ satisfying $0 < \gamma < \tan(\theta/2)$. Let

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

Let

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

Then, for $j = 3, 4, \dots, 2k$, 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 $j = 1, 2, \dots, 2k$, and let $b := (0, 0)'$. Because $b = 0$, the problem is fully degenerate; that is, $\bar{x} = 0$ for all basic solutions \bar{x} . Notice that this implies that either the problem has optimal objective value zero, or the objective value is unbounded on the feasible region.

For $k = 5$, you can choose $\gamma := \frac{1}{2} \tan(\theta/2)$, and then check that the following is a sequence of bases β that are legal for the Worry-Free Simplex Algorithm:

$$\begin{aligned}\beta = & (1, 2) \rightarrow (3, 2) \rightarrow (3, 4) \rightarrow (5, 4) \rightarrow (5, 6) \rightarrow (7, 6) \rightarrow (7, 8) \rightarrow (9, 8) \rightarrow \dots \\ & \rightarrow (2k - 1, 2k) \rightarrow (1, 2k) \rightarrow (1, 2).\end{aligned}$$

You need to check that for every pivot, the incoming basic variable x_{η_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 any software that you find convenient (e.g., Python, MATLAB or Mathematica).

Note that it may seem hard to grasp the picture at all⁵. But see Section 6.1.3 and Figure 4.5; you can look at it from different perspectives using the Jupyter notebook [Circle.ipynb](#) (see Appendix A.7).

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

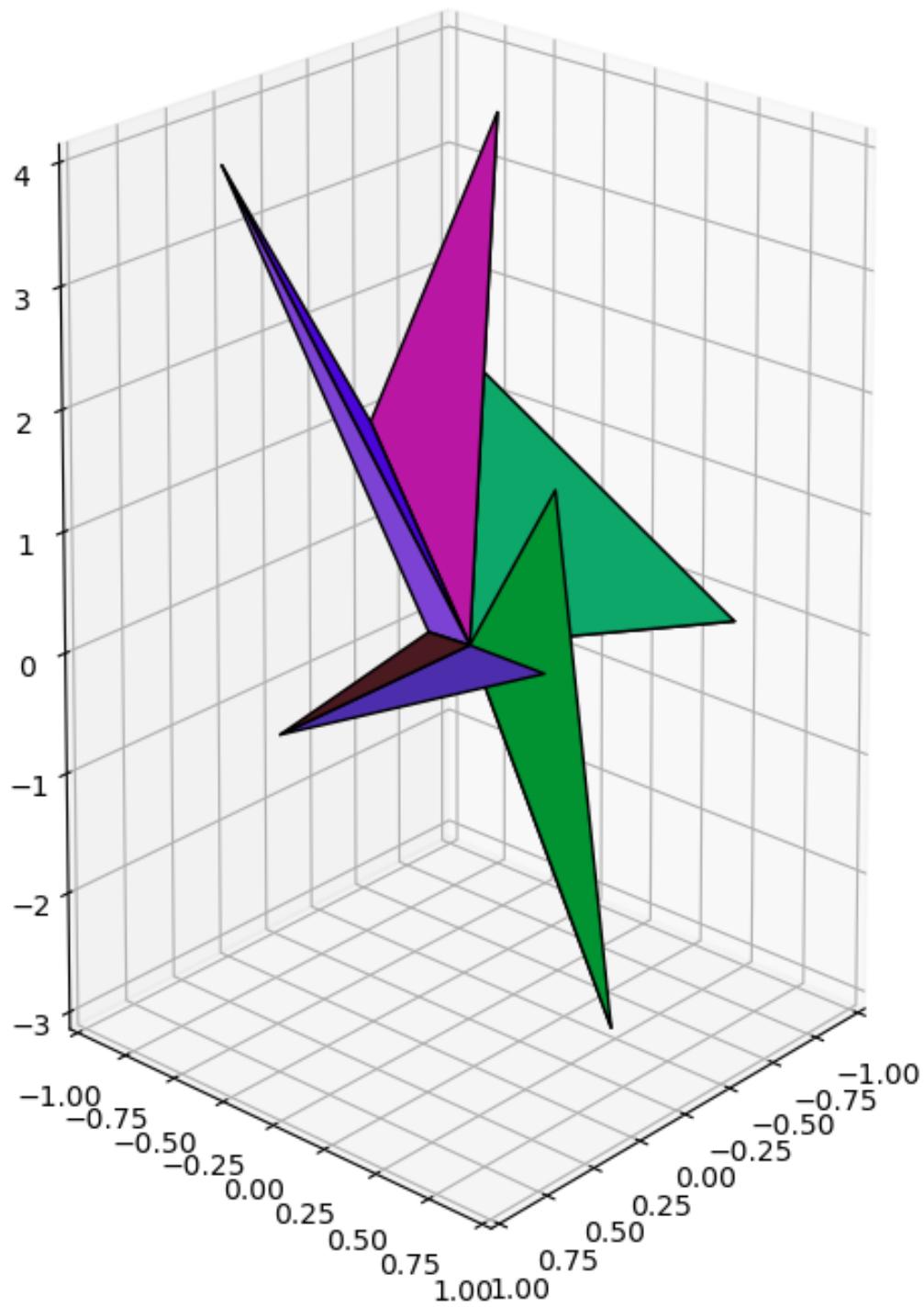
Exercise 4.5

Run the code [pivot_example.ipynb](#) (see Appendix A.5), but with the following line uncommented:

```
#pivot_perturb()      # uncomment to perturb the right-hand side
```

See how this carries out the algebraic-perturbation method from Section 4.3.

Now, using this code as your starting point, solve the example from Exercise 4.4 with $k = 5$ to optimality, using the algebraic-perturbation method. Just change the data to correspond to the example that we want to solve, and do the pivots one at a time, following the rules of the simplex method.

Figure 4.5: 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

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

and its dual

$$\max_{y' A} y'b \leq c'. \tag{D}$$

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 β is a feasible basis and $\bar{c}_\eta \geq 0$, then the primal solution \bar{x} and the dual solution \bar{y} associated with β 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 β 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 β such that the associated basic solution \bar{x} and the associated dual solution \bar{y} are optimal. \square

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



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

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

Theorem 5.3

If β is a basis, then the primal basic solution \bar{x} (feasible or not) and the dual solution \bar{y} (feasible or not) 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

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

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

Theorem 5.4

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

Proof.

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

which is 0 by complementarity. \square

Corollary 5.5 (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 5.4 and the Weak Duality Theorem. \square

Theorem 5.6 (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

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

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

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

□

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

$$\begin{aligned} \min \quad & c'_P x_P + c'_N x_N + c'_U x_U \\ & A_{GP} x_P + A_{GN} x_N + A_{GU} x_U \geq b_G; \\ & A_{LP} x_P + A_{LN} x_N + A_{LU} x_U \leq b_L; \\ & A_{EP} x_P + A_{EN} x_N + A_{EU} x_U = b_E; \\ & x_P \geq \mathbf{0}, \quad x_N \leq \mathbf{0}. \end{aligned} \tag{G}$$

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

$$\begin{aligned} \max \quad & y'_G b_G + y'_L b_L + y'_E b_E \\ & y'_G A_{GP} + y'_L A_{LP} + y'_E A_{EP} \leq c'_P; \\ & y'_G A_{GN} + y'_L A_{LN} + y'_E A_{EN} \geq c'_N; \\ & y'_G A_{GU} + y'_L A_{LU} + y'_E A_{EU} = c'_U; \\ & y'_G \geq \mathbf{0}, \quad y'_L \leq \mathbf{0}. \end{aligned} \tag{H}$$

Theorem 5.7

- **Weak Duality Theorem:** If $(\hat{x}_P, \hat{x}_N, \hat{x}_U)$ is feasible in (G) and $(\hat{y}_G, \hat{y}_L, \hat{y}_E)$ is feasible in (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 (G) has a feasible solution, and (G) is not unbounded, then there exist feasible solutions $(\hat{x}_P, \hat{x}_N, \hat{x}_U)$ for (G) and $(\hat{y}_G, \hat{y}_L, \hat{y}_E)$ for (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 (G) to the standard-form

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

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

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

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

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

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

Theorem 5.8

- **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.

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

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

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	≥ 0	variables
	\leq	≤ 0	
variables	$=$	unres.	constraints
	≥ 0	\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 Farkas⁶, opened the door for studying linear inequalities and optimization.

Theorem 5.9 (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.

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

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

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:

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

Its dual is

$$\max_{y' A} \quad y' b \\ y' A \leq \mathbf{0}' \quad (D)$$

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). \square



Remark 5.10

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 ***ŷ*** 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.

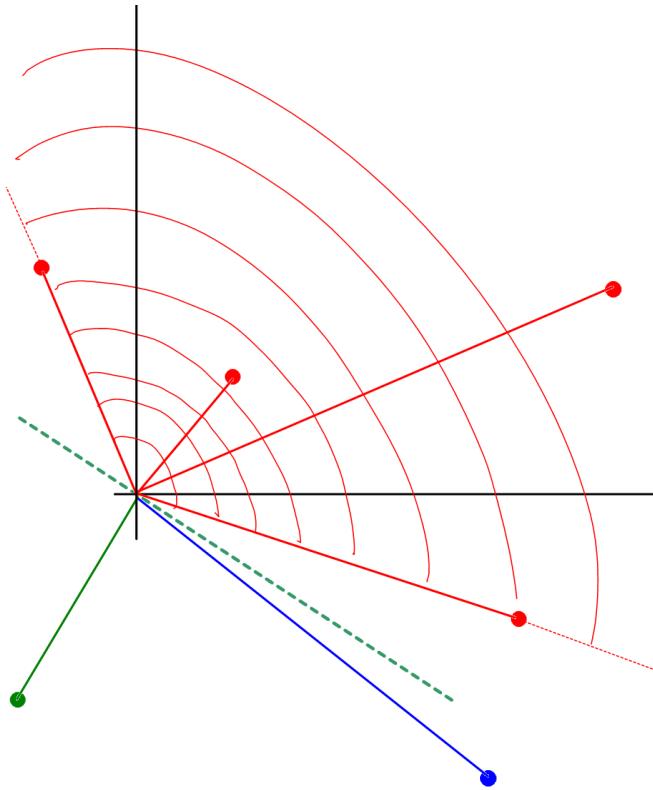


Figure 5.1: Case (II) of the Farkas Lemma

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

Theorem 5.11 (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.

$$Ax \geq b. \quad (\text{I})$$

$$\begin{aligned} y'b &> 0; \\ y'A &= \mathbf{0}'; \\ y &\geq \mathbf{0}. \end{aligned} \quad (\text{II})$$

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

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

which is a clear inconsistency.

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

$$\begin{array}{ll} \min & \mathbf{0}'x \\ Ax & \geq b. \end{array} \quad (\text{P})$$

Its dual is

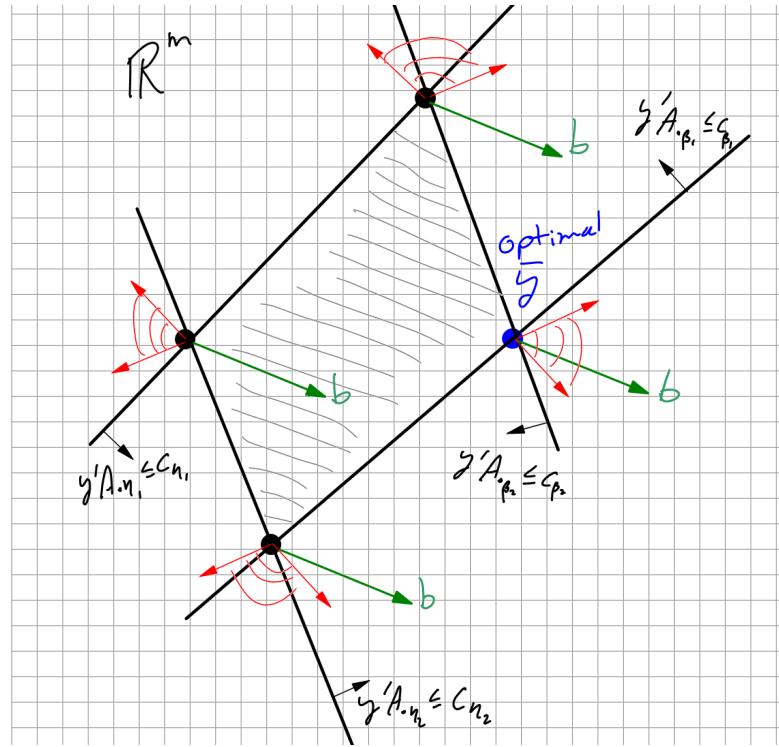
$$\begin{array}{ll} \max & y'b \\ y'A & = \mathbf{0}' \\ y & \geq \mathbf{0}. \end{array} \quad (\text{D})$$

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 greater than any fixed constant). Such a \hat{y} is a solution to (II). \square

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 (Reduced costs as dual values)

In this exercise, we will see that we can regard reduced costs (corresponding to an optimal basic partition) as (optimal) values of dual variables for non-negativity constraints.

Consider the ordinary standard-form problem

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

and let β, η be an optimal basic partition for (P).

We can equivalently see (P) as

$$\begin{array}{ll} z := \min & c'x \\ Ax = b; & \text{dual variables} \\ x \geq \mathbf{0}, & \textcolor{red}{y} \\ & \textcolor{red}{w} \end{array} \quad (\tilde{P})$$

where in (\tilde{P}) , we regard the non-negativity constraints of (P) as ordinary structural constraints — with dual variables.

Define $\bar{w} \in \mathbb{R}^n$ by

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

Prove that together, $\bar{y}' := c'_\beta A_\beta^{-1}$ and \bar{w} are optimal for the dual of (\tilde{P}) .

Exercise 5.3 (Duality and complementarity with Python/Gurobi)

After optimization using Python/Gurobi, it is easy to get more information regarding primal and dual problems. In particular, we can obtain optimal primal and dual solutions, and slacks for these solutions in the primal and dual constraints. See how this is done in [Production.ipynb](#) (Appendix A.3), and verify the concepts of duality and complementarity developed in this chapter.

Exercise 5.4 (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 5.5 (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

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

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

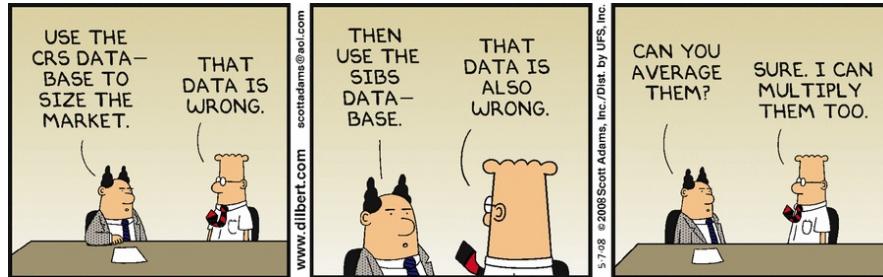
HINT: Let v be the optimal objective value of (P). For each $j = 1, 2, \dots, n$, consider

$$\begin{array}{ll} \max & x_j \\ c'x & \leq v \\ Ax & = b. \\ x & \geq \mathbf{0}. \end{array} \quad (P_j)$$

(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 exactly one of

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

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 5.6 (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 5.7 (A general Theorem of the Alternative)

State and prove a "Theorem of the Alternative" for the system:

$$\begin{aligned} A_{GP}x_P + A_{GN}x_N + A_{GU}x_U &\geq b_G; \\ A_{LP}x_P + A_{LN}x_N + A_{LU}x_U &\leq b_L; \\ A_{EP}x_P + A_{EN}x_N + A_{EU}x_U &= b_E; \\ x_P \geq \mathbf{0}, \quad x_N \leq \mathbf{0}. \end{aligned} \tag{I}$$

Exercise 5.8 (Dual ray)

Consider the linear-optimization problem

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

- 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 *constructively demonstrate* that (D) is unbounded.

Chapter 6

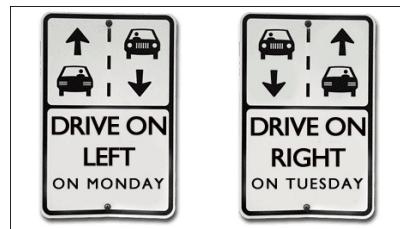
Sensitivity Analysis



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 $f : \mathbb{R}^m \rightarrow \mathbb{R}$ via

$$\begin{aligned} f(b) &:= \min c'x \\ Ax &= b; \\ x &\geq 0. \end{aligned} \tag{P}_b$$

That is, (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 β 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, $c'_\beta - \bar{y}'A_\eta \geq \mathbf{0}'$. Considering the set \mathcal{B} of $b \in \mathbb{R}^m$ such that β is an optimal basis, is it easy to see that \mathcal{B} is just the set of b such that $\bar{x}_\beta := A_\beta^{-1}b \geq \mathbf{0}$. 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 $\bar{x}_{\beta_i} > 0$ for $i = 1, 2, \dots, m$.

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

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

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

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

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

$$L_i := \max_{k : h_k^i > 0} \{-\bar{x}_{\beta_k}/h_k^i\},$$

and

$$U_i := \min_{k : h_k^i < 0} \{-\bar{x}_{\beta_k}/h_k^i\}.$$

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

In summary, for all Δ_i satisfying $L_i \leq \Delta_i \leq U_i$, β is an optimal basis of (P_i) . 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 . \square

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

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

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

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

A function $\check{f} : \mathbb{R}^m \rightarrow \mathbb{R}$ 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

$$\check{f}(u) := \max_{1 \leq i \leq k} \{\bar{f}_i(u)\},$$

for u in the domain of \check{f} , where each \bar{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 $u^1, u^2 \in \mathbb{R}^m$,

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

□

Next we establish a fundamental result concerning the function f (defined as the optimal objective value(P_b)).

Theorem 6.3

f is a convex piecewise-linear function on its domain.

Proof. We refer to the dual

$$f(b) := \max_{y' A \leq c'} y' b \quad (\text{D}_b)$$

of (P_b) .

A basis β 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.⁷ 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 $j = 1, 2, \dots, n$, 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 is of particular interest to think about 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 β , 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_β is an " m -dimensional simplicial cone". Next, we observe that if β is a feasible basis, then K_β 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 β .

In a pivot of the Simplex Algorithm, passing from basis β to basis $\tilde{\beta}$, our goal is to have $K_{\tilde{\beta}}$ intersect the requirement line at a *lower* point than did K_β . In Figure 6.1 ($m = 2$ and the **coordinate axes** are the red lines), we see an example depicting a single pivot. K_β 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.

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



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

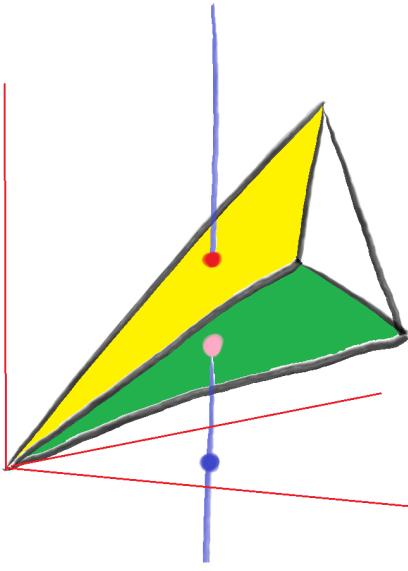


Figure 6.1: A simplex pivot

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

$$\begin{aligned} g(c) &:= \min \quad c'x \\ Ax &= b; \\ x &\geq 0. \end{aligned} \tag{P^c}$$

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 β 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 us assume that \bar{x} is feasible for (P^c) — or, equivalently, $A_\beta^{-1}b \geq 0$. Considering the set \mathcal{C} of $c \in \mathbb{R}^n$ such that β is an optimal basis, is it easy to see that this is just the set of c such that $c'_\eta - c'_\beta A_\beta^{-1} A_\eta \geq 0'$. 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 6.4

The domain of g is a convex set.

A function $g : \mathbb{R}^n \rightarrow \mathbb{R}$ 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 6.5

g is a concave piecewise-linear function on its domain.

6.3 Exercises

Exercise 6.1 (Local sensitivity analysis with Python/Gurobi)

We can easily carry out some local sensitivity analysis with Python/Gurobi. See how this is done in [Production.ipynb](#) (Appendix A.3). Verify the calculations of Python/Gurobi "by hand", using the ideas and formulas in Section 6.1.1 to make the calculations yourself; you may use any convenient software (e.g., Python, MATLAB or Mathematica) to assist you, but *only* for doing arithmetic on scalars, vector and matrices.

Exercise 6.2 (Illustrate global sensitivity analysis using Python/Gurobi)

Using Python/Gurobi, 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, using local analysis to identify successive ranges where the optimal value is linear.

Exercise 6.3 ("I feel that I know the change that is needed." — Mahatma Gandhi)

We are given $2m$ numbers satisfying $L_i \leq 0 \leq U_i$, $i = 1, 2, \dots, m$. Let β be an optimal basis for all of the m problems

$$\begin{aligned} \min \quad & c'x \\ Ax \quad &= b + \Delta_i e_i; \\ x \quad &\geq 0, \end{aligned} \tag{P}_i$$

for all Δ_i satisfying $L_i \leq \Delta_i \leq U_i$. Let's be clear on what this means: For each i *individually*, the basis β is optimal when the i th right-hand side component is changed from b_i to $b_i + \Delta_i$, as long as Δ_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 6.4.

Exercise 6.5 (Concave piecewise-linear function)

Prove Theorem 6.5.

Chapter 7

Large-Scale Linear Optimization



Our goals in this chapter are as follows:

- See some approaches to large-scale linear-optimization problems.
- In particular, 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 problems⁸.

$$\begin{pmatrix} \blacksquare & \cdots & \blacksquare \\ \blacksquare & \ddots & \blacksquare \\ \vdots & & \blacksquare \end{pmatrix}$$

Figure 7.1: Nearly separates

7.1.1 Reformulation via the representation theorem

Theorem 7.1 (The Representation Theorem)

Let

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

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

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

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

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

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

$$\begin{aligned} \sum_{j \in \mathcal{J}} \lambda_j \hat{x}^j + \sum_{k \in \mathcal{K}} \mu_k \hat{z}^k &= \hat{x}; \\ \sum_{j \in \mathcal{J}} \lambda_j &= 1; \\ \lambda_j \geq 0, j \in \mathcal{J}; \mu_k \geq 0, k \in \mathcal{K}. & \end{aligned} \tag{I}$$

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 λ_j and the μ_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

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

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

$$\begin{aligned} \min \quad & -\hat{w}'x \\ Ax &= b; \\ x &\geq \mathbf{0}. \end{aligned} \tag{\hat{P}}$$

(\hat{P}) cannot be unbounded, because $-\hat{w}'z^k \geq 0$, for all $k \in \mathcal{K}$. In addition, every basic feasible solution of (\hat{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{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, $\hat{x} \notin S$. \square

Corollary 7.2 (The Decomposition Theorem)

Let

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

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 of 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 **Main Problem**

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

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 $Ex \geq h$ of (Q), and it is easy to see that (M) is equivalent to (Q). \square

Decomposition is typically applied in a way such that the constraints defining (S) are somehow relatively "nice," and the constraints $Ex \geq h$ 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

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

and

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

We note that there is nothing special here about the "nice" constraints being " $=$ ", and the complicating constraints being " \geq ". The method, with small modifications, can handle any types of constraints; we take the particular form that we do for some convenience.

7.1.2 Solution of the Main Problem (M) 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), the standard-form problem obtained from (M) by adding slack variables for the $Ex \geq 0$ constraints, because the number of constraints of (M), is just one more than the number of constraints in $Ex \leq h$.

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 check.

Toward this end, we define dual variables y and σ for (M).

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

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

Using a vector of slack variables s , we obtain the standard-form problem

$$\begin{aligned} \min \quad & \sum_{j \in \mathcal{J}} (c' \hat{x}^j) \lambda_j + \sum_{k \in \mathcal{K}} (c' \hat{z}^k) \mu_k \\ & \sum_{j \in \mathcal{J}} (E \hat{x}^j) \lambda_j + \sum_{k \in \mathcal{K}} (E \hat{z}^k) \mu_k - Is &= h; \\ & \sum_{j \in \mathcal{J}} \lambda_j &= 1; \\ & \lambda_j \geq 0, j \in \mathcal{J}; & \mu_k \geq 0, k \in \mathcal{K}; & s \geq \mathbf{0}. \end{aligned} \tag{\bar{M}}$$

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

Entering variable. Notice that nonnegativity of the dual variables y in (M), is is equivalently realized in (\bar{M}) via the reduced costs of the slack variables being non-negative. Therefore, a slack variable s_i is eligible to enter the basis if $\bar{y}_i < 0$.

The reduced cost of a variable λ_j is

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

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

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

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 λ_j corresponding to the optimal \hat{x}^j is eligible to enter the current basis of (\bar{M}) . On the other hand, if

the optimal objective value of (SUB) is non-negative, then we have a proof that no non-basic λ_j is eligible to enter the current basis of (\bar{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 μ_k is precisely $(c'\hat{z}^k) - \bar{y}'(E\hat{z}^k) = (c' - \bar{y}'E)\hat{z}^k$, so, in fact, μ_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 (\bar{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

$$\begin{pmatrix} E\hat{z}^k \\ 0 \end{pmatrix} \text{ and } \begin{pmatrix} -e_i \\ 0 \end{pmatrix}.$$

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:

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

if λ_j is entering the basis, or

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

if μ_k is entering the basis, or

$$B^{-1} \begin{pmatrix} -e_i \\ 0 \end{pmatrix}$$

if s_i 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

$$\begin{pmatrix} E\hat{z}^k \\ 0 \end{pmatrix} \text{ and } \begin{pmatrix} -e_i \\ 0 \end{pmatrix}.$$

So organizing the basic variables λ_j , μ_k and s_i into a vector ζ , with their order appropriately matched with the columns of B , the vector $\bar{\zeta}$ of values of ζ 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 $c'\hat{x}^j$, $c'\hat{z}^k$, and 0 of the basic variables λ_j , μ_k and s_i into a vector ξ , 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 (\bar{M}) ; after all, we may not have at hand any basic feasible solutions and rays of (S) . Next, we give a simple recipe. First, we take as \hat{x}^1 *any* basic feasible solution of (P) . Such a solution can be readily obtained by using our usual (phase-one) methodology of the Simplex Algorithm. Our initial basic variables are all of the slack variables s_i and also λ_1 , associated with \hat{x}^1 . So we have the initial basis matrix

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

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

It is very important to realize that we have given a recipe for finding an initial basic solution of (\bar{M}) . This basic solution is feasible precisely when x^1 satisfies the $Ex \geq h$ constraints. If this solution is not feasible, then we would introduce an artificial variable and do a phase-one procedure. Following the methodology of Section 4.4.1, we introduce the single artificial column

$$-B\mathbf{1} = \left(\begin{array}{c} \mathbf{1} - E\hat{x}^1 \\ -1 \end{array} \right),$$

with cost 1. We let the artificial variable enter the basis, removing the slack variable that is the most negative from the basis. This yields a feasible basis for the phase-one problem, with positive objective value. Now we carry out phase-one of the simplex method, *using Decomposition*, minimizing the artificial variable, seeking to drive it down to zero.

A demonstration implementation.



It is not completely trivial to write a small Python/Gurobi code for the Decomposition Algorithm. First of all, we solve the subproblems (SUB) using functionality of Gurobi. Another point is that rather than carry out the simplex method at a detailed level on (\bar{M}) , we just accumulate all columns of (\bar{M}) that we generate, and always solve linear-optimization problems, using functionality of Gurobi, 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 Gurobi does give us a dual solution, so we do not compute that ourselves. Our code is in the Jupyter notebook [Decomp.ipynb](#) (see Appendix A.8).

In Figures 7.2 and 7.3, we see quite good behavior for the Decomposition Algorithm, for a problem with 100 variables, 200 "complicating" constraints (i.e., rows of E), and 50 "nice" constraints (i.e., rows of A).

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

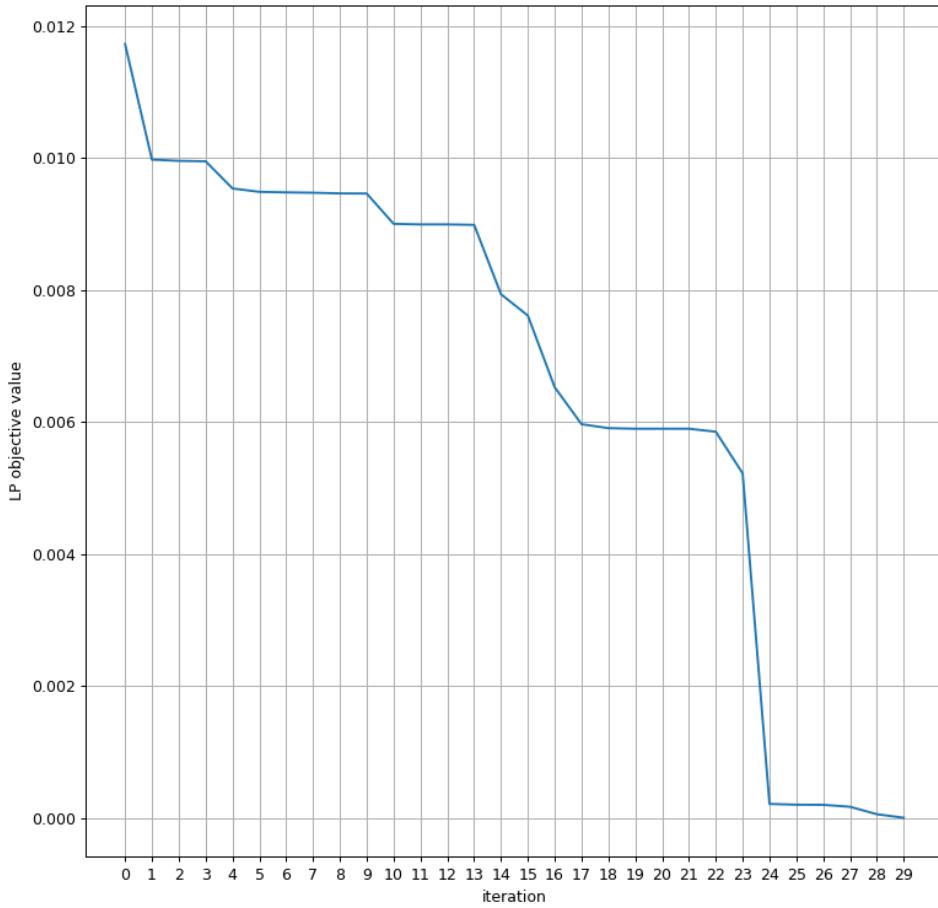


Figure 7.2: Example: Phase-one objective values with Decomposition

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.

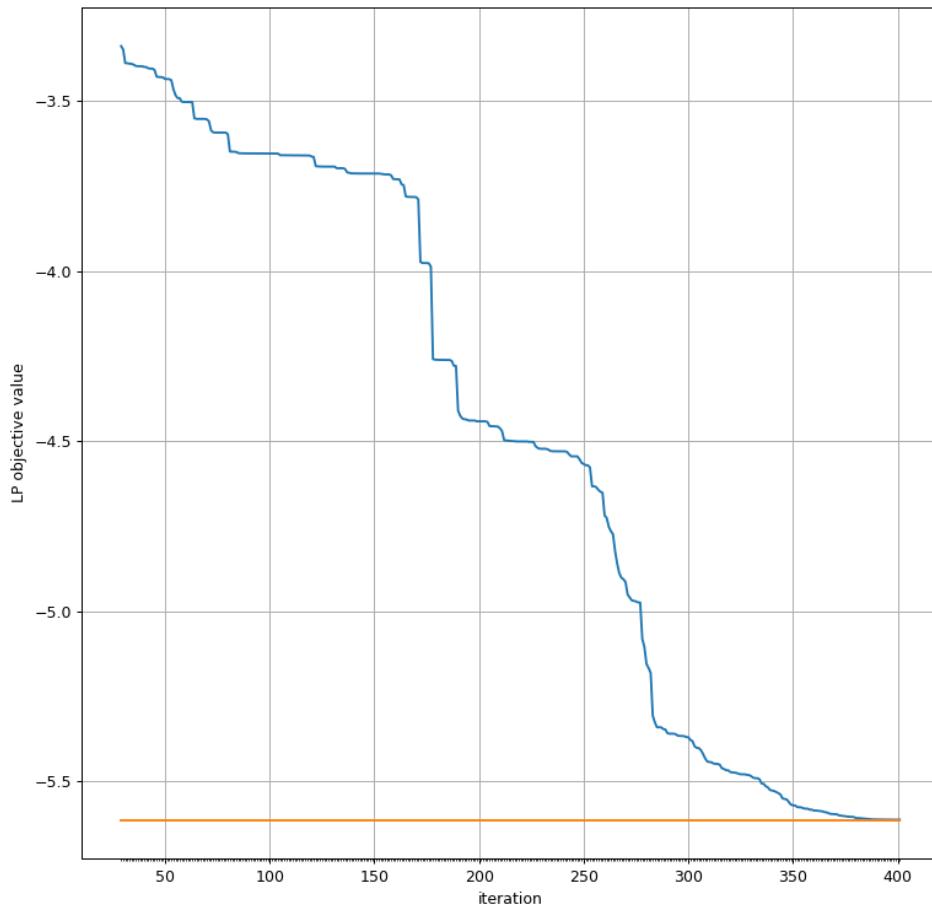


Figure 7.3: Example: Phase-two objective values with Decomposition

7.2 Lagrangian Relaxation



Again, we consider

$$\begin{aligned} z := \min \quad & c'x \\ \text{subject to} \quad & Ex \geq h; \\ & Ax = b; \\ & x \geq \mathbf{0}, \end{aligned} \tag{Q}$$

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: $Ax = b$, $x \geq \mathbf{0}$. In what follows, we assume that (Q) has an optimal solution.

7.2.1 Lagrangian bounds

For any fixed choice of $\hat{y} \geq \mathbf{0}$, consider the following "Lagrangian" optimization problem

$$\begin{aligned} v(\hat{y}) := \hat{y}'h + \min \quad & (c' - \hat{y}'E)x \\ \text{subject to} \quad & Ax = b; \\ & x \geq \mathbf{0}. \end{aligned} \tag{L}_{\hat{y}}$$

Note that the only variables in the minimization are x , because 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

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

The last equation uses the fact that x^* is optimal for (Q), so $z = c'x^*$, and also that $Ex^* \geq h$ and $\hat{y} \geq \mathbf{0}$. \square

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 6.5). 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 $v(y)$ over $y \geq 0$, and the maximum value of $v(y)$ over $y \geq 0$ is z .

In the theorem above, we refer to two duals. The dual of (Q) is:

$$\begin{aligned} \max \quad & y' h + \pi' b \\ & y'E + \pi'A \leq c' ; \\ & y \geq \mathbf{0}. \end{aligned}$$

The dual of $(L_{\hat{y}})$ is:

$$\hat{y}' h + \max_{\pi' A \leq c' - \hat{y}' E} \pi' b$$

Proof. x^* is clearly feasible for $(L_{\hat{y}})$. Because \hat{y} and $\hat{\pi}$ are feasible for the dual of (Q), we have $\hat{y} \geq \mathbf{0}$, and $\hat{y}' E + \hat{\pi}' A \leq c'$. The latter implies that $\hat{\pi}$ 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 $E \hat{x}^* \geq h$ (feasibility of x^* in (Q)), we then have that $(c' - \hat{y}' E) x^* \leq \hat{\pi}' b$. 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,

$$\begin{aligned} z &\geq v(\hat{y}) \quad (\text{by Theorem 7.3}) \\ &= \hat{y}' h + (c' - \hat{y}' E) x^* \quad (\text{because } x^* \text{ is optimal for } (L_{\hat{y}})) \\ &= c' x^* - \hat{y}' (E x^* - h) \\ &= c' x^* \quad (\text{because of strong complementarity: } \hat{y}_i (E_i x^* - h_i) = 0 \text{ for all } i) \\ &= z. \end{aligned}$$

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

Theorem 7.5

Suppose that \hat{y} is a maximizer of $v(y)$ over $y \geq \mathbf{0}$, 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.

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

The third equation follows from taking the dual of the inner (minimization) problem. The last equation follows from seeing that the final maximization (over $y \geq \mathbf{0}$ and π 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 $\hat{\pi}' A \leq c' - \hat{y}' E$ and $y \geq \mathbf{0}$, we have that \hat{y} and $\hat{\pi}$ are optimal for the dual of (Q). \square

Note that the conclusion of Theorem 7.5 gives us an optimal \hat{y} and $\hat{\pi}$ for the dual of (Q), but not an 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 $\hat{y} \geq \mathbf{0}$. Can we find the best such \hat{y} ? This would entail solving the so-called **Lagrangian Dual** problem of maximizing $v(y)$ over all $y \geq \mathbf{0}$ in the domain of v . 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.

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

The inequality follows from the fact that \hat{x} is feasible (but possibly not optimal) for $(L_{\tilde{y}})$. \square

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 "Projected Subgradient Optimization Algorithm" for solving the Lagrangian Dual. The general idea is to iteratively move in the direction of a subgradient.

Projected Subgradient Optimization Algorithm

0. Start with any non-negative $\hat{y}^1 \in \mathbb{R}^m$. 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 $\hat{y}^{k+1} \leftarrow \mathbf{Proj}_{\mathbb{R}_+^m}(\hat{y}^k + \lambda_k \hat{\gamma}^k)$.
4. Let $k \leftarrow k + 1$, and GOTO 1.

A few comments are in order:

- The λ_k are positive "step sizes", and we will discuss how they may be chosen to gain convergence of the algorithm;
- $\mathbf{Proj}_{\mathbb{R}_+^m}(\cdot)$ means "project onto the non-negative orthant". That is, we take the closest non-negative point (in the Euclidean norm) to the argument of the function. In fact, this means just zeroing-out the negative entries.
- The algorithm can break down if $(L_{\hat{y}^k})$ is unbounded in step 1.

Convergence. We have neglected, thus far, to fully specify the Subgradient Optimization Algorithm. We can stop if, at some iteration k , we have $\hat{\gamma}^k = \mathbf{0}$ (or, more generally, if $\hat{y}^k = \mathbf{Proj}_{\mathbb{R}_+^m}(\hat{y}^k + \lambda_k \hat{\gamma}^k)$), 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 $v(y)$ over $y \geq \mathbf{0}$. But this is 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 how the convergence analysis is critically connected with the choice of step sizes. To prove convergence of the algorithm, we must first establish a key technical lemma.

Lemma 7.7

Let y^* be any maximizer of $v(y)$ over $y \geq \mathbf{0}$. Suppose that $\lambda_k > 0$, for all k . Then

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

Proof. Let $w^{i+1} := \hat{y}^i + \lambda_i \hat{\gamma}^i$; that is, the *unprojected* $(i+1)$ -st iterate. For $i \geq 1$, we have

$$\begin{aligned} & \|y^* - \hat{y}^{i+1}\|^2 - \|y^* - \hat{y}^i\|^2 \\ & \leq \|y^* - w^{i+1}\|^2 - \|y^* - \hat{y}^i\|^2 \\ & = \|(y^* - \hat{y}^i) - \lambda_i \hat{\gamma}^i\|^2 - \|y^* - \hat{y}^i\|^2 \\ & = \lambda_i^2 \|\hat{\gamma}^i\|^2 - 2\lambda_i (y^* - \hat{y}^i)' \hat{\gamma}^i \\ & \leq \lambda_i^2 \|\hat{\gamma}^i\|^2 - 2\lambda_i (v(y^*) - v(\hat{y}^i)). \end{aligned}$$

The first inequality uses the fact that the projection of a point onto a convex set is no further to any point in that convex set than the unprojected point. The final inequality uses the assumption that $\lambda_i > 0$ and the subgradient inequality:

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

plugging in y^* for \tilde{y} .

Finally, adding up the established inequality

$$\|y^* - \hat{y}^{i+1}\|^2 - \|y^* - \hat{y}^i\|^2 \leq \lambda_i^2 \|\hat{\gamma}^i\|^2 - 2\lambda_i (v(y^*) - v(\hat{y}^i)),$$

over $i = 1, 2, \dots, k$ yields the result. \square

We will see that the algorithm converges (in a certain sense), if we take a sequence of $\lambda_k > 0$ that in some sense slowly converges; Specifically, we can require 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 $\ln(k+1) < \sum_{k=1}^{\infty} 1/k < \ln(k) + 1$ (so it is not summable) and it turns out that $\sum_{k=1}^{\infty} 1/k^2 = \pi^2/6$ (so it is square summable).

Let

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

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 $v(y)$ over $y \geq 0$. Assume that we take a *basic* solution as the solution of each Lagrangian subproblem. Suppose that $\lambda_i > 0$, for all i . Suppose further that $\sum_{i=1}^{\infty} \lambda_i^2 < +\infty$ and $\sum_{i=1}^{\infty} \lambda_i = +\infty$. Then $\lim_{k \rightarrow \infty} v_k^* = v(y^*)$.

Proof. Because $\|y^* - \hat{y}^{k+1}\|^2$ in the statement of Lemma 7.7 is non-negative, we have

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

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

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

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 $\|\hat{\gamma}^i\|$ is bounded by some constant Γ , 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

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

So, we have

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

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. \square

A more general way to gain convergence is to replace the "square summable" condition with a so-called "diminishing" condition: $\lim_{k \rightarrow \infty} \lambda_k = 0$. For example, $\lambda_k := \alpha/\sqrt{k}$, with $\alpha > 0$, is diminishing but not square summable.

Theorem 7.9 ("Diminishing but not summable" convergence)

Let y^* be any maximizer of $v(y)$ over $y \geq 0$. Assume that we take a *basic* solution as the solution of each Lagrangian subproblem. Suppose that $\lambda_i > 0$, for all i . Suppose further that $\lim_{i \rightarrow \infty} \lambda_i = 0$ and $\sum_{i=1}^{\infty} \lambda_i = +\infty$. Then $\lim_{k \rightarrow \infty} v_k^* = v(y^*)$.

Proof. We simply need to analyze the inequality $(*)$ in the proof of Theorem 7.8 a bit more carefully. We will demonstrate that under the hypotheses of the present result, for every $\epsilon > 0$, there exists a positive integer K , such that $v(y^*) - v_k^* < \epsilon$ for all $k > K$. This will imply the result.

So we consider a fixed but arbitrary $\epsilon > 0$. Our assumption that $\lim_{i \rightarrow \infty} \lambda_i = 0$ implies that there is a positive integer K_1 such that $\lambda_i < \epsilon/\Gamma^2$ for all $i > K_1$. Our hypothesis that $\sum_{i=1}^{\infty} \lambda_i = +\infty$ implies that there is a positive integer K_2 such that

$$\sum_{i=1}^k \lambda_i > \frac{\|y^* - \hat{y}^1\|^2 + \Gamma^2 \sum_{i=1}^{K_1} \lambda_i^2}{\epsilon},$$

for all $k > K_2$. Now, starting from $(*)$, for all $k > K := \max\{K_1, K_2\}$, we have

$$\begin{aligned} v(y^*) - v_k^* &\leq \frac{\|y^* - \hat{y}^1\|^2 + \Gamma^2 \sum_{i=1}^k \lambda_i^2}{2 \sum_{i=1}^k \lambda_i} \\ &= \frac{\|y^* - \hat{y}^1\|^2 + \Gamma^2 \sum_{i=1}^{K_1} \lambda_i^2}{2 \sum_{i=1}^k \lambda_i} + \frac{\Gamma^2 \sum_{i=K_1+1}^k \lambda_i^2}{2 \sum_{i=1}^k \lambda_i} \\ &\leq \frac{\|y^* - \hat{y}^1\|^2 + \Gamma^2 \sum_{i=1}^{K_1} \lambda_i^2}{2 \sum_{i=1}^k \lambda_i} + \frac{\Gamma^2 \sum_{i=K_1+1}^k \lambda_i^2}{2 \sum_{i=K_1+1}^k \lambda_i} \\ &< \frac{\frac{2}{\epsilon} \left(\|y^* - \hat{y}^1\|^2 + \Gamma^2 \sum_{i=1}^{K_1} \lambda_i^2 \right)}{2 \sum_{i=K_1+1}^k \lambda_i} + \frac{\Gamma^2 \sum_{i=K_1+1}^k \lambda_i (\epsilon/\Gamma^2)}{2 \sum_{i=K_1+1}^k \lambda_i} \\ &= \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon. \end{aligned}$$

\square

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 λ_k that depends on a "conservative guess" of the optimal value of $v(\hat{y})$, taking bigger steps when one is far away, and smaller steps when one is close. A further idea

is take shorter steps when the subgradient has a big norm. With these ideas, we can achieve faster practical convergence of the algorithm, using a so-called "Polyak step size"; see Figure 7.5. Instead of working to prove convergence from $(*)$ (which was derived from Lemma 7.7), as we did for Theorems 7.8 and 7.9, we work in a different way. Lemma 7.7 tells us that $\|y^* - \hat{y}^{k+1}\|^2 \leq u_k(\lambda)$, where

$$u_k(\lambda) := \|y^* - \hat{y}^1\|^2 + \sum_{i=1}^k \lambda_i^2 \|\hat{\gamma}^i\|^2 - 2 \sum_{i=1}^k \lambda_i (v(y^*) - v(\hat{y}^i)),$$

for all positive integers k . It is easy to see that $u_k : \mathbb{R}_+^m \rightarrow \mathbb{R}$ is a strictly convex (quadratic) function. Its minimum is achieved at its stationary point, provided that the stationary point is non-negative. Calculating

$$\frac{\partial u_k(\bar{\lambda})}{\partial \lambda_i} = 2\|\hat{\gamma}^i\|^2 \bar{\lambda}_i - 2(v(y^*) - v(\hat{y}^i)),$$

setting this to 0 (for each i), and solving for $\bar{\lambda}_i$, we get the **Polyak step size**

$$\bar{\lambda}_i := \frac{v(y^*) - v(\hat{y}^i)}{\|\hat{\gamma}^i\|^2}, \text{ for } i = 1, \dots, k.$$

This is non-negative, so it minimizes u_k on \mathbb{R}_+^m . Let us put aside, for now, the fact that using this rule seems to require that we know $v(y^*)$.

Theorem 7.10 (Polyak step-size convergence)

Let y^* be any maximizer of $v(y)$ over $y \geq 0$. Assume that we take a *basic* solution as the solution of each Lagrangian subproblem. Suppose that at each iteration, λ_k is chosen according to the Polyak step size. Then $\lim_{k \rightarrow \infty} v_k^* = v(y^*)$.

Proof. From Lemma 7.7, plugging in $\bar{\lambda}$, we have

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

Simplifying and rearranging, we have

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

This implies that

$$\sum_{i=1}^k (v(y^*) - v(\hat{y}^i))^2 \leq \Gamma^2 \|y^* - \hat{y}^1\|^2,$$

and then

$$v(y^*) - v_k^* \leq \Gamma \|y^* - \hat{y}^1\| / \sqrt{k}.$$

Finally, letting k go to infinity, we see that the limit of the right-hand side is 0, and we have our result. \square

A conservative guess. The problem with the Polyak step size is that it requires knowing the maximum value of the function that we seek to maximize. But it is not hard to verify that if we replace $v(y^*)$ in the Polyak step-size formula with an lower bound LB, then either we can stop when we find a \hat{y}^k with $v(\hat{y}^k) \geq LB$, or $\lim_{k \rightarrow \infty} v_k^* = LB$. Practically speaking, the subgradient algorithm becomes a way to rigorously check the validity of the lower bound.

Constant step size. It is very natural to consider the simplest step-size rule, where we take the λ_k to be all equal. We cannot generally converge to an optimum with such a rule, but we can bound the limiting quality of the solution that can be obtained. Considering again the inequality (*) in the proof of Theorem 7.8, if we take instead $\lambda_k = 1/s$, for some fixed $s > 0$, we can observe that $\lim_{k \rightarrow \infty} v_k^* \geq v(y^*) - s\Gamma^2/2$.

A simple implementation. It is very easy to write a small Gurobi/Python code for Subgradient Optimization. Our code is in the Jupyter notebook [SubgradProj.ipynb](#) (see Appendix A.9). Typical behavior is a very bad first iteration, then some iterations to recover from that, and then a slow and steady convergence to an optimum. The method is usually stopped after a predetermined number of iterations or after progress becomes very slow. In Figure 7.4, we see this typical behavior, for a problem with 100 variables, 200 "complicating" constraints (i.e., rows of E), and 50 "nice" constraints (i.e., rows of A).

Dual estimation. From Theorem 7.5, we see that the Subgradient Optimization Method is a way to try and quickly find an *estimate* of an optimal solution to the dual of (Q). At each step, \hat{y} together with the $\hat{\pi}$ that is optimal for the dual of $(L_{\hat{y}})$ give a feasible solution of (Q) with objective value $v(\hat{y})$. 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 an \hat{x} that is optimal for $(L_{\hat{y}})$ will be feasible for (Q).

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 $i = 1, 2, \dots, m$. The **cutting-stock problem** is to find a plan for satisfying demand, using as few stock rolls as possible.⁹

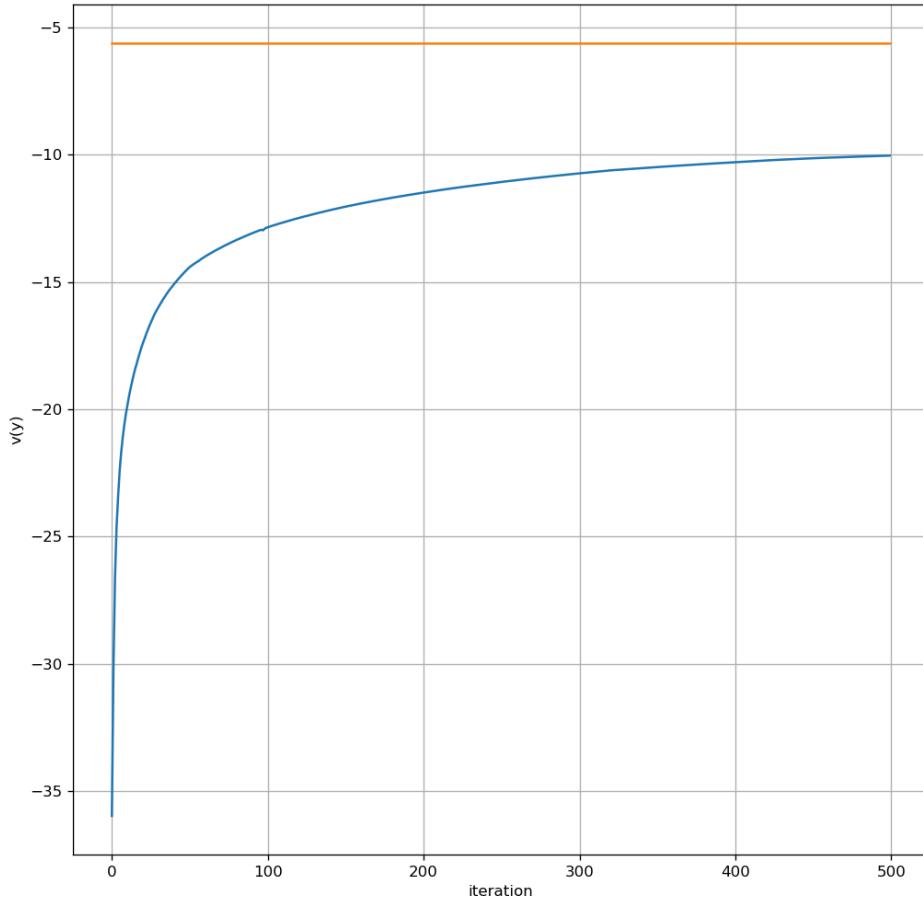


Figure 7.4: Example: Projected subgradient optimization with harmonic step sizes

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

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

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

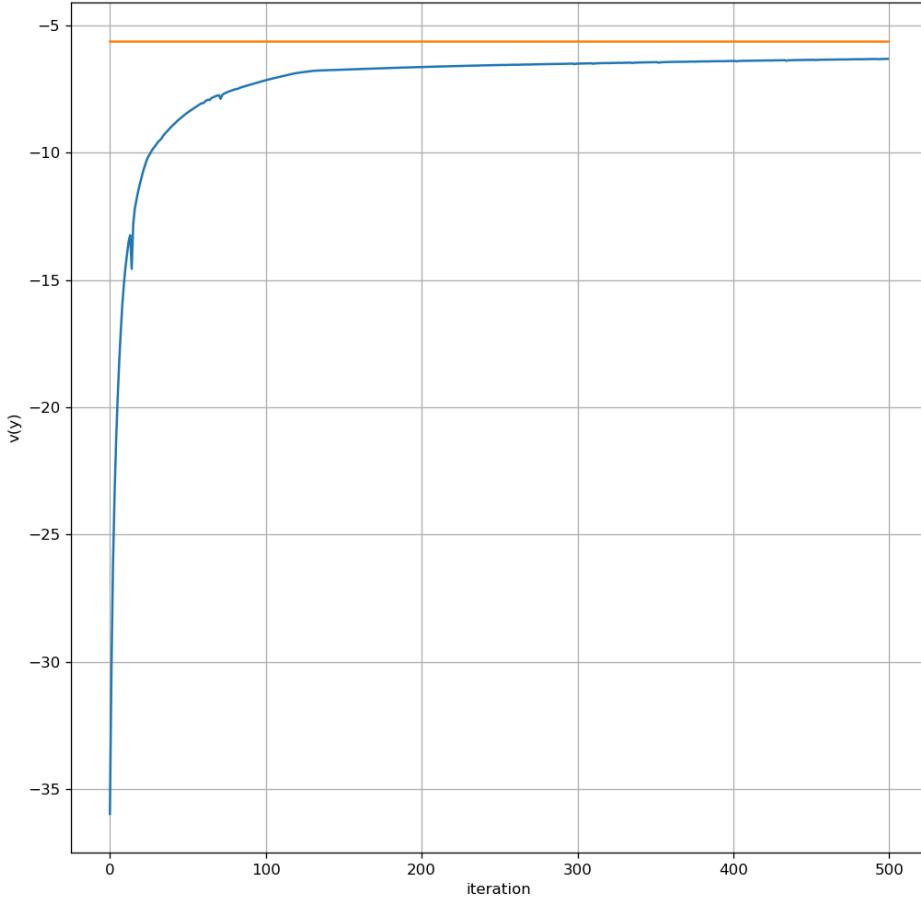


Figure 7.5: Example: Projected subgradient optimization with better step sizes

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

$$\begin{aligned} z := \min \quad & \sum_j x_j \\ \text{subject to} \quad & \sum_j A_j x_j \geq d; \\ & x_j \geq 0 \text{ integer, } \forall j. \end{aligned} \tag{CSP}$$

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

$$\begin{aligned} \underline{z} := \min \quad & \sum_j x_j \\ & \sum_j A_j x_j - t = d; \\ & x_j \geq 0, \forall j; \\ & t \geq \mathbf{0}. \end{aligned} \tag{CSP}$$

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 7.11

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

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

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.

$$\begin{aligned} \underline{z} := \min \quad & \sum_j x_j && \text{dual variables} \\ & \sum_j A_j x_j - t = d; && y \\ & x_j \geq 0, \forall j; && \\ & t \geq \mathbf{0}. && \end{aligned} \tag{CSP}$$

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.

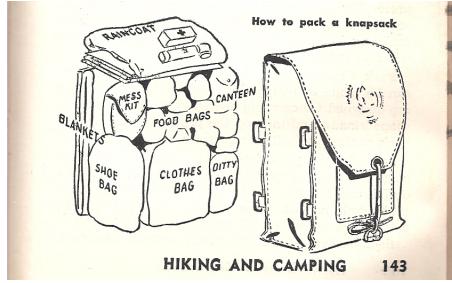
So, moving forward, we may assume that $\bar{y}_i \geq 0$ for all i . We now want to examine the reduced cost of an x_j variable. The reduced cost is simply

$$1 - \bar{y}' A_j = 1 - \sum_{i=1}^m \bar{y}_i a_{ij}.$$

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 **knap-sack problem**

$$\begin{aligned} \max \quad & \sum_{i=1}^m \bar{y}_i a_{ij} \\ & \sum_{i=1}^m w_i a_{ij} \leq W; \\ & a_i \geq 0 \text{ integer}, i = 1, \dots, m, \end{aligned}$$

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*¹⁰). We will solve this problem for all positive integers up through W . That is, we will solve

$$f(s) := \max \quad \begin{array}{l} \sum_{i=1}^m \bar{y}_i a_i \\ \sum_{i=1}^m w_i a_i \leq s; \\ a_i \geq 0 \text{ integer}, i = 1, \dots, m, \end{array}$$

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 $i = 1, \dots, m$.
1. While ($s > 0$)
 - (a) Find $\hat{i} : f(s) = \bar{y}_{\hat{i}} + f(s - w_{\hat{i}})$.
 - (b) Let $a_{\hat{i}} := a_{\hat{i}} + 1$.
 - (c) Let $s := s - w_{\hat{i}}$.
2. Return a_i , for $i = 1, \dots, m$.

Note that in Step 1.a, there must be such an \hat{i} , by virtue of the recursive formula for calculating $f(s)$. In fact, if we like, we can save an appropriate \hat{i} 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 $A_i := \lfloor W/w_i \rfloor e_i$, for $i = 1, 2, \dots, m$. 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 $-e_i$.

Organizing the basic variables x_j and t_i into a vector ζ , with their order appropriately matched with the columns of B , the vector $\bar{\zeta}$ of values of ζ 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 ξ , 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 Python/Gurobi, 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 Gurobi, 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 Gurobi does give us a dual solution, so we do not compute that ourselves. Our full code is in the Jupyter notebook [CSP.ipynb](#) (see Appendix A.10).

On the example provided, 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.

```
***** Solving LP...
***** A:
[[1. 0. 0. 0. 0.]
 [0. 2. 0. 0. 0.]
 [0. 0. 2. 0. 0.]
 [0. 0. 0. 4. 0.]
 [0. 0. 0. 0. 3.]]
***** x:
x[ 0 ]= 205.0
x[ 1 ]= 1160.5
x[ 2 ]= 71.5
x[ 3 ]= 272.25
x[ 4 ]= 39.0
```

```

***** y': [1.      0.5     0.5     0.25    0.3333]

***** Solving Knapsack...
***** Gurobi Knap objval: 1.5
***** DP Knap objval:     1.5
***** Column: [1. 1. 0. 0. 0.]

***** Solving LP...
***** A:
[[1. 0. 0. 0. 0. 1.]
 [0. 2. 0. 0. 0. 1.]
 [0. 0. 2. 0. 0. 0.]
 [0. 0. 0. 4. 0. 0.]
 [0. 0. 0. 0. 3. 0.]]
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 1058.0
x[ 2 ]= 71.5
x[ 3 ]= 272.25
x[ 4 ]= 39.0
x[ 5 ]= 205.0
***** y': [0.5     0.5     0.5     0.25    0.3333]

***** Solving Knapsack...
***** Gurobi Knap objval: 1.25
***** DP Knap objval:     1.25
***** Column: [0. 2. 0. 1. 0.]

***** Solving LP...
***** A:
[[1. 0. 0. 0. 0. 1. 0.]
 [0. 2. 0. 0. 0. 1. 2.]
 [0. 0. 2. 0. 0. 0. 0.]
 [0. 0. 0. 4. 0. 0. 1.]
 [0. 0. 0. 0. 3. 0. 0.]]
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 71.5
x[ 3 ]= 7.75
x[ 4 ]= 39.0
x[ 5 ]= 205.0
x[ 6 ]= 1058.0
***** y': [0.625   0.375   0.5     0.25    0.3333]

***** Solving Knapsack...
***** Gurobi Knap objval: 1.0833333333333333
***** DP Knap objval:     1.0833333333333333
***** Column: [0. 0. 0. 3. 1.]

***** Solving LP...
***** A:

```

```

[[1. 0. 0. 0. 0. 1. 0. 0.]
 [0. 2. 0. 0. 0. 1. 2. 0.]
 [0. 0. 2. 0. 0. 0. 0. 0.]
 [0. 0. 0. 4. 0. 0. 1. 3.]
 [0. 0. 0. 0. 3. 0. 0. 1.]]
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 71.5
x[ 3 ]= 0.0
x[ 4 ]= 35.5556
x[ 5 ]= 205.0
x[ 6 ]= 1058.0
x[ 7 ]= 10.3333
***** y': [0.6111 0.3889 0.5      0.2222 0.3333]

***** Solving Knapsack...
***** Gurobi Knap objval: 1.0555555555555556
***** DP Knap objval:      1.0555555555555556
***** Column: [0. 1. 0. 0. 2.]

***** Solving LP...
***** A:
[[1. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 2. 0. 0. 0. 1. 2. 0. 1.]
 [0. 0. 2. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 4. 0. 0. 1. 3. 0.]
 [0. 0. 0. 0. 3. 0. 0. 1. 2.]]
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 71.5
x[ 3 ]= 0.0
x[ 4 ]= 0.0
x[ 5 ]= 205.0
x[ 6 ]= 1033.3846
x[ 7 ]= 18.5385
x[ 8 ]= 49.2308
***** y': [0.6154 0.3846 0.5      0.2308 0.3077]

***** Solving Knapsack...
***** Gurobi Knap objval: 1.0
***** DP Knap objval:      1.0
***** No more improving columns
***** Pattern generation complete. Main LP solved to optimality.
***** Total number of patterns generated: 9
***** A:
[[1. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 2. 0. 0. 0. 1. 2. 0. 1.]
 [0. 0. 2. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 4. 0. 0. 1. 3. 0.]
 [0. 0. 0. 0. 3. 0. 0. 1. 2.]]

```

```
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 71.5
x[ 3 ]= 0.0
x[ 4 ]= 0.0
x[ 5 ]= 205.0
x[ 6 ]= 1033.3846
x[ 7 ]= 18.5385
x[ 8 ]= 49.2308
***** Optimal LP objective value: 1377.6538461538462
***** rounds up to: 1378.0 (lower bound on rolls needed)
***** x rounded up:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 72.0
x[ 3 ]= 0.0
x[ 4 ]= 0.0
x[ 5 ]= 205.0
x[ 6 ]= 1034.0
x[ 7 ]= 19.0
x[ 8 ]= 50.0
***** Number of rolls used: 1380.0
```

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.

```
***** Now solve the ILP over all patterns generated to try and get a better soution...
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 72.0
x[ 3 ]= 1.0
x[ 4 ]= 1.0
x[ 5 ]= 205.0
x[ 6 ]= 1034.0
x[ 7 ]= 17.0
x[ 8 ]= 49.0
***** Number of rolls used: 1379.0
```

It remains unknown as to whether the optimal solution to this instance is 1378 or 1379.

7.4 Exercises

Exercise 7.1 (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 7.2 (Lagrangian value function)

Using Theorem 6.5, prove that v (from Section 7.2.1) is a concave piecewise-linear function on its domain.

Exercise 7.3 (Play with subgradient optimization)

Play with the Gurobi/Python code in the Jupyter notebook [SubgradProj.ipynb](#) (see Appendix A.9). Try bigger examples. Try different ideas for the step size, with the goal of gaining faster convergence — be a real engineer and think "outside of the box" (you can use any information you like: e.g., the current subgradient \hat{y}^k , the current function value $v(\hat{\gamma}^k)$, an estimate \bar{v} of the maximum value of v , etc.).

Exercise 7.4 (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 [CSP.ipynb](#) that I provided (see Appendix A.10) to try this out.

Exercise 7.5 (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:

- Develop some elementary facility with modeling using integer variables;
- Learn how to recognize when we can expect solutions of linear-optimization problems to be integer automatically;
- Learn the fundamentals of the ideas that most solvers employ to handle integer variables;
- 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. Recapitulating a bit from Section 2.3, 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** $h(e) \in \mathcal{N}$, both nodes. 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 all $e \in \mathcal{A}$. We assume that the 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

$$\sum_{e \in \mathcal{A}} c_e \hat{x}_e.$$

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. This is the $K = 1$ commodity version of the multi-commodity min-cost network-flow problem from Section 2.3.

We can formulate the single-commodity min-cost network-flow problem as follows:

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

As we have stated this, it is just a structured linear-optimization problem. But there are many situations where the given net supplies at the nodes and the given flow capacities on the arcs are integer, and we wish to constrain the flow variables to be integers.

We will see that 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

$$a_{ve} := \begin{cases} 1, & \text{if } v = t(e); \\ -1, & \text{if } v = h(e); \\ 0, & \text{if } v \notin \{t(e), h(e)\}, \end{cases}$$

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

$$\begin{aligned} \min \quad & c'x \\ Ax \quad &= b; \\ x \quad &\leq u; \\ x \quad &\geq \mathbf{0}. \end{aligned}$$

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 **edges**, each one of which is an ordered pair 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 each vertex of the graph meets exactly one edge in M . We assume that there are given **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 all $(i, j) \in E$. Then we can model the problem of finding a perfect matching of G having minimum weight via the formulation:

$$\begin{aligned} \min \quad & \sum_{(i,j) \in E} c_{ij} x_{ij} \\ & \sum_{\substack{j \in V_2 : \\ (i,j) \in E}} x_{ij} = 1, \quad \forall i \in V_1; \\ & \sum_{\substack{i \in V_1 : \\ (i,j) \in E}} x_{ij} = 1, \quad \forall j \in V_2; \\ & x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E. \end{aligned}$$

It will be useful to think of this 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

$$\begin{aligned} \min \quad & c'x \\ Ax &= \mathbf{1}; \\ x &\in \{0, 1\}^E. \end{aligned}$$

Staffing problem. In this problem, we have discrete time periods numbered $1, 2, \dots, m$, and we are given

$$b_i := \text{the minimum number of workers required at time period } i,$$

for each $i = 1, 2, \dots, m$. Additionally, there is an allowable set of "shifts". An allowable shift is simply a given collection of time periods that a worker is allowed to staff. It may well be that not all subsets of $\{1, 2, \dots, m\}$ are allowable; e.g., we may not want to allow too many or too few time periods, and we may not want to allow idle time to be interspersed between non-idle times. We suppose that the allowable shifts are numbered $1, 2, \dots, n$, and we have

$$c_j := \text{the per-worker cost to staff shift } j,$$

for each $j = 1, 2, \dots, n$. 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 $i = 1, 2, \dots, m$, $j = 1, 2, \dots, n$. Letting x be an n -vector of variables, with x_j representing the number of workers assigned to shift j , we can formulate the staffing problem as

$$\begin{array}{ll} \min & c'x \\ Ax & \geq b; \\ x & \geq \mathbf{0} \text{ integer.} \end{array}$$



As we have stated it, this 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 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. This has important implications for the network-flow, assignment, and staffing problems that we introduced.

Let A be an $m \times n$ real matrix. A basis matrix A_β 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 8.1

If A is an integer matrix, all basis matrices of A are unimodular, and b is an integer vector, then every basic solution \bar{x} of

$$\begin{array}{ll} Ax & = b; \\ x & \geq \mathbf{0} \end{array}$$

is an integer vector.

Proof. Of course $\bar{x}_{\eta_j} = 0$, an integer, for $j = 1, 2, \dots, n - m$, so we concentrate now on the basic variables. By Cramer's rule, the basic variables take on the values

$$\bar{x}_{\beta_i} = \frac{\det(A_\beta(i))}{\det(A_\beta)}, \text{ for } i = 1, 2, \dots, m,$$

where $A_\beta(i)$ is defined to be the matrix A_β with its i -th column, A_{β_i} , replaced by b . Because we assume that A and b are all 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_β is ± 1 . That is, the denominator above is ± 1 . So, we have an integer divided by ± 1 , which results in an integer value for \bar{x}_{β_i} . \square

We note that Theorem 8.1 asserts that all basic solutions are integer, whether or not they are feasible. There is a converse to this theorem.

Theorem 8.2

Let A be an integer matrix in $\mathbb{R}^{m \times n}$. If the system

$$\begin{aligned} Ax &= b; \\ x &\geq 0 \end{aligned}$$

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 8.2 is weaker than the conclusion of Theorem 8.1. For Theorem 8.2, we only require integrality for basic *feasible* solutions.

Proof. (Theorem 8.2). Let β be an arbitrary basis, choose an arbitrary i ($1 \leq i \leq m$), and consider the associated basic solution when $b := e_i + \Delta A_\beta \mathbf{1}$. The basic solution \bar{x} has \bar{x}_β equal to the i -th column of A_β^{-1} plus $\Delta 1$. Note that if we choose Δ to be an integer, then b is integer. Furthermore, if we choose Δ to be sufficiently large, then \bar{x}_β is non-negative. Therefore, we can choose Δ 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_β^{-1} plus $\Delta 1$ is an integer vector. But this clearly implies that the i -th column of A_β^{-1} is an integer vector. Now, because i was arbitrary, we conclude that A_β^{-1} is an integer matrix. Of course A_β 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_β and A_β^{-1} are reciprocals. Of course the only integers with integer reciprocal are 1 and -1 . Therefore, the determinant of A_β is 1 or -1 . \square

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.

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 $0, \pm 1$, because the determinant of a 1×1 submatrix is just its single entry. It is quite easy to make an example of even a 2×2 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 1×1 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 8.3

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. \square

Next, we point out some simple transformations that preserve the TU property.

Theorem 8.4

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 8.5

Relationship with transformations of linear-optimization problems. The significance of Theorem 8.4 for linear-optimization problems 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-sensed 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 8.6

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 ± 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 A have determinant ± 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 ± 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 ± 1 . \square

Corollary 8.7

The *single*-commodity min-cost network-flow formulation

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

has an integer optimal solution if: (i) it has an optimal solution, (ii) each b_v is an integer, and (iii) each u_e is an integer or is infinite.

Proof. Recall that we can rewrite the *single*-commodity min-cost network-flow formulation as

$$\begin{aligned} \min \quad & c'x \\ Ax \quad &= b; \\ x \quad &\leq u; \\ x \quad &\geq 0, \end{aligned}$$

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:

$$\begin{aligned} \min \quad & c'x \\ Ax \quad &= b; \\ x + s \quad &= u; \\ x, s \quad &\geq 0. \end{aligned}$$

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. \square

Remark 8.8

Considering Example 2.1, we can see that Corollary 8.7 does not extend to more than one commodity.

Assignments.

Theorem 8.9

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 8.6, the result follows. \square

Corollary 8.10

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.

$$\begin{aligned} \min \quad & \sum_{(i,j) \in E} c_{ij} x_{ij} \\ & \sum_{\substack{j \in V_2 : \\ (i,j) \in E}} x_{ij} = 1, \quad \forall i \in V_1; \\ & \sum_{\substack{i \in V_1 : \\ (i,j) \in E}} x_{ij} = 1, \quad \forall j \in V_2; \\ & x_{ij} \geq 0, \quad \forall (i,j) \in E. \end{aligned}$$

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. \square

A **matching** M of G is a subset of E such that each vertex of the graph is met by *no more than one edge* in M . 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, due to the mathematician König¹¹, tells us that the answer is 'yes' when G is bipartite.

Corollary 8.11 (König's Theorem)

If G is a 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:

$$\begin{aligned} \max \quad & \sum_{(i,j) \in E} x_{ij} \\ & \sum_{\substack{j \in V_2 : \\ (i,j) \in E}} x_{ij} \leq 1, \quad \forall i \in V_1 ; \\ & \sum_{\substack{i \in V_1 : \\ (i,j) \in E}} x_{ij} \leq 1, \quad \forall j \in V_2 ; \\ & x_{ij} \geq 0 \quad \text{integer}, \quad \forall (i,j) \in E. \end{aligned}$$

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 resulting linear-optimization problem is

$$\begin{aligned} \min \quad & \sum_{v \in V} y_v \\ & y_i + y_j \geq 1, \quad \forall (i,j) \in E ; \\ & y_v \geq 0, \quad \forall v \in V. \end{aligned}$$

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 $v \in V$, then decreasing \bar{y}_v to 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 \hat{y} be an optimal 0, 1-valued solution. Let

$$W := \{v \in V : \hat{y}_v = 1\}.$$

It is easy to see that W is a vertex cover of G and that $|W| = \sum_{v \in V} \hat{y}_v$. The result now follows from the strong duality theorem. \square

For studying matching in *non-bipartite graphs*, one can have a look at [3, Chapter 4].

Staffing.

Theorem 8.12

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 ± 1 . (We will see that the ‘*all*’ in the inductive hypothesis will be needed — it will not be enough to consider just $(r - 1) \times (r - 1)$ invertible submatrices of our given matrix A).

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

$$\left(\begin{array}{c|cc|c} 1 & 1 \cdots 1 & 0 \cdots 0 \\ 1 & 1 \cdots 1 & \\ \vdots & \vdots & \vdots \\ 1 & 1 \cdots 1 & \\ \hline 0 & & & \\ \vdots & F & & \\ 0 & & & \end{array} \right) G,$$

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

$$\left(\begin{array}{c|cc|c} 1 & 1 \cdots 1 & 0 \cdots 0 \\ \hline 0 & 0 \cdots 0 & \\ \vdots & \vdots & \vdots \\ 0 & 0 \cdots 0 & \\ \hline 0 & F & G \\ \vdots & & \\ 0 & & \end{array} \right).$$

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,

$$\left(\begin{array}{c|c} 0 \cdots 0 & \\ \vdots & \vdots \\ 0 \cdots 0 & \\ \hline F & G \end{array} \right).$$

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 our inductive hypothesis, its determinant is ± 1 .

□

Corollary 8.13

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

$$\begin{aligned} \min \quad & c'x \\ Ax & \geq b; \\ x & \geq \mathbf{0} \end{aligned}$$

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 8.14

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.

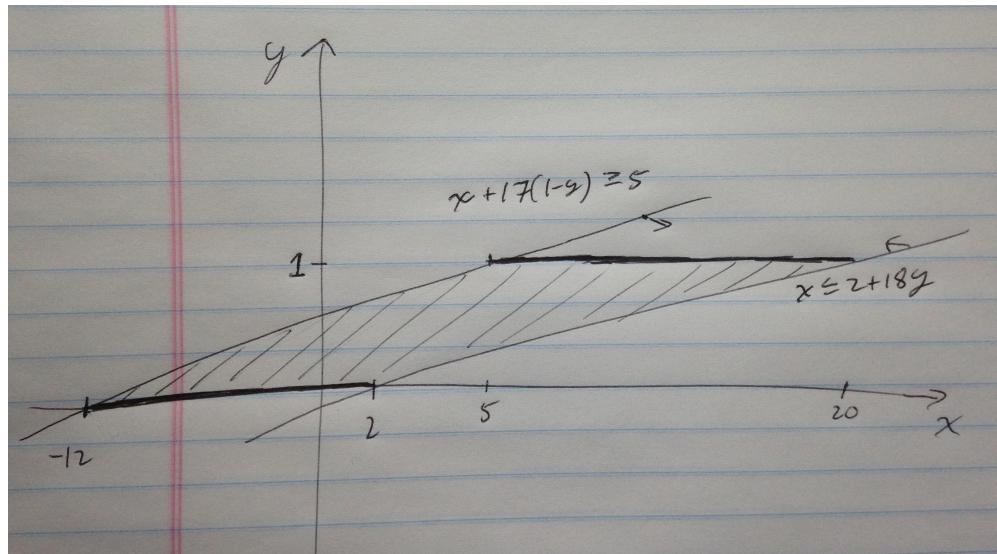


Figure 8.1: Optimal choice of "big M's"

8.2.2 Forcing constraints

The **uncapacitated facility-location problem** involves n customers, numbered $1, 2, \dots, n$ and m facilities, numbered $1, 2, \dots, m$. Associated with each facility, we have

$$f_i := \text{fixed cost for operating facility } i,$$

for $i = 1, \dots, m$. Associated with each customer/facility pair, we have

$$c_{ij} := \text{cost for satisfying all of customer } j's \text{ demand from facility } i,$$

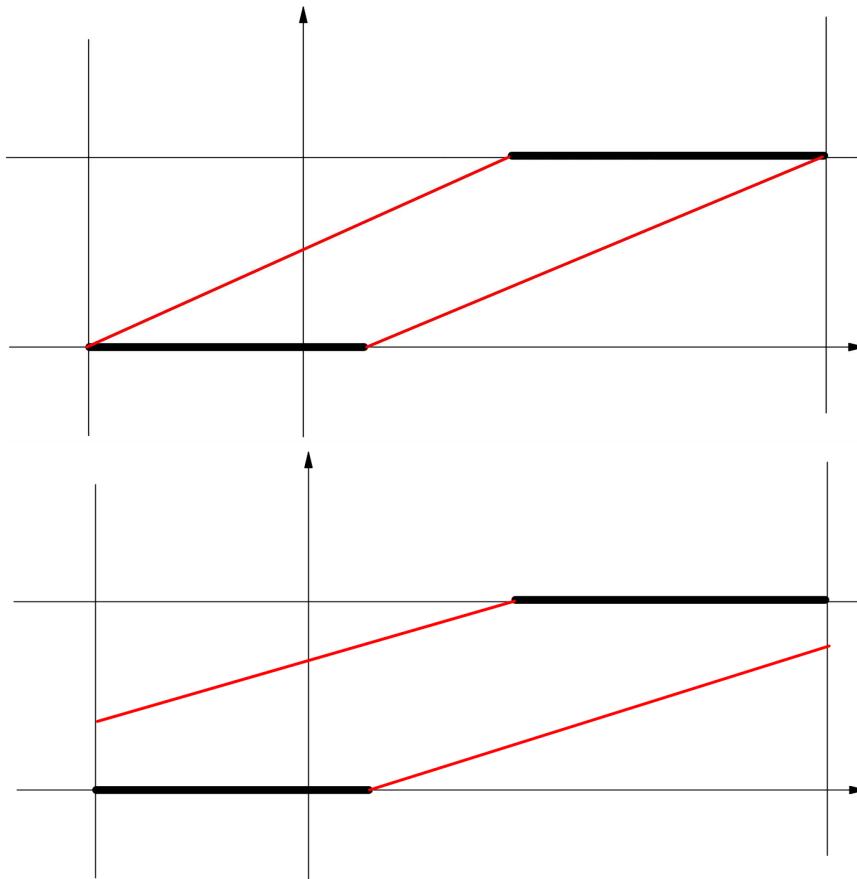


Figure 8.2: Comparing optimal vs non-optimal "big M's"

for $i = 1, \dots, m, j = 1, \dots, n$. The goal is to determine a set of facilities to operate and an allocation of each customer's demand across operating 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

$y_i :=$ indicator variable for operating facility i ,

for $i = 1, \dots, m$, and

$x_{ij} :=$ fraction of customer j demand satisfied by facility i ,

for $i = 1, \dots, m, j = 1, \dots, n$.

Our formulation is as follows:

$$\begin{aligned}
 \min \quad & \sum_{i=1}^m f_i y_i + \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\
 & \sum_{i=1}^m x_{ij} = 1, \quad \text{for } j = 1, \dots, n; \\
 & -y_i + x_{ij} \leq 0, \quad \begin{array}{l} \text{for } i = 1, \dots, m, \\ j = 1, \dots, n; \end{array} \\
 & y_i \in \{0, 1\}, \quad \text{for } i = 1, \dots, m; \\
 & x_{ij} \geq 0, \quad \begin{array}{l} \text{for } i = 1, \dots, m, \\ j = 1, \dots, n. \end{array}
 \end{aligned}$$

All of these constraints are self-explanatory except for the mn constraints:

$$-y_i + x_{ij} \leq 0, \quad \text{for } i = 1, \dots, m, \quad j = 1, \dots, n. \quad (\text{S})$$

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:

$$-ny_i + \sum_{j=1}^n x_{ij} \leq 0, \quad \text{for } i = 1, \dots, m. \quad (\text{W})$$

We can view the coefficient $-n$ of y_i as a "big M", rendering the constraint vacuous when $y_i = 1$. Despite the apparent parsimony of the latter formulation, it turns out that the original formulation is preferred. The Python/Gurobi code is in Jupyter notebook [UFL.ipynb](#) can be used to compare the use of (S) versus (W). (see Appendix A.11).

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 the domain of $f : \mathbb{R} \rightarrow \mathbb{R}$ is the interval $[l, u]$, with $l < u$. For some $n \geq 2$, we choose n **breakpoints** $l = \xi^1 < \xi^2 < \dots < \xi^{n-1} < \xi^n = u$. Then, we approximate f linearly between *adjacent* pairs of breakpoints. That is, we approximate f by

$$\hat{f}(x) := \sum_{j=1}^n \lambda_j f(\xi^j),$$

where we require that

$$\begin{aligned}
 \sum_{j=1}^n \lambda_j &= 1; \\
 \lambda_j &\geq 0, \quad \text{for } j = 1, \dots, n,
 \end{aligned}$$

and **the adjacency condition**:

if λ_j and λ_k are positive for a pair $j < k$, then $k = j + 1$.

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

The situation is depicted in Figure 8.3, where the red curve graphs the non-linear function f .

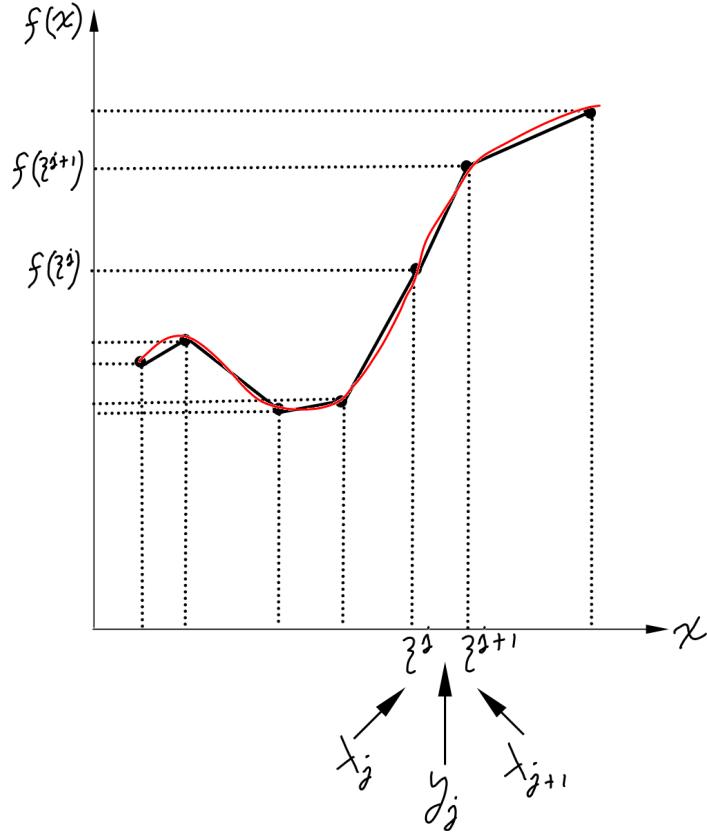


Figure 8.3: Piecewise-linear approximation

We only want to allow one of the $n - 1$ intervals to be activated, so we use the constraint

$$\sum_{j=1}^{n-1} y_j = 1.$$

We only want to allow $\lambda_1 > 0$ if the first interval $[\xi^1, \xi^2]$ is activated. For an internal breakpoint ξ^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

only want to allow $\lambda_n > 0$ if the last interval $[\xi^{n-1}, \xi^n]$ is activated. We can accomplish these restrictions with the constraints

$$\begin{aligned}\lambda_1 &\leq y_1; \\ \lambda_j &\leq y_{j-1} + y_j, \text{ for } j = 2, \dots, n-1; \\ \lambda_n &\leq y_{n-1}.\end{aligned}$$

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 ($j \neq k$), then only λ_k and λ_{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:

$$\begin{aligned}\sum_{j=1}^n \lambda_j &= 1; \\ \sum_{j=1}^{n-1} y_j &= 1; \\ \lambda_1 &\leq y_1; \\ \lambda_j &\leq y_{j-1} + y_j, \text{ for } j = 2, \dots, n-1; \\ \lambda_n &\leq y_{n-1}; \\ \lambda_j &\geq 0, \text{ for } j = 1, \dots, n; \\ y_j &\in \{0, 1\}, \text{ for } j = 1, \dots, n-1.\end{aligned}$$

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 A Prelude to Algorithms

For reasons that will become apparent, for the purpose of developing algorithms for linear-optimization problems in which some variable are required to be integer, it is convenient to assume that our problem has the form

$$\begin{aligned}z := \max \quad &y' b \\ &y' A \leq c'; \\ &y \in \mathbb{R}^m; \\ &y_i \text{ integer, for } i \in \mathcal{I}.\end{aligned} \tag{D}_{\mathcal{I}}$$

The set $\mathcal{I} \subset \{1, 2, \dots, m\}$ allows for a given subset of the variables to be constrained to be integer. This linear-optimization problem has a non-standard form, but it is convenient that the dual of the continuous relaxation has the standard form

$$\begin{aligned}\min \quad &c' x \\ Ax &= b; \\ x &\geq 0.\end{aligned} \tag{P}$$

To prove that an algorithm for $(D_{\mathcal{I}})$ is finite, it is helpful to assume that the feasible region of the continuous relaxation (D) of $(D_{\mathcal{I}})$ is non-empty and bounded.

We saw in Section 7.3.2 that there are situations in which rounding the solution of a continuous relaxation can yield a good solution to an optimization problem involving integer variables. But generally, this is not the case.

Example 8.15

Consider the problem

$$\begin{aligned} \max \quad & y_2 \\ 2ky_1 + y_2 & \leq 2k; \\ -2ky_1 + y_2 & \leq 0; \\ -y_2 & \leq 0; \\ y_1, y_2 & \text{ integer}, \end{aligned}$$

where $k \geq 1$ is a possibly large positive integer. It is easy to check that $(y_1, y_2) = (0, 0)$ and $(y_1, y_2) = (1, 0)$ are both optimal solutions of this problem, but the optimal solution of the continuous relaxation is $(y_1, y_2) = (\frac{1}{2}, k)$. If we consider rounding y_1 up or down in the continuous solution, we do not get a feasible solution, and moreover we are quite far from the optimal solutions.

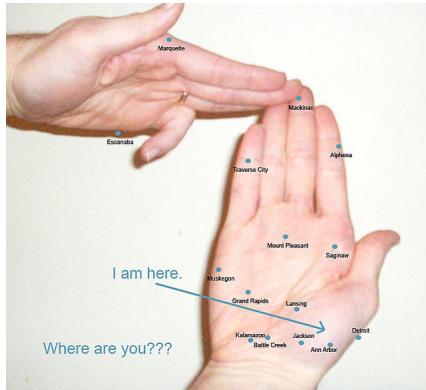
Example 8.16

Consider the problem

$$\begin{aligned} \max \quad & \sum_{i=1}^m y_i \\ y_i + y_\ell & \leq 1, \quad \text{for all } 1 \leq i < \ell \leq m; \\ -y_i & \leq 0, \quad \text{for all } 1 \leq i \leq m; \\ y_i & \text{ integer, for all } 1 \leq i \leq m, \end{aligned}$$

where $m \geq 3$ is a possibly large positive integer. It is easy to check that each integer optimal solution sets any single variable to one and the rest to zero, achieving objective value 1. While the (unique) continuous optimal solution sets all variables to $\frac{1}{2}$, achieving objective value $\frac{m}{2}$. We can see that the continuous optimal solution is not closely related to the integer optimal solutions.

8.4 Branch-and-Bound



Next, we look at a rudimentary framework called **branch-and-bound**, which aims at finding an optimal solution of (D_I) , a linear-optimization problem having some integer variables. We assume that (P) , the dual of the continuous relaxation of (D_I) , has a feasible solution.

Our algorithm maintains a list \mathcal{L} of optimization problems that all have the general form of (D_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 \bar{y}_{LB} of (D_I) that we have seen so far. Initially, we set $LB = -\infty$, and we update it in an increasing fashion.

The algorithm maintains the **key invariant for branch-and-bound**:

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}_{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}_{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.)

Because the key invariant for branch-and-bound is maintained by the processing rules, the following result is evident.

Theorem 8.17

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}_{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 β 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_{down} in (\tilde{P}) . The variable x_{down} has a new column $A_{\text{down}} := e_i$ 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}_{down} of x_{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_{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_{up} in (\tilde{P}) . The variable x_{up} has a new column $A_{\text{up}} := -e_i$ 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}_{up} of x_{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_{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{P}) associated with a child $(\tilde{D}_{\mathcal{I}'})$ of that problem, with the new variable of the child's (\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 upper bound on the optimal objective value of $(\tilde{D}_{\mathcal{I}})$. That is, for any such (\tilde{P}) , we start with the upper bound 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{P}) — its optimal objective value will be no greater than LB — and conclude that the optimal objective value of $(\tilde{D}_{\mathcal{I}})$ 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 $\text{LB} = z$. But we can see that

$$z \leq \text{UB} := \max \{\text{UB}_{\text{better}}, \text{LB}\}.$$

It may be useful to have UB at hand, because we can always stop the computation early, say when $\text{UB} - \text{LB} \leq \tau$, returning the feasible solution \bar{y}_{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 problems *before* we put them on the list.

Selecting a problem from the list. Which problem 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 problem 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 problems.

- A strategy of choosing a problem 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 problems 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 problems. 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. Our constraint system $y' A \leq c'$ can be viewed as $y' A_j \leq c_j$, for $j = 1, 2, \dots, n$; that is, c_j is an upper bound on $y' A_j$. We may wonder if we can also derive lower bounds on $y' A_j$.

Theorem 8.18

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

$$\begin{aligned} z(\Delta_j) := \max_{\substack{y' b \\ y' A \\ y \in \mathbb{R}^m \\ y_i \text{ integer, for } i \in \mathcal{I}}} \quad & y' b \\ & y' A \leq c' + \Delta_j e'_j; \\ & y_i \end{aligned} \tag{D_{\mathcal{I}}(\Delta_j)}$$

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 $(D_{\mathcal{I}})$. \square

It is interesting to consider two special cases of Theorem 8.18:

Corollary 8.19 (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 Exercise 5.2, this is known as **reduced-cost fixing**.

8.5 Cutting Planes

This section is adapted from material in [2] and [4]. In fact, those papers were developed to achieve versions of Gomory cutting-plane algorithms (with finiteness proofs) that would mesh with our column-generation treatment of many topics in this book (i.e., cutting stock, decomposition, and branch-and-bound).

8.5.1 Pure

In this section, we assume that all y_i variables are constrained to be integer. That is, $\mathcal{I} = \{1, 2, \dots, m\}$

We can choose any non-negative $w \in \mathbb{R}^n$, and we see that

$$w \geq \mathbf{0} \text{ and } y' A \leq c' \implies y'(Aw) \leq c'w.$$

Note that this inequality is valid for all solutions of $y' A \leq c'$, integer or not. Next, if Aw is integer, we can exploit the integrality of y . We see that

$$Aw \in \mathbb{Z}^m, y \in \mathbb{Z}^m \implies y'(Aw) \leq \lfloor c'w \rfloor,$$

for all integer solutions of $y' A \leq c'$.

The inequality $y'(Aw) \leq \lfloor c'w \rfloor$ is called a **Chvátal-Gomory cut**. The condition $Aw \in \mathbb{Z}^m$ may seem a little awkward, but usually we have that A is integer, so we can get $Aw \in \mathbb{Z}^m$ by then just choosing $w \in \mathbb{Z}^n$. In fact, for the remainder of this section, we will assume that A and c are integer.

Of course, it is by no means clear how to choose appropriate w , 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 8.15 reveals that *infeasible* integer points can "guard" Chvátal-Gomory cuts from getting close to any feasible integer points.

We would like to develop a concrete algorithmic scheme for generating Chvátal-Gomory cuts. We will do this via basic solutions. Let β be any basis for P . The associated dual basic solution (for the continuous relaxation (D)) is $\bar{y}' := c'_\beta A_\beta^{-1}$. Suppose that \bar{y}_i is not an integer. Our goal is to derive a valid cut for $(D_{\mathcal{I}})$ that is violated by \bar{y} .

Let

$$\tilde{b} := \mathbf{e}_i + A_\beta r,$$

where $r \in \mathbb{Z}^m$, and, as usual, \mathbf{e}_i denotes the i -th standard unit vector in \mathbb{R}^m . Note that by construction, $\tilde{b} \in \mathbb{Z}^m$.

Theorem 8.20

$\bar{y}'\tilde{b}$ is not an integer, and so $y'\tilde{b} \leq \lfloor \bar{y}'\tilde{b} \rfloor$ cuts off \bar{y} .

$$\text{Proof. } \bar{y}'\tilde{b} = \bar{y}'(\mathbf{e}_i + A_\beta r) = \bar{y}_i + (c'_\beta A_\beta^{-1})A_\beta r = \underbrace{\bar{y}_i}_{\notin \mathbb{Z}} + \underbrace{c'_\beta r}_{\in \mathbb{Z}} \notin \mathbb{Z}. \quad \square$$

At this point, we have an inequality $y'\tilde{b} \leq \lfloor \bar{y}'\tilde{b} \rfloor$ which cuts off \bar{y} , but we have not established its validity for $(D_{\mathcal{I}})$.

Let $H_{\cdot i} := A_\beta^{-1}\mathbf{e}_i$, the i -th column of A_β^{-1} . Now let

$$w := H_{\cdot i} + r.$$

Clearly we can choose $r \in \mathbb{Z}^m$ so that $w \geq 0$; we simply choose $r \in \mathbb{Z}^m$ so that

$$r_k \geq -\lfloor h_{ki} \rfloor, \text{ for } k = 1, \dots, m. \quad (*_{\geq})$$

Theorem 8.21

Choosing $r \in \mathbb{Z}^m$ satisfying $(*_{\geq})$, we have that $y'\tilde{b} \leq \lfloor \bar{y}'\tilde{b} \rfloor$ is valid for $(D_{\mathcal{I}})$.

Proof. Because $w \geq 0$ and $y'A \leq c'$, we have the validity of

$$y'A_\beta(A_\beta^{-1}\mathbf{e}_i + r) \leq c'_\beta(A_\beta^{-1}\mathbf{e}_i + r),$$

even for the continuous relaxation (D) of $(D_{\mathcal{I}})$. Simplifying this, we have

$$y'(\mathbf{e}_i + A_\beta r) \leq \bar{y}_i + c'_\beta r.$$

The left-hand side is clearly $y'\tilde{b}$, and the right-hand side is

$$\bar{y}_i + c'_\beta r = \bar{y}_i + \bar{y}'A_\beta r = \bar{y}'(\mathbf{e}_i + A_\beta r) = \bar{y}'\tilde{b}.$$

So we have that $y'\tilde{b} \leq \bar{y}'\tilde{b}$ is valid even for (D) . Finally, observing that $\tilde{b} \in \mathbb{Z}^m$ and y is constrained to be in \mathbb{Z}^m for $(D_{\mathcal{I}})$, we can round down the right-hand side and get the result. \square

So, given any non-integer basic dual solution \bar{y} , we have a way to produce a valid inequality for $(D_{\mathcal{I}})$ that cuts it off. This cut for $(D_{\mathcal{I}})$ is used as a column for (P) : the column is \tilde{b} with objective coefficient $\lfloor \bar{y}'\tilde{b} \rfloor$. Taking β to be an optimal basis for (P) , the new variable corresponding to this column is the unique variable eligible to enter the basis in the context of the primal simplex algorithm applied to (P) — the reduced cost is precisely

$$\lfloor \bar{y}'\tilde{b} \rfloor - \bar{y}'\tilde{b} < 0.$$

The new column for A is \tilde{b} which is integer. The new objective coefficient for c is $\lfloor \bar{y}'\tilde{b} \rfloor$ which is an integer. So the original assumption that A and c are integer is maintained, and we can repeat. In this way, we get a legitimate cutting-plane framework for $(D_{\mathcal{I}})$ — though we emphasize that we do our computations as column generation with respect to (P) .

There is clearly a lot of flexibility in how r can be chosen. Next, we demonstrate that in a very concrete sense, it is always best to choose a minimal $r \in \mathbb{Z}^m$ satisfying $(*_{\geq})$.

Theorem 8.22

Let $r \in \mathbb{Z}^m$ be defined by

$$r_k := -\lfloor h_{ki} \rfloor, \text{ for } k = 1, \dots, m. \quad (*_=)$$

The the associated cut dominates (on (D)) any cut associated with an r only satisfying $(*_\geq)$.

Proof. It is easy to check that our cut $y' \tilde{b} \leq \lfloor \bar{y}' \tilde{b} \rfloor$ can generally be re-expressed as

$$y_i \leq \lfloor \bar{y}_i \rfloor + (c'_\beta - y' A_\beta) r.$$

Noting that $c'_\beta - y' A_\beta \geq 0$ for all y that are feasible for (D), we see that the strongest inequality is obtained by choosing $r \in \mathbb{Z}^m$ to be minimal (i.e., satisfying $(*_=)$). \square

Example 8.23

We work through an example in [pure_gomory_example_1.ipynb](#) (see Appendix A12) which uses again [pivot_tools.ipynb](#) (see Appendix A.6). The function library [pivot_tools.ipynb](#) contains two (additional) useful tools for this: `pure_gomory()` and `dual_plot(,)`

Let

$$A = \begin{pmatrix} 7 & 8 & -1 & 1 & 3 \\ 5 & 6 & -1 & 2 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 26 \\ 19 \end{pmatrix},$$

and $c' = (126 \ 141 \ -10 \ 5 \ 67).$

So, the integer program (D_I) which we seek to solve is defined by five inequalities in the two variables y_0 and y_1 . For the basis of (P), $\beta = (0, 1)$, we have

$$A_\beta = \begin{pmatrix} 7 & 8 \\ 5 & 6 \end{pmatrix}, \text{ and hence } A_\beta^{-1} = \begin{pmatrix} 3 & -4 \\ -5/2 & 7/2 \end{pmatrix}.$$

It is easy to check that for this choice of basis, we have

$$\bar{x}_\beta = \begin{pmatrix} 2 \\ 3/2 \end{pmatrix},$$

and for the non-basis $\eta = (2, 3, 4, 5)$, we have $\bar{c}'_\eta = (5 \ 1/2 \ 1)$, which are both non-negative, and so this basis is optimal for (P). The associated dual basic solution depicted in Figure 8.4 is

$$\bar{y}' = (51/2 \ -21/2), \text{ and the objective value is } z = 463 1/2.$$

Because both \bar{y}_0 and \bar{y}_1 are not integer, we can derive a cut for (D_I) from either. Recalling the procedure, for any fraction \bar{y}_i , we start with the i -th column $H_{\cdot i}$ of $H := A_\beta^{-1}$, and we get a new $A_{\cdot j} := \mathbf{e}_i + A_\beta r$. Throughout we will choose r via $(*_=)$. So we have,

$$H_{\cdot 0} = \begin{pmatrix} 3 \\ -5/2 \end{pmatrix} \Rightarrow r = \begin{pmatrix} -3 \\ 3 \end{pmatrix} \Rightarrow \tilde{b} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 7 & 8 \\ 5 & 6 \end{pmatrix} \begin{pmatrix} -3 \\ 3 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \end{pmatrix} =: A_{\cdot 5}$$

$$H_{\cdot 1} = \begin{pmatrix} -4 \\ 7/2 \end{pmatrix} \Rightarrow r = \begin{pmatrix} 4 \\ -3 \end{pmatrix} \Rightarrow \tilde{b} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 7 & 8 \\ 5 & 6 \end{pmatrix} \begin{pmatrix} 4 \\ -3 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \end{pmatrix}.$$

In fact, for this iteration of this example, we get the same cut for either choice of i . To calculate the right-hand side of the cut, we have

$$\bar{y}' \tilde{b} = (51/2 \ -21/2) \begin{pmatrix} 4 \\ 3 \end{pmatrix} = 70 1/2,$$

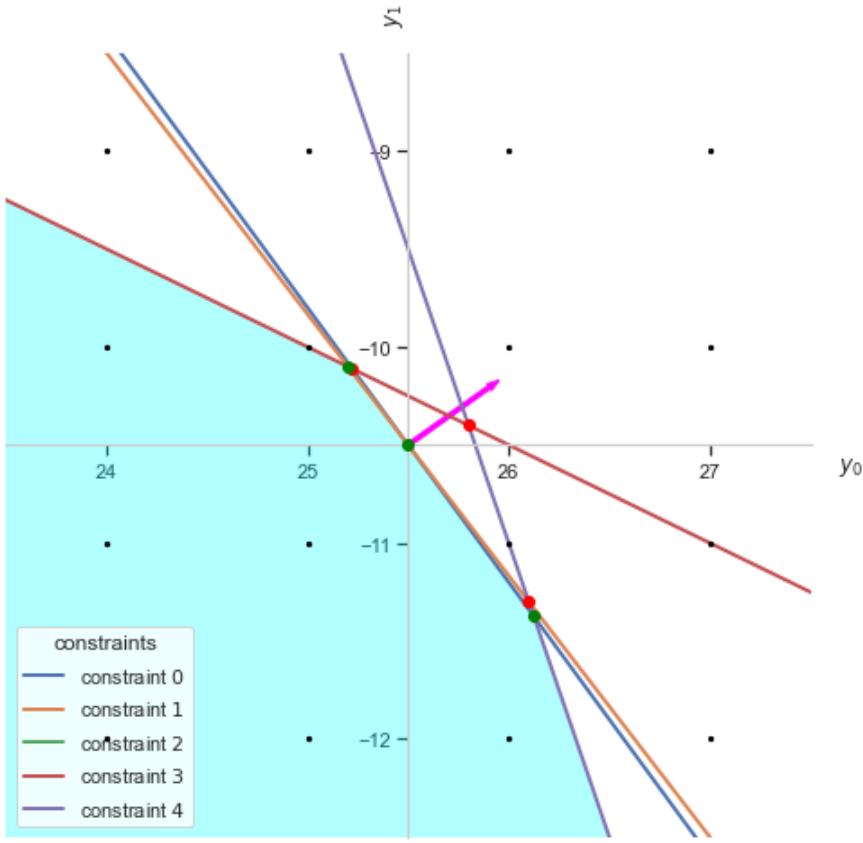


Figure 8.4:

so the cut for $(D_{\mathcal{I}})$ is

$$4y_0 + 3y_1 \leq 70.$$

Now, we do our simplex-method calculations with respect to (P) .

The new column for (P) is $A_{.5}$ (above) with objective coefficient $c_5 := 70$. Following the ratio test, when index 5 enters the basis, index 2 leaves the basis, and so the new basis is $\beta = (0, 5)$, with

$$A_{\beta} = \begin{pmatrix} 7 & 4 \\ 5 & 3 \end{pmatrix},$$

with objective value 462, a decrease. At this point, index 4 has a negative reduced cost, and index 0 leaves the basis. So we now have $\beta = (4, 5)$, which turns out to be optimal.

The associated dual basic solution depicted in Figure 8.5 is

$$\bar{y}' = \left(\begin{array}{c} 131/5 \\ -58/5 \end{array} \right), \text{ and the objective value is } z = 460 4/5.$$

We observe that the objective function has decreased, but unfortunately both \bar{y}_0 and \bar{y}_1 are not integers. So we must continue. We have

$$A_{\beta} = \begin{pmatrix} 3 & 4 \\ 1 & 3 \end{pmatrix}, \text{ and hence } A_{\beta}^{-1} = \begin{pmatrix} 3/5 & -4/5 \\ -1/5 & 3/5 \end{pmatrix}.$$

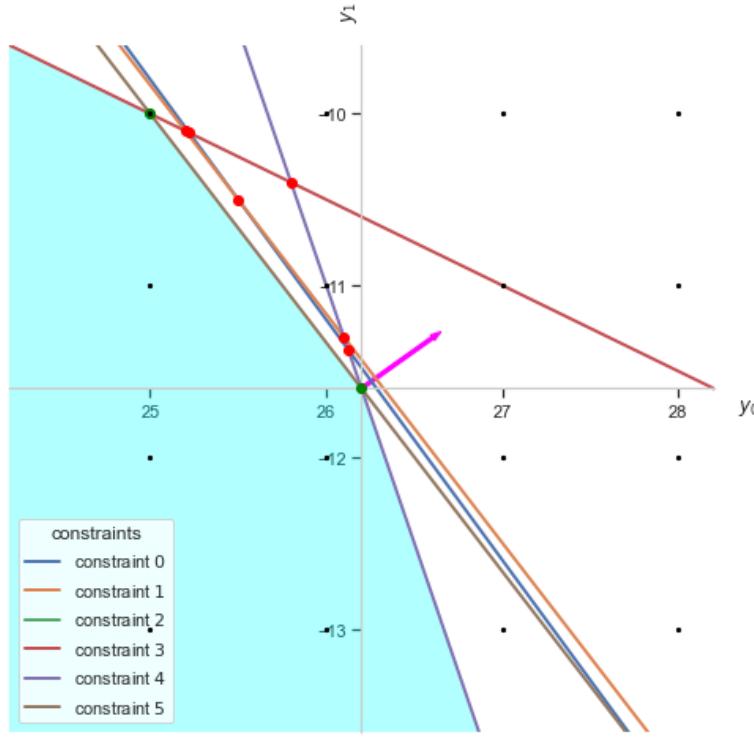


Figure 8.5:

We observe that the objective function has decreased, but because both \bar{y}_0 and \bar{y}_1 are not integers, we can again derive a cut for (D_I) from either. We calculate

$$H_{.0} = \begin{pmatrix} 3/5 \\ -1/5 \end{pmatrix} \Rightarrow r = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Rightarrow \tilde{b} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 3 & 4 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \end{pmatrix} =: A_{.6}$$

$$H_{.1} = \begin{pmatrix} -4/5 \\ 3/5 \end{pmatrix} \Rightarrow r = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow \tilde{b} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 3 & 4 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} =: A_{.7}.$$

Correspondingly, we have $\bar{y}' A_{.6} = 96 \frac{1}{5}$ and $\bar{y}' A_{.7} = 55 \frac{2}{5}$, giving us $c_6 := 96$ and $c_7 := 55$. So, we have two possible cuts for (D_I) :

$$5y_0 + 3y_1 \leq 96 \text{ and } 3y_0 + 2y_1 \leq 55.$$

Choosing to incorporate both as columns for (P) , and letting index 7 enter the basis, index 5 leaves (according to the ratio test), and it turns out that we reach an optimal basis $\beta = (7, 5)$ after this single pivot. The associated dual basic solution is depicted in Figure 8.6 (the second graphic is zoomed in)

$$\bar{y}' = (25 \quad -10), \text{ and the objective value is } z = 460.$$

Not only has the objective decreased, but now all of the \bar{y}_i are integers, so we have an optimal solution for (D_I) .

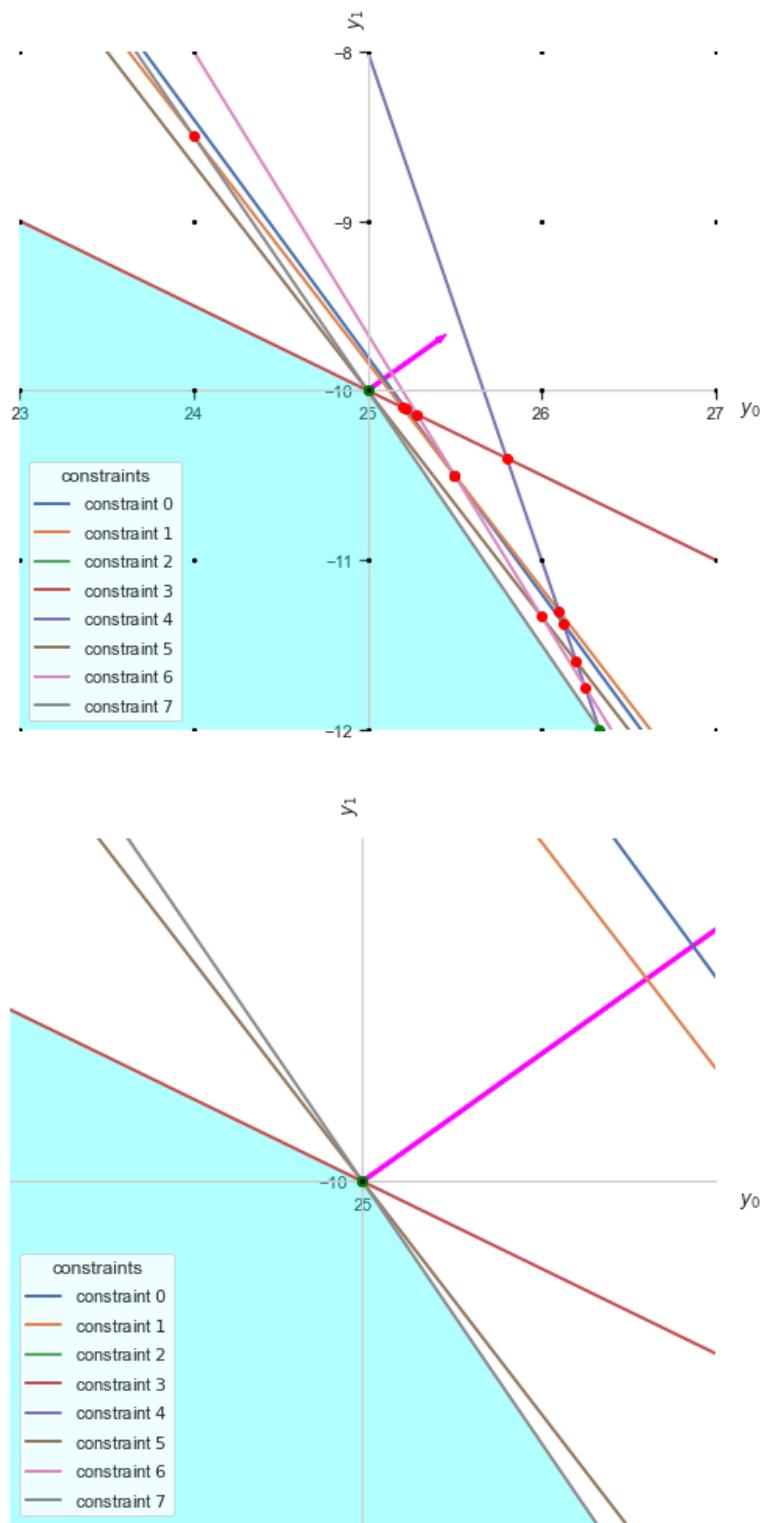


Figure 8.6:

8.5.2 Mixed

In this section, we no longer assume that all y_i variables are constrained to be integer. That is, we only assume that non-empty $\mathcal{I} \subset \{1, 2, \dots, m\}$. The cuts from the previous section cannot be guaranteed to be valid, so we start anew.

Let β be any basis partition for (P), and let \bar{y} be the associated dual basic solution. Suppose that $\bar{y}_i \notin \mathbb{Z}$, for some $i \in \mathcal{I}$. We aim to find a cut, valid for $(D_{\mathcal{I}})$ and violated by \bar{y} .

Let

$$\tilde{b}^1 := \mathbf{e}_i + A_{\beta}r,$$

and $r \in \mathbb{R}^m$ will be determined later. We will accumulate the conditions we need to impose on r , as we go.

Let w^1 be the basic solution associated with the basis β and the "right-hand side" \tilde{b}^1 . So $w_{\beta}^1 = h_{\cdot i} + r$, where $h_{\cdot i}$ is defined as the i -th column of A_{β}^{-1} , and $w_{\eta}^1 = \mathbf{0}$. Choosing $r \geq -h_{\cdot i}$, we can make $w^1 \geq \mathbf{0}$. Moreover, $c'w^1 = c'_{\beta}(h_{\cdot i} + r) = c'_{\beta}h_{\cdot i} + c'_{\beta}r = \bar{y}_i + c'_{\beta}r$, so because we assume that $\bar{y}_i \notin \mathbb{Z}$, we can choose $r \in \mathbb{Z}^m$, and we have that $c'w^1 \notin \mathbb{Z}$.

Next, let

$$\tilde{b}^2 := A_{\beta}r.$$

Let w^2 be the basic solution associated with the basis β and the "right-hand side" \tilde{b}^2 . So, now further choosing $r \geq \mathbf{0}$, we have $w_{\beta}^2 = r \geq \mathbf{0}$, $w_{\eta}^2 = \mathbf{0}$, and $c'w^2 = c'_{\beta}r$.

So, we choose $r \in \mathbb{Z}^m$ so that:

$$r_k \geq \max\{-\lfloor h_{ki} \rfloor, 0\}, \text{ for } k = 1, \dots, m, \quad (*_{\geq+})$$

Because we have chosen w^1 and w^2 to be non-negative, forming $(y' A)w^l \leq c'w^l$, for $l = 1, 2$, we get a pair of valid inequalities for D. They have the form $y'\tilde{b}^l \leq c'w^l$, for $l = 1, 2$. Let α'_j denote the j -th row of A_{β} . Then our inequalities have the form:

$$(1 + \alpha'_i r)y_i + \sum_{j:j \neq i} (\alpha'_j r)y_j \leq \bar{y}_i + \bar{y}' A_{\beta}r, \quad (I1)$$

$$(\alpha'_i r)y_i + \sum_{j:j \neq i} (\alpha'_j r)y_j \leq \bar{y}' A_{\beta}r. \quad (I2)$$

Now, defining $z := \sum_{j:j \neq i} (\alpha'_j r)y_j$, we have the following inequalities in the two variables y_i and z :

$$(1 + \alpha'_i r)y_i + z \leq \bar{y}_i + \bar{y}' A_{\beta}r \quad \begin{array}{c} \text{slope} \\ -1/(1 + \alpha'_i r) \end{array} \quad (B1)$$

$$(\alpha'_i r)y_i + z \leq \bar{y}' A_{\beta}r \quad \begin{array}{c} \text{slope} \\ -1/\alpha'_i r \end{array} \quad (B2)$$

Note that the intersection point (y_i^*, z^*) of the lines associated with these inequalities (subtract the second equation from the first) has $y_i^* = \bar{y}_i$ and $z^* = \sum_{j:j \neq i} (\alpha'_j r)\bar{y}_j$. Also, the "slopes" indicated regard y_i as the ordinate and z as the abscissa.

Bearing in mind that we choose $r \in \mathbb{Z}^m$ and that A is assumed to be integer, we have that $\alpha'_i r \in \mathbb{Z}$. There are now two cases to consider:

- $\alpha'_i r \geq 0$, in which case the first line has negative slope and the second line has more negative slope (or infinite $\alpha'_i r = 0$);
- $\alpha'_i r \leq -1$, in which case the second line has positive slope and the first line has more positive slope (or infinite $\alpha'_i r = -1$).

See Figures 8.7 and 8.8.

In both cases, we are interested in the point (z^1, y_i^1) where the first line intersects the line $y_i = \lfloor \bar{y}_i \rfloor + 1$ and the point (z^2, y_i^2) where the second line intersects the line $y_i = \lfloor \bar{y}_i \rfloor$.

We can check that

$$\begin{aligned} z^1 &= \bar{y}_i + \bar{y}' A_\beta r - (1 + \alpha'_i r) (\lfloor \bar{y}_i \rfloor + 1), \\ z^2 &= \bar{y}' A_\beta r - (\alpha'_i r) \lfloor \bar{y}_i \rfloor. \end{aligned}$$

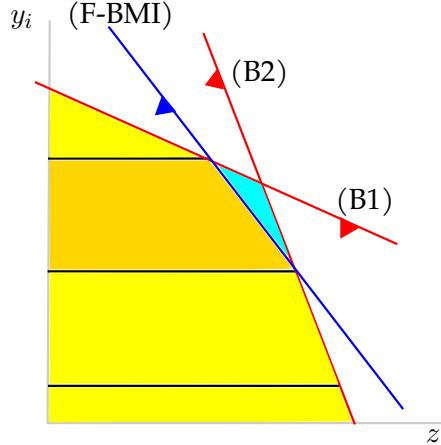
Subtracting, we have

$$z_1 - z_2 = \underbrace{(\bar{y}_i - \lfloor \bar{y}_i \rfloor)}_{\in (0,1)} - (1 + \underbrace{\alpha'_i r}_{\in \mathbb{Z}}),$$

so we see that: $z^1 < z^2$ precisely when $\alpha'_i r \geq 0$; $z^2 < z^1$ precisely when $\alpha'_i r \leq -1$. Moreover, the slope of the line through the pair of points (z^1, y_i^1) and (z^2, y_i^2) is just

$$\frac{1}{z^1 - z^2} = \frac{1}{(\bar{y}_i - \lfloor \bar{y}_i \rfloor) - (1 + \alpha'_i r)}.$$

Figure 8.7: (F-BMI) cut when $\alpha'_i r \geq 0$



We now define the inequality

$$((\bar{y}_i - \lfloor \bar{y}_i \rfloor) - (1 + \alpha'_i r)) (y_i - \lfloor \bar{y}_i \rfloor) \geq z - \bar{y}' A_\beta r + (\alpha'_i r) \lfloor \bar{y}_i \rfloor,$$

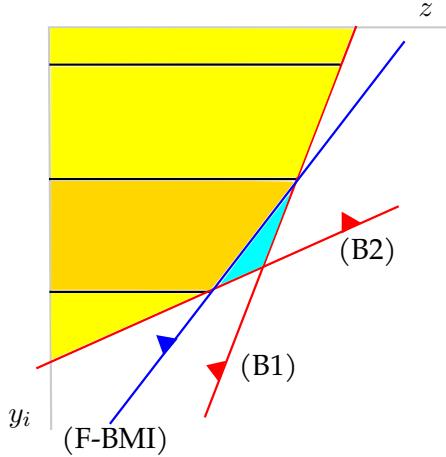
which has the more convenient form

$$((1 + \alpha'_i r) - (\bar{y}_i - \lfloor \bar{y}_i \rfloor)) y_i + z \leq \bar{y}' A_\beta r - (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) \lfloor \bar{y}_i \rfloor. \quad (\text{F-BMI})$$

By construction, we have the following two results.

Lemma 8.24

(F-BMI) is satisfied at equality by both of the points (z^1, y_i^1) and (z^2, y_i^2) .

Figure 8.8: (F-BMI) cut when $\alpha'_i r \leq -1$ **Lemma 8.25**

(F-BMI) is valid for

$$\{(y_i, z) \in \mathbb{R}^2 : (\text{B1}), y_i \geq \lceil \bar{y}_i \rceil\} \cup \{(y_i, z) \in \mathbb{R}^2 : (\text{B2}), y_i \leq \lfloor \bar{y}_i \rfloor\}.$$

Lemma 8.26

(F-BMI) is violated by the point (y_i^*, z^*) .

Proof. Plugging (y_i^*, z^*) into (F-BMI), and making some if-and-only-if manipulations, we obtain

$$(\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1)(\bar{y}_i - \lfloor \bar{y}_i \rfloor) \geq 0,$$

which is not satisfied. \square

Finally, translating (F-BMI) back to the original variables $y \in \mathbb{R}^m$, we get

$$((1 + \alpha'_i r) - (\bar{y}_i - \lfloor \bar{y}_i \rfloor)) y_i + \sum_{j:j \neq i} (\alpha'_j r) y_j \leq \bar{y}' A_\beta r - (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) \lfloor \bar{y}_i \rfloor,$$

or,

$$-(\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) y_i + y' A_\beta r \leq \bar{y}' A_\beta r - (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) \lfloor \bar{y}_i \rfloor,$$

which, finally has the convenient form

$$y' (A_\beta r - (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) e_i) \leq c'_\beta r - (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) \lfloor \bar{y}_i \rfloor. \quad (\text{F-GMI})$$

We immediately have:

Theorem 8.27

(F-GMI) is violated by the point \bar{y} .

Finally, we have:

Theorem 8.28

(F-GMI) is valid for the following relaxation of the feasible region of (D):

$$\{y \in \mathbb{R}^m : y' A_\beta \leq c'_\beta, y_i \geq \lceil \bar{y}_i \rceil\} \cup \{y \in \mathbb{R}^m : y' A_\beta \leq c'_\beta, y_i \leq \lfloor \bar{y}_i \rfloor\}.$$

Proof. The proof, maybe obvious, is by a simple disjunctive argument. We will argue that (F-BMI) is valid for both $S_1 := \{y \in \mathbb{R}^m : y' A_\beta \leq c'_\beta, -y_i \leq -\lfloor \bar{y}_i \rfloor - 1\}$ and $S_2 := \{y \in \mathbb{R}^m : y' A_\beta \leq c'_\beta, y_i \leq \lfloor \bar{y}_i \rfloor\}$.

The inequality (F-BMI) is simply the sum of (B1) and the scalar $\bar{y}_i - \lfloor \bar{y}_i \rfloor$ times $-y_i \leq -\lfloor \bar{y}_i \rfloor - 1$. It follows then that taking (I1) plus $\bar{y}_i - \lfloor \bar{y}_i \rfloor$ times $-y_i \leq -\lfloor \bar{y}_i \rfloor - 1$, we get an inequality equivalent to (F-GMI).

Similarly, it is easy to check that the inequality (F-BMI) is simply (B2) plus $1 - (\bar{y}_i - \lfloor \bar{y}_i \rfloor)$ times $y_i \leq \lfloor \bar{y}_i \rfloor$. It follows then that taking (I2) plus $1 - (\bar{y}_i - \lfloor \bar{y}_i \rfloor)$ times $y_i \leq \lfloor \bar{y}_i \rfloor$, we also get an inequality equivalent to (F-GMI). \square

In our algorithm, we append columns to (P), rather than cuts to (D). The column for (P) corresponding to (F-GMI) is

$$A_\beta r - (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) e_i,$$

and the associated cost coefficient is

$$c'_\beta r - (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) \lfloor \bar{y}_i \rfloor.$$

So A_β^{-1} times the column is

$$r - (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) h_{\cdot i}.$$

Agreeing with what we calculated in Proposition 8.26, we have the following result.

Proposition 8.29

The reduced cost of the column for (P) corresponding to (F-GMI) is

$$(\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) (\bar{y}_i - \lfloor \bar{y}_i \rfloor) < 0.$$

Proof.

$$\begin{aligned} c'_\beta r - (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) \lfloor \bar{y}_i \rfloor &- c'_\beta (r - (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) h_{\cdot i}) \\ &= (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) (c'_\beta h_{\cdot i} - \lfloor \bar{y}_i \rfloor) \\ &= (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1) (\bar{y}_i - \lfloor \bar{y}_i \rfloor). \end{aligned}$$

\square

Next, we come to the choice of r .

Theorem 8.30

Let $r \in \mathbb{Z}^m$ be defined by

$$r_k = \max\{0, -\lfloor h_{ki} \rfloor\}, \text{ for } k = 1, 2, \dots, m, \quad (*_{=+})$$

and suppose that $\hat{r} \in \mathbb{Z}^m$ satisfies $(*_{\geq+})$ and $r \leq \hat{r}$. Then the cut determined by r dominates the cut determined by \hat{r} .

Proof. We simply rewrite (F-GMI) as

$$(c'_\beta - y' A_\beta)r \geq (\bar{y}_i - \lfloor \bar{y}_i \rfloor - 1)(\lfloor \bar{y}_i \rfloor - y_i).$$

Observing that $c'_\beta - y' A_\beta \geq 0$ for y that are feasible for (D), we see that the tightest inequality of this type, satisfying $(*_{\geq+})$, arises by choosing a minimal r . The result follows. \square

8.5.3 Finite termination

Making a version of our Gomory cutting-plane scheme that we can prove is finitely terminating is rather technical. Though it can be done in essentially the same manner for both pure and mixed cases. We need to treat the objective-function value as an additional variable (numbered first), employ the Simplex Algorithm adapted to the ϵ -perturbed problem, always choose the least-index $i \in \mathcal{I}$ having $\bar{y}_i \notin \mathbb{Z}$ and choose r via $(*_=)$ or $(*_{=+})$ as appropriate to generate the Gomory cuts. Details can be found in [2] and [4].

8.5.4 Branch-and-Cut

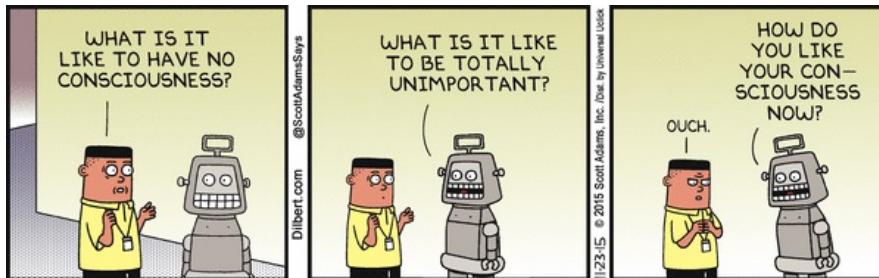
State-of-the-art algorithms for (mixed-)integer linear optimization (like Gurobi, Cplex and Express) combine cuts with branch-and-bound. There are a lot of software design and tuning issues that make this work successfully.

8.6 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 Python/Gurobi, 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)

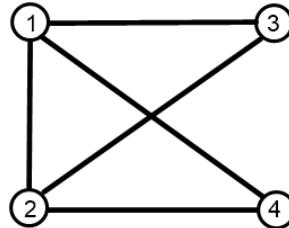
Consider the systems:

$$\begin{aligned} S_1 : \quad & 2x_1 + 2x_2 + x_3 + x_4 \leq 2; \\ & x_j \leq 1; \\ & -x_j \leq 0. \end{aligned}$$

$$\begin{aligned} S_2 : \quad & x_1 + x_2 + x_3 \leq 1; \\ & x_1 + x_2 + x_4 \leq 1; \\ & -x_j \leq 0. \end{aligned}$$

$$\begin{aligned} S_3 : \quad & x_1 + x_2 \leq 1; \\ & x_1 + x_3 \leq 1; \\ & x_1 + x_4 \leq 1; \\ & x_2 + x_3 \leq 1; \\ & x_2 + x_4 \leq 1; \\ & -x_j \leq 0. \end{aligned}$$

Notice that each system has precisely the same set of integer solutions. In fact, each system chooses, via its feasible integer (0/1) solutions, the "vertex packings" of the following graph.



A **vertex packing** of a graph is a set of vertices with no edges between them. For this particular graph we can see that the packings are: $\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{3, 4\}$.

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 $i = 1, \dots, m$ and $j = 1, \dots, n$, or the m constraints: $-ny_i + \sum_{j=1}^n x_{ij} \leq 0$, for $i = 1, \dots, m$. 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 $i = 1, \dots, m$). The Jupyter notebook [UFL.ipynb](#) can be used to perform experiments comparing the use of (S) versus (W). (see Appendix A.11).

Exercise 8.5 (Comparing piecewise-linear formulations)

We have seen that the adjacency condition for piecewise-linear univariate functions can be modeled by

$$\begin{aligned}\lambda_1 &\leq y_1; \\ \lambda_j &\leq y_{j-1} + y_j, \text{ for } j = 2, \dots, n-1; \\ \lambda_n &\leq y_{n-1}.\end{aligned}$$

An alternative formulation is

$$\begin{aligned}\sum_{i=1}^j y_i &\leq \sum_{i=1}^{j+1} \lambda_i, \text{ for } j = 1, \dots, n-2; \\ \sum_{i=j}^{n-1} y_i &\leq \sum_{i=j}^n \lambda_i, \text{ for } j = 2, \dots, n-1.\end{aligned}$$

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 $i = 1, \dots, n-1$. (Note that for both formulations, we require $\lambda_i \geq 0$, for $i = 1, \dots, n$, $\sum_{i=1}^n \lambda_i = 1$, and $\sum_{i=1}^{n-1} y_i = 1$).

Exercise 8.6 (Variable fixing)

Prove Corollary 8.19.

Exercise 8.7 (Gomory cuts)

Prove that we need at least k Chvátal-Gomory cuts to solve Example 8.15. You can observe this bad behavior specifically for Gomory cuts in [pure_gomory_example_2.ipynb](#) (see Appendix A.12)

Exercise 8.8 (Solve pure integer problems using Gomory cuts)

Extend what you did for Exercise 4.1 to now solve pure integer problems using Gomory cuts. [pivot_tools.ipynb](#) (see Appendix A.6) contains two (additional) useful tools for this: `pure_gomory()` and `dual1_plot(,)`. Using only the functions in [pivot_tools.ipynb](#), extend your code from Exercise 4.1 to solve pure integer problems using Gomory cuts. As before, do not worry about degeneracy/anti-cycling. Make some small examples to fully illustrate your code.

Exercise 8.9 (Make amends)

Find an *interesting applied problem*, model it as a pure- or mixed-integer linear-optimization problem, and test your model with Python/Gurobi.



Credit will be given for *deft* modeling, *sophisticated* use of Python/Gurobi, testing on meaningfully-large instances, and *insightful* analysis. Try to play with Gurobi *integer* solver options (they can be set through Python) 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 L^AT_EX template



L^AT_EX Template

December 20, 2022

Your actual name (youremail@umich.edu)

This template can serve as a starting point for learning L^AT_EX. You may download MiK_TeX from <https://miktex.org> to get started, or you can work on Overleaf if you prefer: <https://www.overleaf.com>. 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:

$$\sum_{j=1}^n a_{ij}x_j \quad \forall i = 1, \dots, m.$$

And here is a matrix:

$$\begin{pmatrix} 1 & \pi & 2 & \frac{1}{2} & \nu \\ 6.2 & r & 2 & 4 & 5 \\ |y'| & \mathcal{R} & \mathbb{R} & \underline{r} & \hat{R} \end{pmatrix}.$$

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 typeset a standard-form linear-optimization problem

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

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.

4 Graphics

This is how to include and refer to Figure 1 with pdfLaTeX.

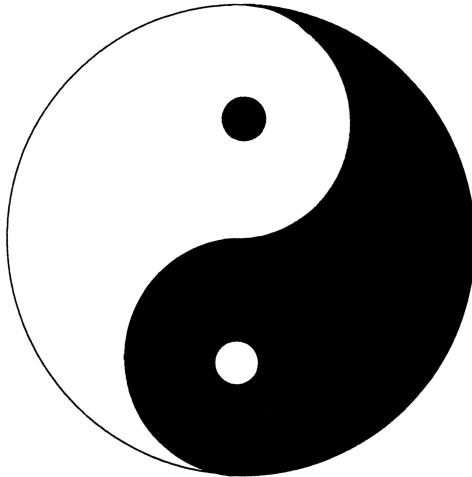


Figure 1: Another duality

5 The L^AT_EX commands to produce this document

Look at the L^AT_EX 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 L^AT_EX document verbatim.

```
% LaTeX_Template.tex // J. Lee
%
% -----
% AMS-LaTeX ****
% **** -----
\documentclass{article}
\usepackage{graphicx,amsmath,amsthm,amsfonts}
\usepackage{hyperref}
\usepackage{verbatim}
\usepackage[letterpaper, 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}{\rightarrowtail}
\newcommand{\BX}{\mathbf{B}(X)}
\newcommand{\A}{\mathcal{A}}
% -----
\begin{document}

\title{\LaTeX~ Template}

\date{\today}

\maketitle

\href{mailto:youremail@umich.edu}{Your actual name (youremail@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 <https://miktex.org> to get started, or you can work on Overleaf if you prefer: <https://www.overleaf.com>. 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:

```
\[
\sum_{j=1}^n a_{ij} x_j ~ \forall i=1,\dots,m.
\]
```

And here is a matrix:

```
\[
\left(
\begin{array}{ccccc}
1 & \pi & 2 & \frac{1}{2} & \nu \\
6.2 & r & 2 & 4 & 5 \\
|y'| & \mathcal{R} & \mathbb{R} & \underline{r} & \hat{R} \\
\end{array}
\right).
\]
```

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!]
\centering
\includegraphics[width=0.4\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
```

```
% -----
```

```
\end{document}  
%
```

A.2 MatrixLP.ipynb

Example: Setting up and solving a matrix-style LP with Python/Gurobi

[Jon Lee \(https://sites.google.com/site/jonleewebpage/home\)](https://sites.google.com/site/jonleewebpage/home),
10/20/2020

$$\begin{aligned} & \min c'x + f'w \\ & Ax + Bw \leq b; \\ & Dx = g; \\ & x \geq \mathbf{0}, w \leq \mathbf{0}. \end{aligned}$$

Note that we have the following dual, but we do not model it:

$$\begin{aligned} & \max y'b + v'g \\ & y'A + v'D \leq c'; \\ & y'B \geq f'; \\ & y \leq \mathbf{0}, v \text{ unrestricted}. \end{aligned}$$

Rather, we recover its solution from Gurobi.

References:

- Jon Lee, "A First Course in Linear Optimization", Fourth Edition (Version 4.05), Reex Press, 2013-22.

MIT License

Copyright (c) 2020 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
In [1]: %reset -f
import numpy as np
import gurobipy as gp
from gurobipy import GRB

class StopExecution(Exception):
    def __render_traceback__(self):
        pass
```

```
In [2]: # setting the matrix sizes and random data
n1=7
n2=15
m1=2
m2=4
np.random.seed(56) # set seed to be able to repeat the same random data
A=np.random.rand(m1,n1)
B=np.random.rand(m1,n2)
D=np.random.rand(m2,n1)

# Organize the situation (i.e., choose the right-hand side coefficients)
# so that the primal problem has a feasible solution
xs=np.random.rand(n1)
ws=-np.random.rand(n2)
b=np.matmul(A,xs)+np.matmul(B,ws)+0.01*np.random.rand(m1)
g=np.matmul(D,xs)

# Organize the situation (i.e., choose the objective coefficients)
# so that the dual problem has a feasible solution
ys=-np.random.rand(m1)
vs=np.random.rand(m2)-np.random.rand(m2)
c=np.matmul(np.transpose(A),ys)+np.matmul(np.transpose(D),vs)+0.01*np.random.r
and(n1)
f=np.matmul(np.transpose(B),ys)-0.01*np.random.rand(n2)
```

```
In [3]: model = gp.Model()
model.reset()
x = model.addMVar(n1) # default is a nonnegative continuous variable
w = model.addMVar(n2, ub=0.0, lb=-GRB.INFINITY)
objective = model.setObjective(c@x+f@w, GRB.MINIMIZE)
constraints1 = model.addConstr(A@x+B@w <= b)
constraints2 = model.addConstr(D@x == g)
```

Set parameter Username

Academic license - for non-commercial use only - expires 2025-06-14

Discarded solution information

```
In [4]: model.optimize()
if model.status != GRB.Status.OPTIMAL:
    print("***** Gurobi solve status:", model.status)
    print("***** This is a problem. Model does not have an optimal solution")
    raise StopExecution
```

Gurobi Optimizer version 11.0.2 build v11.0.2rc0 (win64 - Windows 10.0 (19045.2))

CPU model: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz, instruction set [SSE2|AVX2]

Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 6 rows, 22 columns and 72 nonzeros

Model fingerprint: 0x734450bc

Coefficient statistics:

Matrix range [2e-03, 1e+00]

Objective range [1e-01, 1e+00]

Bounds range [0e+00, 0e+00]

RHS range [7e-01, 3e+00]

Presolve time: 0.01s

Presolved: 6 rows, 22 columns, 72 nonzeros

Iteration	Objective	Primal Inf.	Dual Inf.	Time
0	-7.2547823e+30	1.946982e+31	7.254782e+00	0s
9	2.6453973e+00	0.000000e+00	0.000000e+00	0s

Solved in 9 iterations and 0.02 seconds (0.00 work units)

Optimal objective 2.645397253e+00

```
In [5]: print("***** Primal solution:")
for j in range(0,n1): print("x[",j,"]=", 
    np.format_float_positional(x[j].X,4,pad_right=4))
print(" ")
for j in range(0,n2): print("w[",j,"]=", 
    np.format_float_positional(w[j].X,4,pad_right=4))
print(" ")
print("***** Dual solution:")
for i in range(0,m1): print("y[",i,"]=", 
    np.format_float_positional(constraints1[i].Pi,4,pad_right=4))
print(" ")
for i in range(0,m2): print("v[",i,"]=", 
    np.format_float_positional(constraints2[i].Pi,4,pad_right=4))
```

***** Primal solution:

```
x[ 0 ]= 0.2689
x[ 1 ]= 0.0080
x[ 2 ]= 1.3952
x[ 3 ]= 0.
x[ 4 ]= 0.4962
x[ 5 ]= 0.
x[ 6 ]= 0.
```

```
w[ 0 ]= 0.
w[ 1 ]= 0.
w[ 2 ]= 0.
w[ 3 ]= 0.
w[ 4 ]= 0.
w[ 5 ]= 0.
w[ 6 ]= 0.
w[ 7 ]= 0.
w[ 8 ]= 0.
w[ 9 ]= 0.
w[ 10 ]= -4.7348
w[ 11 ]= 0.
w[ 12 ]= -4.392
w[ 13 ]= 0.
w[ 14 ]= 0.
```

***** Dual solution:

```
y[ 0 ]= -0.4424
y[ 1 ]= -0.7261
```

```
v[ 0 ]= -0.8196
v[ 1 ]= -0.6668
v[ 2 ]= -0.0458
v[ 3 ]= 0.1904
```


A.3 Production.ipynb

Production model: constraint-style LP with Python/Gurobi

[Jon Lee \(https://sites.google.com/site/jonleewebpage/home\)](https://sites.google.com/site/jonleewebpage/home),
10/20/2020

Notes:

- This example is meant to show how to:
 - do constraint-style LP's (as opposed to matrix style), though the model we are setting up is $\max\{c'x : Ax \leq b, x \geq 0\}$.
 - extract and print: primal and dual solutions, primal and dual slacks, and sensitivity information
 - pass constraint names to Gurobi and then retrieve constraints from Gurobi by these names

References:

- Jon Lee, "A First Course in Linear Optimization", Fourth Edition (Version 4.05), Reex Press, 2013-22.

MIT License

Copyright (c) 2020 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
In [1]: %reset -f
import numpy as np
import gurobipy as gp
from gurobipy import GRB

class StopExecution(Exception):
    def __render_traceback__(self):
        pass
```

```
In [2]: # Some toy data
m=3
n=2
M=list(range(0,m))
N=list(range(0,n))
A = np.array([ [8, 5], [8, 6], [8, 7] ])
b = np.array([32, 33, 35])
c = np.array([3 ,2])
```

```
In [3]: model = gp.Model()
model.reset()
x = model.addMVar(n)
revenueobjective = model.setObjective(sum(c[j]*x[j] for j in N), GRB.MAXIMIZE)
for i in M:      # naming the constraints r0,r1,r2,... (inside Gurobi)
    model.addConstr(sum(A[i,j]*x[j] for j in N) <= b[i], name='r'+str(i))
```

Set parameter Username

Academic license - for non-commercial use only - expires 2025-06-14

Discarded solution information

```
In [4]: model.optimize()
if model.status != GRB.Status.OPTIMAL:
    print("***** Gurobi solve status:", model.status)
    print("***** This is a problem. Model does not have an optimal solution")
    raise StopExecution
print(" ")
print("primal var,      dual slack,      obj delta-lb,      obj delta-ub")
for j in N: print("x[",j,"]=",np.format_float_positional(x[j].X,4,pad_right=4),
                  " t[",j,"]=", np.format_float_positional(x[j].RC,4,pad_right=4),
                  " L[",j,"]=", np.format_float_positional(x[j].SAObjLow-c[j],4,pad_right=4),
                  " U[",j,"]=", np.format_float_positional(x[j].SAObjUp-c[j],4,pad_right=4))
print(" ")
print("dual vars,      primal slack,      rhs delta-lb,      rhs delta-ub")
for i in M:
    constr=model.getConstrByName('r'+str(i)) # retriving from Gurobi the named constraints r0,r1,r2,...
    print("y[",i,"]=",np.format_float_positional(constr.Pi,4,pad_right=4),
          " s[",i,"]=", np.format_float_positional(constr.Slack,4,pad_right=4),
          " L[",i,"]=", np.format_float_positional(constr.SARHSLow-b[i],4,pad_right=4),
          " U[",i,"]=", np.format_float_positional(constr.SARHSUp-b[i],4,pad_right=4))
```

Gurobi Optimizer version 11.0.2 build v11.0.2rc0 (win64 - Windows 10.0 (19045.2))

CPU model: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz, instruction set [SSE2|AVX2]

Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 3 rows, 2 columns and 6 nonzeros

Model fingerprint: 0x3e75bd93

Coefficient statistics:

Matrix range [5e+00, 8e+00]

Objective range [2e+00, 3e+00]

Bounds range [0e+00, 0e+00]

RHS range [3e+01, 4e+01]

Presolve time: 0.00s

Presolved: 3 rows, 2 columns, 6 nonzeros

Iteration	Objective	Primal Inf.	Dual Inf.	Time
0	5.0000000e+30	5.250000e+30	5.000000e+00	0s
3	1.2125000e+01	0.000000e+00	0.000000e+00	0s

Solved in 3 iterations and 0.01 seconds (0.00 work units)

Optimal objective 1.212500000e+01

primal var, dual slack, obj delta-lb, obj delta-ub
x[0]= 3.3750 t[0]= 0. L[0]= -0.3333 U[0]= 0.2
x[1]= 1. t[1]= 0. L[1]= -0.125 U[1]= 0.25

dual vars, primal slack, rhs delta-lb, rhs delta-ub
y[0]= 0.2500 s[0]= 0. L[0]= -1. U[0]= 1.
y[1]= 0.125 s[1]= 0. L[1]= -1. U[1]= 0.5
y[2]= 0. s[2]= 1.0000 L[2]= -1. U[2]= inf

A.4 Multi-commodityFlow.ipynb

Multi-Commodity Network-Flow model: constraint-style LP with Python/Gurobi

[Jon Lee \(https://sites.google.com/site/jonleewebpage/home\)](https://sites.google.com/site/jonleewebpage/home),
02/2020

$$\begin{aligned}
 \min \quad & \sum_{k=1}^K \sum_{e \in \mathcal{A}} c_e^k x_e^k \\
 \text{subject to} \quad & \sum_{e \in \mathcal{A} : t(e)=v} x_e^k - \sum_{e \in \mathcal{A} : h(e)=v} x_e^k = b_v^k, \quad \text{for } v \in \mathcal{N}, k = 1, 2, \dots, K; \\
 & \sum_{k=1}^K x_e^k \leq u_e, \quad \text{for } e \in \mathcal{A}; \\
 & x_e^k \geq 0, \quad \text{for } e \in \mathcal{A}, k = 1, 2, \dots, K.
 \end{aligned}$$

Notes:

- K=1 is ordinary single-commodity network flow. Integer solutions for free when node-supplies and arc capacities are integer.
- K=2 example below with integer data gives a fractional basic optimum. This example doesn't have any feasible integer flow at all.

References:

- Jon Lee, "A First Course in Linear Optimization", Fourth Edition (Version 4.05), Reex Press, 2013-22.

MIT License

Copyright (c) 2021 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
In [1]: %reset -f
import itertools
import numpy as np
%matplotlib notebook
#%matplotlib inline
import matplotlib.pyplot as plt
import gurobipy as gp
from gurobipy import GRB
import networkx as nx

class StopExecution(Exception):
    def __render_traceback__(self):
        pass
```

```
In [2]: # parameters
solveLPOnly=True      # set False to solve as an IP
```

```
In [3]: # # Some toy data: 1 commodity
# Supplies= {
# # node i: [supply commodity[1] ... supply commodity[K]],
#     1: [12.],
#     2: [6.],
#     3: [-2.],
#     4: [0.],
#     5: [-9.],
#     6: [-7.]}

# CapacityCosts = {
# # arc (i,j): [capacity, cost commodity[1] ... cost commodity[K]],
#     (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]}

# Some toy data: 2 commodities with a fractional LP basic optimum
Supplies= {
# node i: [supply commodity[1] ... supply commodity[K]],
    1: [1., 0.],
    2: [0., -1.],
    3: [0., 0.],
    4: [0., 0.],
    5: [0., 0.],
    6: [0., 0.],
    7: [0., 1.],
    8: [-1., 0.]}}

CapacityCosts = {
# arc (i,j): [capacity, cost commodity[1] ... cost commodity[K]],
    (1,2): [1., 1, 1],
    (1,3): [1., 1, 1],
    (2,5): [1., 1, 1],
    (3,4): [1., 1, 1],
    (4,1): [1., 1, 1],
    (4,7): [1., 1, 1],
    (5,6): [1., 1, 1],
    (6,2): [1., 1, 1],
    (6,8): [1., 1, 1],
    (7,3): [1., 1, 1],
    (7,8): [1., 1, 1],
    (8,5): [1., 1, 1]}
```

```
In [4]: Nodes=list(Supplies.keys()) # get node list from supply data
K=len(Supplies[Nodes[0]]) # get number of commodities from supply data
Commods=list(range(1,K+1)) # name the commodities 1,2,...,K
Arcs=list(CapacityCosts.keys()) # get arc list from Capacity/Cost data
ArcsCrossCommods=list(itertools.product(Arcs,Commods)) # make cross product of
Arcs and Commods for variable indexing
```

```
In [5]: model = gp.Model()
if solveLPOnly==True:
    x = model.addVars(ArcsCrossCommods)
else:
    x = model.addVars(ArcsCrossCommods,vtype=GRB.INTEGER)
model.setObjective(sum(sum(CapacityCosts[i,j][k]*x[(i,j),k] for (i,j) in Arcs)
for k in Commods), GRB.MINIMIZE)
model.addConstrs(sum(x[(i,j),k] for k in Commods) <= CapacityCosts[i,j][0] for
(i,j) in Arcs)
model.addConstrs(
    (sum(x[(i, j),k] for j in Nodes if (i, j) in Arcs) - sum(x[(j, i),k] for j in
Nodes if (j,i) in Arcs)
    == Supplies[i][k-1] for i in Nodes for k in Commods))
model.update()
```

Set parameter Username

Academic license - for non-commercial use only - expires 2025-06-14

```
In [6]: model.optimize()
if model.status != GRB.Status.OPTIMAL:
    print("***** Gurobi solve status:", model.status)
    print("***** This is a problem. Model does not have an optimal solution")
    raise StopExecution
print(" ")
print("***** Flows:")
for (i,j) in Arcs:
    arcflow=""
    for k in Commods:
        arcflow += str(round(x[(i,j),k].X,4))
        arcflow += " "
    print("x[(" , i , ", " , j , "), *]=", arcflow, "capacity:", CapacityCosts[i,j][0])
```

Gurobi Optimizer version 11.0.2 build v11.0.2rc0 (win64 - Windows 10.0 (19045.2))

CPU model: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz, instruction set [SSE2|AVX2]

Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 28 rows, 24 columns and 72 nonzeros

Model fingerprint: 0xf7e9da00

Coefficient statistics:

Matrix range	[1e+00, 1e+00]
Objective range	[1e+00, 1e+00]
Bounds range	[0e+00, 0e+00]
RHS range	[1e+00, 1e+00]

Presolve removed 26 rows and 22 columns

Presolve time: 0.01s

Presolved: 2 rows, 2 columns, 4 nonzeros

Iteration	Objective	Primal Inf.	Dual Inf.	Time
0	8.0000000e+00	1.000000e+00	0.000000e+00	0s
1	8.0000000e+00	0.000000e+00	0.000000e+00	0s

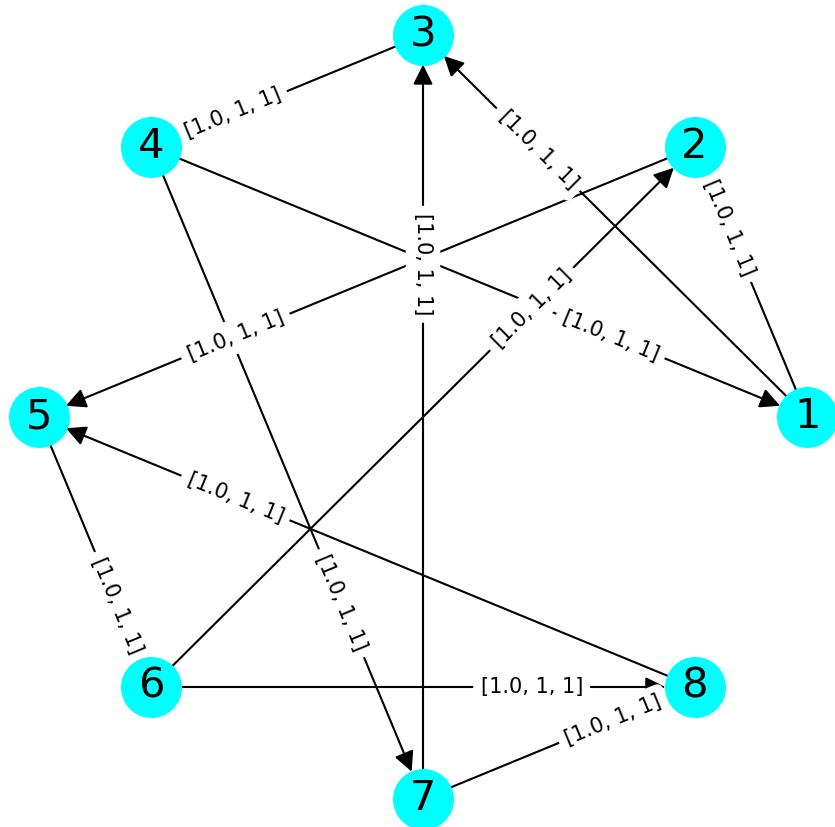
Solved in 1 iterations and 0.02 seconds (0.00 work units)

Optimal objective 8.000000000e+00

***** Flows:

x[(1 , 2), *]= 0.5 0.5	capacity: 1.0
x[(1 , 3), *]= 0.5 0.0	capacity: 1.0
x[(2 , 5), *]= 0.5 0.0	capacity: 1.0
x[(3 , 4), *]= 0.5 0.5	capacity: 1.0
x[(4 , 1), *]= 0.0 0.5	capacity: 1.0
x[(4 , 7), *]= 0.5 0.0	capacity: 1.0
x[(5 , 6), *]= 0.5 0.5	capacity: 1.0
x[(6 , 2), *]= 0.0 0.5	capacity: 1.0
x[(6 , 8), *]= 0.5 0.0	capacity: 1.0
x[(7 , 3), *]= 0.0 0.5	capacity: 1.0
x[(7 , 8), *]= 0.5 0.5	capacity: 1.0
x[(8 , 5), *]= 0.0 0.5	capacity: 1.0

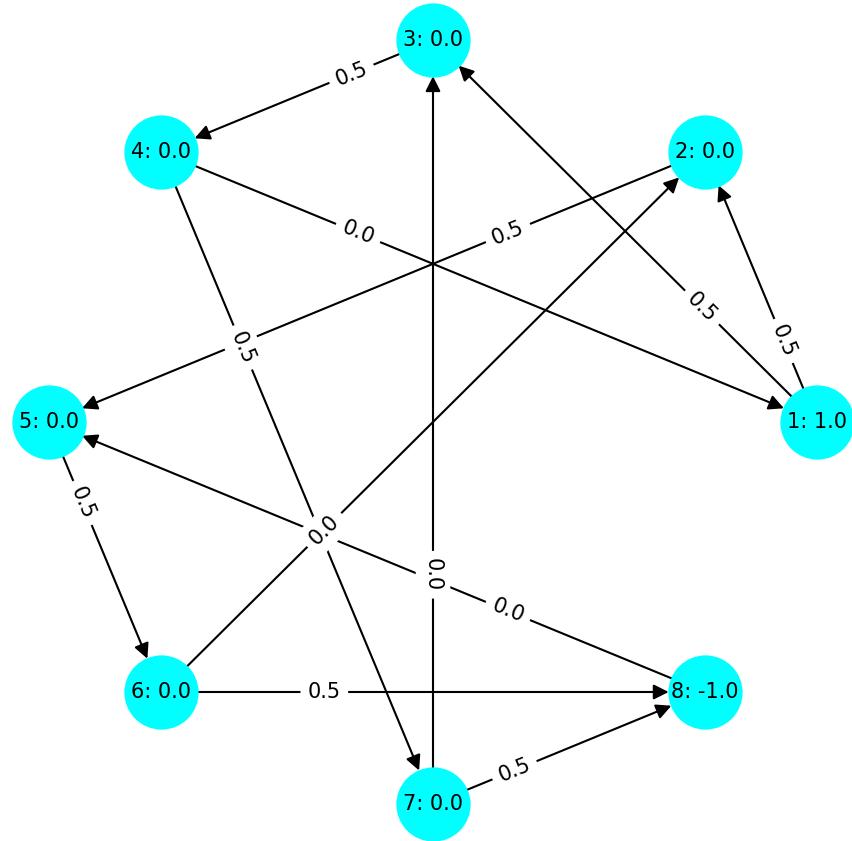
```
In [7]: G = nx.DiGraph()
G.add_nodes_from(Nodes)
G.add_edges_from(Arcs)
plt.figure(figsize=(8,8))
edge_labels=nx.draw_networkx_edge_labels(G,edge_labels=CapacityCosts,
    pos=nx.shell_layout(G), label_pos=0.3, font_size=10)
nx.draw_shell(G, with_labels=True, node_color='cyan', node_size=800,
    font_size=20, arrowsize=20)
print("Network with node labels and capacities/costs on arcs")
```



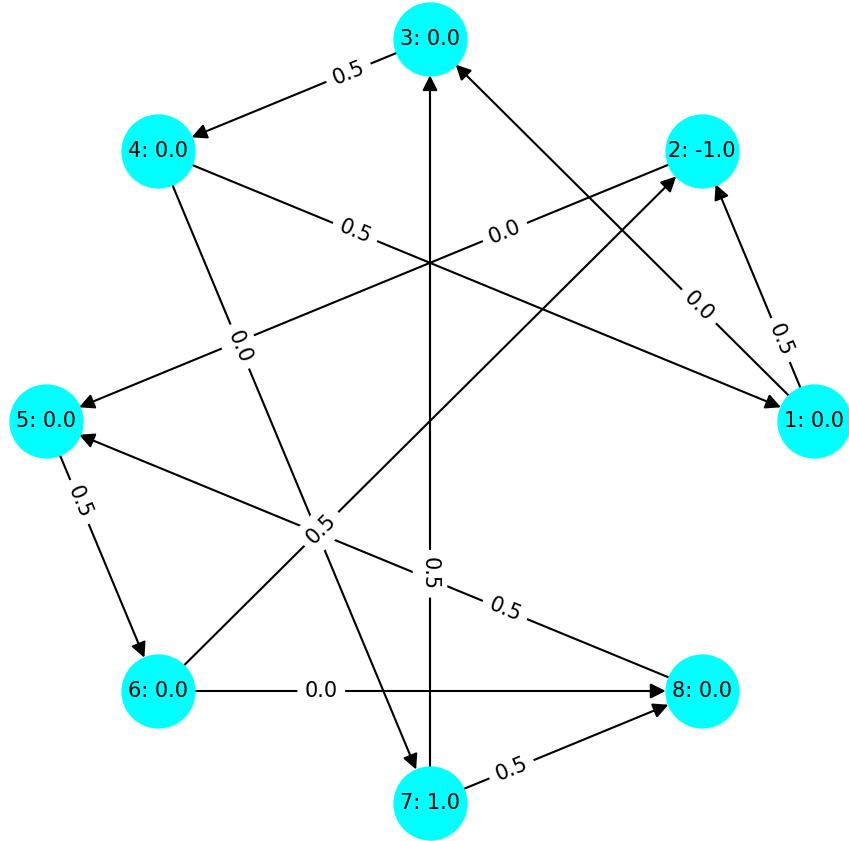
Network with node labels and capacities/costs on arcs

```
In [8]: #k=2
for k in Commods:
    Supply1_label={}
    for i in Nodes:
        Supply1_label[i]= str(i)+': '+str(Supplies[i][k-1])

    Flow0=np.zeros(len(Arcs))
    Flow=dict(zip(list(Arcs), Flow0))
    for (i,j) in Arcs: Flow[i,j]= str(round(x[(i,j),k].X,4))
    H=nx.relabel_nodes(G, Supply1_label)
    plt.figure(figsize=(8,8))
    edge_labels=nx.draw_networkx_edge_labels(H,edge_labels=Flow,
                                             pos=nx.shell_layout(G), label_pos=0.7, font_size=10)
    nx.draw_shell(H, with_labels=True, node_color='cyan',
                  node_size=1200, font_size=10, arrowsize=15)
    print("Network with supplies and flows for commodity ",k)
```



Network with supplies and flows for commodity 1



Network with supplies and flows for commodity 2

A.5 pivot_example.ipynb

Example: pivot tools for standard form linear-optimization problem P

[Jon Lee \(https://sites.google.com/site/jonleewebpage/home\)](https://sites.google.com/site/jonleewebpage/home),
11/21/2020

For standard-form problems

$$\begin{aligned} z &:= \min c'x \\ Ax &= b; \\ x &\geq 0. \end{aligned} \tag{P}$$

Notes:

- Can work with ϵ perturbed right-hand side

References:

- Jon Lee, "A First Course in Linear Optimization", Fourth Edition (Version 4.05), Reex Press, 2013-22.

MIT License

Copyright (c) 2020 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

In [1]: %reset -f

In [2]: %run ./pivot_tools.ipynb

```
pivot_tools loaded: pivot_perturb, pivot_algebra, N, pivot_ratios, pivot_swaps, pivot_plot, pure_gomory, mixed_gomory, dual_plot
```

```
In [3]: A = sym.Matrix(([1, 2, 1, 0, 0, 0],
                     [3, 1, 0, 1, 0, 0],
                     [sym.Rational(3,2), sym.Rational(3,2), 0, 0, 1, 0],
                     [0, 1, 0, 0, 0, 1]))
m = A.shape[0]
n = A.shape[1]
c = sym.Matrix([6, 7, -2, 0, 4, sym.Rational(9,2)])
b = sym.Matrix([7, 9, 6, sym.Rational(33,10)])
beta = [0,1,3,5]
eta = list(set(list(range(n)))-set(beta))
A_beta = copy.copy(A[:,beta])
A_eta = copy.copy(A[:,eta])
c_beta = copy.copy(c[beta,0])
c_eta = copy.copy(c[eta,0])
Perturb=False ##### do NOT change this!!!!!!!!!!!!!!!!!!!! You
can perturb later
```

In [4]: A

Out[4]:

$$\begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 & 0 \\ \frac{3}{2} & \frac{3}{2} & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In [5]: c

Out[5]:

$$\begin{bmatrix} 6 \\ 7 \\ -2 \\ 0 \\ 4 \\ \frac{9}{2} \end{bmatrix}$$

In [6]: #pivot_perturb() # uncomment to perturb the right-hand side

In [7]: b

Out[7]:

$$\begin{bmatrix} 7 \\ 9 \\ 6 \\ \frac{33}{10} \end{bmatrix}$$

In [8]: beta

Out[8]: [0, 1, 3, 5]

In [9]: eta

Out[9]: [2, 4]

```
In [10]: A_beta
```

```
Out[10]: 
$$\begin{bmatrix} 1 & 2 & 0 & 0 \\ 3 & 1 & 1 & 0 \\ \frac{3}{2} & \frac{3}{2} & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

```

```
In [11]: A_eta
```

```
Out[11]: 
$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

```

```
In [12]: pivot_algebra()
```

```
pivot_algebra() done
```

```
In [13]: objval
```

```
Out[13]: 
$$\frac{567}{20}$$

```

```
In [14]: xbar_beta
```

```
Out[14]: 
$$\begin{bmatrix} 1 \\ 3 \\ 3 \\ \frac{3}{10} \end{bmatrix}$$

```

```
In [15]: cbar_eta
```

```
Out[15]: 
$$\begin{bmatrix} \frac{3}{2} \\ -\frac{7}{3} \end{bmatrix}$$

```

In [16]: `pivot_ratios(1)`

$$\begin{bmatrix} \frac{3}{4} \\ \infty \\ \infty \\ \frac{9}{20} \end{bmatrix}$$

$\bar{x} + \lambda \bar{z} :$

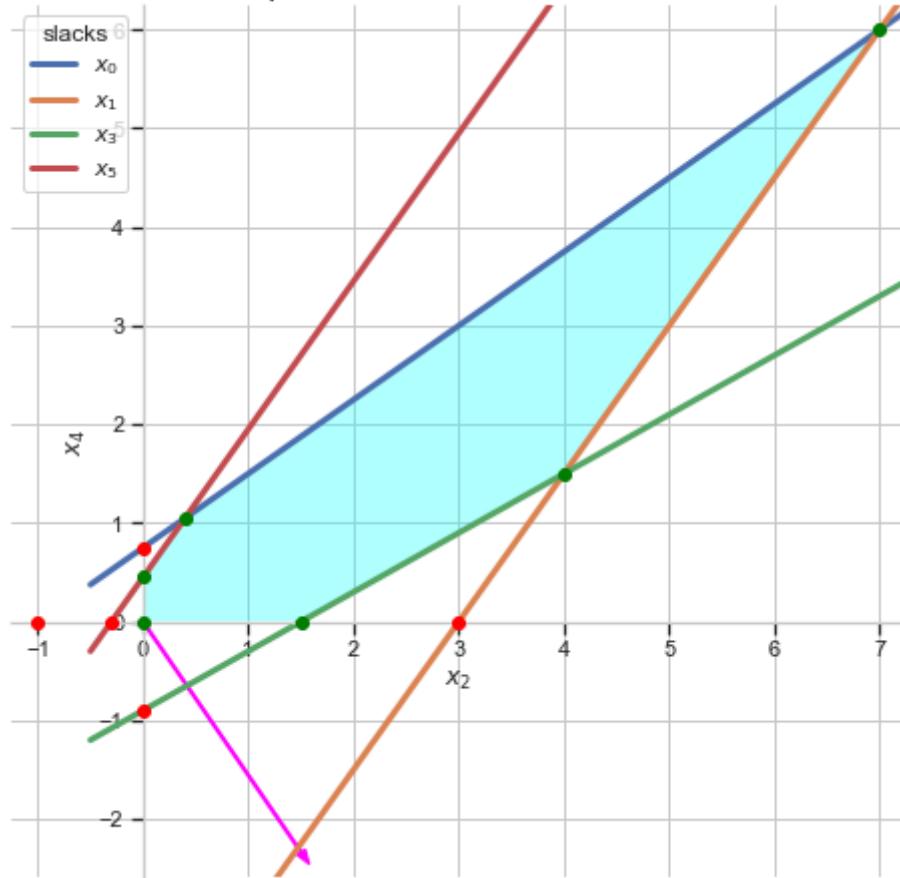
$$\begin{bmatrix} 1 - \frac{4\lambda}{3} \\ \frac{2\lambda}{3} + 3 \\ 0 \\ \frac{10\lambda}{3} + 3 \\ \lambda \\ \frac{3}{10} - \frac{2\lambda}{3} \end{bmatrix}$$

In [17]: `c.dot(zbar) # agrees with cbar_eta(1)`

Out[17]: $-\frac{7}{3}$

In [18]: `pivot_plot()`

In the space of the non-basic variables



In [19]: `pivot_swap(1,3)`

swap accepted --- new partition:

```
eta: [2, 5]
beta: [0, 1, 3, 4]
*** MUST APPLY pivot_algebra()! ***
```

In [20]: `pivot_algebra()`

`pivot_algebra()` done

In [21]: `sym.N(objval)`

Out[21]: 27.3

In [22]: `xbar_beta`

$$\begin{bmatrix} \frac{2}{5} \\ \frac{33}{10} \\ \frac{9}{2} \\ \frac{9}{20} \end{bmatrix}$$

In [23]: `cbar_eta`

$$\begin{bmatrix} -2 \\ \frac{7}{2} \end{bmatrix}$$

In [24]: `pivot_ratios(0)`

$$\begin{bmatrix} \frac{2}{5} \\ \infty \\ \infty \\ \infty \end{bmatrix}$$

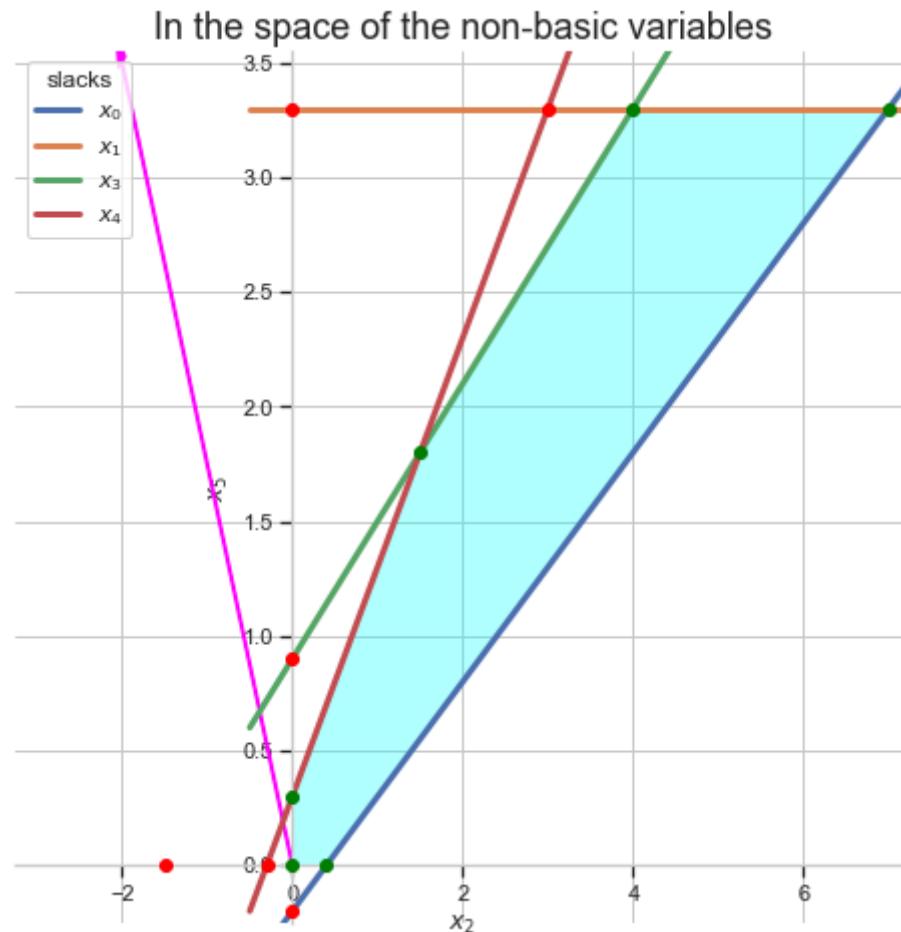
$\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} \frac{2}{5} - \lambda \\ \frac{33}{10} \\ \lambda \\ 3\lambda + \frac{9}{2} \\ \frac{3\lambda}{2} + \frac{9}{20} \\ 0 \end{bmatrix}$$

In [25]: `c.dot(zbar) # agrees with cbar_eta(0)`

Out[25]: -2

In [26]: `pivot_plot()`



In [27]: `pivot_swap(0,0)`

swap accepted --- new partition:

```
eta: [0, 5]
beta: [2, 1, 3, 4]
*** MUST APPLY pivot_algebra()! ***
```

In [28]: `pivot_algebra()`

`pivot_algebra()` done

In [29]: `sym.N(objval)`

Out[29]: 26.5

In [30]: `xbar_beta`

Out[30]:

$$\begin{bmatrix} \frac{2}{5} \\ \frac{33}{10} \\ \frac{57}{10} \\ \frac{21}{20} \end{bmatrix}$$

```
In [31]: cbar_eta
```

```
Out[31]: 
$$\begin{bmatrix} 2 \\ -\frac{1}{2} \end{bmatrix}$$

```

```
In [32]: pivot_ratios(1)
```

$$\begin{bmatrix} \infty \\ \frac{33}{10} \\ \infty \\ \infty \end{bmatrix}$$

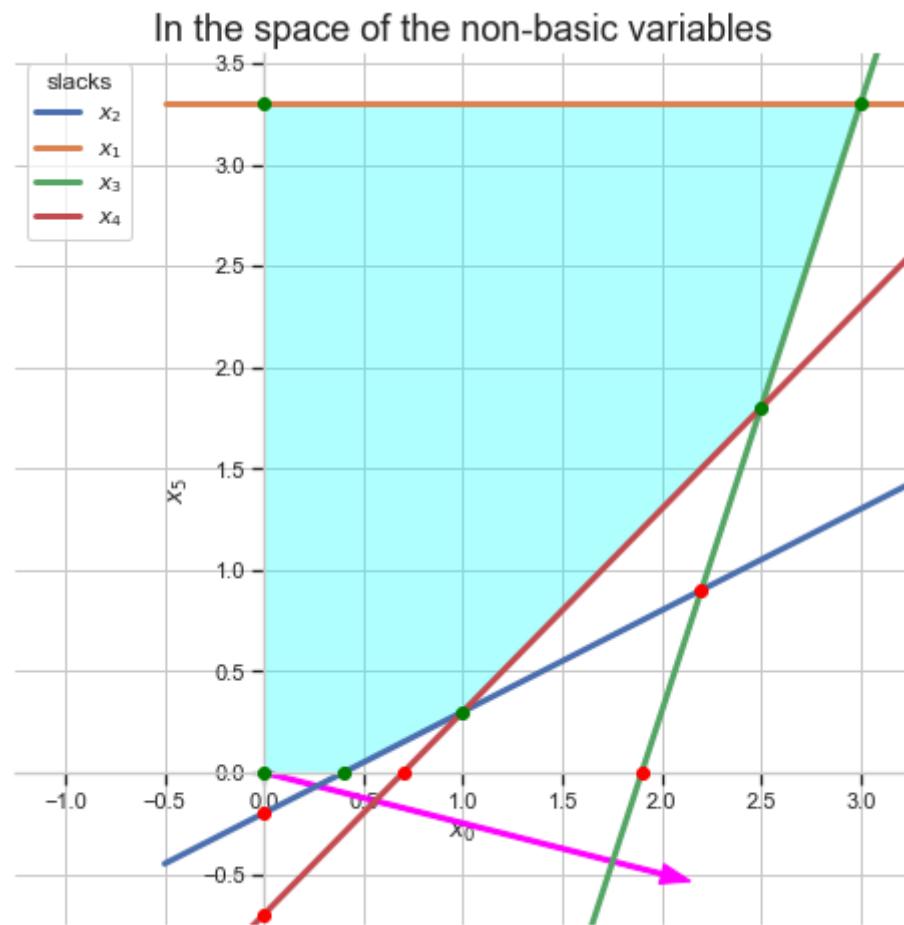
$$\bar{x} + \lambda \bar{z} :$$

$$\begin{bmatrix} 0 \\ \frac{33}{10} - \lambda \\ 2\lambda + \frac{2}{5} \\ \lambda + \frac{57}{10} \\ \frac{3\lambda}{2} + \frac{21}{20} \\ \lambda \end{bmatrix}$$

```
In [33]: c.dot(zbar) # agrees with cbar_eta(1)
```

```
Out[33]: - $\frac{1}{2}$ 
```

In [34]: `pivot_plot()`



In [35]: `pivot_swap(1,1)`

swap accepted --- new partition:

```
eta: [0, 1]
beta: [2, 5, 3, 4]
*** MUST APPLY pivot_algebra()! ***
```

In [36]: `pivot_algebra()`

`pivot_algebra()` done

In [37]: `sym.N(objval)`

Out[37]: 24.85

In [38]: `xbar_beta`

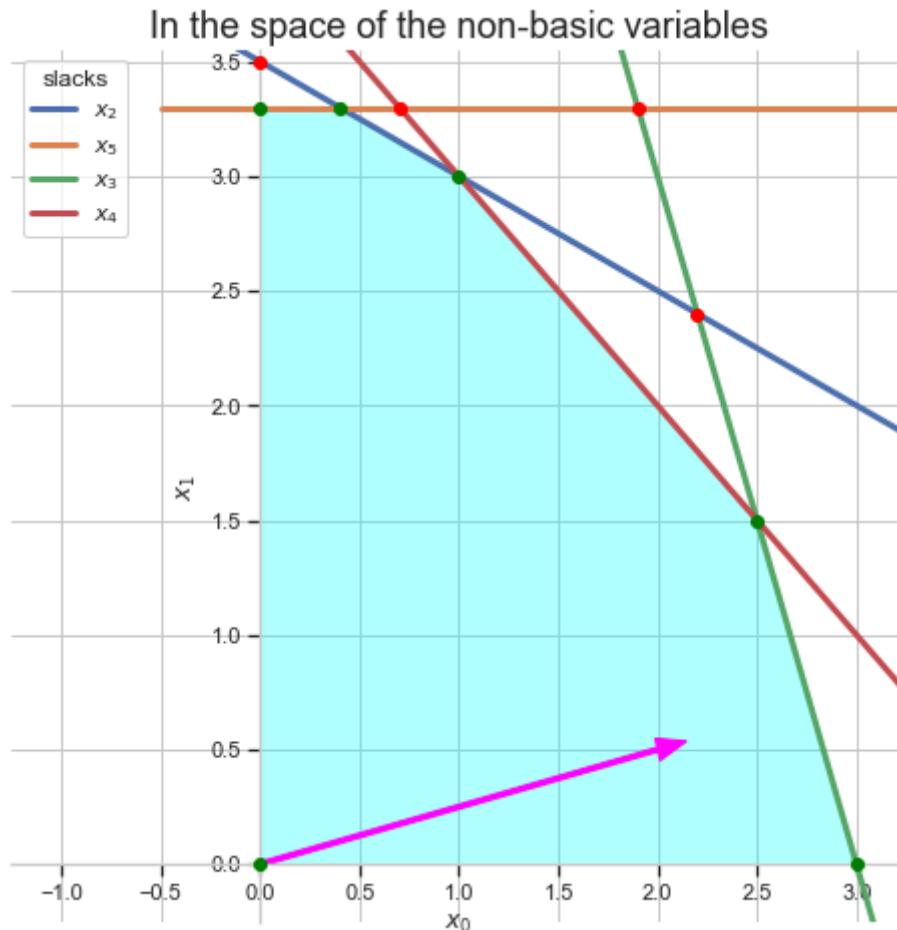
Out[38]:

$$\begin{bmatrix} 7 \\ \frac{33}{10} \\ 9 \\ 6 \end{bmatrix}$$

In [39]: cbar_eta

Out[39]: $\begin{bmatrix} 2 \\ \frac{1}{2} \end{bmatrix}$

In [40]: pivot_plot()



In [41]: xbar

Out[41]: $\begin{bmatrix} 0 \\ 0 \\ 7 \\ 9 \\ 6 \\ \frac{33}{10} \end{bmatrix}$

In [42]: objval

Out[42]: $\frac{497}{20}$

```
In [43]: c.dot(xbar) # reality check
```

```
Out[43]:  $\frac{497}{20}$ 
```

```
In [44]: c_beta.dot(xbar_beta) # reality check
```

```
Out[44]:  $\frac{497}{20}$ 
```

```
In [45]: ybar.dot(b) # reality check
```

```
Out[45]:  $\frac{497}{20}$ 
```

```
In [46]: sym.transpose(c)-sym.transpose(ybar)*A # reality check
```

```
Out[46]:  $\begin{bmatrix} 2 & \frac{1}{2} & 0 & 0 & 0 & 0 \end{bmatrix}$ 
```

```
In [47]: b-A*xbar # reality check
```

```
Out[47]:  $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ 
```


A.6 pivot_tools.ipynb

Pivot tools for standard-form linear-optimization problem P

Jon Lee (<https://sites.google.com/site/jonleewebpage/home>), 11/21/2020

For standard-form problems

$$\begin{aligned} z &:= \min c'x \\ Ax &= b; \\ x &\geq \mathbf{0}. \end{aligned} \tag{P}$$

Notes:

- Can work with ϵ perturbed right-hand side.
- $\beta = (\beta_0, \beta_1, \dots, \beta_{m-1})$ has m entries from $\{0, 1, \dots, n - 1\}$.
- $\eta = (\eta_0, \eta_1, \dots, \eta_{n-m-1})$ has $n - m$ entries from $\{0, 1, \dots, n - 1\}$.
- So, for the purpose of selecting j (corresponding to η_j entering the basis), we view $\bar{c}_\eta = (\bar{c}_{\eta_0}, \bar{c}_{\eta_1}, \dots, \bar{c}_{\eta_{n-m-1}})$.
- For `pivot_ratios(j)`: j must be in $\{0, 1, \dots, n - m - 1\}$. The output of `pivot_ratios(j)` is m numbers, and they correspond to the basic variables numbered $\beta_0, \beta_1, \dots, \beta_{m-1}$. So, for the purpose of selecting i (correspond to β_i leaving the basis), i must be in $\{0, 1, \dots, m - 1\}$.
- For `pivot_swap(j,i)`: j must be in $\{0, 1, \dots, n - m - 1\}$ and i must be in $\{0, 1, \dots, m - 1\}$.

References:

- Jon Lee, "A First Course in Linear Optimization", Fourth Edition (Version 4.05), Reex Press, 2013-22.

MIT License

Copyright (c) 2020 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
In [ ]: import numpy as np
import sympy as sym
sym.init_printing(use_latex='mathjax') # Can replace with sym.init_printing()
if LaTeX is installed
import copy
import operator
eps = sym.symbols('epsilon')
lam = sym.symbols('lambda')
x1z = sym.Symbol('\bar{x}+\lambda \bar{z} :')
from IPython.display import Latex, Math
#####
### CHOOSE A BACKEND --- IF YOU SWITCH, RESTART THE KERNEL
# evaluates faster than 'notebook':
%matplotlib inline
# evaluates slower than 'inline' (gives interactive plots, though delayed when
running all cells):
%matplotlib notebook
#####
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
from matplotlib.ticker import AutoMinorLocator, MultipleLocator
from scipy.spatial import ConvexHull, convex_hull_plot_2d
import itertools
import seaborn as sns; sns.set(); sns.set_style("whitegrid"); color_list = sn
s.color_palette("muted")
```

```
In [ ]: # perturb
def pivot_perturb():
    global m, b, Perturb, eps
    Perturb = True
    for i in range(m):
        for j in range(m):
            b[i] += A_beta[i,j]*eps**(j+1)
    print('pivot_perturb() done')
```

```
In [ ]: # algebra
def pivot_algebra():
    global m, n, objval, xbar, xbar_beta, xbar_eta, ybar, cbar_eta, ratios
    xbar_beta = A_beta.solve(b)
    xbar_eta = sym.zeros(n-m,1)
    objval = c_beta.dot(xbar_beta)
    xbar = sym.zeros(n,1)
    for i in range(m): xbar[beta[i]] = xbar_beta[i]
    for j in range(n-m): xbar[eta[j]] = xbar_eta[j]
    ybar = A_beta.transpose().solve(c[beta,0])
    #cbar_eta = c_eta.transpose() - ybar.transpose()*A_eta
    cbar_eta = c_eta - A_eta.transpose()*ybar
    ratios=sym.oo*sym.ones(m,1)
    print('pivot_algebra() done')
```

```
In [ ]: # numerical version of a d-by-1 array
def N(parray):
    for i in range(parray.shape[0]): display(sym.N(parray[i]))
```

```
In [ ]: # ratios (and direction) for a given nonbasic index eta_j
def pivot_ratios(j):
    global ratios, zbar
    if j>n-m-1:
        display(Latex("error: $j$ is out of range."))
    else:
        A_etaj=copy.copy(A[:,eta[j]])
        Abar_etaj = A_beta.solve(A_etaj)
        for i in range(m):
            if Abar_etaj[i] > 0:
                ratios[i] = xbar_beta[i] / Abar_etaj[i]
            else:
                ratios[i] = sym.oo
        display(ratios)
        zbar=sym.zeros(n,1)
        for i in range(m): zbar[beta[i]] = -Abar_etaj[i]
        zbar[eta[j]] = 1
        display(xlz,xbar+lam*zbar)
```

```
In [ ]: # swap nonbasic eta_j in and basic beta_i out
def pivot_swap(j,i):
    global A_beta, A_eta, c_beta, c_eta
    if i>m-1 or j>n-m-1:
        display(Latex("error: $j$ or $i$ is out of range. swap not accepted"))
    else:
        save = copy.copy(beta[i])
        beta[i] = copy.copy(eta[j])
        eta[j] = save
        A_beta = copy.copy(A[:,beta])
        A_eta = copy.copy(A[:,eta])
        c_beta = copy.copy(c[beta,0])
        c_eta = copy.copy(c[eta,0])
        display(Latex("swap accepted --- new partition:"))
        print('eta:',eta)
        print('beta:',beta)
        print('*** MUST APPLY pivot_algebra()! ***')
```

```
In [ ]: # plot
def pivot_plot():
    if n-m != 2 or Perturb == True:
        display(Latex("Hey friend --- give me a break!"))
        display(Latex("This plotting only works if there are $n-m=2$ nonbasic
variables and no rhs perturbation"))
    return
A_beta_inv = A_beta.inv()
Abar_eta = A_beta_inv*A_eta
M = sym.zeros(n,n-m)
M[0:m,:] = Abar_eta
M[m:n,:] = -sym.eye(n-m)
h = sym.zeros(n,1)
h[0:m,0] = xbar_beta
feaspoints=np.empty((0,2))
infeaspoints=np.empty((0,2))
bbar=sym.zeros(2,1)
M2=sym.zeros(2,2)
for i in range(n-1):
    for j in range(i+1,n):
        bbar[0]=h[i]
        bbar[1]=h[j]
        M2[0,:]=M[i,:]
        M2[1,:]=M[j,:]
        if abs(sym.det(M2)) >0.0001:
            xy = M2.solve(bbar)
            if min(h - M*xy) >= -0.00001:
                feaspoints=np.r_[feaspoints,np.transpose(xy)]
            else:
                infeaspoints=np.r_[infeaspoints,np.transpose(xy)]
hull = ConvexHull(feaspoints)
fig, ax = plt.subplots(figsize=(8,8))
ax.set(xlabel=r"$x_{}$".format(eta[0]), ylabel=r"$x_{}$".format(eta[1]))
ax.spines['left'].set_position(('data',0.0))
ax.spines['bottom'].set_position(('data',0.0))
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
plt.xlim(float(min(cbar_eta[0],min(feaspoints[:,0])))-1.25, float(max(feas
points[:,0]))+0.25)
plt.ylim(float(min(cbar_eta[1],min(feaspoints[:,1])))-0.25, float(max(feas
points[:,1]))+0.25)
plt.fill(feaspoints[hull.vertices,0], feaspoints[hull.vertices,1], 'cyan',
alpha=0.3)
```

```

x = np.linspace(float(min(feaspoints[:,0]))-0.5, float(max(feaspoints[:,0]))+0.5, 100)
for i in range(m):
    if Abar_eta[i,1] != 0:
        y = (xbar_beta[i] - Abar_eta[i,0]*x) / Abar_eta[i,1]
        plt.plot(x, y, linewidth=3, label=r"$x_{\{}$\format(beta[i])")
    else:
        plt.vlines(float(xbar_beta[i]/ Abar_eta[i,0]), float(min(cbar_eta[1],min(feaspoints[:,1]))),
                  float(max(feaspoints[:,0])), label=r"$x_{\{}$\format(beta[i]))
    for simplex in hull.simplices:
        plt.fill(feaspoints[simplex, 0], feaspoints[simplex, 1], 'cyan', alpha=0.5)
        arrow=plt.arrow(0,0, float(cbar_eta[0]),float(cbar_eta[1]), color='magenta',
                        width = 0.02, head_width = 0.1, label=r"$\bar{c} \backslash \eta$")
        ax.scatter(feaspoints[:,0], feaspoints[:,1], color='green', zorder=8)
        ax.scatter(infeaspoints[:,0], infeaspoints[:,1], color='red', zorder=7)
        plt.legend(loc="upper left", title="slack")
    plt.title(r"In the space of the non-basic variables", size=18)
#ax.grid()
plt.show()

```

Gomory cutting-plane tool for dual-form pure-integer problem D_I

[Jon Lee \(<https://sites.google.com/site/jonleewebpage/home>\)](https://sites.google.com/site/jonleewebpage/home),
11/21/2020

For dual-form pure-integer problem

$$\begin{aligned}
 & \max y' b \\
 & y' A \leq c' ; \\
 & y \in \mathbb{Z}^m .
 \end{aligned} \tag{D_I}$$

Notes:

- A and c **MUST** be integer.
- The variables are y_0, y_1, \dots, y_{m-1} , so valid input arguments for `pure_gomory(i)` are $i \in \{0, 1, \dots, m-1\}$.

Reference:

- Qi He, Jon Lee. Another pedagogy for pure-integer Gomory. *RAIRO – Operations Research*, 51:189–197, 2017.

```
In [ ]: # pure gomory cut
def pure_gomory(i):
    global A, c, A_beta, A_eta, c_eta, cbar_eta, m, n, beta, eta
    if i>m-1:
        display(Latex("error: $i$ is out of range."))
    else:
        ei = sym.zeros(m,1)
        ei[i]=1 # ei is the i-th standard unit column
        hi = A_beta.solve(ei) # i-th column of basis inverse
        #r = -sym.floor(hi) # best choice of r
        r = -(hi.applyfunc(sym.floor))
        btilde = ei + A_beta*r # new column for P
        A = A.row_join(btilde)
        c = c.col_join(sym.Matrix(([sym.floor(ybar.dot(btilde))])))
        eta.insert(n-m,n)
        n += 1
        A_eta = copy.copy(A[:,eta])
        c_eta = copy.copy(c[eta,0])
        cbar_eta = c_eta - A_eta.transpose()*ybar
    print('*** PROBABLY WANT TO APPLY pivot_algebra()! ***')
```

```
In [ ]: # dual plot
def dual_plot(delta=None, center=None):
    if delta==None: delta=2
    if center==None: center=ybar
    if m != 2 or Perturb == True:
        display(Latex("Hey friend --- give me a break!"))
        display(Latex("This plotting only works if there are $m=2$ dual variables and no rhs perturbation"))
    return
M=sym.transpose(A)
feaspoints=np.empty((0,2))
infeaspoints=np.empty((0,2))
c2=sym.zeros(2,1)
M2=sym.zeros(2,2)
for i in range(n-1):
    for j in range(i+1,n):
        c2[0]=c[i]
        c2[1]=c[j]
        M2[0,:]=M[i,:]
        M2[1,:]=M[j,:]
        if abs(sym.det(M2)) > 0.0001:
            y0y1 = M2.solve(c2)
            if min(c - M*y0y1) >= -0.00001:
                feaspoints=np.r_[feaspoints,np.transpose(y0y1)]
            else:
                infeaspoints=np.r_[infeaspoints,np.transpose(y0y1)]
hull = ConvexHull(feaspoints)
fig, ax = plt.subplots(figsize=(8,8))
ax.xaxis.set_label_coords(1.05, 0.49)
ax.yaxis.set_label_coords(0.5, 1.05)
ax.set(xlabel=r"$y_0$".format(0), ylabel=r"$y_1$".format(1))
ax.spines['left'].set_position(('data',ybar[0]))
ax.spines['bottom'].set_position(('data',ybar[1]))
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
# set major ticks to show every 1 (integer)
ax.xaxis.set_major_locator(MultipleLocator(1))
ax.yaxis.set_major_locator(MultipleLocator(1))
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
plt.xlim(float(center[0])-delta, float(center[0])+delta)
plt.ylim(float(center[1])-delta, float(center[1])+delta)
plt.fill(feaspoints[hull.vertices,0], feaspoints[hull.vertices,1], 'cyan',
alpha=0.3)
y1 = np.linspace(float(min(feaspoints[:,0]))-0.5, float(max(feaspoints[:,0]))+0.5, 100)
for j in range(n):
    if M[j,1] != 0:
        y2 = (c[j] - M[j,0]*y1) / M[j,1]
        plt.plot(y1, y2, linewidth=2, label=r"constraint ${}$.format(j))
    else:
        plt.vlines(float(c[j]/ M[j,0]), float(center[1])-delta,
                  float(center[1])+delta, linewidth=2, label=r"constraint
${}$.format(j))
```

```
for simplex in hull.simplices:
    plt.fill(feaspoints[simplex, 0], feaspoints[simplex, 1], 'cyan', alpha=0.5)
    arrow=plt.arrow(float(ybar[0]),float(ybar[1]),0.5*float(b[0]/(b.dot(b))**0.5),0.5*float(b[1]/(b.dot(b))**0.5), color='magenta', width = 0.01*delta, head_width = 0.02*delta, label=r"$b$")
    ax.scatter(feaspoints[:,0], feaspoints[:,1], color='green', zorder=8)
    ax.scatter(infeaspoints[:,0], infeaspoints[:,1], color='red', zorder=7)
    # the integer grid
    xp = np.arange(np.floor(float(center[0])-delta)-1, np.ceil(float(center[0])+delta)+2)
    yp = np.arange(np.floor(float(center[1])-delta)-1, np.ceil(float(center[1])+delta)+2)
    pp = itertools.product(xp, yp)
    plt.scatter(*zip(*pp), marker='o', s=5, color='black', zorder=9)
    # sorting plot legend entries by label
    handles, labels = ax.get_legend_handles_labels()
    hl = sorted(zip(handles, labels), key=operator.itemgetter(1))
    handles2, labels2 = zip(*hl)
    ax.legend(handles2, labels2, loc="lower left", title="constraints")
    ax.grid(which='major')
plt.show()
```

In []: `print('pivot_tools loaded: pivot_perturb, pivot_algebra, N, pivot_ratios, pivot_swap, pivot_plot, pure_gomory, mixed_gomory, dual_plot')`

A.7 Circle.ipynb

Hoffman's circle

[Jon Lee \(https://sites.google.com/site/jonleewebpage/home\)](https://sites.google.com/site/jonleewebpage/home),
11/20/2020

Reference:

- Jon Lee. Hoffman's circle untangled. *SIAM Review*, 39(1):98-105, 1997.

MIT License

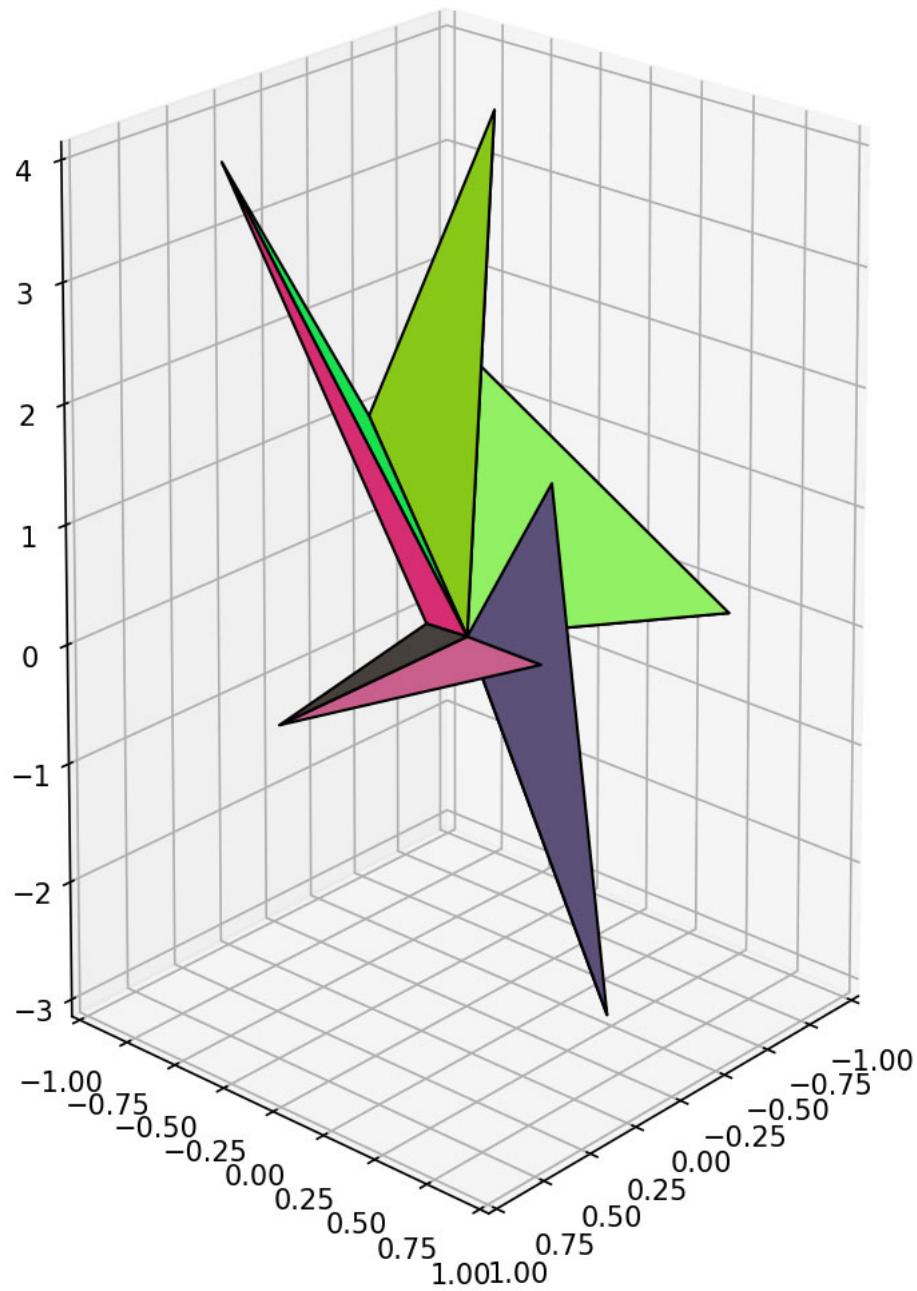
Copyright (c) 2020 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
In [1]: import numpy as np
%matplotlib notebook
#matplotlib inline
import mpl_toolkits.mplot3d as a3
import matplotlib.colors as colors
import pylab as pl
t = 2 * np.pi/5
c = np.cos(t)
s = np.sin(t)
M = np.array([[c, -s, 0], [s, c, 0], [(c - 1)/c, s/c, 1]])
x = np.array((1,0,0))
y = np.array((0, 0.5*np.tan(t/2), 0))
T=np.row_stack((x,y, M.dot(x), M.dot(y), M.dot(M.dot(x)), M.dot(M.dot(y)),
                 M.dot(M.dot(M.dot(x))), M.dot(M.dot(M.dot(y))),
                 M.dot(M.dot(M.dot(M.dot(x)))), M.dot(M.dot(M.dot(M.dot(y)))),
                 x))
ax = a3.Axes3D(pl.figure(figsize=(5,8)),azim=42,elev=15)
for i in range(10):
    vtx = np.row_stack(([0,0,0],T[i],T[i+1]))
    tri = a3.art3d.Poly3DCollection([vtx])
    tri.set_color(colors.rgb2hex(np.random.rand(3)))
    tri.set_edgecolor('k')
    ax.add_collection3d(tri)
ax.set_xlim3d(-1,1)
ax.set_ylim3d(-1,1)
ax.set_zlim3d(-3,4)
pl.show()
```



A.8 Decomp.ipynb

Decomposition Algorithm with Python/Gurobi

[Jon Lee \(https://sites.google.com/site/jonleewebpage/home\)](https://sites.google.com/site/jonleewebpage/home), 11/2/2020

Apply the (Dantzig-Wolfe) Decomposition Algorithm to:

$$\begin{aligned} z := \min \quad & c'x \\ \text{subject to} \quad & Ex \geq h; \\ & Ax = b; \\ & x \geq 0, \end{aligned} \tag{Q}$$

treating $Ex \geq h$ as the "complicating constraints".

Notes:

- In this implementation, we never delete generated columns

References:

- Jon Lee, "A First Course in Linear Optimization", Fourth Edition (Version 4.05), Reex Press, 2013-22.

MIT License

Copyright (c) 2020 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
In [1]: %reset -f
import numpy as np
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import gurobipy as gp
from gurobipy import GRB

class StopExecution(Exception):
    def __init__(self):
        pass
```

```
In [2]: MAXIT = 500
# generate a random example
n = 100 # number of variables
m1 = 200 # number of equations to relax
m2 = 50 # number of equations to keep
np.random.seed(25) # change the seed for a different example
E=0.01*np.random.randint(-5,high=5,size=(m1,n)).astype(float) #np.random.randn(m1,nt)
A=0.01*np.random.randint(-2,high=3,size=(m2,n)).astype(float) #np.random.randn(m2,nt)

# choose the right-hand sides so that Q will be feasible
xfeas=0.1*np.random.randint(0,high=5,size=n).astype(float)
h=E.dot(xfeas) - 0.1*np.random.randint(0,high=10,size=m1).astype(float)
b=A.dot(xfeas)

# choose the objective function so that the dual of Q will be feasible
yfeas=0.1*np.random.randint(0,high=5,size=m1).astype(float)
pifeas=0.1*np.random.randint(-5,high=5,size=m2).astype(float)
c=np.transpose(E)@yfeas + np.transpose(A)@pifeas + 0.1*np.random.randint(0,high=1,size=n).astype(float)
```

```
In [3]: print("***** Solve as one big LP --- for comparison purposes")
modelQ = gp.Model()
modelQ.reset()
xQ = modelQ.addMVar(n)
objective = modelQ.setObjective(c@xQ, GRB.MINIMIZE)
constraintsQ1 = modelQ.addConstr(E@xQ >= h)
constraintsQ2 = modelQ.addConstr(A@xQ == b)
modelQ.optimize()
if modelQ.status != GRB.Status.OPTIMAL:
    print("***** Gurobi solve status:", modelQ.status)
    print("***** This is a problem. Stopping execution.")
    raise StopExecution
print(" ")
print("***** Proceed to Decomposition")

# initialization for Decomposition
results1=[]
results2=[]
ITER=0
xgen=0
zgen=0
y=np.zeros(m1)

# set up the Subproblem model and get one basic feasible solution
models = gp.Model()
models.setParam('OutputFlag', 0) # quiet the Gurobi output
x = models.addMVar(n)
constraintsS = models.addConstr(A@x == b)
#models.setObjective(c@x, GRB.MINIMIZE)
models.optimize()
if models.status != GRB.Status.OPTIMAL:
    print("***** Gurobi (initial) Subproblem solve status:", models.status)
    print("***** This is a problem. Stopping execution.")
    raise StopExecution
xgen += 1

# construct a basis
XZ=np.reshape(x.X,(n,1))
#Z=np.r_[np.zeros((n-m1,m1)),np.eye(m1)]
#Z=np.empty((n,0), dtype=float)
h1=np.r_[h,(1)]
#B=np.c_[np.r_[np.eye(m1),np.zeros((1,m1))],np.r_[E@x.X,(1)]]
B=np.c_[-np.r_[np.eye(m1),np.zeros((1,m1))],np.r_[E@x.X,(1)]]

# set up the Main Phase-2 model
modelM2 = gp.Model()
s = modelM2.addMVar(m1+1)
modelM2.setObjective(c@x.X*s[m1], GRB.MINIMIZE)
modelM2.addConstrs((-s[i] + E[i,:]@x.X*s[m1] == h[i] for i in range(m1)))
modelM2.addConstr(s[m1]==1)
modelM2.update()
constraintsM2=modelM2.getConstrs()

# Identify if the constructed basis is feasible to see if Phase 1 is needed
if min(np.linalg.solve(B, h1)) >= -1e-10:
    print('***** Phase I not needed')
    Phase=2
    modelM=modelM2
else:
    print('***** Phase I needed')
    Phase=1
    ITERphaseI=1
    modelM1=modelM2.copy()
    modelM1.setParam('OutputFlag', 0) # quiet the Gurobi output
    constraintsM1=modelM1.getConstrs()
    # create the artificial variable
    newcol=gp.Column(-np.r_[E@x.X-np.ones(m1),(-1)],constraintsM1)
    modelM1.setObjective(0.0, GRB.MINIMIZE)
    modelM1.addVar(obj=1.0, column=newcol, name='artificial')
    modelM=modelM1
    modelM.setParam('OutputFlag', 0) # quiet the Gurobi output

while True:
    ITER += 1
    print(" ")
```

```

print("***** Currently in Phase", Phase, ". Iteration number", ITER)
print("***** Solving Main LP...")
modelM.optimize()
if modelM.status != GRB.Status.OPTIMAL:
    print("***** Gurobi Main solve status:", modelM.status)
    print("***** This is a problem. Stopping execution.")
    raise StopExecution
results1=np.append(results1,ITER-1)
results2=np.append(results2,modelM.Objval)
if Phase==1 and modelM.Objval < 0.000001:
    print("***** Phase I succeeded")
    print("LP iter", " LP val")
    print("-----")
    for j in range(ITER):
        print(int(results1[j]), " ", np.round(results2[j],9))
fig, ax = plt.subplots(figsize=(10,10))
ax.plot(results1[0:ITER], results2[0:ITER])
ax.set(xlabel='iteration', ylabel='LP objective value')
ax.set_xticks(ticks=results1, minor=False)
ax.grid()
plt.show()
ITERphaseI=ITER
Phase=2
# switch to the Phase II model
modelM=modelM2
modelM.setParam('OutputFlag', 0) # quiet the Gurobi output
modelM.optimize()
# overwrite last iteration result with phase-II objective value
results2[ITER-1]=modelM.Objval
if ITER == MAXIT: break

constraintsM=modelM.getConstrs()
for i in range(m1):
    y[i]=constraintsM[i].Pi
sigma=constraintsM[m1].Pi
if Phase==1: modelS.setObjective((-y.dot(E))@x, GRB.MINIMIZE)
else: modelS.setObjective((c-y.dot(E))@x, GRB.MINIMIZE)
print(" ")
print("***** Solving Subproblem LP...")
modelS.optimize()
if modelS.status != GRB.Status.OPTIMAL and modelS.status != GRB.Status.UNBOUNDED:
    print("***** Gurobi Subproblem solve status:", modelS.status)
    print("***** This is a problem. Stopping execution.")
    raise StopExecution
if modelS.status == GRB.Status.OPTIMAL:
    print("***** Gurobi Subproblem solve status:", modelS.status)
    reducedcost = -sigma + modelS.Objval
    print("***** sigma=",sigma)
    print("***** reduced cost=",reducedcost)
    if reducedcost < -0.0001:
        xnew=x.X
        if Phase==1:
            newcol=gp.Column(np.r_[E@xnew,(1)],constraintsM1)
            modelM1.addVar(obj=0.0, column=newcol)
            newcol=gp.Column(np.r_[E@xnew,(1)],constraintsM2)
            modelM2.addVar(obj=c@xnew, column=newcol)
            XZ=np.c_[XZ,xnew]
            xgen += 1
        else:
            if Phase==1:
                print("***** No more improving columns for Main")
                print("***** Phase I finished without a feasible solution")
                print("***** Phase I objective", modelM.Objval)
                break
            else: # Phase 2
                print("***** No more improving columns for Main")
                print("***** Phase II finished")
                print("***** Phase II objective", modelM.Objval)
                break
    if modelS.status == GRB.Status.UNBOUNDED:
        print("***** Gurobi Subproblem solve status:", modelS.status)
        znew=x.UnbdRay
        if Phase==1:
            newcol=gp.Column(np.r_[E@znew,(0)],constraintsM1)
            modelM1.addVar(obj=0.0, column=newcol)
            reducedcost = -y.dot(E)@znew

```

```

newcol=gp.Column(np.r_[E@znew,(0)],constraintsM2)
modelM2.addVar(obj=c@znew, column=newcol)
if Phase==2:
    reducedcost = (c-y.dot(E))@znew
    print("***** reduced cost=", reducedcost)
    #if reducedcost > 0.0001: input()
    XZ=np.c_[XZ,znew]
    zgen += 1

print("LP iter", " LP val")
print("----- -----")
for j in range(ITERphaseI-1,ITER):
    print(int(results1[j]), " ", np.round(results2[j],9))
# recover the solution in the original variables x
greekvar=modelM2.getVars()[m1:ITER+m1]
greekval=np.zeros(ITER)
for i in range(ITER):
    greekval[i] = greekvar[i].X
xhat=XZ@greekval
print("***** Reality check: recover the optimal x found by decomposition.")
print("***** Its objective value is:", np.round(c@xhat,9))
print(" ")
print("***** Compare with LP value calculated without decomposition:",np.round(modelQ.Objval,9))

if ITER > ITERphaseI:
    fig, ax = plt.subplots(figsize=(10,10))
    ax.plot(results1[ITERphaseI-1:ITER], results2[ITERphaseI-1:ITER])
    ax.plot(results1[ITERphaseI-1:ITER], modelQ.Objval*np.ones(ITER-ITERphaseI+1))
    ax.set(xlabel='iteration', ylabel='LP objective value')
    ax.set_xticks(ticks=results1[ITERphaseI-1:ITER], minor=True)
    ax.grid()
    plt.show()
print(" ")
print("***** Number of basic-feasible solutions generated:", xgen)
print(" ")
print("***** Number of basic-feasible rays generated:", zgen)

```

```
***** Solve as one big LP --- for comparison purposes
Set parameter Username
Academic license - for non-commercial use only - expires 2025-06-14
Discarded solution information
Gurobi Optimizer version 11.0.2 build v11.0.2rc0 (win64 - Windows 10.0 (19045.2))

CPU model: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 250 rows, 100 columns and 21957 nonzeros
Model fingerprint: 0xd5eae979
Coefficient statistics:
  Matrix range      [1e-02, 5e-02]
  Objective range   [2e-02, 5e-01]
  Bounds range      [0e+00, 0e+00]
  RHS range         [3e-18, 1e+00]
Presolve time: 0.01s
Presolved: 250 rows, 100 columns, 21957 nonzeros

Iteration    Objective       Primal Inf.     Dual Inf.     Time
      0    -4.2122000e+31    1.799360e+33    4.212200e+01      0s
     211    -5.6119344e+00    0.000000e+00    0.000000e+00      0s

Solved in 211 iterations and 0.04 seconds (0.03 work units)
Optimal objective -5.611934358e+00

***** Proceed to Decomposition
***** Phase I needed

***** Currently in Phase 1 . Iteration number 1
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.8888889762412218

***** Currently in Phase 1 . Iteration number 2
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.2224915654958144

***** Currently in Phase 1 . Iteration number 3
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.4792036653947186

***** Currently in Phase 1 . Iteration number 4
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.22440797623489983

***** Currently in Phase 1 . Iteration number 5
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.5340356492257962

***** Currently in Phase 1 . Iteration number 6
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.2279241802577182

***** Currently in Phase 1 . Iteration number 7
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
```

```
***** reduced cost= -0.09110396626994344
***** Currently in Phase 1 . Iteration number 8
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.3968156834747276

***** Currently in Phase 1 . Iteration number 9
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.41978215586145495

***** Currently in Phase 1 . Iteration number 10
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0018015352823892161

***** Currently in Phase 1 . Iteration number 11
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.00010250271284489854

***** Currently in Phase 1 . Iteration number 12
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.07981521492913896
***** reduced cost= -0.30466811062938265

***** Currently in Phase 1 . Iteration number 13
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.16499004968754516

***** Currently in Phase 1 . Iteration number 14
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0072739512112391935

***** Currently in Phase 1 . Iteration number 15
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.08536314880703423
***** reduced cost= -0.281317880821247

***** Currently in Phase 1 . Iteration number 16
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.08442387544888776
***** reduced cost= -0.24745008025160126

***** Currently in Phase 1 . Iteration number 17
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.1684466794844134

***** Currently in Phase 1 . Iteration number 18
```

```
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.06948938457101494

***** Currently in Phase 1 . Iteration number 19
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.006733993980674313

***** Currently in Phase 1 . Iteration number 20
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.07516966535204106
***** reduced cost= -0.36478214781102325

***** Currently in Phase 1 . Iteration number 21
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0430249309229985

***** Currently in Phase 1 . Iteration number 22
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.22159149090059715

***** Currently in Phase 1 . Iteration number 23
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0013215098274264037

***** Currently in Phase 1 . Iteration number 24
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.07630635705376518
***** reduced cost= -0.29016678213482927

***** Currently in Phase 1 . Iteration number 25
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.076377719236034
***** reduced cost= -0.2509001623629846

***** Currently in Phase 1 . Iteration number 26
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.07759125749970142
***** reduced cost= -0.2339324542081378

***** Currently in Phase 1 . Iteration number 27
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -8.780118001394815

***** Currently in Phase 1 . Iteration number 28
***** Solving Main LP...
```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.09568091915114302

***** Currently in Phase 1 . Iteration number 29
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.0926157983598297

***** Currently in Phase 1 . Iteration number 30
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.03448309553914619

***** Currently in Phase 1 . Iteration number 31
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.04629229883919406

***** Currently in Phase 1 . Iteration number 32
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.04508764252266495

***** Currently in Phase 1 . Iteration number 33
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.017109936553204252

***** Currently in Phase 1 . Iteration number 34
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.00031773789890373813

***** Currently in Phase 1 . Iteration number 35
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.00033478801307243455

***** Currently in Phase 1 . Iteration number 36
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.09408705091609795
***** reduced cost= -0.34365765953823413

***** Currently in Phase 1 . Iteration number 37
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.5415315094663868

***** Currently in Phase 1 . Iteration number 38
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.03279177734678951

***** Currently in Phase 1 . Iteration number 39
```

```
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.1777604722368462

***** Currently in Phase 1 . Iteration number 40
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0022424935270347057

***** Currently in Phase 1 . Iteration number 41
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.006344928562024843

***** Currently in Phase 1 . Iteration number 42
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.03357562461679886

***** Currently in Phase 1 . Iteration number 43
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0033744569016675996

***** Currently in Phase 1 . Iteration number 44
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.0900098187938332
***** reduced cost= -0.24911084537826017

***** Currently in Phase 1 . Iteration number 45
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.08951197542846455
***** reduced cost= -0.25300943398893777

***** Currently in Phase 1 . Iteration number 46
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.0900284281545457
***** reduced cost= -0.22984670329947943

***** Currently in Phase 1 . Iteration number 47
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.7980892258922567

***** Currently in Phase 1 . Iteration number 48
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.6027100362089755

***** Currently in Phase 1 . Iteration number 49
***** Solving Main LP...

***** Solving Subproblem LP...
```

```
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.11427322014188163

***** Currently in Phase 1 . Iteration number 50
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -51.69189583031917

***** Currently in Phase 1 . Iteration number 51
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.08402811150926937

***** Currently in Phase 1 . Iteration number 52
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.05998116887431593

***** Currently in Phase 1 . Iteration number 53
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.021820372253347764

***** Currently in Phase 1 . Iteration number 54
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.010578662076998535

***** Currently in Phase 1 . Iteration number 55
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.08684779851282028
***** reduced cost= -0.2535630666754055

***** Currently in Phase 1 . Iteration number 56
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.14782297344379247

***** Currently in Phase 1 . Iteration number 57
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0007692065993953889

***** Currently in Phase 1 . Iteration number 58
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.08862970101149362
***** reduced cost= -0.19950241557667941

***** Currently in Phase 1 . Iteration number 59
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.08817363273849654
***** reduced cost= -0.1855363772369614
```

```

***** Currently in Phase 1 . Iteration number 60
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.34313372063073194

***** Currently in Phase 1 . Iteration number 61
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.05891729554389308

***** Currently in Phase 1 . Iteration number 62
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.0930959874491292
***** reduced cost= -0.22265628494346093

***** Currently in Phase 1 . Iteration number 63
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.09292839693236621
***** reduced cost= -0.1611028800244138

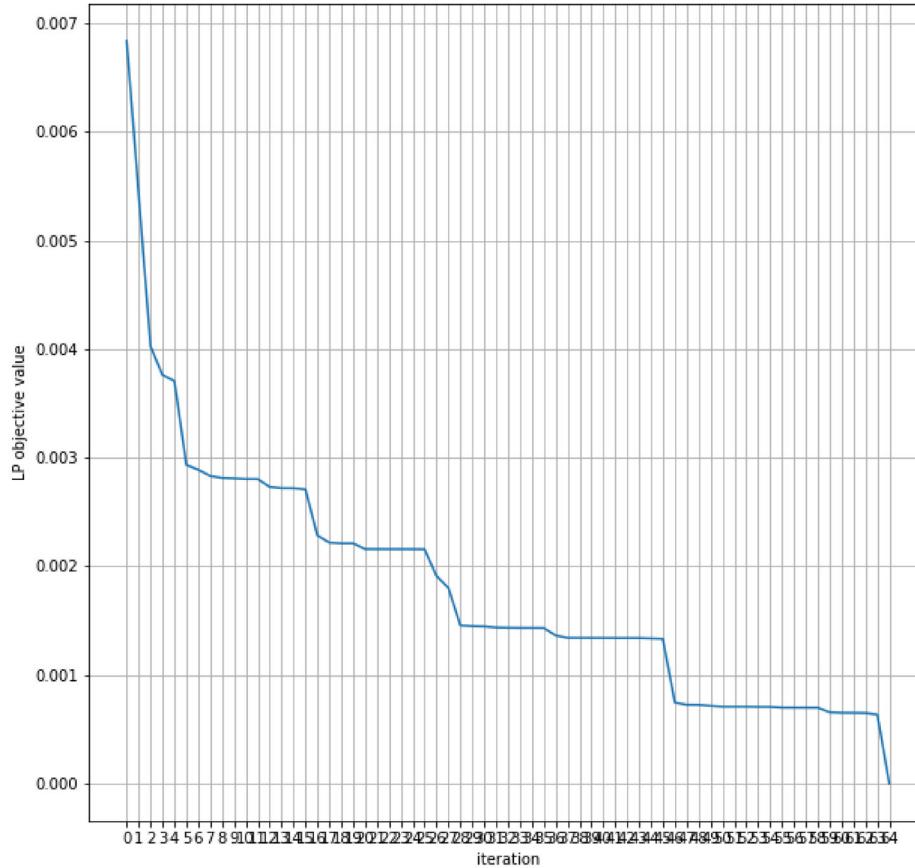
***** Currently in Phase 1 . Iteration number 64
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.09135367132685802
***** reduced cost= -0.12990632255737256

***** Currently in Phase 1 . Iteration number 65
***** Solving Main LP...
***** Phase I succeeded
LP iter      LP val
----- -----
0            0.006835071
1            0.005424588
2            0.004025531
3            0.003760887
4            0.003707445
5            0.002933915
6            0.002890196
7            0.002832538
8            0.002813929
9            0.002809714
10           0.002804215
11           0.00280392
12           0.002730938
13           0.002719706
14           0.002719284
15           0.002708503
16           0.002285412
17           0.002218841
18           0.002212064
19           0.002211955
20           0.002159507
21           0.002158732
22           0.002158548
23           0.002158547
24           0.00215847
25           0.002157155
26           0.001911139
27           0.001801355
28           0.001457574
29           0.00145069
30           0.001447753
31           0.001437339
32           0.001434842

```

```
33      0.00143382
34      0.001433319
35      0.001433002
36      0.001363848
37      0.001341909
38      0.00134136
39      0.001341098
40      0.001340498
41      0.00133999
42      0.001339711
43      0.0013397
44      0.001336228
45      0.001332589
46      0.000746315
47      0.000725292
48      0.000724647
49      0.000715795
50      0.000708423
51      0.000708421
52      0.000708411
53      0.000706712
54      0.000706446
55      0.000699676
56      0.000699205
57      0.000699204
58      0.000698759
59      0.000656658
60      0.0006522
61      0.000651253
62      0.000649771
63      0.000635845
64      0.0
```



```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -77.59813656922532

***** Currently in Phase 2 . Iteration number 66
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.8845080684140214

***** Currently in Phase 2 . Iteration number 67
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 6.747602727886344
***** reduced cost= -29.07400478763364

***** Currently in Phase 2 . Iteration number 68
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 6.995560329924679
***** reduced cost= -26.669662747087166

***** Currently in Phase 2 . Iteration number 69
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -33.447966681929664

***** Currently in Phase 2 . Iteration number 70
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -5.867959499555795

***** Currently in Phase 2 . Iteration number 71
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -3.5416420378710427

***** Currently in Phase 2 . Iteration number 72
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.953450757417702

***** Currently in Phase 2 . Iteration number 73
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 3.3177536085183448
***** reduced cost= -19.4910179774486

***** Currently in Phase 2 . Iteration number 74
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 3.643872277826989
***** reduced cost= -17.178481511314704

***** Currently in Phase 2 . Iteration number 75
***** Solving Main LP...

***** Solving Subproblem LP...
```

```
***** Gurobi Subproblem solve status: 5
***** reduced cost= -25.754882197827328

***** Currently in Phase 2 . Iteration number 76
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.19204463268852123

***** Currently in Phase 2 . Iteration number 77
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 5.480692390939993
***** reduced cost= -26.231815504934033

***** Currently in Phase 2 . Iteration number 78
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 5.558949784367123
***** reduced cost= -20.39590414253263

***** Currently in Phase 2 . Iteration number 79
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 5.978762315590637
***** reduced cost= -17.034380500931952

***** Currently in Phase 2 . Iteration number 80
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -2.9661361210598693

***** Currently in Phase 2 . Iteration number 81
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -178.8674525701388

***** Currently in Phase 2 . Iteration number 82
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -213.3039731263416

***** Currently in Phase 2 . Iteration number 83
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -41.932909535680665

***** Currently in Phase 2 . Iteration number 84
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -23.8096702650674

***** Currently in Phase 2 . Iteration number 85
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.8013202986781484
```

```
***** Currently in Phase 2 . Iteration number 86
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.5937964640419291

***** Currently in Phase 2 . Iteration number 87
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -4.300060441744137

***** Currently in Phase 2 . Iteration number 88
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -24.144139593867067

***** Currently in Phase 2 . Iteration number 89
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -2.0372112302400183

***** Currently in Phase 2 . Iteration number 90
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.8961642268885157

***** Currently in Phase 2 . Iteration number 91
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -18.10650391685317

***** Currently in Phase 2 . Iteration number 92
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -2.2774276227451082

***** Currently in Phase 2 . Iteration number 93
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.5541118487919903

***** Currently in Phase 2 . Iteration number 94
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.19682180858411158

***** Currently in Phase 2 . Iteration number 95
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.06640473154281201

***** Currently in Phase 2 . Iteration number 96
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.1527322518869445
```

```
***** Currently in Phase 2 . Iteration number 97
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.038430942826860104

***** Currently in Phase 2 . Iteration number 98
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.019650058339762744

***** Currently in Phase 2 . Iteration number 99
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 1.5095325048527433
***** reduced cost= -8.798952570283403

***** Currently in Phase 2 . Iteration number 100
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -4.462468155605427

***** Currently in Phase 2 . Iteration number 101
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -10.633421894208592

***** Currently in Phase 2 . Iteration number 102
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.9535365218046206

***** Currently in Phase 2 . Iteration number 103
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -52.23792187286517

***** Currently in Phase 2 . Iteration number 104
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.24983630736404905

***** Currently in Phase 2 . Iteration number 105
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.015426017505934642

***** Currently in Phase 2 . Iteration number 106
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 1.490433397676426
***** reduced cost= -7.102806602433631

***** Currently in Phase 2 . Iteration number 107
***** Solving Main LP...
```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -10.459199652548968

***** Currently in Phase 2 . Iteration number 108
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.2066645781908183
***** reduced cost= -8.890023142084338

***** Currently in Phase 2 . Iteration number 109
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.1544445928388076
***** reduced cost= -7.666295118457549

***** Currently in Phase 2 . Iteration number 110
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 1.7002322636321772
***** reduced cost= -6.667988704920401

***** Currently in Phase 2 . Iteration number 111
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -65.87045319634314

***** Currently in Phase 2 . Iteration number 112
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -138.6431282537927

***** Currently in Phase 2 . Iteration number 113
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -360.2275123490729

***** Currently in Phase 2 . Iteration number 114
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -29.7495537385512

***** Currently in Phase 2 . Iteration number 115
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.8274548648716173

***** Currently in Phase 2 . Iteration number 116
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -60.44740006637323

***** Currently in Phase 2 . Iteration number 117
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -18.606525055024996
```

```
***** Currently in Phase 2 . Iteration number 118
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -8.103175956905375

***** Currently in Phase 2 . Iteration number 119
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -15.843869913974487

***** Currently in Phase 2 . Iteration number 120
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.3442275767961398

***** Currently in Phase 2 . Iteration number 121
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.0972013486697225

***** Currently in Phase 2 . Iteration number 122
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.0618564397606747

***** Currently in Phase 2 . Iteration number 123
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.6576864780605192

***** Currently in Phase 2 . Iteration number 124
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -3.2864497198288376

***** Currently in Phase 2 . Iteration number 125
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.3092776738417342

***** Currently in Phase 2 . Iteration number 126
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.30721439320360844

***** Currently in Phase 2 . Iteration number 127
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.37626743466725543

***** Currently in Phase 2 . Iteration number 128
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
```

```
***** reduced cost= -0.6273360120061398
***** Currently in Phase 2 . Iteration number 129
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.9773417006620093

***** Currently in Phase 2 . Iteration number 130
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.10982473440113694

***** Currently in Phase 2 . Iteration number 131
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.018157208555427573

***** Currently in Phase 2 . Iteration number 132
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.15096420908662367

***** Currently in Phase 2 . Iteration number 133
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.021876813059961364

***** Currently in Phase 2 . Iteration number 134
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.02623235856623296

***** Currently in Phase 2 . Iteration number 135
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.014711304736778175

***** Currently in Phase 2 . Iteration number 136
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.012693565832705644

***** Currently in Phase 2 . Iteration number 137
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.002543819096491262

***** Currently in Phase 2 . Iteration number 138
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0009401535819364426

***** Currently in Phase 2 . Iteration number 139
***** Solving Main LP...
***** Solving Subproblem LP...
```

```
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0005667619282654357

***** Currently in Phase 2 . Iteration number 140
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.5178069587968883
***** reduced cost= -8.538847186253673

***** Currently in Phase 2 . Iteration number 141
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -2.5727486243465023

***** Currently in Phase 2 . Iteration number 142
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.2175282747472238

***** Currently in Phase 2 . Iteration number 143
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.18609930244647188

***** Currently in Phase 2 . Iteration number 144
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.29938439563720465

***** Currently in Phase 2 . Iteration number 145
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.5420478617519393

***** Currently in Phase 2 . Iteration number 146
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0679423144801774

***** Currently in Phase 2 . Iteration number 147
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.149466934587328
***** reduced cost= -6.679301397356322

***** Currently in Phase 2 . Iteration number 148
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.8579449659727345

***** Currently in Phase 2 . Iteration number 149
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.13428816483011374

***** Currently in Phase 2 . Iteration number 150
```

```
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.016910048037431125

***** Currently in Phase 2 . Iteration number 151
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0003619339601499405

***** Currently in Phase 2 . Iteration number 152
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.1035124292384637
***** reduced cost= -6.874582185144883

***** Currently in Phase 2 . Iteration number 153
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.16605561040659822

***** Currently in Phase 2 . Iteration number 154
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.002009479316289031

***** Currently in Phase 2 . Iteration number 155
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.123094242923779
***** reduced cost= -6.787138320401189

***** Currently in Phase 2 . Iteration number 156
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -4.093915773885044

***** Currently in Phase 2 . Iteration number 157
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.03515880730867327

***** Currently in Phase 2 . Iteration number 158
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.8296467088026134

***** Currently in Phase 2 . Iteration number 159
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.09822839841839937

***** Currently in Phase 2 . Iteration number 160
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
```

```
***** reduced cost= -0.017995439163954075
***** Currently in Phase 2 . Iteration number 161
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0015200109858142241

***** Currently in Phase 2 . Iteration number 162
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.4608824442317996
***** reduced cost= -8.029157084762566

***** Currently in Phase 2 . Iteration number 163
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.427463780032653
***** reduced cost= -8.325000316119878

***** Currently in Phase 2 . Iteration number 164
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.002339532857217108

***** Currently in Phase 2 . Iteration number 165
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.605970239721347
***** reduced cost= -8.398729996228145

***** Currently in Phase 2 . Iteration number 166
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -42.37156574096746

***** Currently in Phase 2 . Iteration number 167
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.8879131410663064

***** Currently in Phase 2 . Iteration number 168
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.14347420860039573

***** Currently in Phase 2 . Iteration number 169
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.11494557288887464

***** Currently in Phase 2 . Iteration number 170
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.012063082135078673

***** Currently in Phase 2 . Iteration number 171
```

```
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.003576860592608977

***** Currently in Phase 2 . Iteration number 172
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 1.956178507004239
***** reduced cost= -7.9394339681884105

***** Currently in Phase 2 . Iteration number 173
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -5.6086277599311245

***** Currently in Phase 2 . Iteration number 174
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.121128853143852
***** reduced cost= -8.220354487900273

***** Currently in Phase 2 . Iteration number 175
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.1283819090767775
***** reduced cost= -7.679052280910091

***** Currently in Phase 2 . Iteration number 176
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.7598982361215834
***** reduced cost= -8.041425024146768

***** Currently in Phase 2 . Iteration number 177
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -3.4252709811061415

***** Currently in Phase 2 . Iteration number 178
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -3.830344779457903

***** Currently in Phase 2 . Iteration number 179
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -11.173160401534915

***** Currently in Phase 2 . Iteration number 180
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.8146575245527856

***** Currently in Phase 2 . Iteration number 181
***** Solving Main LP...
```

```

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.621234786565779

***** Currently in Phase 2 . Iteration number 182
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.3872393670202214
***** reduced cost= -8.750106837633666

***** Currently in Phase 2 . Iteration number 183
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.3919750867339
***** reduced cost= -7.754577563163368

***** Currently in Phase 2 . Iteration number 184
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.45682652945019
***** reduced cost= -5.5787990979757796

***** Currently in Phase 2 . Iteration number 185
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 3.2882849831115735
***** reduced cost= -8.771693519604801

***** Currently in Phase 2 . Iteration number 186
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -9.350808936216076

***** Currently in Phase 2 . Iteration number 187
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -14.452997696265527

***** Currently in Phase 2 . Iteration number 188
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 3.3506879113665686
***** reduced cost= -13.360812432455402

***** Currently in Phase 2 . Iteration number 189
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 3.3148312000260827
***** reduced cost= -10.422785141554797

***** Currently in Phase 2 . Iteration number 190
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 3.1353091637756183
***** reduced cost= -5.899531527977356

***** Currently in Phase 2 . Iteration number 191
***** Solving Main LP...

```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.9343622992598295
***** reduced cost= -5.221362665161161

***** Currently in Phase 2 . Iteration number 192
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -41.04849673391038

***** Currently in Phase 2 . Iteration number 193
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.5025382055600387

***** Currently in Phase 2 . Iteration number 194
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -5.980813683394501

***** Currently in Phase 2 . Iteration number 195
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.6422621860830304

***** Currently in Phase 2 . Iteration number 196
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.7094893157880335

***** Currently in Phase 2 . Iteration number 197
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.40719112533348323

***** Currently in Phase 2 . Iteration number 198
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 6.676554306007255
***** reduced cost= -15.062674122148405

***** Currently in Phase 2 . Iteration number 199
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 6.860078681738104
***** reduced cost= -14.281310357859784

***** Currently in Phase 2 . Iteration number 200
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 6.839719893661485
***** reduced cost= -17.177667549387337

***** Currently in Phase 2 . Iteration number 201
***** Solving Main LP...

***** Solving Subproblem LP...
```

```
***** Gurobi Subproblem solve status: 5
***** reduced cost= -6.511548061971832

***** Currently in Phase 2 . Iteration number 202
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -8.01711213563312

***** Currently in Phase 2 . Iteration number 203
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -25.646982731462458

***** Currently in Phase 2 . Iteration number 204
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -8.555817141740249

***** Currently in Phase 2 . Iteration number 205
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.25010422091950435

***** Currently in Phase 2 . Iteration number 206
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -52.17066904401702

***** Currently in Phase 2 . Iteration number 207
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.8602186873547453

***** Currently in Phase 2 . Iteration number 208
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.4729762600281333

***** Currently in Phase 2 . Iteration number 209
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.8323589931887625

***** Currently in Phase 2 . Iteration number 210
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.6475418609929875

***** Currently in Phase 2 . Iteration number 211
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.20329758663353914

***** Currently in Phase 2 . Iteration number 212
***** Solving Main LP...
```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.10036207137568454

***** Currently in Phase 2 . Iteration number 213
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 4.821453051093709
***** reduced cost= -11.25418532535505

***** Currently in Phase 2 . Iteration number 214
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 4.921294979196924
***** reduced cost= -10.471891993839586

***** Currently in Phase 2 . Iteration number 215
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 4.862491650860447
***** reduced cost= -10.276274913375604

***** Currently in Phase 2 . Iteration number 216
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -16.009697821449265

***** Currently in Phase 2 . Iteration number 217
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.4044692743808378

***** Currently in Phase 2 . Iteration number 218
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.6415089699867877

***** Currently in Phase 2 . Iteration number 219
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.07862449773215019

***** Currently in Phase 2 . Iteration number 220
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.12031299211801638

***** Currently in Phase 2 . Iteration number 221
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.1664911349730005

***** Currently in Phase 2 . Iteration number 222
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.21833191805626484
```

```
***** Currently in Phase 2 . Iteration number 223
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 5.251253500707105
***** reduced cost= -9.501198299103049

***** Currently in Phase 2 . Iteration number 224
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -3.484012380460136

***** Currently in Phase 2 . Iteration number 225
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.06800720402732097

***** Currently in Phase 2 . Iteration number 226
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 5.085236386591818
***** reduced cost= -9.060447816508935

***** Currently in Phase 2 . Iteration number 227
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.15239819591579717

***** Currently in Phase 2 . Iteration number 228
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 5.423776177294972
***** reduced cost= -9.488497600477446

***** Currently in Phase 2 . Iteration number 229
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 5.2528221701649365
***** reduced cost= -8.534163500519437

***** Currently in Phase 2 . Iteration number 230
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 6.326657375151726
***** reduced cost= -9.327296713295427

***** Currently in Phase 2 . Iteration number 231
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 6.552560106721817
***** reduced cost= -11.203465594563358

***** Currently in Phase 2 . Iteration number 232
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 8.086802536664067
```

```
***** reduced cost= -13.512357524810595
***** Currently in Phase 2 . Iteration number 233
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 5.113193092849183
***** reduced cost= -7.730229870461857
***** Currently in Phase 2 . Iteration number 234
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -377.23971574376367
***** Currently in Phase 2 . Iteration number 235
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 4.168357603084864
***** reduced cost= -16.224501234095904
***** Currently in Phase 2 . Iteration number 236
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 4.173093837340323
***** reduced cost= -15.404331659635092
***** Currently in Phase 2 . Iteration number 237
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 4.408015715244671
***** reduced cost= -9.579551252253708
***** Currently in Phase 2 . Iteration number 238
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 5.26324223342772
***** reduced cost= -7.859742097065015
***** Currently in Phase 2 . Iteration number 239
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -15.011091607419335
***** Currently in Phase 2 . Iteration number 240
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.9340266559871111
***** Currently in Phase 2 . Iteration number 241
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.9794114989566935
***** Currently in Phase 2 . Iteration number 242
***** Solving Main LP...
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.05133416738215146
```

```
***** Currently in Phase 2 . Iteration number 243
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 6.0769852176727035
***** reduced cost= -19.274410061433745

***** Currently in Phase 2 . Iteration number 244
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 6.089311583920809
***** reduced cost= -16.30015110365849

***** Currently in Phase 2 . Iteration number 245
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 6.38819037758944
***** reduced cost= -9.841769186673703

***** Currently in Phase 2 . Iteration number 246
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 7.728724293234618
***** reduced cost= -8.220704395710113

***** Currently in Phase 2 . Iteration number 247
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 5.392446426478844
***** reduced cost= -7.448409458269741

***** Currently in Phase 2 . Iteration number 248
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.979059218182322
***** reduced cost= -3.5741472975920177

***** Currently in Phase 2 . Iteration number 249
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.8657767164370422
***** reduced cost= -3.138816651040244

***** Currently in Phase 2 . Iteration number 250
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.4634319483141693
***** reduced cost= -3.5956477114930467

***** Currently in Phase 2 . Iteration number 251
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.071215769512394
***** reduced cost= -3.888432630433099

***** Currently in Phase 2 . Iteration number 252
***** Solving Main LP...
```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 4.219742524344925
***** reduced cost= -4.029951847718467

***** Currently in Phase 2 . Iteration number 253
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -16.73549297953592

***** Currently in Phase 2 . Iteration number 254
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -6.550307697826611

***** Currently in Phase 2 . Iteration number 255
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -2.0787542975450606

***** Currently in Phase 2 . Iteration number 256
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.6512752041418821

***** Currently in Phase 2 . Iteration number 257
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.15445806962920017

***** Currently in Phase 2 . Iteration number 258
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.37919485005350695
***** reduced cost= -5.777034814128528

***** Currently in Phase 2 . Iteration number 259
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -1.6024437015977213

***** Currently in Phase 2 . Iteration number 260
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.0014296658935899487

***** Currently in Phase 2 . Iteration number 261
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.5676148562203531
***** reduced cost= -4.180515701341796

***** Currently in Phase 2 . Iteration number 262
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.4895221481108929
```

```
***** reduced cost= -2.87019455337653

***** Currently in Phase 2 . Iteration number 263
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 5
***** reduced cost= -0.08736167751194479

***** Currently in Phase 2 . Iteration number 264
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.02876615335577737
***** reduced cost= -4.207153212823049

***** Currently in Phase 2 . Iteration number 265
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.13155967796001172
***** reduced cost= -3.1488642079231624

***** Currently in Phase 2 . Iteration number 266
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.8106438088795097
***** reduced cost= -5.269182434377169

***** Currently in Phase 2 . Iteration number 267
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.0901719278226158
***** reduced cost= -1.7893626477385673

***** Currently in Phase 2 . Iteration number 268
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.46799261912294043
***** reduced cost= -5.296272399085914

***** Currently in Phase 2 . Iteration number 269
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.0932493576419082
***** reduced cost= -3.1184887112057855

***** Currently in Phase 2 . Iteration number 270
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.4194250026289923
***** reduced cost= -2.465626072431696

***** Currently in Phase 2 . Iteration number 271
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.9073042913925894
***** reduced cost= -2.6024533723052268

***** Currently in Phase 2 . Iteration number 272
***** Solving Main LP...
```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.6551970700886848
***** reduced cost= -2.74709400363024

***** Currently in Phase 2 . Iteration number 273
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.8624173917611622
***** reduced cost= -2.012696627175741

***** Currently in Phase 2 . Iteration number 274
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 2.445948737371542
***** reduced cost= -3.621584206891452

***** Currently in Phase 2 . Iteration number 275
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.3517385050480418
***** reduced cost= -1.513782750158894

***** Currently in Phase 2 . Iteration number 276
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 1.5504069881903857
***** reduced cost= -2.8647114365797055

***** Currently in Phase 2 . Iteration number 277
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.18992897399000752
***** reduced cost= -1.5216297771910923

***** Currently in Phase 2 . Iteration number 278
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.7626567635478297
***** reduced cost= -1.71106586295417

***** Currently in Phase 2 . Iteration number 279
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.6281495584449459
***** reduced cost= -1.3635239043105738

***** Currently in Phase 2 . Iteration number 280
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.07204584032749062
***** reduced cost= -4.304242050686414

***** Currently in Phase 2 . Iteration number 281
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.4315495476363276
***** reduced cost= -3.3235284655897956
```

```
***** Currently in Phase 2 . Iteration number 282
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.1867770783676051
***** reduced cost= -3.2654970578282017

***** Currently in Phase 2 . Iteration number 283
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.34720411970897963
***** reduced cost= -1.7426422450304169

***** Currently in Phase 2 . Iteration number 284
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.4595495519411771
***** reduced cost= -2.6566898824439296

***** Currently in Phase 2 . Iteration number 285
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 1.143840280675614
***** reduced cost= -2.514188427767097

***** Currently in Phase 2 . Iteration number 286
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8493119425888271
***** reduced cost= -1.3827902050932401

***** Currently in Phase 2 . Iteration number 287
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.2910609623580704
***** reduced cost= -3.0358184197544675

***** Currently in Phase 2 . Iteration number 288
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1146584990861799
***** reduced cost= -2.1576099744662716

***** Currently in Phase 2 . Iteration number 289
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0259094835048082
***** reduced cost= -1.8451826294514384

***** Currently in Phase 2 . Iteration number 290
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7777218591661296
***** reduced cost= -1.357147531516254

***** Currently in Phase 2 . Iteration number 291
***** Solving Main LP...
```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8005272105873584
***** reduced cost= -0.9259415516855212

***** Currently in Phase 2 . Iteration number 292
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.6387839947076799
***** reduced cost= -1.3056625612200818

***** Currently in Phase 2 . Iteration number 293
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1844362448402623
***** reduced cost= -1.1474960548263076

***** Currently in Phase 2 . Iteration number 294
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7958356421761349
***** reduced cost= -0.7638860728599827

***** Currently in Phase 2 . Iteration number 295
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7128876224705228
***** reduced cost= -1.2413471339272073

***** Currently in Phase 2 . Iteration number 296
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.30334009352744024
***** reduced cost= -1.007809358267505

***** Currently in Phase 2 . Iteration number 297
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.5366759108736098
***** reduced cost= -1.2663365177380848

***** Currently in Phase 2 . Iteration number 298
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.5460268437249901
***** reduced cost= -1.2354556023158878

***** Currently in Phase 2 . Iteration number 299
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.48671257632935605
***** reduced cost= -0.9841431653946624

***** Currently in Phase 2 . Iteration number 300
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.1200571135312195
***** reduced cost= -0.9249292189617208
```

```
***** Currently in Phase 2 . Iteration number 301
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.5393484158107146
***** reduced cost= -0.8882708226470127

***** Currently in Phase 2 . Iteration number 302
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8672599776175991
***** reduced cost= -1.5078224993726748

***** Currently in Phase 2 . Iteration number 303
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7033457539394122
***** reduced cost= -1.1379947265647261

***** Currently in Phase 2 . Iteration number 304
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.5619970471296043
***** reduced cost= -0.6928745419750206

***** Currently in Phase 2 . Iteration number 305
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0315517928018125
***** reduced cost= -1.0777239072113853

***** Currently in Phase 2 . Iteration number 306
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8991007978351786
***** reduced cost= -1.0018584843406664

***** Currently in Phase 2 . Iteration number 307
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7881048677936828
***** reduced cost= -0.959268444613885

***** Currently in Phase 2 . Iteration number 308
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.73138094422645
***** reduced cost= -1.437022961340618

***** Currently in Phase 2 . Iteration number 309
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.6355757623564515
***** reduced cost= -1.2160293653801075

***** Currently in Phase 2 . Iteration number 310
***** Solving Main LP...
```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.6920498524690217
***** reduced cost= -1.012130541720161

***** Currently in Phase 2 . Iteration number 311
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.6450813709083704
***** reduced cost= -0.8722612923342602

***** Currently in Phase 2 . Iteration number 312
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7241983198400936
***** reduced cost= -0.8790706670187349

***** Currently in Phase 2 . Iteration number 313
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.6636090358160702
***** reduced cost= -0.5492246454602305

***** Currently in Phase 2 . Iteration number 314
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7506082789518541
***** reduced cost= -1.1777070917639376

***** Currently in Phase 2 . Iteration number 315
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.13092969497819643
***** reduced cost= -2.0528679131816525

***** Currently in Phase 2 . Iteration number 316
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7148161799523205
***** reduced cost= -1.073665001007796

***** Currently in Phase 2 . Iteration number 317
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7343546503989297
***** reduced cost= -0.5254554555413019

***** Currently in Phase 2 . Iteration number 318
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.881500634919983
***** reduced cost= -0.6186066341867336

***** Currently in Phase 2 . Iteration number 319
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.49584011714712856
***** reduced cost= -0.5334632060665669
```

```
***** Currently in Phase 2 . Iteration number 320
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.3099791035113908
***** reduced cost= -1.228112665104114

***** Currently in Phase 2 . Iteration number 321
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.642822383265704
***** reduced cost= -0.6725687732463203

***** Currently in Phase 2 . Iteration number 322
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.4894542658870467
***** reduced cost= -0.7928184674509291

***** Currently in Phase 2 . Iteration number 323
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.38773115163030986
***** reduced cost= -0.5919819076901756

***** Currently in Phase 2 . Iteration number 324
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.6069594531956879
***** reduced cost= -0.5188988959217697

***** Currently in Phase 2 . Iteration number 325
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7744054710510503
***** reduced cost= -0.41155820033431334

***** Currently in Phase 2 . Iteration number 326
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= 0.02304733745994017
***** reduced cost= -1.1548894226513342

***** Currently in Phase 2 . Iteration number 327
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.5333026276010084
***** reduced cost= -0.48941211413789687

***** Currently in Phase 2 . Iteration number 328
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.827346820292534
***** reduced cost= -0.38930755724796673

***** Currently in Phase 2 . Iteration number 329
***** Solving Main LP...
```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0441547879787043
***** reduced cost= -0.30486952114433685

***** Currently in Phase 2 . Iteration number 330
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9125934367808235
***** reduced cost= -0.33980253149543715

***** Currently in Phase 2 . Iteration number 331
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0493833249377034
***** reduced cost= -0.4192332112111812

***** Currently in Phase 2 . Iteration number 332
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.697323979588887
***** reduced cost= -0.4432243747216411

***** Currently in Phase 2 . Iteration number 333
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8887398138997408
***** reduced cost= -0.3613909144085291

***** Currently in Phase 2 . Iteration number 334
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.46958877966602136
***** reduced cost= -0.5509172369484214

***** Currently in Phase 2 . Iteration number 335
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8720764052680718
***** reduced cost= -0.41640939521940923

***** Currently in Phase 2 . Iteration number 336
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9710344217475733
***** reduced cost= -0.30622747937395656

***** Currently in Phase 2 . Iteration number 337
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0214993880549195
***** reduced cost= -0.3412411175428407

***** Currently in Phase 2 . Iteration number 338
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.862861645888226
***** reduced cost= -0.28218046426435894
```

```
***** Currently in Phase 2 . Iteration number 339
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1759175017167427
***** reduced cost= -0.3469879209521547

***** Currently in Phase 2 . Iteration number 340
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9329555175403094
***** reduced cost= -0.4245391933091295

***** Currently in Phase 2 . Iteration number 341
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.2875270932073455
***** reduced cost= -0.4905225296941047

***** Currently in Phase 2 . Iteration number 342
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0018972563622044
***** reduced cost= -0.30805499586010443

***** Currently in Phase 2 . Iteration number 343
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9646003985180743
***** reduced cost= -0.3300265733624733

***** Currently in Phase 2 . Iteration number 344
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.2634708699918644
***** reduced cost= -0.5053050470306097

***** Currently in Phase 2 . Iteration number 345
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.105671213171536
***** reduced cost= -0.46706752376841987

***** Currently in Phase 2 . Iteration number 346
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8885390089071409
***** reduced cost= -0.3128997457364916

***** Currently in Phase 2 . Iteration number 347
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.219954204235106
***** reduced cost= -0.7523634672046255

***** Currently in Phase 2 . Iteration number 348
***** Solving Main LP...
```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0773688641392358
***** reduced cost= -0.5425416030419066

***** Currently in Phase 2 . Iteration number 349
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8882438470137308
***** reduced cost= -0.37291884034208267

***** Currently in Phase 2 . Iteration number 350
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8065982120653563
***** reduced cost= -0.400981356178868

***** Currently in Phase 2 . Iteration number 351
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8686408579556317
***** reduced cost= -0.2866711326984981

***** Currently in Phase 2 . Iteration number 352
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9777524652319455
***** reduced cost= -0.35416939174233353

***** Currently in Phase 2 . Iteration number 353
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9274721793111043
***** reduced cost= -0.21847531254078967

***** Currently in Phase 2 . Iteration number 354
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9796763451714521
***** reduced cost= -0.33442219450190125

***** Currently in Phase 2 . Iteration number 355
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0435526322113102
***** reduced cost= -0.6405107596107833

***** Currently in Phase 2 . Iteration number 356
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9488486138043807
***** reduced cost= -0.3470775604186406

***** Currently in Phase 2 . Iteration number 357
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8319887558621174
***** reduced cost= -0.28505640010110367
```

```
***** Currently in Phase 2 . Iteration number 358
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.5386035651964265
***** reduced cost= -0.2663823937376777

***** Currently in Phase 2 . Iteration number 359
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0399511611036072
***** reduced cost= -0.2177384544636929

***** Currently in Phase 2 . Iteration number 360
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0542995798536854
***** reduced cost= -0.29366047015352303

***** Currently in Phase 2 . Iteration number 361
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0116236143299377
***** reduced cost= -0.1755463319663424

***** Currently in Phase 2 . Iteration number 362
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.6238900420868269
***** reduced cost= -0.2784485170802731

***** Currently in Phase 2 . Iteration number 363
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.6262646740455227
***** reduced cost= -0.34876892168053

***** Currently in Phase 2 . Iteration number 364
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.6145796319611032
***** reduced cost= -0.1669637002970792

***** Currently in Phase 2 . Iteration number 365
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.6064174209763349
***** reduced cost= -0.17365852134922077

***** Currently in Phase 2 . Iteration number 366
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.6224560545451711
***** reduced cost= -0.16403489818626105

***** Currently in Phase 2 . Iteration number 367
***** Solving Main LP...
```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.862611110727525
***** reduced cost= -0.13587267514307877

***** Currently in Phase 2 . Iteration number 368
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7921774538488445
***** reduced cost= -0.09389551857191292

***** Currently in Phase 2 . Iteration number 369
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.7948726228731876
***** reduced cost= -0.1469847498677357

***** Currently in Phase 2 . Iteration number 370
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8393050401431252
***** reduced cost= -0.14447240058296518

***** Currently in Phase 2 . Iteration number 371
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9722474194495598
***** reduced cost= -0.12118873407492847

***** Currently in Phase 2 . Iteration number 372
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9306438233904313
***** reduced cost= -0.15294997869168458

***** Currently in Phase 2 . Iteration number 373
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.908574494667906
***** reduced cost= -0.0986616716724944

***** Currently in Phase 2 . Iteration number 374
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.990971607013656
***** reduced cost= -0.06249894619884988

***** Currently in Phase 2 . Iteration number 375
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.014264036697978
***** reduced cost= -0.13218081134651838

***** Currently in Phase 2 . Iteration number 376
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.002441033708764
***** reduced cost= -0.1514773467370072
```

```
***** Currently in Phase 2 . Iteration number 377
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.8660320826314147
***** reduced cost= -0.1155509477046025

***** Currently in Phase 2 . Iteration number 378
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.967510867404249
***** reduced cost= -0.09152694857819133

***** Currently in Phase 2 . Iteration number 379
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0077844575952195
***** reduced cost= -0.16506201597597814

***** Currently in Phase 2 . Iteration number 380
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0681437484955427
***** reduced cost= -0.09865963030726954

***** Currently in Phase 2 . Iteration number 381
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9958306843603136
***** reduced cost= -0.11708722438771513

***** Currently in Phase 2 . Iteration number 382
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.099765310275433
***** reduced cost= -0.09897387987630824

***** Currently in Phase 2 . Iteration number 383
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0855828705362118
***** reduced cost= -0.10279275712504532

***** Currently in Phase 2 . Iteration number 384
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.035392117484975
***** reduced cost= -0.07715823008471179

***** Currently in Phase 2 . Iteration number 385
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1267838584400351
***** reduced cost= -0.11135204424088463

***** Currently in Phase 2 . Iteration number 386
***** Solving Main LP...
```

```
***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.059437510489916
***** reduced cost= -0.08580587016535635

***** Currently in Phase 2 . Iteration number 387
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -0.9984321379757355
***** reduced cost= -0.10090711501543859

***** Currently in Phase 2 . Iteration number 388
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0815847646091183
***** reduced cost= -0.06365925571362596

***** Currently in Phase 2 . Iteration number 389
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0130486293264502
***** reduced cost= -0.05672871817390002

***** Currently in Phase 2 . Iteration number 390
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0628656211448146
***** reduced cost= -0.03752897876471395

***** Currently in Phase 2 . Iteration number 391
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1342855634809568
***** reduced cost= -0.05452206752117417

***** Currently in Phase 2 . Iteration number 392
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0363617839202355
***** reduced cost= -0.05536979895826377

***** Currently in Phase 2 . Iteration number 393
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0790189396081118
***** reduced cost= -0.046765184666803794

***** Currently in Phase 2 . Iteration number 394
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1367844190498735
***** reduced cost= -0.025355887278889933

***** Currently in Phase 2 . Iteration number 395
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1255727332592862
***** reduced cost= -0.04122492270932443
```

```
***** Currently in Phase 2 . Iteration number 396
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0254247167921087
***** reduced cost= -0.03628979482735217

***** Currently in Phase 2 . Iteration number 397
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1603766618949773
***** reduced cost= -0.033811963685412394

***** Currently in Phase 2 . Iteration number 398
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1342169898532752
***** reduced cost= -0.02665567323289908

***** Currently in Phase 2 . Iteration number 399
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1415945167915311
***** reduced cost= -0.026220933131823276

***** Currently in Phase 2 . Iteration number 400
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1679474074650202
***** reduced cost= -0.01866052012211128

***** Currently in Phase 2 . Iteration number 401
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0314549738289904
***** reduced cost= -0.024247997009293254

***** Currently in Phase 2 . Iteration number 402
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0950963735929404
***** reduced cost= -0.029137251289869326

***** Currently in Phase 2 . Iteration number 403
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.08651450215326
***** reduced cost= -0.023060847803603624

***** Currently in Phase 2 . Iteration number 404
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1884190012657216
***** reduced cost= -0.028231143299900507

***** Currently in Phase 2 . Iteration number 405
***** Solving Main LP...
```

```

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1747689797614305
***** reduced cost= -0.014605991818261277

***** Currently in Phase 2 . Iteration number 406
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1529907889789042
***** reduced cost= -0.010118643740379074

***** Currently in Phase 2 . Iteration number 407
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0969751553470564
***** reduced cost= -0.012102488604773143

***** Currently in Phase 2 . Iteration number 408
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0983780556909235
***** reduced cost= -0.02104011211513379

***** Currently in Phase 2 . Iteration number 409
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0988657834779454
***** reduced cost= -0.0004530312607824438

***** Currently in Phase 2 . Iteration number 410
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0988550707033164
***** reduced cost= -0.0011817432286369556

***** Currently in Phase 2 . Iteration number 411
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.0974755554811053
***** reduced cost= -0.0005920768733964632

***** Currently in Phase 2 . Iteration number 412
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.1005686567858526
***** reduced cost= -0.0007940409383935254

***** Currently in Phase 2 . Iteration number 413
***** Solving Main LP...

***** Solving Subproblem LP...
***** Gurobi Subproblem solve status: 2
***** sigma= -1.099630761196768
***** reduced cost= 4.218847493575595e-15
***** No more improving columns for Main
***** Phase II finished
***** Phase II objective -5.611934358015313
LP iter      LP val
-----
64          -3.179640386
65          -3.183607069
66          -3.183681585

```

67	-3.185509645
68	-3.24286009
69	-3.245694743
70	-3.245964306
71	-3.246757344
72	-3.246940876
73	-3.247916478
74	-3.259547866
75	-3.259810745
76	-3.25981113
77	-3.259915031
78	-3.260472407
79	-3.55405747
80	-3.556724726
81	-3.568083607
82	-3.568187261
83	-3.568308846
84	-3.572000686
85	-3.572593336
86	-3.573353887
87	-3.574452988
88	-3.574827758
89	-3.574830518
90	-3.574834634
91	-3.575214915
92	-3.575217859
93	-3.575218948
94	-3.575389887
95	-3.575410601
96	-3.575575919
97	-3.575583647
98	-3.57558461
99	-3.590991833
100	-3.591285993
101	-3.591787396
102	-3.59209745
103	-3.592191422
104	-3.592202657
105	-3.592208454
106	-3.595291685
107	-3.595298696
108	-3.59530319
109	-3.595342283
110	-3.647978661
111	-3.651964835
112	-3.6525323
113	-3.655381338
114	-3.656517362
115	-3.656661982
116	-3.657460535
117	-3.659380904
118	-3.659529174
119	-3.6598364
120	-3.659857582
121	-3.659861321
122	-3.659862039
123	-3.66002445
124	-3.660036742
125	-3.660040375
126	-3.660072286
127	-3.660084695
128	-3.660090374
129	-3.660102069
130	-3.660107512
131	-3.660111108
132	-3.660111395
133	-3.660112336
134	-3.660112493
135	-3.660113029
136	-3.66011303
137	-3.660113042
138	-3.660113053
139	-3.660113197
140	-3.660739744
141	-3.660856784
142	-3.660861492

143	-3.660877106
144	-3.660877676
145	-3.660878086
146	-3.660879523
147	-3.660997567
148	-3.660999136
149	-3.661000189
150	-3.661000203
151	-3.661000206
152	-3.661002921
153	-3.661004352
154	-3.66100438
155	-3.661211954
156	-3.661270499
157	-3.661277123
158	-3.661283969
159	-3.661285716
160	-3.661285848
161	-3.661285885
162	-3.661293787
163	-3.661313606
164	-3.661314006
165	-3.662486748
166	-3.66256658
167	-3.662567322
168	-3.662585513
169	-3.66260164
170	-3.662602434
171	-3.662603146
172	-3.663715404
173	-3.663732759
174	-3.663738751
175	-3.664260432
176	-3.664375004
177	-3.66518909
178	-3.665240105
179	-3.66524597
180	-3.665252947
181	-3.66529345
182	-3.665349717
183	-3.666120242
184	-3.677509246
185	-3.696578242
186	-3.698351476
187	-3.698623232
188	-3.699074881
189	-3.701336131
190	-3.708826972
191	-3.914459203
192	-3.916772006
193	-3.920773626
194	-3.921778909
195	-3.921800548
196	-3.92206773
197	-3.922076274
198	-3.924855329
199	-3.995946954
200	-4.048971915
201	-4.050905088
202	-4.052384963
203	-4.052713845
204	-4.05306294
205	-4.05335395
206	-4.054215978
207	-4.054220674
208	-4.054231984
209	-4.054286983
210	-4.054296902
211	-4.0543035
212	-4.054304608
213	-4.054417013
214	-4.054475844
215	-4.057896619
216	-4.05793275
217	-4.058208444
218	-4.058214232

219 -4.058215258
220 -4.058218144
221 -4.058220741
222 -4.058222096
223 -4.05833102
224 -4.058352949
225 -4.058353855
226 -4.058620743
227 -4.058634022
228 -4.058979972
229 -4.096617825
230 -4.115563774
231 -4.216983662
232 -4.32548286
233 -4.379490891
234 -4.38100782
235 -4.381030973
236 -4.382179371
237 -4.38636008
238 -4.549728001
239 -4.554894529
240 -4.554981648
241 -4.555034025
242 -4.555034581
243 -4.555198328
244 -4.559168739
245 -4.576976859
246 -4.610068737
247 -4.670694058
248 -4.686122134
249 -4.694245643
250 -4.714292046
251 -4.781425699
252 -4.816637721
253 -4.816903654
254 -4.81690847
255 -4.81691604
256 -4.81698198
257 -4.816987981
258 -4.817221189
259 -4.817307942
260 -4.8173134
261 -4.826715564
262 -4.848685847
263 -4.8486911
264 -4.848802424
265 -4.850859905
266 -4.855474131
267 -4.883490218
268 -4.895204708
269 -4.938859966
270 -4.942895667
271 -4.949369293
272 -4.952635886
273 -4.995402368
274 -5.006888261
275 -5.0212257
276 -5.02702113
277 -5.040098633
278 -5.060324309
279 -5.128596996
280 -5.139106314
281 -5.210490859
282 -5.213063532
283 -5.243266366
284 -5.261648332
285 -5.275636775
286 -5.290210158
287 -5.29065953
288 -5.291337387
289 -5.297174829
290 -5.301428984
291 -5.303551532
292 -5.303755793
293 -5.304853817
294 -5.324754862

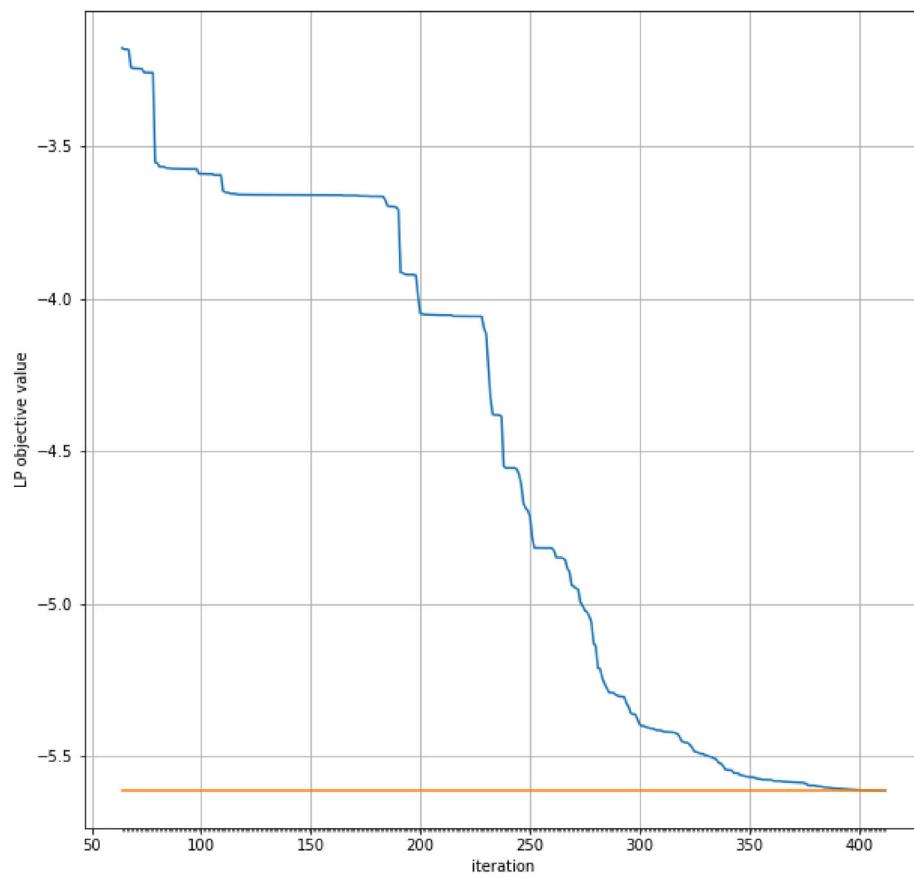
295 -5.337811349
296 -5.359363482
297 -5.361173491
298 -5.362725867
299 -5.377433564
300 -5.394435833
301 -5.399926355
302 -5.400153264
303 -5.403936667
304 -5.406016934
305 -5.408127075
306 -5.408383596
307 -5.412232187
308 -5.414027554
309 -5.414183037
310 -5.415355312
311 -5.419219011
312 -5.419274621
313 -5.419992826
314 -5.420150625
315 -5.421104897
316 -5.422345559
317 -5.425109368
318 -5.432315219
319 -5.447997518
320 -5.452316659
321 -5.454375879
322 -5.455523278
323 -5.461947439
324 -5.471527216
325 -5.483557558
326 -5.485560503
327 -5.488624144
328 -5.491152347
329 -5.491486188
330 -5.496512779
331 -5.497923986
332 -5.500801271
333 -5.502613042
334 -5.506677019
335 -5.509717334
336 -5.51992052
337 -5.52360472
338 -5.529947012
339 -5.541969164
340 -5.543834788
341 -5.544318162
342 -5.545261624
343 -5.554260741
344 -5.554451563
345 -5.555349346
346 -5.561723016
347 -5.562184923
348 -5.564719888
349 -5.565769899
350 -5.566326728
351 -5.568035292
352 -5.568269185
353 -5.571205108
354 -5.573211907
355 -5.57385168
356 -5.575908629
357 -5.57602477
358 -5.576283108
359 -5.576556364
360 -5.576874157
361 -5.580187331
362 -5.580277667
363 -5.580533002
364 -5.580887014
365 -5.581740558
366 -5.582582489
367 -5.582852991
368 -5.583461578
369 -5.583996348
370 -5.584432226

```
371      -5.584959799
372      -5.585188673
373      -5.585345532
374      -5.585903312
375      -5.587109255
376      -5.589392016
377      -5.594572861
378      -5.594790924
379      -5.59489712
380      -5.595122519
381      -5.597521883
382      -5.598158632
383      -5.598497815
384      -5.60063271
385      -5.601213795
386      -5.602054601
387      -5.602359239
388      -5.60406748
389      -5.604319114
390      -5.604732044
391      -5.605363027
392      -5.605709417
393      -5.606210877
394      -5.606792331
395      -5.607295951
396      -5.608052637
397      -5.608416853
398      -5.608583449
399      -5.609217635
400      -5.6098819
401      -5.610238131
402      -5.61096805
403      -5.611136318
404      -5.611160151
405      -5.61130637
406      -5.611587229
407      -5.611798692
408      -5.611901026
409      -5.611902114
410      -5.611908427
411      -5.611932178
412      -5.611934358
```

***** Reality check: recover the optimal x found by decomposition.

***** Its objective value is: -5.611934358

***** Compare with LP value calculated without decomposition: -5.611934358



***** Number of basic-feasible solutions generated: 229

***** Number of basic-feasible rays generated: 184

A.9 SubgradProj.ipynb

Subgradient Optimization with Python/Gurobi

Jon Lee (<https://sites.google.com/site/jonleewebpage/home>),
11/2/2020

Apply Subgradient Optimization to:

$$\begin{aligned} z := \min \quad & c'x \\ \text{subject to } & Ex \geq h ; \\ & Ax = b ; \\ & x \geq \mathbf{0} , \end{aligned} \tag{Q}$$

relaxing $Ex \geq h$ in the Lagrangian.

References:

- Jon Lee, "A First Course in Linear Optimization", Fourth Edition (Version 4.05), Reex Press, 2013-22.

MIT License

Copyright (c) 2020 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
In [1]: %reset -f
import numpy as np
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import gurobipy as gp
from gurobipy import GRB

class StopExecution(Exception):
    def __render_traceback__(self):
        pass
```

```
In [2]: MAXIT = 500
HarmonicStepSize = True          # If you choose False, then you have to guess a
'target value'
GUESS = -5.6                      # but don't guess a target value higher than
z!!!!
SmartInitialization = True        # Set 'False' to initialize with y=0.

# generate a random example
n = 100  # number of variables
m1 = 200 # number of equations to relax
m2 = 50  # number of equations to keep
np.random.seed(25)   # change the seed for a different example
E=0.01*np.random.randint(-5,high=5,size=(m1,n)).astype(float) #np.random.randn
(m1,nt)
A=0.01*np.random.randint(-2,high=3,size=(m2,n)).astype(float) #np.random.randn
(m2,nt)

# choose the right-hand sides so that Q will be feasible
xfeas=0.1*np.random.randint(0,high=5,size=n).astype(float)
h=E.dot(xfeas) - 0.1*np.random.randint(0,high=10,size=m1).astype(float)
b=A.dot(xfeas)

# choose the objective function so that the dual of Q will be feasible
yfeas=0.1*np.random.randint(0,high=5,size=m1).astype(float)
pifeas=0.1*np.random.randint(-5,high=5,size=m2).astype(float)
c=np.transpose(E)@yfeas + np.transpose(A)@pifeas + 0.1*np.random.randint(0,hi
gh=1,size=n).astype(float)
```

```
In [3]: # solve the problem as one big LP --- for comparison purposes
modelQ = gp.Model()
modelQ.reset()
x = modelQ.addMVar(n)
objective = modelQ.setObjective(c@x, GRB.MINIMIZE)
constraintsQ1 = modelQ.addConstr(E@x >= h)
constraintsQ2 = modelQ.addConstr(A@x == b)
modelQ.optimize()
if modelQ.status != GRB.Status.OPTIMAL:
    print("***** Gurobi solve status:", modelQ.status)
    print("***** This is a problem. Model Q does not have an optimal solution")
raise StopExecution
```

Set parameter Username
 Academic license - for non-commercial use only - expires 2025-06-14
 Discarded solution information
 Gurobi Optimizer version 11.0.2 build v11.0.2rc0 (win64 - Windows 10.0 (19045.2))

CPU model: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz, instruction set [SSE2|AVX2]
 Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 250 rows, 100 columns and 21957 nonzeros
 Model fingerprint: 0xd5eae979
 Coefficient statistics:

Matrix range	[1e-02, 5e-02]
Objective range	[2e-02, 5e-01]
Bounds range	[0e+00, 0e+00]
RHS range	[3e-18, 1e+00]

Presolve time: 0.01s
 Presolved: 250 rows, 100 columns, 21957 nonzeros

Iteration	Objective	Primal Inf.	Dual Inf.	Time
0	-4.2122000e+31	1.799360e+33	4.212200e+01	0s
211	-5.6119344e+00	0.000000e+00	0.000000e+00	0s

Solved in 211 iterations and 0.03 seconds (0.03 work units)
 Optimal objective -5.611934358e+00

```
In [4]: # 'SmartInitialization' chooses the initial y so that the dual of the Lagrangian Subproblem has (pi=0 as)
# a feasible solution, thus making sure that the initial Lagrangian Subproblem is not unbounded.
if SmartInitialization:
    modelY = gp.Model()
    modelY.reset()
    yvar = modelY.addMVar(m1)
    constraintsY = modelY.addConstr(np.transpose(E)@yvar <= c)
    modelY.optimize()
    y=yvar.X
else: y=np.zeros(m1)

# initialization
k=1
bestlb = -np.Inf

# set up the Lagrangian relaxation
modelL = gp.Model()
modelL.reset()
modelL.setParam('OutputFlag', 0)    # quiet the Gurobi output
x = modelL.addMVar(n)
constraintsL = modelL.addConstr(A@x == b)
objective = modelL.setObjective((c-y.dot(E))@x, GRB.MINIMIZE)

modelL.optimize()
if modelL.status != GRB.Status.OPTIMAL:
    print("***** Gurobi solve status:", modelL.status)
    print("***** This is a problem. Lagrangian Subproblem is unbounded.")
    print("***** The algorithm cannot work with this starting y.")
    raise StopExecution
v = y.dot(h) + modelL.Objval
results1=[0]
results2=[v]
bestlb = v
```

Discarded solution information

Gurobi Optimizer version 11.0.2 build v11.0.2rc0 (win64 - Windows 10.0 (19045.2))

CPU model: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz, instruction set [SSE2|AVX2]

Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 100 rows, 200 columns and 17965 nonzeros

Model fingerprint: 0x64395335

Coefficient statistics:

Matrix range [1e-02, 5e-02]

Objective range [0e+00, 0e+00]

Bounds range [0e+00, 0e+00]

RHS range [2e-02, 5e-01]

Presolve time: 0.01s

Presolved: 100 rows, 200 columns, 17965 nonzeros

Iteration	Objective	Primal Inf.	Dual Inf.	Time
0	0.0000000e+00	1.684880e+02	0.000000e+00	0s
84	0.0000000e+00	0.000000e+00	0.000000e+00	0s

Solved in 84 iterations and 0.03 seconds (0.01 work units)

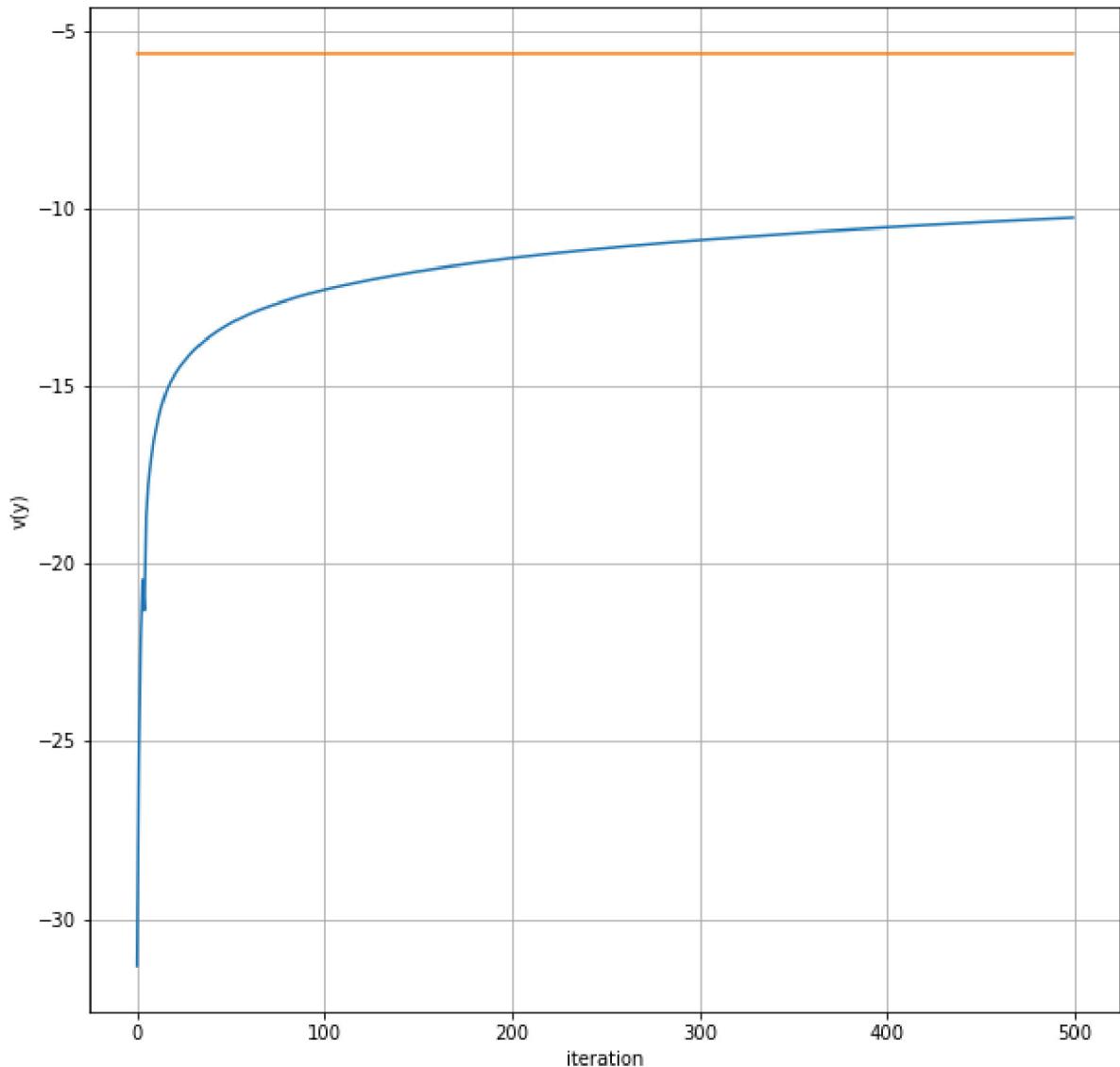
Optimal objective 0.000000000e+00

Discarded solution information

```
In [5]: while k < MAXIT:
    k += 1
    g = h - E.dot(x.X)
    if HarmonicStepSize:
        stepsize = 1/k #0.5/np.sqrt(k) # Harmonic converges in theory, but it
is slow.
    else: # Instead, you can make a GUESS at the
max
        stepsize = (GUESS - v)/(g@g) # and then use this 'Polyak' stepsiz
e
    y = np.maximum(y + stepsize*g, np.zeros(m1)) # The projection keeps y>=
0.
    objective = modelL.setObjective((c-y.dot(E))@x, GRB.MINIMIZE)
    modelL.optimize()
    if modelL.status != GRB.Status.OPTIMAL:
        k -= 1
        print("***** Gurobi solve status:", modelL.status)
        print("***** This is a problem. Lagrangian Subproblem is unbounded.")
        print("***** The algorithm cannot continue after k =",k)
        break
    v = y.dot(h) + modelL.Objval
    bestlb = np.max((bestlb,v))
    results1=np.append(results1,k-1)
    results2=np.append(results2,v)
print("***** z:", modelQ.Objval)
print("***** first lower bound:", results2[0])
print("***** best lower bound:", bestlb)

***** z: -5.611934358015314
***** first lower bound: -31.32869250311665
***** best lower bound: -10.246259680166116
```

```
In [6]: if k > 1:  
    fig, ax = plt.subplots(figsize=(10,10))  
    ax.plot(results1, results2)  
    ax.plot(results1, modelQ.Objval*np.ones(k))  
    ax.set(xlabel='iteration', ylabel='v(y)')  
    ax.grid()  
    plt.show()
```



A.10 CSP.ipynb

Cutting-Stock model: column generation with Python/Gurobi

[Jon Lee \(<https://sites.google.com/site/jonleewebpage/home>\)](https://sites.google.com/site/jonleewebpage/home),
10/20/2020

$$\begin{aligned} \min \quad & \mathbf{e}'x \\ \text{subject to} \quad & Ax - t = d ; \\ & x, t \geq \mathbf{0} , \end{aligned}$$

where the columns of A are cutting patterns, and d is the demand vector.

Notes:

- In this implementation, we never delete generated columns (i.e., patterns)
- Knapsack subproblems solved by DP or ILP (Gurobi) or both [user options]
- At the end, we solve the ILP over all columns generated, aiming to improve on the rounded-up LP solution from column-generation

References:

- Jon Lee, "A First Course in Linear Optimization", Fourth Edition (Version 4.05), Reex Press, 2013-22.

MIT License

Copyright (c) 2020 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
In [1]: %reset -f
import numpy as np
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set(); sns.set_style("whitegrid"); color_list = sns.color_palette("muted")
import gurobipy as gp
from gurobipy import GRB

class StopExecution(Exception):
    def __render_traceback__(self):
        pass
```

```
In [2]: # set at least one of the following two parameters to 'True'
# if both are set to 'True', then DP overwrites what IP calculates (but we can still compare)
IP=True # set True for solution of knapsack problem by IP (i.e., Gurobi)
DP=True # set True for solution of knapsack problem by DP
results1=[]
results2=[]
ITER=0
```

```
In [3]: # Some toy data
W=110
m=5; M=range(m)
Widths=np.array([70.0,40.0,55.0,25.0,35.0])
Demands=np.array([205,2321,143,1089,117])
```

```
In [4]: # set up the Main LP model
LP = gp.Model()
LP.setParam('OutputFlag', 0) #comment out to see more Gurobi output
minsum = LP.setObjective(0, GRB.MINIMIZE)
s=LP.addVars(m)
for i in M:
    LP.addConstr(-s[i] == Demands[i])
LP.update()
demandconstraints=LP.getConstrs()
# initialize with elementary patterns
nPAT=0
A = np.zeros((m,m))
for i in M:
    nPAT += 1
    A[i,nPAT-1] = np.floor(W/Widths[i])
    newcol=gp.Column(A[:,i],demandconstraints)
    LP.addVar(obj=1.0, column=newcol)
LP.update()
```

Set parameter Username

Academic license - for non-commercial use only - expires 2025-06-14

The Knapsack model for generating an improving column

$$\begin{aligned} \max \quad & \sum_{i=1}^m \bar{y}_i a_i \\ \text{subject to} \quad & \sum_{i=1}^m w_i a_i \leq W ; \\ & a_i \geq 0 \text{ and integer, for } i = 1, \dots, m . \end{aligned}$$

```
In [5]: # set up for solving the knapsack subproblems: either by DP or IP (or both)
#
y=np.zeros(m)
if IP==True:
    # set up the Subproblem ILP knapsack model for Gurobi
    Knap = gp.Model()
    Knap.setParam('OutputFlag', 0) #comment out to see more Gurobi output
    a = Knap.addMVar(m,vtype=GRB.INTEGER)
    knapsackobjective = Knap.setObjective(y@a, GRB.MAXIMIZE)
    knapsackconstraint = Knap.addConstr(Widths@a <= W)
if DP==True:
    # DP for knapsack. Local notation: max c'x, s.t. a'x <= b, x>=0 int.
    def Knapf(a,b,c):
        m=np.size(a)
        f=np.zeros(b+1)
        i=-np.ones(b+1,dtype=int)
        v=-np.Inf*np.ones(m)
        for s in range(min(a),b+1):
            for j in range(m):
                if a[j]<=s: v[j]=c[j] + f[s-a[j]]
                else: v[j]=-np.Inf
            f[s]=max(v)
            i[s]=np.argmax(v) # save the index j where the max occurred for that s
        #
        x=np.zeros(m)
        s=b+0
        while s>=min(a):
            x[i[s]] += 1
            s=s-a[i[s]]
        return f[b], x
```

```
In [6]: # fancy output function
def fancyoutput():
    plt.figure()
    print("***** Patterns / Widths:", Widths, "Stock roll width:", w)
    Aw=np.zeros((m,nPAT))
    for i in M:
        for j in range(nPAT):
            Aw[i,j]=A[i,j]*Widths[i]
    Aw=np.c_[ Aw, np.zeros(m) ]
    wlist=[''] * m
    for i in M:
        wlist[i]='w'+str(i)
    K=np.diagflat(Widths)
    Bw=np.c_[Aw,K]
    T = np.arange(Bw.shape[1])
    for i in range(Bw.shape[0]):
        plt.bar(T, Bw[i],
                tick_label = np.concatenate((np.arange(nPAT),np.array([' ']),wlist)),
                bottom = np.sum(Bw[:i], axis = 0),
                color = color_list[i % len(color_list)])
    plt.show()

    print("***** A:")
    print(A)
```

```
In [7]: while True:
    print(" ")
    print("***** Solving LP...")
    ITER += 1
    LP.optimize()
    if LP.status != GRB.Status.OPTIMAL:
        print("***** Gurobi solve status:", LP.status)
        print("***** This is a problem. LP does not have an optimal solution")
        raise StopExecution
    results1=np.append(results1,ITER-1)
    results2=np.append(results2,LP.Objval)
    print("***** A:")
    print(A)
    print("***** x:")
    x = LP.getVars()
    for j in range(nPAT):
        print("x[",j,"]=",round(x[j+m].X,4))
    for i in M:
        y[i]=demandconstraints[i].Pi
    print("***** y':",np.round(y,4))
    #
    if IP==True:
        knapsackobjective = Knap.setObjective(y@a, GRB.MAXIMIZE)
        print(" ")
        print("***** Solving Knapsack...")
        Knap.optimize()
        if Knap.status != GRB.Status.OPTIMAL:
            print("***** Gurobi solve status:", Knap.status)
            print("***** This is a problem. Knapsack IP does not have an optimal solution")
            raise StopExecution
        print("***** Gurobi Knap objval:",Knap.Objval)
        reducedcost = 1.0-Knap.Objval
        pattern=a.X+np.zeros(m)
    #
    if DP==True:
        results = Knapf(Widths.astype(int),W,y)
        print("***** DP Knap objval: ",results[0])
        reducedcost = 1.0-results[0]
        pattern=results[1]
    #
    if reducedcost < -0.0001:
        print("***** Column:",pattern)
        A=np.c_[ A, pattern ]
        nPAT += 1
        newcol=gp.Column(pattern,demandconstraints)
        LP.addVar(obj=1.0, column=newcol)
    else:
        print("***** No more improving columns")
        break

    print("***** Pattern generation complete. Main LP solved to optimality.")
    print("***** Total number of patterns generated: ", nPAT)
    print("***** A:")
    print(A)
    print("***** x:")
```

```
x = LP.getVars()
for j in range(nPAT):
    print("x[",j,"]=",round(x[j+m].X,4))
print("***** Optimal LP objective value:", LP.Objval)
print("***** rounds up to: ", np.ceil(LP.Objval), "(lower bound on rolls needed)")
print("***** x rounded up:")
for j in range(nPAT):
    print("x[",j,"]=",np.ceil(x[j+m].X))
print("***** Number of rolls used:", sum(np.ceil(x[j+m].X) for j in range(nPAT)))
fancyoutput()
fig, ax = plt.subplots(figsize=(10, 10))
ax.plot(results1[0:ITER], results2[0:ITER])
ax.plot(results1, np.ceil(LP.Objval)*np.ones(ITER))
ax.plot(results1, sum(np.ceil(x[j+m].X) for j in range(nPAT))*np.ones(ITER))
ax.set(xlabel='LP iteration', ylabel='LP objective value')
ax.set_xticks(ticks=results1, minor=False)
ax.grid()
plt.show()
print("LP iter", " LP val")
print("----- -----")
for j in range(ITER):
    print(int(results1[j]), " ", np.round(results2[j],4))
print(" ")
```

```
***** Solving LP...
***** A:
[[1. 0. 0. 0. 0.]
 [0. 2. 0. 0. 0.]
 [0. 0. 2. 0. 0.]
 [0. 0. 0. 4. 0.]
 [0. 0. 0. 0. 3.]]
***** x:
x[ 0 ]= 205.0
x[ 1 ]= 1160.5
x[ 2 ]= 71.5
x[ 3 ]= 272.25
x[ 4 ]= 39.0
***** y': [1.      0.5     0.5     0.25    0.3333]

***** Solving Knapsack...
***** Gurobi Knap objval: 1.5
***** DP Knap objval: 1.5
***** Column: [1. 1. 0. 0. 0.]

***** Solving LP...
***** A:
[[1. 0. 0. 0. 0. 1.]
 [0. 2. 0. 0. 0. 1.]
 [0. 0. 2. 0. 0. 0.]
 [0. 0. 0. 4. 0. 0.]
 [0. 0. 0. 0. 3. 0.]]
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 1058.0
x[ 2 ]= 71.5
x[ 3 ]= 272.25
x[ 4 ]= 39.0
x[ 5 ]= 205.0
***** y': [0.5     0.5     0.5     0.25    0.3333]

***** Solving Knapsack...
***** Gurobi Knap objval: 1.25
***** DP Knap objval: 1.25
***** Column: [0. 2. 0. 1. 0.]

***** Solving LP...
***** A:
[[1. 0. 0. 0. 0. 1. 0.]
 [0. 2. 0. 0. 0. 1. 2.]
 [0. 0. 2. 0. 0. 0. 0.]
 [0. 0. 0. 4. 0. 0. 1.]
 [0. 0. 0. 0. 3. 0. 0.]]
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 71.5
x[ 3 ]= 7.75
x[ 4 ]= 39.0
x[ 5 ]= 205.0
x[ 6 ]= 1058.0
```

```

***** y': [0.625  0.375  0.5    0.25   0.3333]

***** Solving Knapsack...
***** Gurobi Knap objval: 1.0833333333333333
***** DP Knap objval:     1.0833333333333333
***** Column: [0. 0. 0. 3. 1.]

***** Solving LP...
***** A:
[[1. 0. 0. 0. 1. 0. 0.]
 [0. 2. 0. 0. 0. 1. 2. 0.]
 [0. 0. 2. 0. 0. 0. 0. 0.]
 [0. 0. 0. 4. 0. 0. 1. 3.]
 [0. 0. 0. 0. 3. 0. 0. 1.]]
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 71.5
x[ 3 ]= 0.0
x[ 4 ]= 35.5556
x[ 5 ]= 205.0
x[ 6 ]= 1058.0
x[ 7 ]= 10.3333
***** y': [0.6111 0.3889 0.5    0.2222 0.3333]

***** Solving Knapsack...
***** Gurobi Knap objval: 1.0555555555555556
***** DP Knap objval:     1.0555555555555556
***** Column: [0. 1. 0. 0. 2.]

***** Solving LP...
***** A:
[[1. 0. 0. 0. 1. 0. 0. 0.]
 [0. 2. 0. 0. 0. 1. 2. 0. 1.]
 [0. 0. 2. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 4. 0. 0. 1. 3. 0.]
 [0. 0. 0. 0. 3. 0. 0. 1. 2.]]
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 71.5
x[ 3 ]= 0.0
x[ 4 ]= 0.0
x[ 5 ]= 205.0
x[ 6 ]= 1033.3846
x[ 7 ]= 18.5385
x[ 8 ]= 49.2308
***** y': [0.6154 0.3846 0.5    0.2308 0.3077]

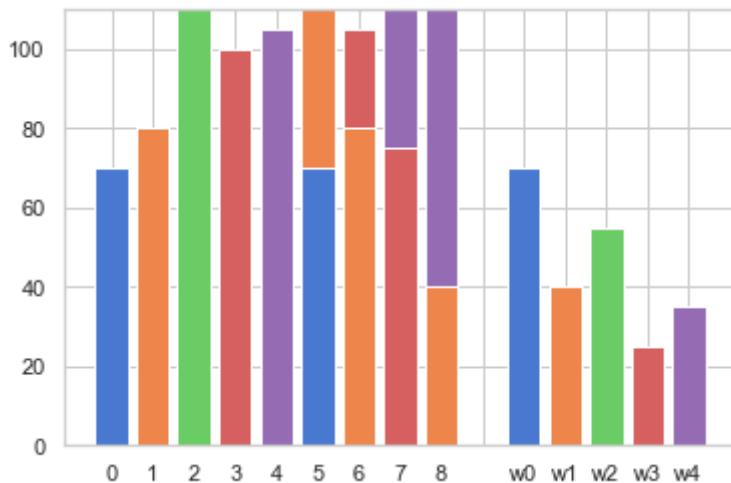
***** Solving Knapsack...
***** Gurobi Knap objval: 1.0
***** DP Knap objval:     1.0
***** No more improving columns
***** Pattern generation complete. Main LP solved to optimality.
***** Total number of patterns generated: 9
***** A:
[[1. 0. 0. 0. 0. 1. 0. 0. 0.]]

```

```

[0. 2. 0. 0. 0. 1. 2. 0. 1.]
[0. 0. 2. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 4. 0. 0. 1. 3. 0.]
[0. 0. 0. 0. 3. 0. 0. 1. 2.]]
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 71.5
x[ 3 ]= 0.0
x[ 4 ]= 0.0
x[ 5 ]= 205.0
x[ 6 ]= 1033.3846
x[ 7 ]= 18.5385
x[ 8 ]= 49.2308
***** Optimal LP objective value: 1377.6538461538462
***** rounds up to: 1378.0 (lower bound on rolls needed)
***** x rounded up:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 72.0
x[ 3 ]= 0.0
x[ 4 ]= 0.0
x[ 5 ]= 205.0
x[ 6 ]= 1034.0
x[ 7 ]= 19.0
x[ 8 ]= 50.0
***** Number of rolls used: 1380.0
***** Patterns / Widths: [70. 40. 55. 25. 35.] Stock roll width: 110

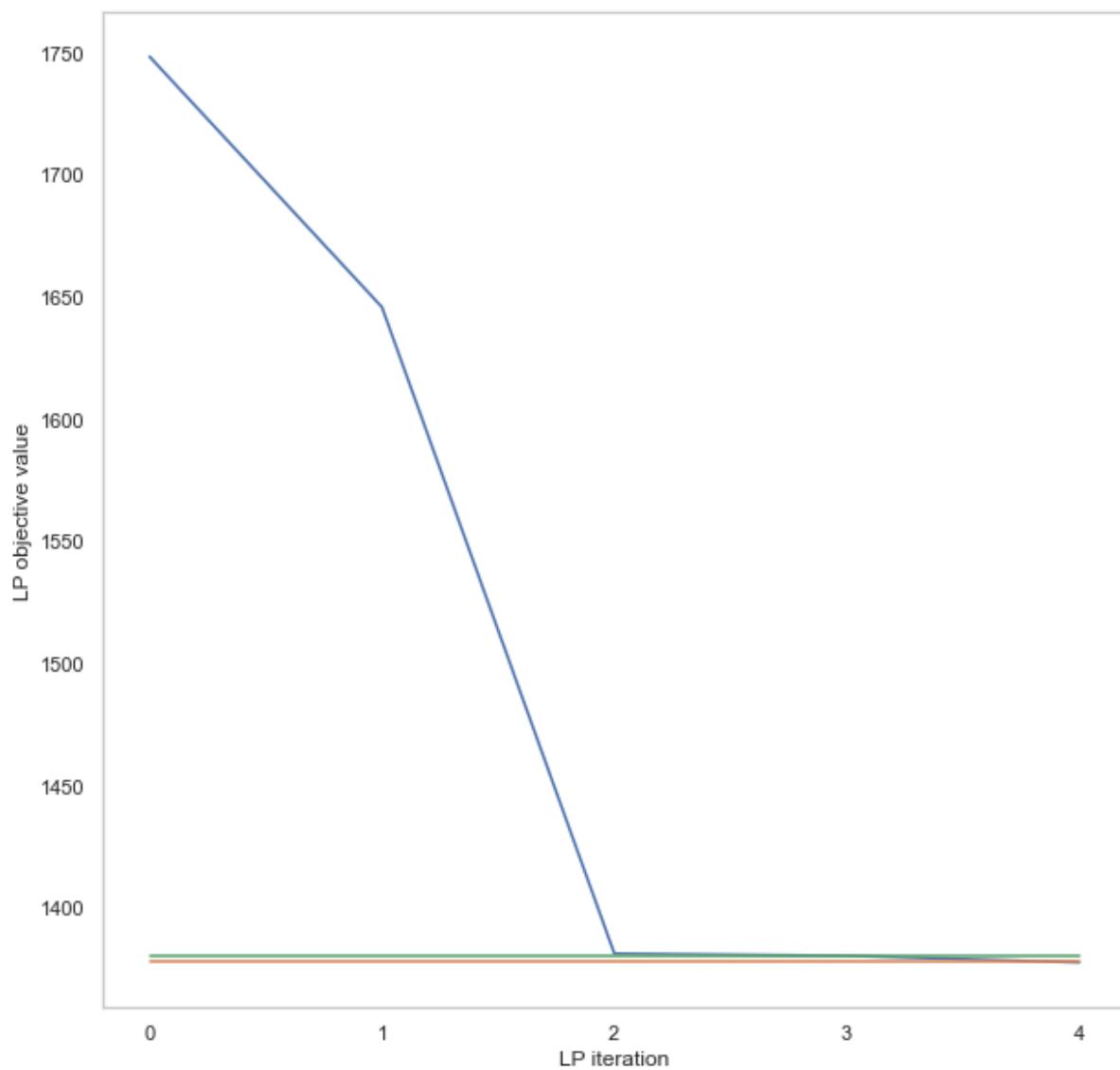
```



```

***** A:
[[1. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 2. 0. 0. 0. 1. 2. 0. 1.]
 [0. 0. 2. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 4. 0. 0. 1. 3. 0.]
 [0. 0. 0. 0. 3. 0. 0. 1. 2.]]

```



LP iter	LP val
0	1748.25
1	1645.75
2	1381.25
3	1380.3889
4	1377.6538

```
In [8]: print(" ")
print("***** Now solve the ILP over all patterns generated to try and get a better soution...")
for var in LP.getVars():
    var.vtype=GRB.INTEGER
LP.setParam('OutputFlag', 1)
LP.optimize()
if LP.status != GRB.Status.OPTIMAL:
    print("***** Gurobi solve status:", LP.status)
    print("***** This is a problem. Hit enter to continue")
    input()
print("***** x:")
for j in range(nPAT):
    print("x[",j,"]=",round(x[j+m].X+0,4))
print("***** Number of rolls used:", sum(np.ceil(x[j+m].X) for j in range(nPAT)))
fancyoutput()
```

```
***** Now solve the ILP over all patterns generated to try and get a better solution...
Set parameter OutputFlag to value 1
Gurobi Optimizer version 11.0.2 build v11.0.2rc0 (win64 - Windows 10.0 (1904
5.2))

CPU model: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz, instruction set [SSE2|AVX2]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 5 rows, 14 columns and 18 nonzeros
Model fingerprint: 0x5d331310
Variable types: 0 continuous, 14 integer (0 binary)
Coefficient statistics:
    Matrix range      [1e+00, 4e+00]
    Objective range   [1e+00, 1e+00]
    Bounds range     [0e+00, 0e+00]
    RHS range        [1e+02, 2e+03]
Found heuristic solution: objective 1750.0000000
Presolve removed 2 rows and 9 columns
Presolve time: 0.00s
Presolved: 3 rows, 5 columns, 8 nonzeros
Found heuristic solution: objective 1382.0000000
Variable types: 0 continuous, 5 integer (0 binary)

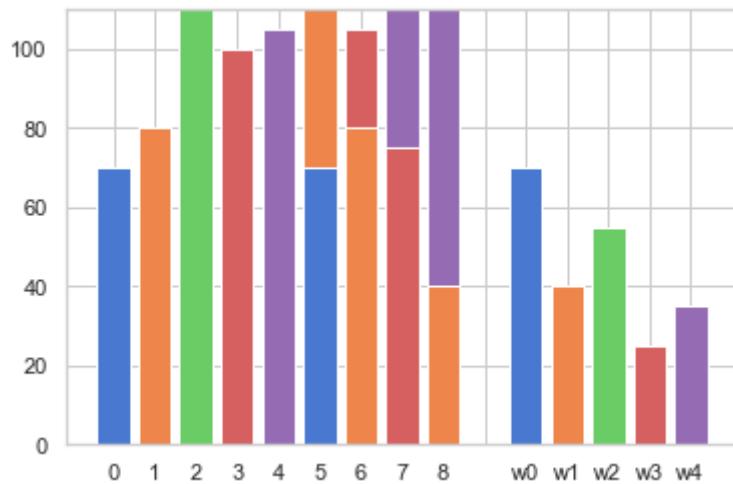
Root relaxation: objective 1.378154e+03, 2 iterations, 0.00 seconds (0.00 work units)

          Nodes       |       Current Node       |       Objective Bounds       |       Work
Expl Unexpl |   Obj  Depth IntInf |   Incumbent     BestBd   Gap | It/Node Time
          0       0 1378.15385     0       3 1382.00000 1378.15385  0.28%   -      0s
H          0       0                   1379.0000000 1378.15385  0.06%   -      0s
          0       0 1378.15385     0       3 1379.00000 1378.15385  0.06%   -      0s

Explored 1 nodes (2 simplex iterations) in 0.04 seconds (0.00 work units)
Thread count was 8 (of 8 available processors)

Solution count 3: 1379 1382 1750

Optimal solution found (tolerance 1.00e-04)
Best objective 1.379000000000e+03, best bound 1.379000000000e+03, gap 0.0000%
***** x:
x[ 0 ]= 0.0
x[ 1 ]= 0.0
x[ 2 ]= 72.0
x[ 3 ]= 1.0
x[ 4 ]= 0.0
x[ 5 ]= 205.0
x[ 6 ]= 1033.0
x[ 7 ]= 18.0
x[ 8 ]= 50.0
***** Number of rolls used: 1379.0
***** Patterns / Widths: [70. 40. 55. 25. 35.] Stock roll width: 110
```



***** A:

```
[[1. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 2. 0. 0. 0. 1. 2. 0. 1.]
 [0. 0. 2. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 4. 0. 0. 1. 3. 0.]
 [0. 0. 0. 0. 3. 0. 0. 1. 2.]]
```

A.11 UFL.ipynb

Uncapacitated-Facility-Location models with Python/Gurobi

[Jon Lee \(<https://sites.google.com/site/jonleewebpage/home>\)](https://sites.google.com/site/jonleewebpage/home),
10/20/2020

The base model that we work with is

$$\begin{aligned} \min \quad & \sum_{i \in M} f_i y_i + \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} \\ & \sum_{i \in M} x_{ij} = 1, \text{ for } j \in N; \\ & x_{ij} \geq 0, \text{ for } i \in M, j \in N; \\ & 0 \leq y_i \leq 1, \text{ and integer, for } i \in M. \end{aligned}$$

Notes:

- We make two solves, first with the weak forcing constraints

$$\sum_{j \in N} x_{ij} \leq n y_i, \text{ for } i \in M,$$

and then with the strong forcing constraints

$$x_{ij} \leq y_i, \text{ for } i \in M, j \in N.$$

- Random instances with m facilities and n customers. Play with m, n and possibly with *demand* and scale factor in f .

References:

- Jon Lee, "A First Course in Linear Optimization", Fourth Edition (Version 4.05), Reex Press, 2013-22.

MIT License

Copyright (c) 2020 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
In [1]: %reset -f
import numpy as np
#%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi, voronoi_plot_2d
import gurobipy as gp
from gurobipy import GRB

class StopExecution(Exception):
    def __render_traceback__(self):
        pass
```

```
In [2]: # parameters
m=75                      # number of facilities
n=4000                     # number of customers
M=list(range(0,m))
N=list(range(0,n))
np.random.seed(10)          # set seed to be able to repeat the same random data
solveLPsOnly=False          # set True to only solve LP relaxations

# random Locations in the unit square
fPx=np.random.rand(m)
fPy=np.random.rand(m)
cPx=np.random.rand(n)
cPy=np.random.rand(n)

# cost data
demand=10*np.random.rand(n) # these will be 'baked' into the shipping costs
f=200*np.random.rand(m)    # facility costs
c=np.zeros((m,n))
for i in range(0,m):
    for j in range(0,n):
        c[i,j]=demand[j]*np.sqrt(np.square(fPx[i]-cPx[j])+np.square(fPy[i]-cPy[j])) # = demand times per-unit transportation costs (distance)
```

```
In [3]: # set up the weak model
model = gp.Model()
model.reset()
#model.setParam('Threads', 1)                                # uncomment to ask for 1 thread
if solveLPsOnly==True:
    y=model.addVars(m,ub=1.0)
else:
    y=model.addVars(m,vtype=GRB.BINARY)
x=model.addVars(m,n)
model.setObjective(sum(f[i]*y[i] for i in M) + sum(sum(c[i,j]*x[i,j] for i in M) for j in N), GRB.MINIMIZE)
demandconstraints = model.addConstrs((sum(x[i,j] for i in M) == 1 for j in N))
weakforceconstraints = model.addConstrs((sum(x[i,j] for j in N) <= n*y[i] for i in M))
```

Set parameter Username

Academic license - for non-commercial use only - expires 2025-06-14

Discarded solution information

```
In [4]: # solve the weak model
model.optimize()
if model.status != GRB.Status.OPTIMAL:
    print("***** Gurobi solve status:", model.status)
    print("***** This is a problem. Model does not have an optimal solution")
    raise StopExecution
for i in M: print("y[",i,"]=",round(y[i].X,4))
ytot=round(sum (y[i].X for i in M))
print("y total =",ytot)
```

Gurobi Optimizer version 11.0.2 build v11.0.2rc0 (win64 - Windows 10.0 (19045.2))

CPU model: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz, instruction set [SSE2|AVX2]

Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 4075 rows, 300075 columns and 600075 nonzeros

Model fingerprint: 0xfc880efe

Variable types: 300000 continuous, 75 integer (75 binary)

Coefficient statistics:

Matrix range [1e+00, 4e+03]

Objective range [3e-04, 2e+02]

Bounds range [1e+00, 1e+00]

RHS range [1e+00, 1e+00]

Found heuristic solution: objective 17544.136375

Presolve time: 0.64s

Presolved: 4075 rows, 300075 columns, 600075 nonzeros

Variable types: 300000 continuous, 75 integer (75 binary)

Root relaxation: objective 1.229656e+03, 0 iterations, 0.49 seconds (0.32 work units)

Nodes			Current Node			Objective Bounds			Work		
	Expl	Unexpl	Obj	Depth	IntInf	Incumbent	BestBd	Gap	It/Node	Time	
H	0	0				8324.6715293	1229.65553	85.2%	-	1s	
H	0	0				4673.5132638	1229.65553	73.7%	-	2s	
	0	0	1438.63680	0	71	4673.51326	1438.63680	69.2%	-	5s	
H	0	0				4519.6169212	1547.13906	65.8%	-	11s	
H	0	0				3962.4962098	1547.13906	61.0%	-	11s	
	0	0	1547.13906	0	72	3962.49621	1547.13906	61.0%	-	11s	
	0	0	1747.92186	0	73	3962.49621	1747.92186	55.9%	-	21s	
	0	0	1795.39708	0	72	3962.49621	1795.39708	54.7%	-	26s	
	0	0	1795.39708	0	72	3962.49621	1795.39708	54.7%	-	26s	
H	0	0				3033.9534976	1920.99022	36.7%	-	45s	
	0	0	1920.99022	0	70	3033.95350	1920.99022	36.7%	-	46s	
H	0	0				2958.9643419	2075.18630	29.9%	-	71s	
H	0	0				2946.5871499	2075.18630	29.6%	-	71s	
H	0	0				2921.4140285	2075.18630	29.0%	-	71s	
H	0	0				2842.0681101	2075.18630	27.0%	-	71s	
H	0	0				2802.9809811	2075.18630	26.0%	-	71s	
H	0	0				2792.0637739	2075.18630	25.7%	-	71s	
H	0	0				2779.7296742	2075.18630	25.3%	-	71s	
H	0	0				2763.1285431	2075.18630	24.9%	-	71s	
	0	0	2075.18630	0	71	2763.12854	2075.18630	24.9%	-	71s	
H	0	0				2760.9864797	2185.57671	20.8%	-	88s	
H	0	0				2745.1240381	2185.57671	20.4%	-	88s	
H	0	0				2735.8698677	2185.57671	20.1%	-	88s	
H	0	0				2727.9013440	2185.57671	19.9%	-	88s	
H	0	0				2714.4940747	2185.57671	19.5%	-	88s	
H	0	0				2714.1367239	2185.57671	19.5%	-	88s	
H	0	0				2708.3760559	2185.57671	19.3%	-	88s	
H	0	0				2704.6679993	2185.57671	19.2%	-	88s	
	0	0	2185.57671	0	71	2704.66800	2185.57671	19.2%	-	88s	
	0	0	2186.19758	0	69	2704.66800	2186.19758	19.2%	-	89s	
	0	0	2297.59324	0	68	2704.66800	2297.59324	15.1%	-	114s	

0	0	2392.42688	0	67	2704.66800	2392.42688	11.5%	-	140s
0	0	2392.48398	0	67	2704.66800	2392.48398	11.5%	-	141s
0	0	2493.11160	0	69	2704.66800	2493.11160	7.82%	-	168s
0	0	2504.48700	0	62	2704.66800	2504.48700	7.40%	-	188s
0	0	2504.50603	0	62	2704.66800	2504.50603	7.40%	-	189s
H	0	0			2692.1722596	2504.50603	6.97%	-	222s
H	0	0			2692.1675541	2504.50603	6.97%	-	222s
H	0	0			2686.4068862	2504.50603	6.77%	-	222s
H	0	0			2682.6377814	2504.50603	6.64%	-	222s
*	0	0		0	2538.4262791	2538.42628	0.00%	-	233s

Cutting planes:

Implied bound: 11418

Flow cover: 33

Relax-and-lift: 26

Explored 1 nodes (13523 simplex iterations) in 233.94 seconds (70.99 work units)

Thread count was 8 (of 8 available processors)

Solution count 10: 2538.43 2682.64 2692.17 ... 2792.06

Optimal solution found (tolerance 1.00e-04)

Best objective 2.538426279075e+03, best bound 2.538426279075e+03, gap 0.0000%

```
y[ 0 ]= 0.0
y[ 1 ]= 0.0
y[ 2 ]= 0.0
y[ 3 ]= 0.0
y[ 4 ]= 0.0
y[ 5 ]= 0.0
y[ 6 ]= 0.0
y[ 7 ]= 0.0
y[ 8 ]= -0.0
y[ 9 ]= 1.0
y[ 10 ]= 0.0
y[ 11 ]= 0.0
y[ 12 ]= 0.0
y[ 13 ]= 1.0
y[ 14 ]= 0.0
y[ 15 ]= 0.0
y[ 16 ]= 0.0
y[ 17 ]= 0.0
y[ 18 ]= 1.0
y[ 19 ]= 1.0
y[ 20 ]= 1.0
y[ 21 ]= 1.0
y[ 22 ]= 0.0
y[ 23 ]= 1.0
y[ 24 ]= 0.0
y[ 25 ]= 0.0
y[ 26 ]= 1.0
y[ 27 ]= 0.0
y[ 28 ]= 0.0
y[ 29 ]= 0.0
y[ 30 ]= 0.0
y[ 31 ]= 0.0
y[ 32 ]= 0.0
```

```
y[ 33 ]= 1.0
y[ 34 ]= 0.0
y[ 35 ]= 0.0
y[ 36 ]= 1.0
y[ 37 ]= 0.0
y[ 38 ]= 1.0
y[ 39 ]= 1.0
y[ 40 ]= 0.0
y[ 41 ]= 0.0
y[ 42 ]= 1.0
y[ 43 ]= 0.0
y[ 44 ]= 0.0
y[ 45 ]= 0.0
y[ 46 ]= 1.0
y[ 47 ]= 0.0
y[ 48 ]= 0.0
y[ 49 ]= 0.0
y[ 50 ]= 0.0
y[ 51 ]= 0.0
y[ 52 ]= 1.0
y[ 53 ]= 0.0
y[ 54 ]= 1.0
y[ 55 ]= 0.0
y[ 56 ]= 0.0
y[ 57 ]= -0.0
y[ 58 ]= 0.0
y[ 59 ]= 0.0
y[ 60 ]= -0.0
y[ 61 ]= 1.0
y[ 62 ]= 0.0
y[ 63 ]= 0.0
y[ 64 ]= 0.0
y[ 65 ]= 0.0
y[ 66 ]= 0.0
y[ 67 ]= 1.0
y[ 68 ]= 0.0
y[ 69 ]= 1.0
y[ 70 ]= 0.0
y[ 71 ]= -0.0
y[ 72 ]= -0.0
y[ 73 ]= 0.0
y[ 74 ]= -0.0
y total = 19
```

```
In [5]: # set up and solve the strong model
model.reset()
model.remove(weakforceconstraints)
strongforceconstraints = model.addConstrs((x[i,j] <= y[i]  for i in M for j in N))
model.optimize()
if model.status != GRB.Status.OPTIMAL:
    print("***** Gurobi solve status:", model.status)
    print("***** This is a problem. Model does not have an optimal solution")
    raise StopExecution
for i in M: print("y[",i,"]=",round(y[i].X,4))
print("y total =", round(sum (y[i].X for i in M),4))
```

Discarded solution information
Gurobi Optimizer version 11.0.2 build v11.0.2rc0 (win64 - Windows 10.0 (1904 5.2))

CPU model: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz, instruction set [SSE2|AVX2]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 304000 rows, 300075 columns and 900000 nonzeros
Model fingerprint: 0xdcd5646b
Variable types: 300000 continuous, 75 integer (75 binary)
Coefficient statistics:

Matrix range	[1e+00, 1e+00]
Objective range	[3e-04, 2e+02]
Bounds range	[1e+00, 1e+00]
RHS range	[1e+00, 1e+00]

Found heuristic solution: objective 17544.136375
Presolve time: 0.97s
Presolved: 304000 rows, 300075 columns, 900000 nonzeros
Variable types: 300000 continuous, 75 integer (75 binary)
Deterministic concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only...

Root barrier log...

Ordering time: 0.17s

Barrier statistics:

Dense cols :	75
AA' NZ :	6.000e+05
Factor NZ :	1.253e+06 (roughly 260 MB of memory)
Factor Ops :	3.365e+07 (less than 1 second per iteration)
Threads :	2

Iter	Objective		Residual			Time
	Primal	Dual	Primal	Dual	Compl	
0	8.86040033e+04	7.66864953e+03	7.50e+00	3.04e+00	1.75e+00	2s
1	1.03781491e+04	-6.11496955e+04	4.96e-02	6.49e-01	1.09e-01	3s

Barrier performed 1 iterations in 2.52 seconds (1.55 work units)
Barrier solve interrupted - model solved by another algorithm

Concurrent spin time: 0.43s (can be avoided by choosing Method=3)

Solved with dual simplex

Root relaxation: objective 2.538426e+03, 11556 iterations, 1.33 seconds (0.55 work units)

Nodes	Current Node			Objective Bounds			Work		
Expl	Unexpl	Obj	Depth	IntInf	Incumbent	BestBd	Gap	It/Node	Time
*	0	0		0	2538.4262791	2538.42628	0.00%	-	2s

Explored 1 nodes (11556 simplex iterations) in 2.66 seconds (1.60 work units)
Thread count was 8 (of 8 available processors)

Solution count: 2538.43 17544.1

Optimal solution found (tolerance 1.00e-04)
Best objective 2.538426279075e+03, best bound 2.538426279075e+03, gap 0.0000%
y[0]= -0.0
y[1]= -0.0
y[2]= -0.0
y[3]= -0.0
y[4]= -0.0
y[5]= -0.0
y[6]= -0.0
y[7]= -0.0
y[8]= -0.0
y[9]= 1.0
y[10]= -0.0
y[11]= -0.0
y[12]= -0.0
y[13]= 1.0
y[14]= -0.0
y[15]= -0.0
y[16]= -0.0
y[17]= -0.0
y[18]= 1.0
y[19]= 1.0
y[20]= 1.0
y[21]= 1.0
y[22]= -0.0
y[23]= 1.0
y[24]= -0.0
y[25]= -0.0
y[26]= 1.0
y[27]= -0.0
y[28]= -0.0
y[29]= -0.0
y[30]= -0.0
y[31]= -0.0
y[32]= -0.0
y[33]= 1.0
y[34]= -0.0
y[35]= -0.0
y[36]= 1.0
y[37]= -0.0
y[38]= 1.0
y[39]= 1.0
y[40]= -0.0
y[41]= -0.0
y[42]= 1.0
y[43]= -0.0
y[44]= -0.0
y[45]= -0.0
y[46]= 1.0
y[47]= -0.0
y[48]= -0.0
y[49]= -0.0
y[50]= -0.0
y[51]= -0.0

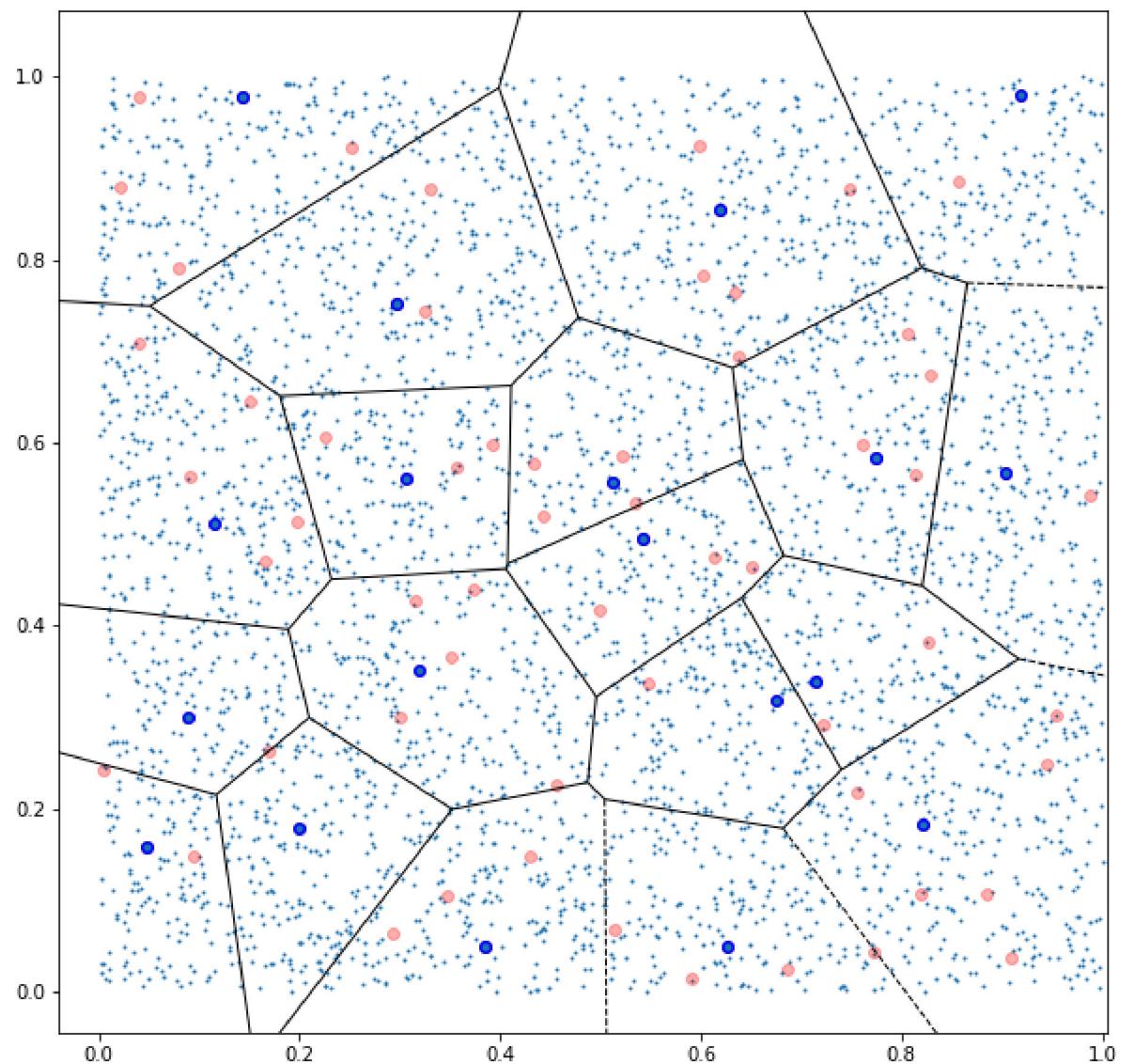
```
y[ 52 ]= 1.0
y[ 53 ]= -0.0
y[ 54 ]= 1.0
y[ 55 ]= -0.0
y[ 56 ]= -0.0
y[ 57 ]= -0.0
y[ 58 ]= -0.0
y[ 59 ]= -0.0
y[ 60 ]= -0.0
y[ 61 ]= 1.0
y[ 62 ]= -0.0
y[ 63 ]= -0.0
y[ 64 ]= -0.0
y[ 65 ]= -0.0
y[ 66 ]= -0.0
y[ 67 ]= 1.0
y[ 68 ]= -0.0
y[ 69 ]= 1.0
y[ 70 ]= -0.0
y[ 71 ]= -0.0
y[ 72 ]= -0.0
y[ 73 ]= -0.0
y[ 74 ]= -0.0
y total = 19.0
```

```
In [6]: # plot the results
#
if solveLPsOnly == False:
    fxopen=np.zeros(ytot)
    fyopen=np.zeros(ytot)
    count=-1
    for i in M:
        if round(y[i].X)==1:
            count += 1
            fxopen[count]=fPx[i]
            fyopen[count]=fPy[i]

    # Get current figure size
    fig_size = plt.rcParams["figure.figsize"]
    #print("Current size:", fig_size)
    fig_size[0] = 10
    fig_size[1] = 10
    plt.rcParams["figure.figsize"] = fig_size

    # vornoi diagram for the open facilities
    points=np.column_stack((fxopen,fyopen))
    vor = Voronoi(points)
    fig = voronoi_plot_2d(vor,show_vertices=False)

    # open facilities are blue, closed failities are opaque red,
    # vornoi cells capture the customers assigned to each open facility
    plt.scatter(cPx,cPy,s=1)
    plt.scatter(fPx,fPy,c='red',alpha=0.3)
    plt.scatter(fxopen,fyopen,c='blue')
```



A.12 pure_gomory_example_1.ipynb

Example 1: Gomory cutting-planes for dual-form pure-integer problem $D_{\mathcal{I}}$

[Jon Lee \(https://sites.google.com/site/jonleewebpage/home\)](https://sites.google.com/site/jonleewebpage/home),
11/21/2020

For dual-form pure-integer problem

$$\begin{aligned} \max \quad & y' b \\ \text{s.t.} \quad & y' A \leq c' ; \\ & y \in \mathbb{Z}^m . \end{aligned} \tag{D_{\mathcal{I}}}$$

Notes:

- A and c **MUST** be integer

Reference:

- Qi He, Jon Lee. Another pedagogy for pure-integer Gomory. *RAIRO – Operations Research*, 51:189–197, 2017.

MIT License

Copyright (c) 2020 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

In [1]: %reset -f

In [2]: %run ./pivot_tools.ipynb

```
pivot_tools loaded: pivot_perturb, pivot_algebra, N, pivot_ratios, pivot_swaps, pivot_plot, pure_gomory, mixed_gomory, dual_plot
```

```
In [3]: A = sym.Matrix(([7, 8, -1, 1, 3],
                  [5, 6, -1, 2, 1]))
m = A.shape[0]
n = A.shape[1]
c = sym.Matrix([126, 141, -10, 5, 67])
b = sym.Matrix([26, 19])
beta = [0,1]
eta = list(set(list(range(n)))-set(beta))
A_beta = copy.copy(A[:,beta])
A_eta = copy.copy(A[:,eta])
c_beta = copy.copy(c[beta,0])
c_eta = copy.copy(c[eta,0])
Perturb=False ##### do NOT change this!!!!!!!!!!!!!!!!!!!! You
can perturb later
```

In [4]: A

Out[4]:
$$\begin{bmatrix} 7 & 8 & -1 & 1 & 3 \\ 5 & 6 & -1 & 2 & 1 \end{bmatrix}$$

In [5]: c

Out[5]:
$$\begin{bmatrix} 126 \\ 141 \\ -10 \\ 5 \\ 67 \end{bmatrix}$$

In [6]: #pivot_perturb()

In [7]: b

Out[7]:
$$\begin{bmatrix} 26 \\ 19 \end{bmatrix}$$

In [8]: pivot_algebra()

pivot_algebra() done

In [9]: xbar_beta

Out[9]:
$$\begin{bmatrix} 2 \\ \frac{3}{2} \end{bmatrix}$$

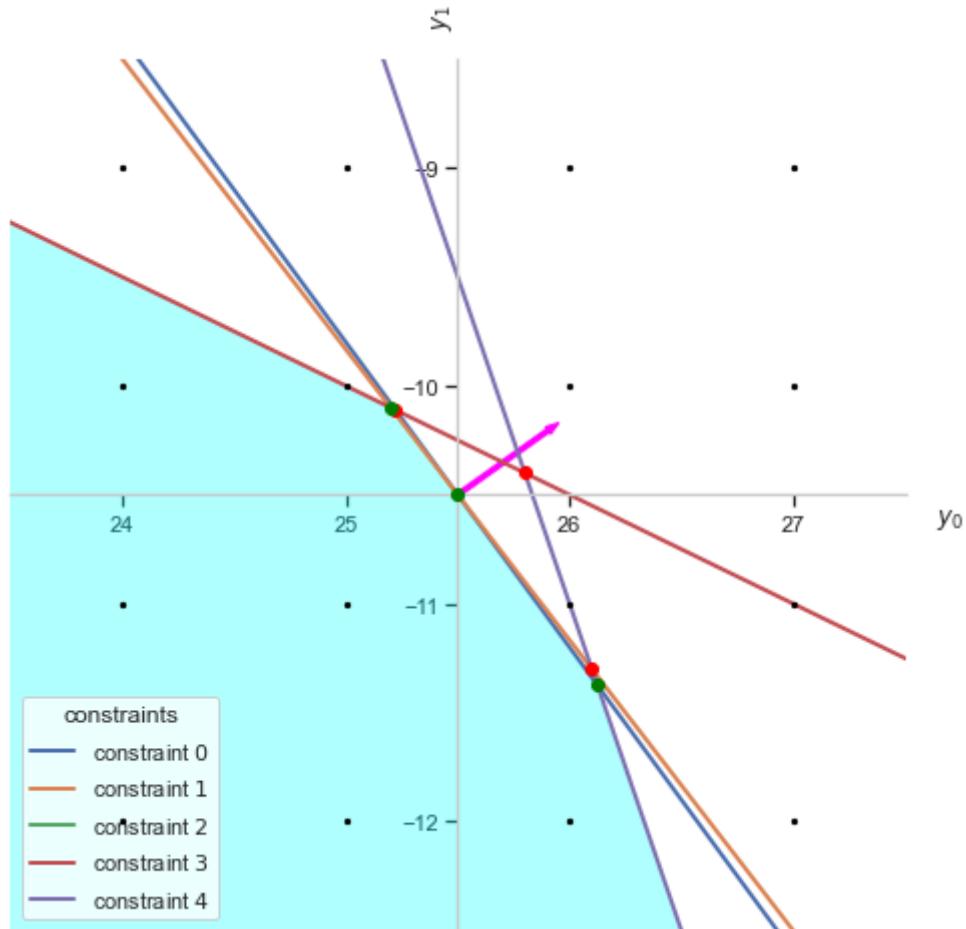
In [10]: cbar_eta

Out[10]:
$$\begin{bmatrix} 5 \\ \frac{1}{2} \\ 1 \end{bmatrix}$$

In [11]: `ybar`

Out[11]: $\begin{bmatrix} \frac{51}{2} \\ -\frac{21}{2} \end{bmatrix}$

In [12]: `dual_plot()`



In [13]: `pure_gomory(1)`

*** PROBABLY WANT TO APPLY `pivot_algebra()`! ***

In [14]: `pivot_algebra()`

`pivot_algebra()` done

In [15]: `A`

Out[15]: $\begin{bmatrix} 7 & 8 & -1 & 1 & 3 & 4 \\ 5 & 6 & -1 & 2 & 1 & 3 \end{bmatrix}$

In [16]: c

Out[16]:

$$\begin{bmatrix} 126 \\ 141 \\ -10 \\ 5 \\ 67 \\ 70 \end{bmatrix}$$

In [17]: eta

Out[17]: [2, 3, 4, 5]

In [18]: cbar_eta

Out[18]:

$$\begin{bmatrix} 5 \\ \frac{1}{2} \\ 1 \\ -\frac{1}{2} \end{bmatrix}$$

In [19]: pivot_ratios(3)

$$\begin{bmatrix} \infty \\ 3 \end{bmatrix}$$
 $\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} 2 \\ \frac{3}{2} - \frac{\lambda}{2} \\ 0 \\ 0 \\ 0 \\ \lambda \end{bmatrix}$$

In [20]: pivot_swap(3,1)

swap accepted --- new partition:

```
eta: [2, 3, 4, 1]
beta: [0, 5]
*** MUST APPLY pivot_algebra()! ***
```

In [21]: pivot_algebra()

pivot_algebra() done

In [22]: `xbar_beta`

Out[22]: $\begin{bmatrix} 2 \\ 3 \end{bmatrix}$

In [23]: `cbar_eta`

Out[23]: $\begin{bmatrix} 4 \\ 5 \\ -3 \\ 1 \end{bmatrix}$

In [24]: `pivot_ratios(2)`

$$\begin{bmatrix} \frac{2}{5} \\ \infty \end{bmatrix}$$

$\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} 2 - 5\lambda \\ 0 \\ 0 \\ 0 \\ \lambda \\ 8\lambda + 3 \end{bmatrix}$$

In [25]: `pivot_swap(2,0)`

swap accepted --- new partition:

```
eta: [2, 3, 0, 1]
beta: [4, 5]
*** MUST APPLY pivot_algebra()! ***
```

In [26]: `pivot_algebra()`

`pivot_algebra()` done

In [27]: `xbar_beta`

Out[27]: $\begin{bmatrix} \frac{2}{5} \\ \frac{31}{5} \end{bmatrix}$

In [28]: `cbar_eta`

Out[28]:

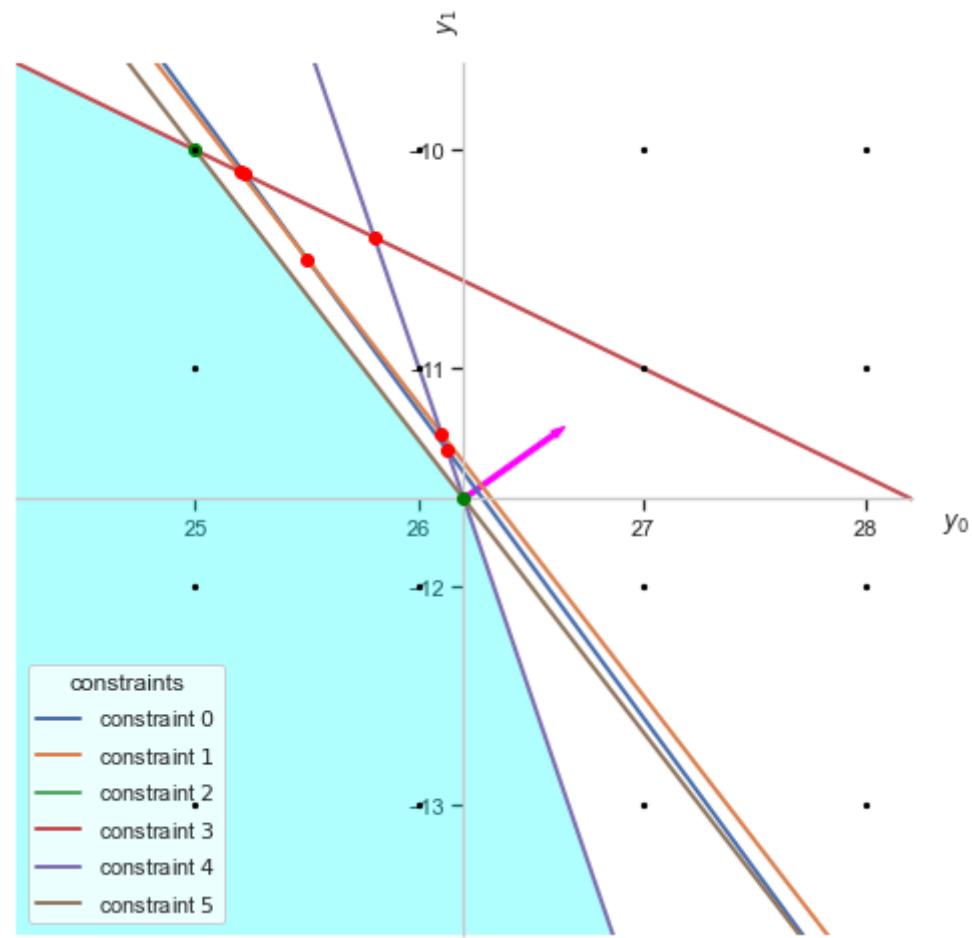
$$\begin{bmatrix} \frac{23}{5} \\ 2 \\ \frac{3}{5} \\ 1 \end{bmatrix}$$

In [29]: `ybar`

Out[29]:

$$\begin{bmatrix} \frac{131}{5} \\ -\frac{58}{5} \end{bmatrix}$$

In [30]: `dual_plot()`



In [31]: `pure_gomory(0)`

*** PROBABLY WANT TO APPLY `pivot_algebra()`! ***

In [32]: `pure_gomory(1)`

*** PROBABLY WANT TO APPLY `pivot_algebra()`! ***

In [33]: `pivot_algebra()`

`pivot_algebra()` done

In [34]: `cbar_eta`

Out[34]:

$$\begin{bmatrix} \frac{23}{5} \\ 2 \\ \frac{3}{5} \\ 1 \\ -\frac{1}{5} \\ -\frac{2}{5} \end{bmatrix}$$

In [35]: `pivot_ratios(5)`

$$\begin{bmatrix} 2 \\ \frac{31}{3} \end{bmatrix}$$

$\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{2}{5} - \frac{\lambda}{5} \\ \frac{31}{5} - \frac{3\lambda}{5} \\ 0 \\ \lambda \end{bmatrix}$$

In [36]: `pivot_swap(5,0)`

swap accepted --- new partition:

```
eta: [2, 3, 0, 1, 6, 4]
beta: [7, 5]
*** MUST APPLY pivot_algebra()! ***
```

In [37]: `pivot_algebra()`

`pivot_algebra()` done

In [38]: `xbar_beta`

Out[38]:

$$\begin{bmatrix} 2 \\ 5 \end{bmatrix}$$

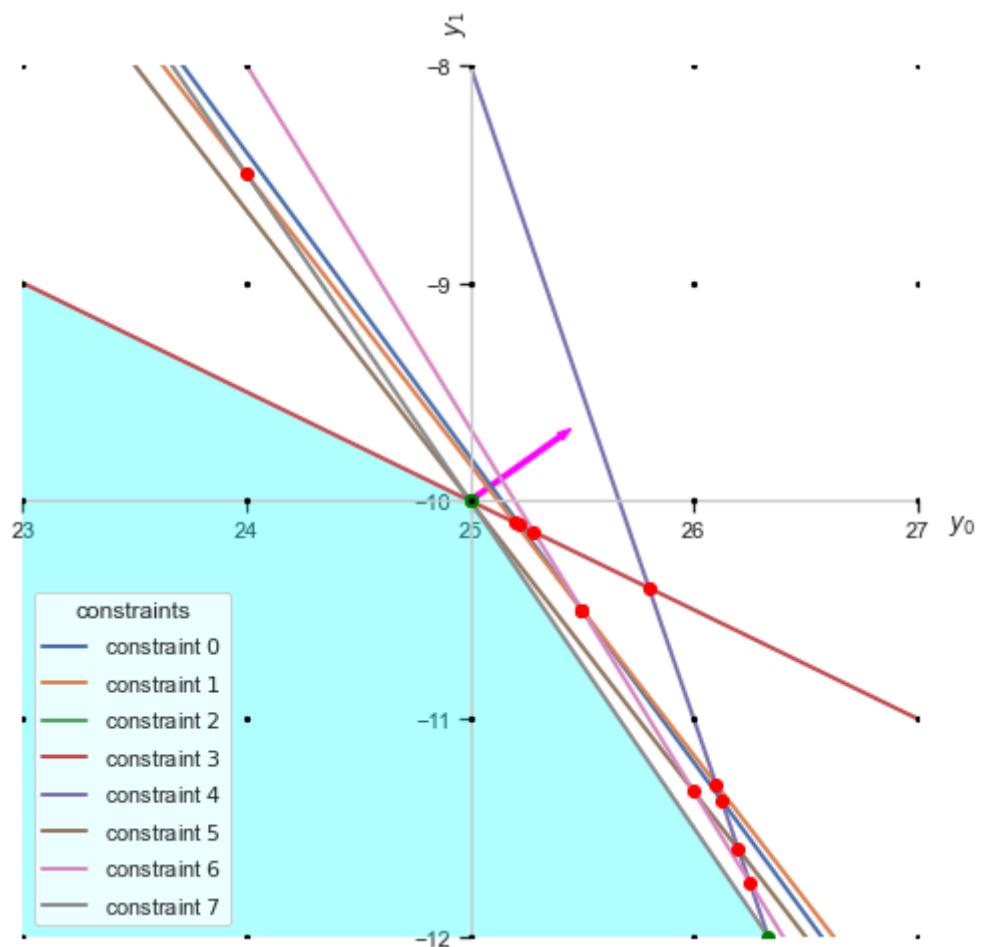
```
In [39]: cbar_eta
```

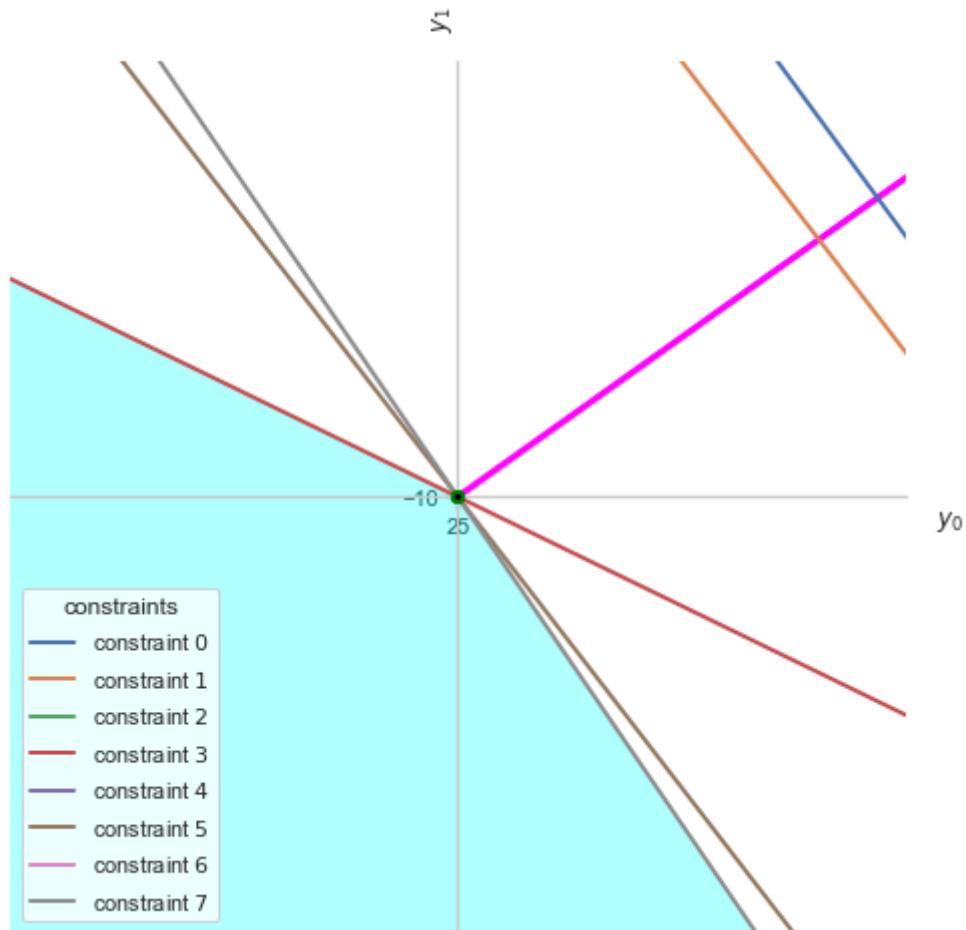
```
Out[39]:  $\begin{bmatrix} 5 \\ 0 \\ 1 \\ 1 \\ 1 \\ 2 \end{bmatrix}$ 
```

```
In [40]: ybar
```

```
Out[40]:  $\begin{bmatrix} 25 \\ -10 \end{bmatrix}$ 
```

```
In [41]: dual_plot()
```



In [42]: `dual_plot(.1)`

A.13 pure_gomory_example_2.ipynb

Example 2: Gomory cutting-planes for dual-form pure-integer problem $D_{\mathcal{I}}$

[Jon Lee \(https://sites.google.com/site/jonleewebpage/home\)](https://sites.google.com/site/jonleewebpage/home),
11/21/2020

For dual-form pure-integer problem

$$\begin{aligned} \max \quad & y' b \\ \text{s.t.} \quad & y' A \leq c' ; \\ & y \in \mathbb{Z}^m . \end{aligned} \tag{D_{\mathcal{I}}}$$

Notes:

- A and c **MUST** be integer

Reference:

- Qi He, Jon Lee. Another pedagogy for pure-integer Gomory. *RAIRO – Operations Research*, 51:189–197, 2017.

MIT License

Copyright (c) 2020 Jon Lee

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

In [1]: %reset -f

In [2]: %run ./pivot_tools.ipynb

```
pivot_tools loaded: pivot_perturb, pivot_algebra, N, pivot_ratios, pivot_swaps, pivot_plot, pure_gomory, mixed_gomory, dual_plot
```

```
In [3]: k=3
A = sym.Matrix(([2*k, -2*k, 0],
               [1, 1, -1]))
m = A.shape[0]
n = A.shape[1]
c = sym.Matrix([2*k, 0, 1])
b = sym.Matrix([0,1])
beta = [0,1]
eta = list(set(list(range(n)))-set(beta))
A_beta = copy.copy(A[:,beta])
A_eta = copy.copy(A[:,eta])
c_beta = copy.copy(c[beta,0])
c_eta = copy.copy(c[eta,0])
Perturb=False ##### do NOT change this!!!!!!!!!!!!!!!!!!!!!! You
can perturb Later
```

In [4]: A

Out[4]:
$$\begin{bmatrix} 6 & -6 & 0 \\ 1 & 1 & -1 \end{bmatrix}$$

In [5]: c

Out[5]:
$$\begin{bmatrix} 6 \\ 0 \\ 1 \end{bmatrix}$$

In [6]: #pivot_perturb()

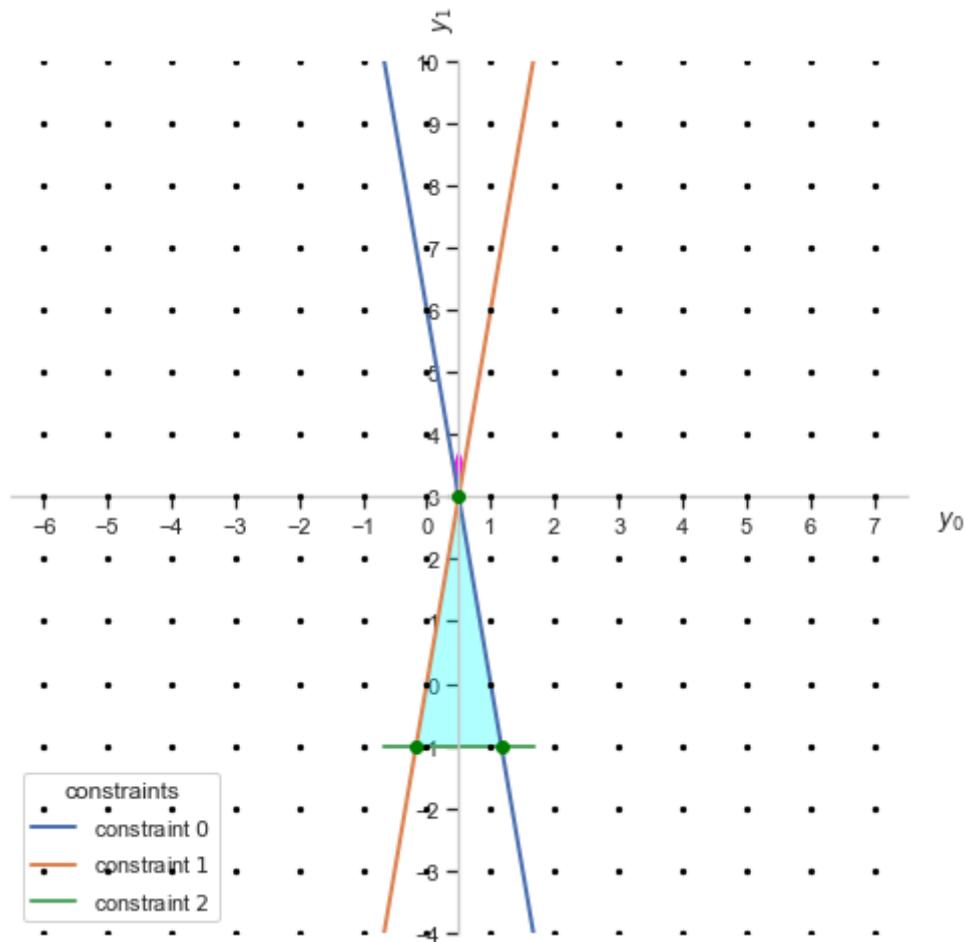
In [7]: b

Out[7]:
$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

In [8]: pivot_algebra()

pivot_algebra() done

In [9]: `dual_plot(2*k+1)`



In [10]: `ybar`

Out[10]: $\begin{bmatrix} \frac{1}{2} \\ 2 \\ 3 \end{bmatrix}$

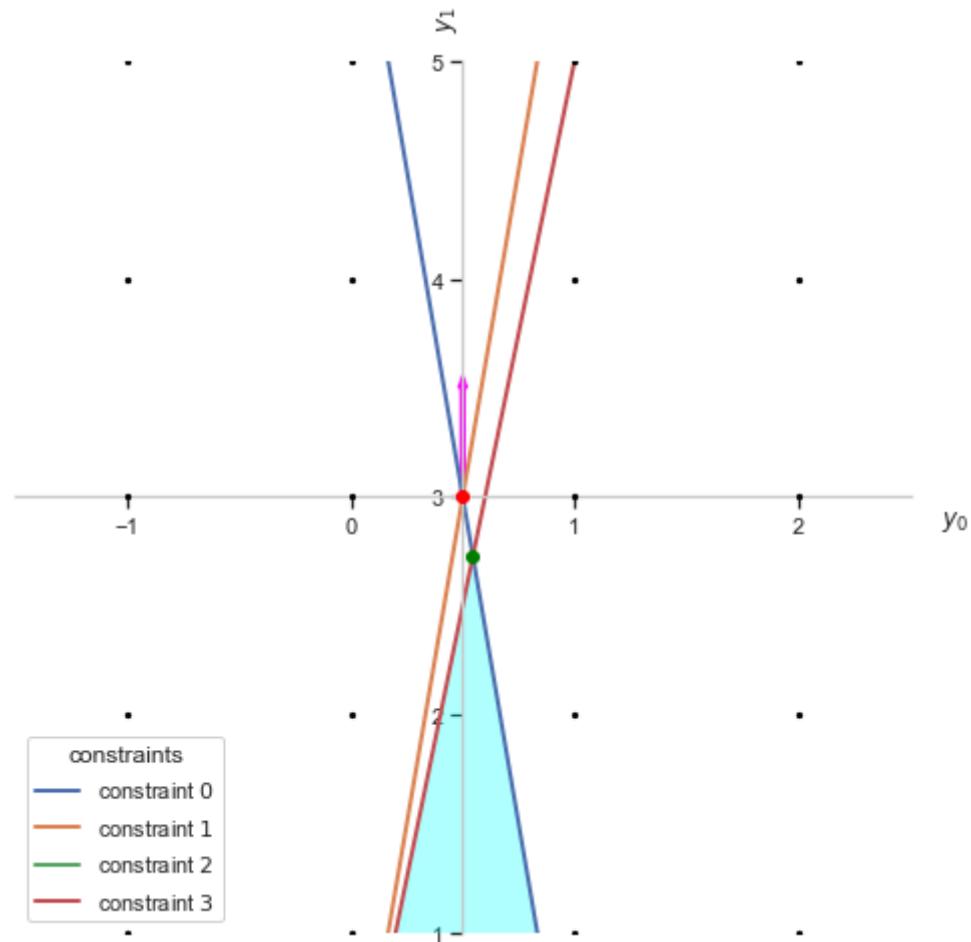
In [11]: `pure_gomory(0)`

*** PROBABLY WANT TO APPLY `pivot_algebra()`! ***

In [12]: `pivot_algebra()`

`pivot_algebra()` done

In [13]: `dual_plot()`



In [14]: `cbar_eta`

Out[14]: $\begin{bmatrix} 4 \\ -\frac{1}{2} \end{bmatrix}$

In [15]: `pivot_ratios(1)`

$$\begin{bmatrix} 6 \\ \frac{6}{11} \end{bmatrix}$$

$\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} \frac{1}{2} - \frac{\lambda}{12} \\ \frac{1}{2} - \frac{11\lambda}{12} \\ 0 \\ \lambda \end{bmatrix}$$

In [16]: `pivot_swap(1,1)`

swap accepted --- new partition:

```
eta: [2, 1]
beta: [0, 3]
*** MUST APPLY pivot_algebra()! ***
```

In [17]: `pivot_algebra()`

`pivot_algebra()` done

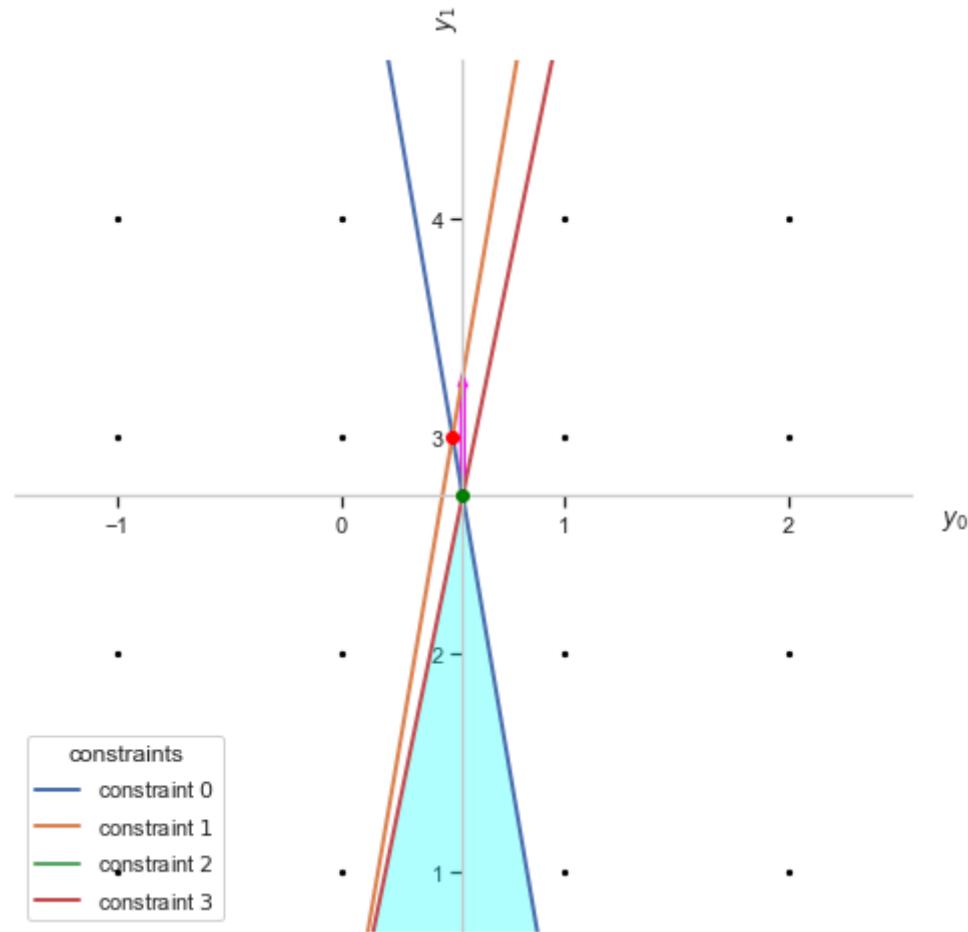
In [18]: `cbar_eta`

$$\begin{bmatrix} \frac{41}{11} \\ \frac{11}{11} \\ \frac{6}{11} \end{bmatrix}$$

In [19]: `xbar_beta`

$$\begin{bmatrix} \frac{5}{11} \\ \frac{11}{11} \\ \frac{6}{11} \end{bmatrix}$$

In [20]: `dual_plot()`



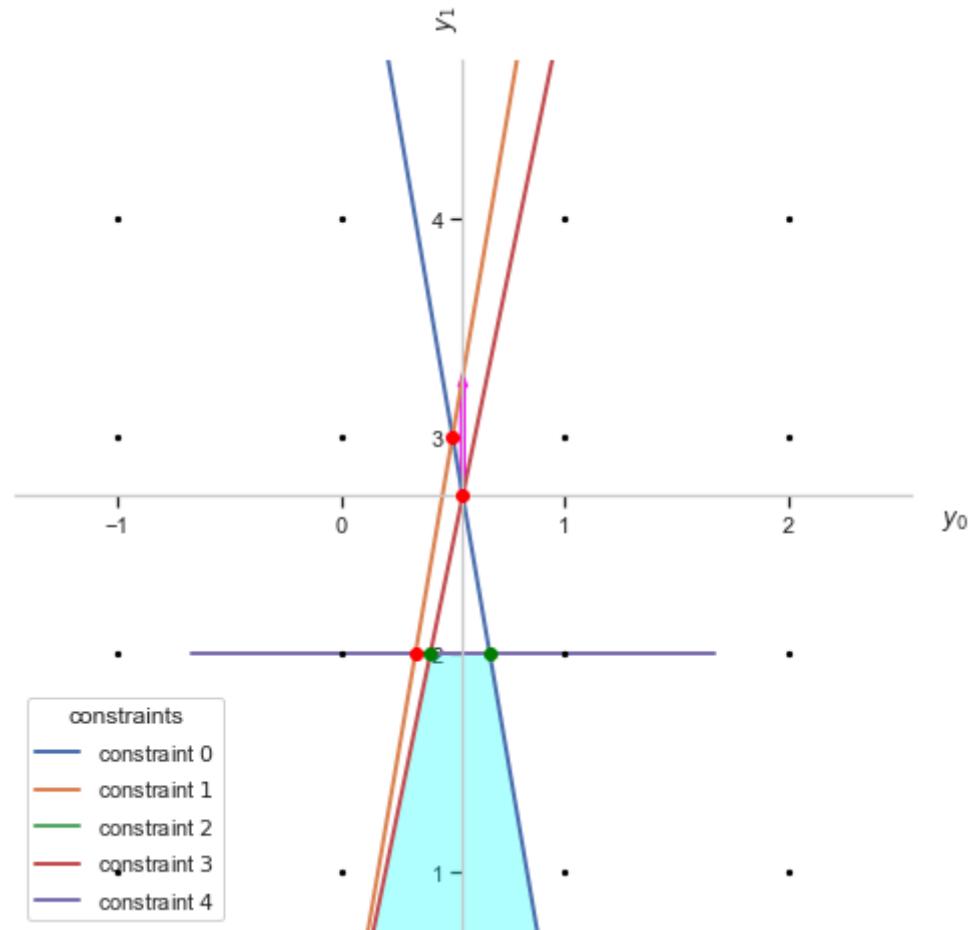
In [21]: `ybar`

Out[21]: $\begin{bmatrix} \frac{6}{11} \\ \frac{30}{11} \end{bmatrix}$

In [22]: `pure_gomory(1)`

*** PROBABLY WANT TO APPLY `pivot_algebra()`! ***

In [23]: `dual_plot()`



In [24]: `pivot_algebra()`

`pivot_algebra()` done

In [25]: `cbar_eta`

Out[25]: $\begin{bmatrix} \frac{41}{11} \\ \frac{6}{11} \\ -\frac{8}{11} \end{bmatrix}$

In [26]: `pivot_ratios(2)`

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

$\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} \frac{5}{11} - \frac{5\lambda}{11} \\ 0 \\ 0 \\ \frac{6}{11} - \frac{6\lambda}{11} \\ \lambda \end{bmatrix}$$



In [27]: `pivot_swap(2,0)`

swap accepted --- new partition:

```
eta: [2, 1, 0]
beta: [4, 3]
*** MUST APPLY pivot_algebra()! ***
```

In [28]: `pivot_algebra()`

`pivot_algebra()` done

In [29]: `cbar_eta`

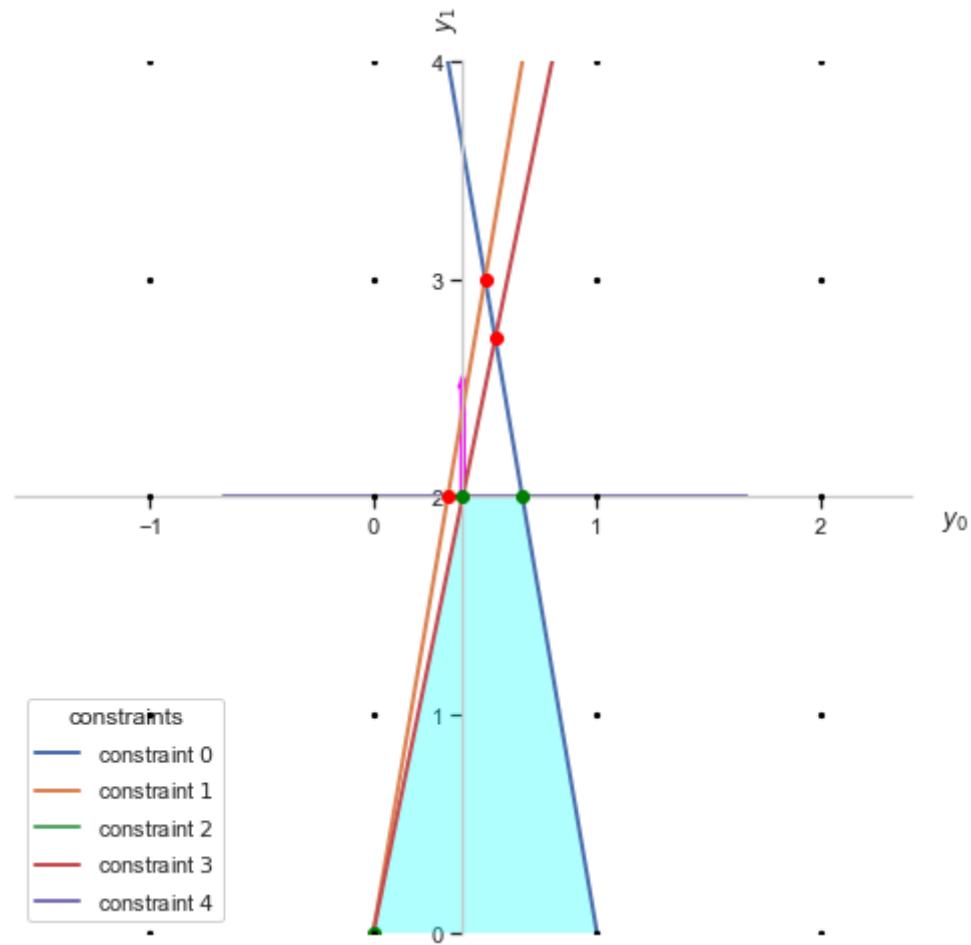
$$\begin{bmatrix} 3 \\ \frac{2}{5} \\ \frac{8}{5} \end{bmatrix}$$



In [30]: `xbar_beta`

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

In [31]: `dual_plot()`



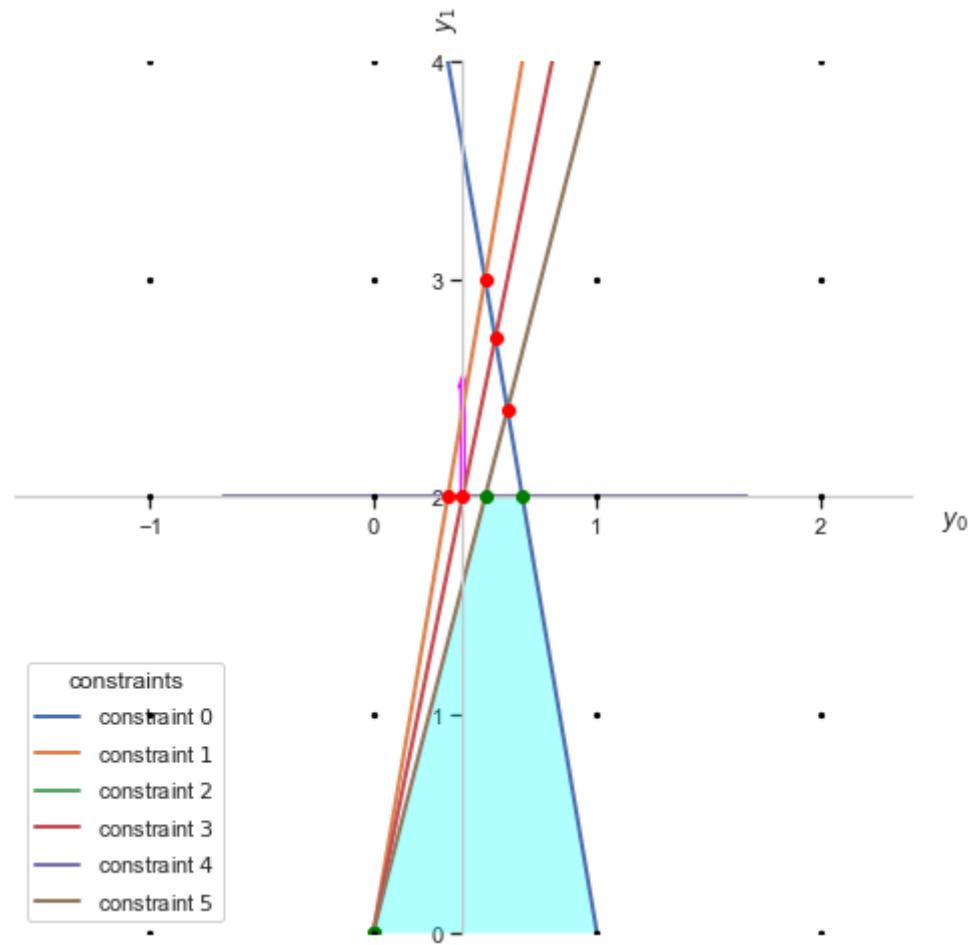
In [32]: `ybar`

Out[32]: $\begin{bmatrix} \frac{2}{5} \\ 2 \end{bmatrix}$

In [33]: `pure_gomory(0)`

*** PROBABLY WANT TO APPLY `pivot_algebra()`! ***

In [34]: `dual_plot()`



In [35]: `pivot_algebra()`

`pivot_algebra()` done

In [36]: `cbar_eta`

Out[36]:

$$\begin{bmatrix} 3 \\ \frac{2}{5} \\ \frac{8}{5} \\ -\frac{2}{5} \end{bmatrix}$$

In [37]: `pivot_ratios(3)`

$$\begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

$\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ -\frac{4\lambda}{5} \\ 1 - \frac{\lambda}{5} \\ \lambda \end{bmatrix}$$



In [38]: `pivot_swap(3,1)`

swap accepted --- new partition:

```
eta: [2, 1, 0, 3]
beta: [4, 5]
*** MUST APPLY pivot_algebra()! ***
```

In [39]: `pivot_algebra()`

`pivot_algebra()` done

In [40]: `cbar_eta`

Out[40]:

$$\begin{bmatrix} 3 \\ 1 \\ 1 \\ \frac{1}{2} \end{bmatrix}$$



In [41]: `xbar_beta`

Out[41]:

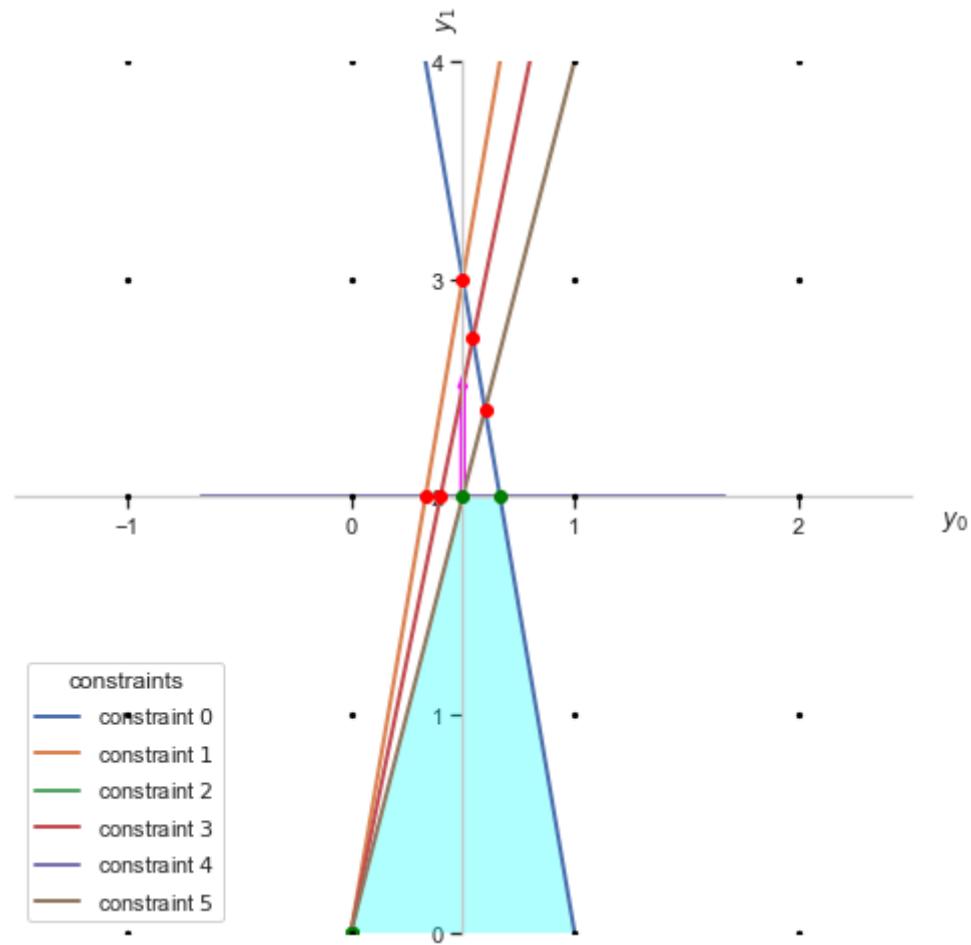
$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

In [42]: `ybar`

Out[42]:

$$\begin{bmatrix} \frac{1}{2} \\ 2 \end{bmatrix}$$

In [43]: `dual_plot()`



In [44]: `pure_gomory(0)`

*** PROBABLY WANT TO APPLY `pivot_algebra()`! ***

In [45]: `pivot_algebra()`

`pivot_algebra()` done

In [46]: `cbar_eta`

Out[46]:

$$\begin{bmatrix} 3 \\ 1 \\ 1 \\ \frac{1}{2} \\ -\frac{1}{2} \end{bmatrix}$$

In [47]: `pivot_ratios(4)`

$$\begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 - \frac{\lambda}{4} \\ -\frac{3\lambda}{4} \\ \lambda \end{bmatrix}$$

In [48]: `pivot_swap(4,1)`

swap accepted --- new partition:

```
eta: [2, 1, 0, 3, 5]
beta: [4, 6]
*** MUST APPLY pivot_algebra()! ***
```

In [49]: `pivot_algebra()`

`pivot_algebra()` done

In [50]: `cbar_eta`

Out[50]:

$$\begin{bmatrix} 3 \\ 2 \\ 0 \\ \frac{4}{3} \\ \frac{2}{3} \end{bmatrix}$$

In [51]: `xbar_beta`

Out[51]:

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

In [52]: `ybar`

Out[52]:

$$\begin{bmatrix} \frac{2}{3} \\ 2 \end{bmatrix}$$

In [53]: `pure_gomory(0)`

```
*** PROBABLY WANT TO APPLY pivot_algebra()! ***
```

In [54]: `pivot_algebra()`

`pivot_algebra()` done

In [55]: `cbar_eta`

Out[55]:

$$\begin{bmatrix} 3 \\ 2 \\ 0 \\ \frac{4}{3} \\ \frac{2}{3} \\ -\frac{2}{3} \end{bmatrix}$$

In [56]: `pivot_ratios(5)`

$$\begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

$\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 - \frac{\lambda}{3} \\ 0 \\ -\frac{2\lambda}{3} \\ \lambda \end{bmatrix}$$

In [57]: `pivot_swap(5,1)`

swap accepted --- new partition:

eta: [2, 1, 0, 3, 5, 6]
 beta: [4, 7]
 *** MUST APPLY `pivot_algebra()`! ***

In [58]: `pivot_algebra()`

`pivot_algebra()` done

In [59]: cbar_eta

Out[59]:

$$\begin{bmatrix} 3 \\ 4 \\ -2 \\ 3 \\ 2 \\ 1 \end{bmatrix}$$

In [60]: pivot_ratios(2)

$$\begin{bmatrix} \frac{1}{4} \\ \infty \end{bmatrix}$$

 $\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} \lambda \\ 0 \\ 0 \\ 0 \\ 1 - 4\lambda \\ 0 \\ 0 \\ 3\lambda \end{bmatrix}$$

In [61]: pivot_swap(2,0)

swap accepted --- new partition:

```
eta: [2, 1, 4, 3, 5, 6]
beta: [0, 7]
*** MUST APPLY pivot_algebra()! ***
```

In [62]: pivot_algebra()

pivot_algebra() done

In [63]: cbar_eta

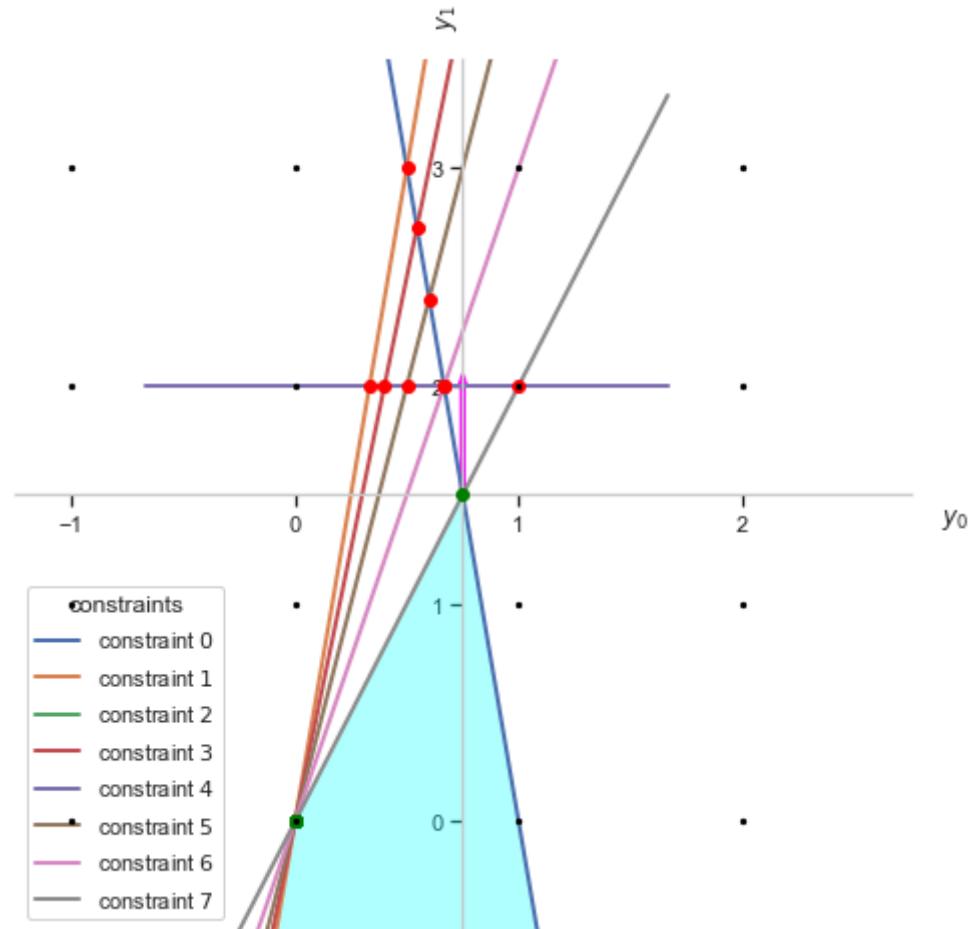
Out[63]:

$$\begin{bmatrix} \frac{5}{2} \\ 2 \\ 3 \\ \frac{1}{2} \\ \frac{9}{4} \\ \frac{3}{2} \\ \frac{3}{4} \end{bmatrix}$$

In [64]: `xbar_beta`

Out[64]: $\begin{bmatrix} \frac{1}{4} \\ \frac{3}{4} \end{bmatrix}$

In [65]: `dual_plot()`



In [66]: `ybar`

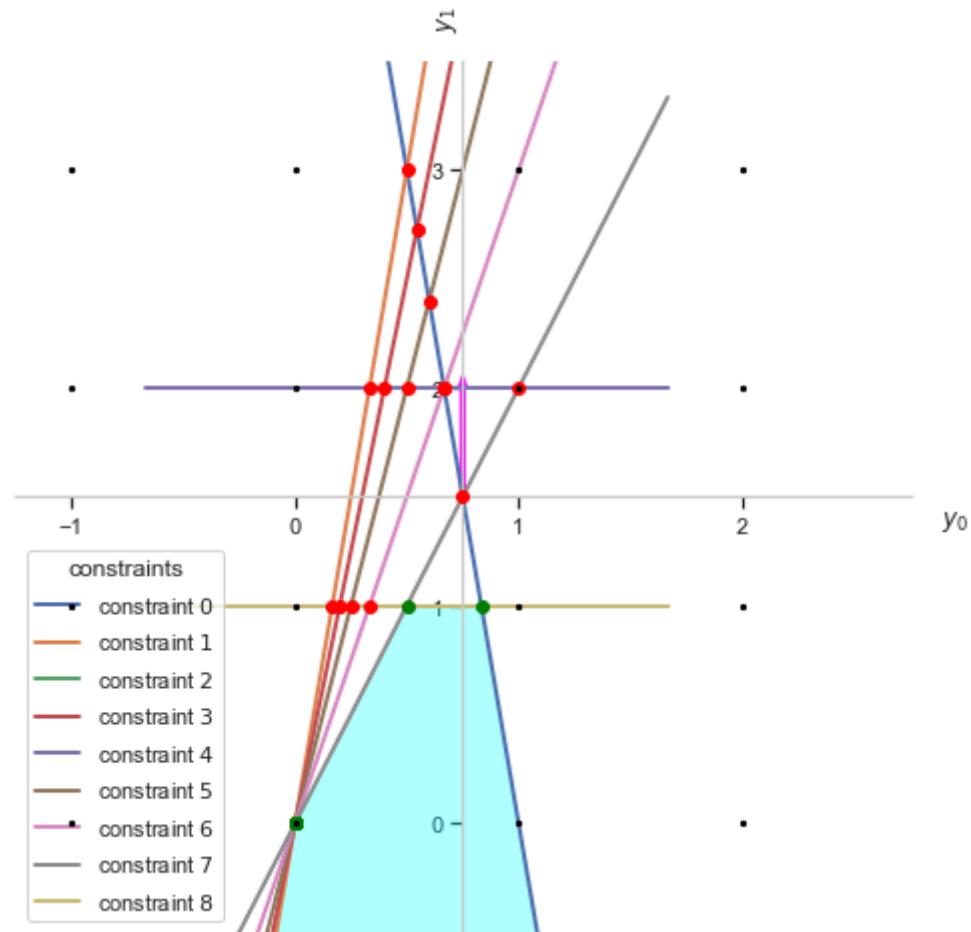
Out[66]: $\begin{bmatrix} \frac{3}{4} \\ \frac{3}{2} \end{bmatrix}$

In [67]: `pure_gomory(1)`

*** PROBABLY WANT TO APPLY `pivot_algebra()`! ***

In [68]: `pivot_algebra()`

`pivot_algebra()` done

In [69]: `dual_plot()`In [70]: `cbar_eta`

Out[70]:

$$\begin{bmatrix} \frac{5}{2} \\ 3 \\ \frac{1}{2} \\ \frac{9}{4} \\ \frac{3}{2} \\ \frac{3}{4} \\ -\frac{1}{2} \end{bmatrix}$$

In [71]: `pivot_ratios(6)`

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

$\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} \frac{1}{4} - \frac{\lambda}{4} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{3}{4} - \frac{3\lambda}{4} \\ \lambda \end{bmatrix}$$



In [72]: `pivot_swap(6,1)`

swap accepted --- new partition:

```
eta: [2, 1, 4, 3, 5, 6, 7]
beta: [0, 8]
*** MUST APPLY pivot_algebra()! ***
```

In [73]: `pivot_algebra()`

`pivot_algebra()` done

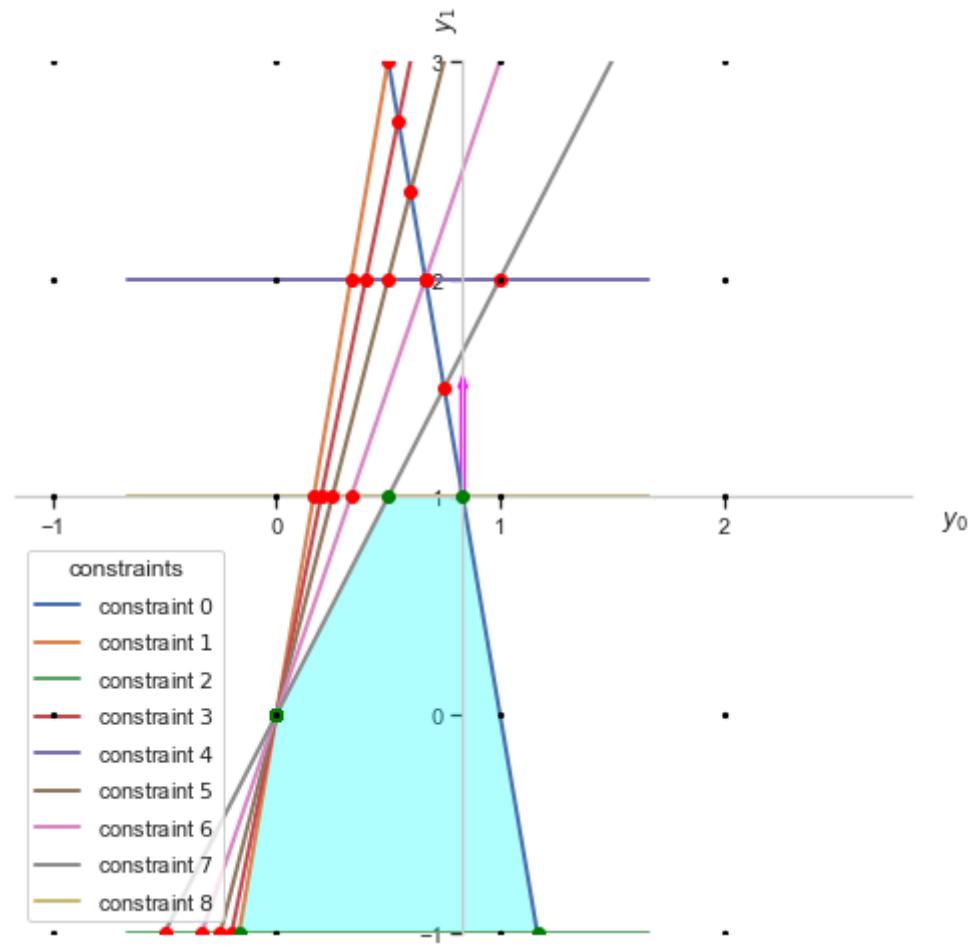
In [74]: `cbar_eta`

Out[74]:

$$\begin{bmatrix} 2 \\ 4 \\ 1 \\ \frac{19}{6} \\ \frac{7}{3} \\ \frac{3}{2} \\ \frac{2}{3} \end{bmatrix}$$



In [75]: `dual_plot()`



In [76]: `ybar`

Out[76]: $\begin{bmatrix} \frac{5}{6} \\ 1 \end{bmatrix}$

In [77]: `pure_gomory(0)`

*** PROBABLY WANT TO APPLY `pivot_algebra()`! ***

In [78]: `pivot_algebra()`

`pivot_algebra()` done

In [79]: cbar_eta

Out[79]:

$$\begin{bmatrix} 2 \\ 4 \\ 1 \\ \frac{19}{6} \\ \frac{7}{3} \\ \frac{3}{2} \\ \frac{2}{3} \\ -\frac{5}{6} \end{bmatrix}$$

In [80]: pivot_ratios(7)

$$\begin{bmatrix} 0 \\ \frac{6}{5} \end{bmatrix}$$

 $\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} -\frac{\lambda}{6} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 - \frac{5\lambda}{6} \\ \lambda \end{bmatrix}$$

In [81]: pivot_swap(7,0)

swap accepted --- new partition:

```
eta: [2, 1, 4, 3, 5, 6, 7, 0]
beta: [9, 8]
*** MUST APPLY pivot_algebra()! ***
```

In [82]: pivot_algebra()

pivot_algebra() done

In [83]: cbar_eta

Out[83]:

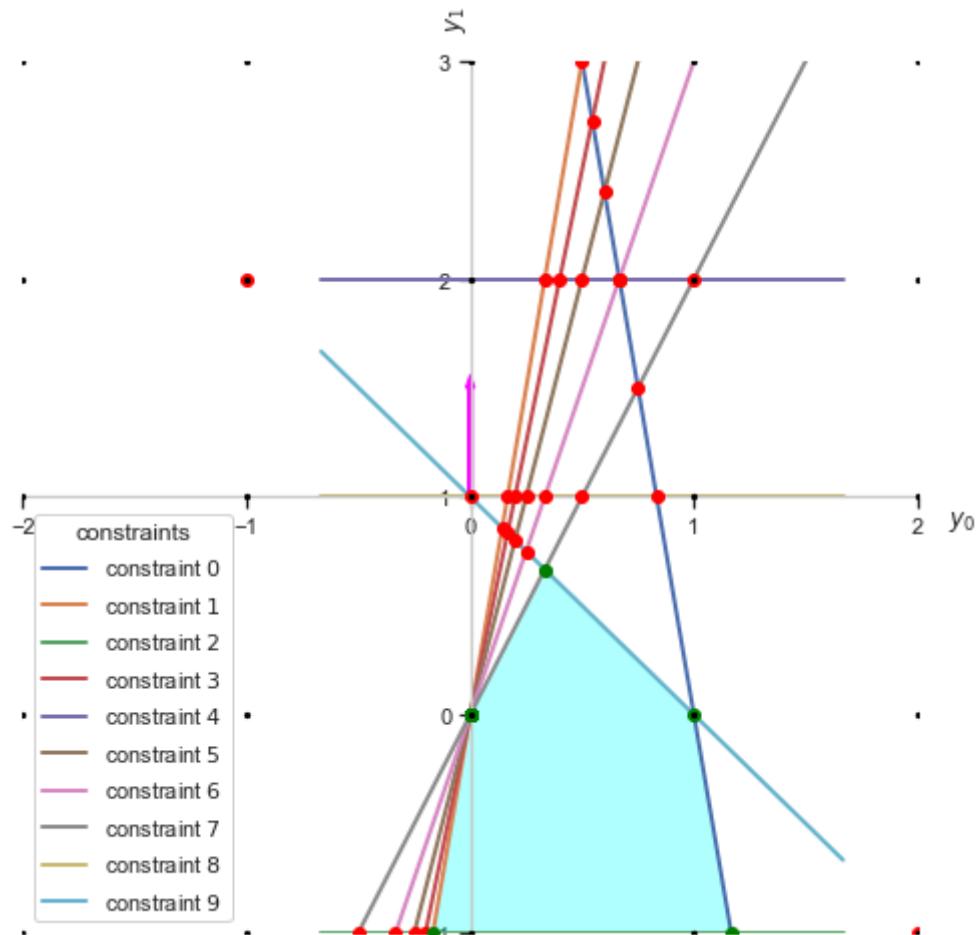
$$\begin{bmatrix} 2 \\ -1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 5 \end{bmatrix}$$

In [84]: xbar_beta

Out[84]:

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

In [85]: dual_plot()



In [86]: `pivot_ratios(6)`

$$\begin{bmatrix} \infty \\ \frac{1}{3} \end{bmatrix}$$

$\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \lambda \\ 1 - 3\lambda \\ 2\lambda \end{bmatrix}$$



In [87]: `pivot_swap(6,1)`

swap accepted --- new partition:

```
eta: [2, 1, 4, 3, 5, 6, 8, 0]
beta: [9, 7]
*** MUST APPLY pivot_algebra()! ***
```

In [88]: `pivot_algebra()`

`pivot_algebra()` done

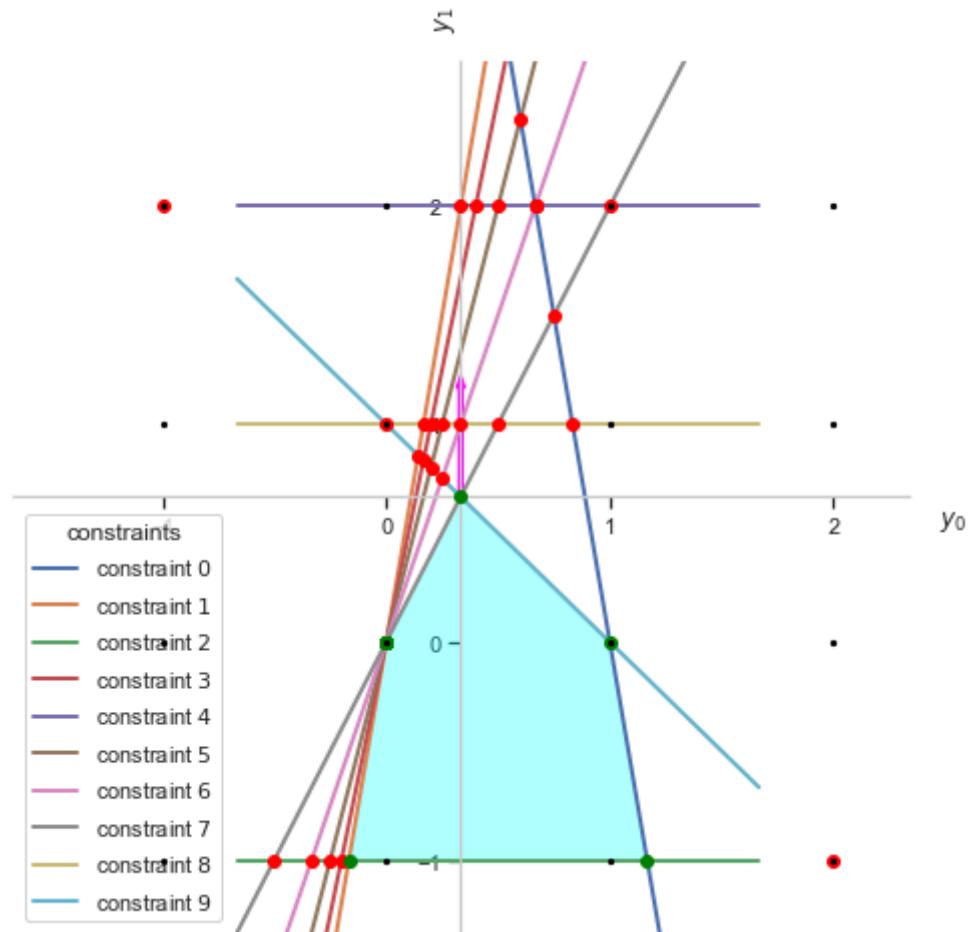
In [89]: `cbar_eta`

Out[89]:

$$\begin{bmatrix} \frac{5}{3} \\ \frac{4}{3} \\ \frac{4}{3} \\ \frac{1}{3} \\ \frac{2}{3} \\ \frac{1}{3} \\ \frac{1}{3} \\ \frac{10}{3} \end{bmatrix}$$



In [90]: `dual_plot()`



In [91]: `ybar`

Out[91]: $\begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \end{bmatrix}$

In [92]: `pure_gomory(1)`

*** PROBABLY WANT TO APPLY `pivot_algebra()`! ***

In [93]: `pivot_algebra()`

`pivot_algebra()` done

In [94]: cbar_eta

Out[94]:

$$\begin{bmatrix} \frac{5}{3} \\ \frac{4}{3} \\ \frac{4}{3} \\ \frac{1}{3} \\ 1 \\ \frac{2}{3} \\ \frac{1}{3} \\ \frac{1}{3} \\ \frac{10}{3} \\ -\frac{2}{3} \end{bmatrix}$$

In [95]: pivot_ratios(8)

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

 $\bar{x} + \lambda \bar{z} :$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{3} - \frac{\lambda}{3} \\ 0 \\ \frac{2}{3} - \frac{2\lambda}{3} \\ \lambda \end{bmatrix}$$

In [96]: pivot_swap(8,1)

swap accepted --- new partition:

```
eta: [2, 1, 4, 3, 5, 6, 8, 0, 7]
beta: [9, 10]
*** MUST APPLY pivot_algebra()! ***
```

In [97]: pivot_algebra()

pivot_algebra() done

In [98]: cbar_eta

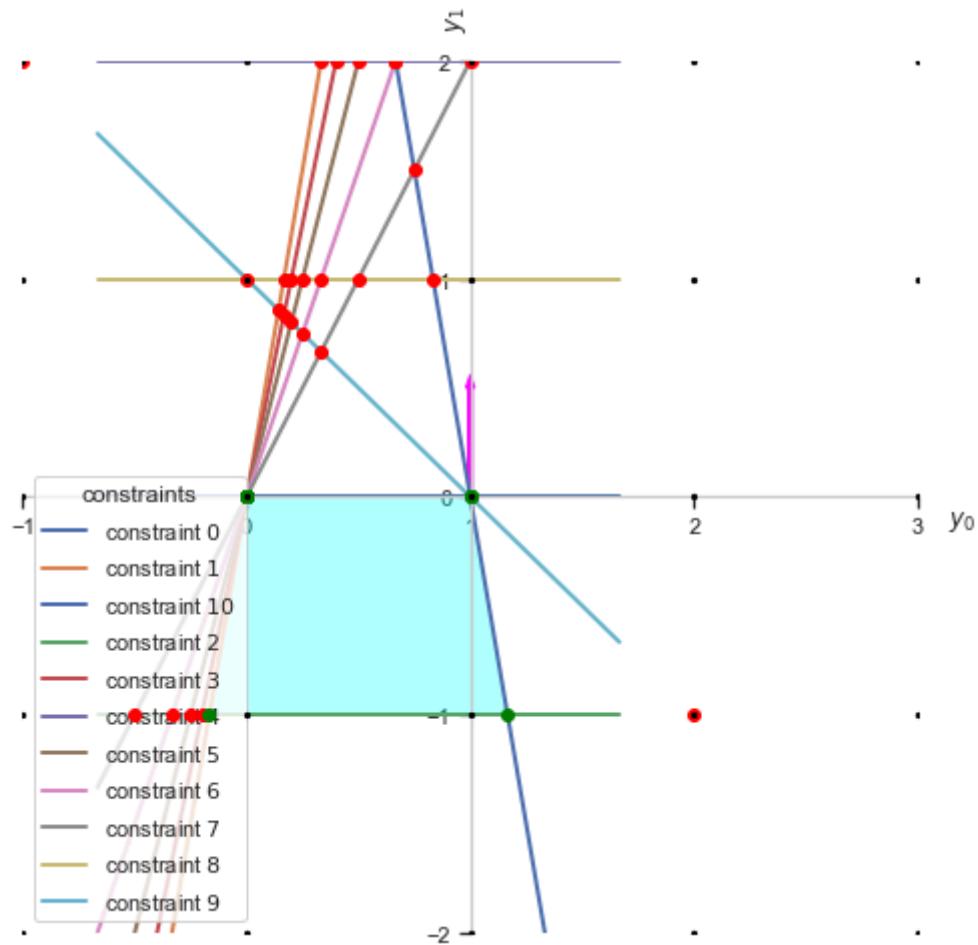
Out[98]: $\begin{bmatrix} 1 \\ 6 \\ 2 \\ 5 \\ 4 \\ 3 \\ 1 \\ 0 \\ 2 \end{bmatrix}$

In [99]: xbar_beta

Out[99]: $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$

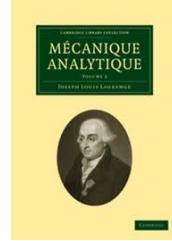
In [100]: ybar

Out[100]: $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$

In [101]: `dual_plot()`

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 "The testing of this hypothesis, however, will be postponed until it is programmed for an electronic computer." — **Ailsa H. Land and Alison G. Doig** (inventors of branch-and-bound), last line of: An Automatic Method of Solving Discrete Programming Problems, *Econometrica*, 1960, Vol. 28, No. 3, pp. 497–520.



³ "Il est facile de voir que...", "il est facile de conclure que...", etc. — **Pierre-Simon Laplace**, frequently in "Traité de Mécanique Céleste".



⁴ "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.



⁵ "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.



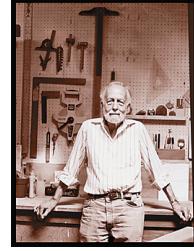
⁶ "Fourier hat sich selbst vielfach um Ungleichungen bemüht, aber ohne erheblichen Erfolg." — **Gyula Farkas**, "Über die Theorie der Einfachen Ungleichungen," *Journal für die Reine und Angewandte Mathematik*, vol. 124:1–27.



⁷ "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.



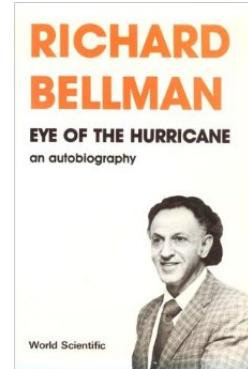
⁸ "George would often call me in and talk about something on his mind. One day in around 1959, he told me about a couple of problem areas: something that Ray Fulkerson worked on, something else whose details I forget. In both cases, he was using a linear programming model and the simplex method on a 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.



⁹ "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 – swissssssh – 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.



¹⁰ "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.



¹¹ "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 <https://web.mit.edu/15.053/www/AMP.htm>. [Cited on page vii]
- [2] Qi He and Jon Lee. Another pedagogy for pure-integer Gomory. *RAIRO-Operations Research*, 51(1):189–197, 2017. [Cited on pages 114 and 124]
- [3] Jon Lee. *A First Course in Combinatorial Optimization*. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, 2004. [Cited on page 101]
- [4] Jon Lee and Angelika Wiegele. Another pedagogy for mixed-integer Gomory. *EURO Journal on Computational Optimization*, 5:455–466, 2017. [Cited on pages 114 and 124]
- [5] Katta G. Murty. *Linear and Combinatorial Programming*. John Wiley & Sons Inc., New York, 1976. [Cited on page vii]

Index of definitions

- 1-norm, 13
- ∞ -norm, 13
- (matrix) product, 3
- concave piecewise-linear function, 65
- convex piecewise-linear function, 61

- affine function, 61
- algebraic perturbation, 36
- arcs, 14, 93
- artificial variable, 39

- basic direction, 26
- basic feasible direction relative to the basic feasible solution, 26
- basic feasible ray, 27
- basic feasible solution, 22
- basic partition, 21
- basic solution, 22
- basis, 4, 21
- basis matrix, 21
- best bound, 113
- big M, 105
- bipartite graph, 95
- branch-and-bound, 110
- breakpoints, 107

- Chvátal-Gomory cut, 114
- column space, 5
- complementary, 49, 51
- concave function, 65
- consecutive-ones matrix, 96
- conservative, 14, 94
- convex function, 61
- convex set, 23
- cost, 14, 94
- Cramer's rule, 7
- cutting pattern, 83
- cutting-stock problem, 82
- cycling, 36

- Dantzig-Wolfe Decomposition, 67
- demand nodes, 19
- determinant, 6
- dimension, 4
- diving, 112
- dot product, 4
- down branch, 111
- dual solution, 29
- dual-feasible basis, 29

- edge weights, 95
- edges, 95
- elementary row operations, 5
- extreme point, 23
- extreme ray, 27

- feasible, 2
- feasible basis, 22
- feasible direction relative to the feasible solution, 26
- feasible region, 2
- flow, 14, 93
- full column rank, 5
- full row rank, 5

- Gauss-Jordan elimination, 6

- head, 14, 93

- identity matrix, 6
- inverse, 6
- invertible, 6

- key invariant for branch-and-bound, 110
- knapsack problem, 85

- Lagrangian Dual, 77
- Laplace expansion, 6
- linear combination, 4
- linear constraints, 1
- linear function, 61
- Linear optimization, 1
- linearly independent, 4
- lower bound, 110

- Main Problem, 69
- matching, 100
- max-norm, 13
- most fractional, 113
- multi-commodity min-cost network-flow problem, 14

- network, 14, 93
- network matrix, 94
- nodes, 14, 93
- non-basis, 21
- non-degeneracy hypothesis, 33
- null space, 5

- objective function, 1
- optimal, 2
- overly complementary, 56

- perfect matching, 95
phase-one problem, 39
phase-two problem, 39
pivot, 34, 125
Polyak step size, 81
polyhedron, 1
rank, 5
ratio test, 32
ray, 27
recursive optimization, 86
reduced costs, 29
reduced-cost fixing, 114
row space, 5
scalar product, 4
Sherman-Morrison formula, 6
single-commodity min-cost network-flow problem, 94
slack variable, 2
solution, 2
span, 4
standard form, 2
subgradient, 77
sufficient unboundedness criterion, 32
supply nodes, 19
surplus variable, 2
tail, 14, 93
the adjacency condition, 107
totally unimodular (TU), 97
transportation problem, 19
trivial, 4
uncapacitated facility-location problem, 105
unimodular, 96
up branch, 111
vertex cover, 101
vertex packing, 125
vertex-edge incidence matrix of the bipartite graph, 95
vertices, 95

Index of Jupyter notebooks

- Circle.ipynb, 45
- CSP.ipynb, 87, 91
- Decomp.ipynb, 72
- MatrixLP.ipynb, 8
- Multi-commodityFlow.ipynb, 16, 19
- pivot_example.ipynb, 28, 43, 45
- pivot_tools.ipynb, 28, 43, 116, 126
- Production.ipynb, 16, 18, 56, 65
- pure_gomory_example_1.ipynb, 116
- pure_gomory_example_2.ipynb, 126
- SubgradProj.ipynb, 82, 91
- UFL.ipynb, 107, 125

