ORMS 1020

Operations Research
with GNU Octave

Tommi Sottinen

tommi.sottinen@uwasa.fi
www.uwasa.fi/~tsottine/or_with_octave/

October 19, 2011

Contents

Preface

> These are the opinions upon which I base my facts. — Winston Churchill

These lecture notes are for the undergraduate course ORMS 1020 "Operations Research" for fall 2010 in the University of Vaasa . This is a 5 credit course with 36 hours lectures and 12 hours of exercises. These notes are a radically modified version of the notes [3] for falls 2008 and 2009 of the same course. The key changes to [3] are:

- We use GNU Octave instead of GNU Linear Programming Kit.
- Computer programming is emphasized.
- Manual labor is no longer considered important. E.g. solving LPs graphically or manually with simplex are only mentioned in passing.
- The Big M method for solving LPs is omitted completely.

The notes concentrate on linear (LP) and mixed linear integer (MILP) optimization modeling and implementing the models as computer programs with Octave. Of course, the course deals with optimization theory also: most notably with the simplex algorithm. The chapters, or sections of chapters, marked with an asterisk (*) may be omitted if time is scarce. The exercises marked with an asterisk (*) are projects. These are longer exercises that require programming. The projects are intended for those who want to show their OR skills for the credits without taking the course. Each project is worth 0–2 points. A solved project gives 1 point and an elegantly solved project gives 2 points. Minimal requirements for a solved project (code) are: (1) It works. Minimal requirements for an elegantly solved project (code) are: (1) It is a solved project. (2) The code is easy to understand. (3) The code is documented (or commented) well. (4) There are input-checkings. 4 points gives the grade 1, 6 points grade 2, and so on.

The author wishes to acknowledge that these lecture notes are collected from the references listed in Bibliography, and from many other sources the author has forgotten. **The author claims no originality**, and hopes not to be sued for plagiarizing or for violating the sacred © laws.

Vaasa October 19, 2011 T. S.

Bibliography

[1] Eaton, J. (2009) *GNU Octave Manual Version 3*.
www.gnu.org/software/octave/doc/interpreter/.

[2] Laaksonen, M. (2005) *TMA.101 Operaatioanalyysi*. Lecture notes.
www.uwasa.fi/~mla/orms1020/oa.html.

[3] Sottinen, T. (2009) *ORMS 1020 Operations Research with GNU Linear Programming Kit*. Lecture notes.
www.uwasa.fi/~tsottine/or_with_glpk/or.pdf.

[4] Winston, W. (2004) *Operations Research: Applications and Algorithms*. International ed edition, Brooks Cole.

Part I

Introduction and Preliminaries

Chapter 1

Selection of Optimization Problems

> It isn't that they can't see the solution. It's that they can't see the problem.
> — G. K. Chesterton

> The Science Graduate asks "How does it work?"
> The Economics Graduate asks "How much does it cost?"
> The Engineering Graduate asks "How can we make it?"
> The Liberal Arts Graduate asks "Do you want fries with that?"
> — Jesse N. Schell.

> Education isn't how much you have committed to memory, or even how much you know. It's being able to differentiate between what you do know and what you don't.
> — Anatole France

We jump right into water and consider optimization problems, their mathematical modeling and solving them with Octave. Have a nice swim!

1.1 Product Selection Problem

> In a **product selection problem** one has to decide how much to produce each product in order to maximize one's profit.
>
> The profit and the constraints on production are linear and, in particular, fractional production is allowed.

Product selection problem is a typical **linear program** (LP).

1.1.1 Example (Giapetto's Problem)**.** Giapetto's Woodcarving Inc. manufactures wooden toy soldiers and trains. A soldier sells for €27 and uses €10 worth of raw materials. Each soldier built increases Giapetto's labor costs by €14. A train sells for €21 and uses €9 worth of raw materials. Each train built increases Giapetto's labor costs by €10. The manufacture of wooden soldiers and trains requires two types of skilled labor: carpentry and finishing. A soldier requires 2 hours of finishing labor and 1 hour of carpentry labor. A train requires 1 hour of finishing and 1 hour of carpentry labor. Each week, Giapetto can obtain all the needed raw material but only 100 finishing hours and 80 carpentry hours. Demand for trains is unlimited, but at most 40 soldier are bought each week.

Giapetto wants to maximize weekly profits (revenues − costs).

To model mathematically optimization problems like 1.1.1 one may follow the following three-step procedure:

1.1.2 Algorithm (Optimization modeling)**.**

Step 1 Find the **decision variables**, i.e. find out what are the variables whose values you can choose.

Step 2 Find the **objective function**, i.e. find out how your objective to be minimized or maximized depends on the decision variables.

Step 3 Find the **constraints**, i.e. find out the (in)equalities that the decision variables must satisfy. (Don't forget the possible sign constraints!)

Let us then start the **mathematical modeling phase** following Algorithm 1.1.2 above.

Giapetto produces soldiers and trains. So, the **decision variables** are:

$$\begin{aligned} x_1 &= \text{number of soldiers produced each week,} \\ x_2 &= \text{number of trains produced each week.} \end{aligned}$$

Once the decision variables are known, the **objective function** z of this problem is simply the revenue minus the costs for each toy, x_1 and x_2:

$$\begin{aligned} z &= (27-10-14)x_1 + (21-9-10)x_2 \\ &= 3x_1 + 2x_2. \end{aligned}$$

Note that the profit z depends linearly on x_1 and x_2. So, this is a linear problem so far (the constraints must turn out to be linear, too).

It may seem at first glance that the profit can be maximized by simply increasing x_1 and x_2. Well, if life were that easy, let's all start manufacturing trains and soldiers and move to Jamaica ☻! Unfortunately, there are **constraints** that

limit the decisions (or else the model is very likely to be wrong). Indeed, we need finishing time and carpentry time for the soldiers and trains, and these are scarce resources. We also need raw material, but this is not scarce. Also, there is a limitation on how many soldiers can be sold.

Let us consider the finishing time constraint then. Both soldiers and trains require finishing time. Suppose e.g. that 10 soldiers and 20 trains were built. The amount of finishing hours needed for that would be 10 times 2 hours (for soldiers) plus 20 times 1 hour (for trains), for a total of 40 hours of finishing labor. The general constraint in terms of the decision variables is:

$$2x_1 + x_2 \quad \leq \quad 100.$$

This is a linear constraint, so we are still dealing with a linear program.

Now that the constraint for the finishing hours is ready, the carpentry hours constraint is found in the same way:

$$x_1 + x_2 \quad \leq \quad 80.$$

This constraint is a linear one. So, we are still in the domain of LPs.

There's only one more constraint remaining for this problem: the weekly demand for soldiers. At most 40 soldiers can be sold each week:

$$x_1 \quad \leq \quad 40,$$

again a linear constraint. There is no demand constraint for the trains.

The mathematical modeling phase is finished, and we have the following LP:

$$\begin{array}{llrcrcrl} & \max z & = & 3x_1 & + & 2x_2 & & \text{(objective function)} \\ & \text{s.t.} & & 2x_1 & + & x_2 & \leq \; 100 & \text{(finishing constraint)} \\ (1.1.3) & & & x_1 & + & x_2 & \leq \; 80 & \text{(carpentry constraint)} \\ & & & x_1 & & & \leq \; 40 & \text{(demand for soldiers)} \\ & & & & & x_1, x_2 & \geq \; 0 & \text{(sign constraints)} \end{array}$$

Note the last **sign constraints**. They ensures that the values of the decision variables will always be positive. The problem does not state this explicitly, but it's still important (and obvious). The problem also implies that the decision variables are integers. So, we should also have **integrity constraints**. However, we are not dealing with IPs (Integer Programs) yet. So, we will just hope that the optimal solution will turn out to be an integer one (it will, but that's just luck).

Now that the mathematical modeling phase is finished, there is the **implementation phase**, i.e. we have to tell the problem to a computer so that it can solve it. Here is the Octave code that solves the Giapetto's product selection problem 1.1.1 by using the Octave's LP solver function `glpk`:

```
octave:1> c = [3,2]';
octave:2> A = [2,1; 1,1; 1,0];
octave:3> b = [100,80,40]';
octave:4> [x_max,z_max] = glpk(c,A,b,[0,0]',[],"UUU","CC",-1)
x_max =

   20
   60

z_max =  180
```

The answer means that Giapetto should produce $x_1 = 20$ soldiers and $x_2 = 60$ trains. Then he will get the maximal possible profit $z = €180$.

If you don't understand the code above, don't worry. We will explain it in detail in Section 2.5. If you are impatient and you have Octave installed, type `help glpk` in Octave's prompt and all the secrets will be unveiled.

1.2 Knapsack Problem

> In a **knapsack problem** a person has to decide which products and how many should he put on his knapsack in order to maximize his comfort.

The difference between the knapsack problem and the product selection problem is that you can only take whole products in your knapsack. So, you can take e.g. 1 compass, or 2 compasses, but not 0.5 compasses. So, the knapsack problem is an **integer program** (IP): an IP is an LP with the additional constraint that all the decision variables must be integers.

1.2.1 Example (Hlynur's Knapsack Problem)**.** Hlynur is going to hike on Eyjafjallajökull. He has a 20 liter knapsack he wants to fill with some tins of hákarl and bottles of svarti dauði. Hákarl comes in 3 liter tins and svarti dauði in 0.75 liter bottles. A tin of hákarl gives 5 units of comfort and a bottle of svarti dauði gives 1 unit of comfort.

Hlynur can only take whole tins and bottles in his knapsack. How should Hlynur pack his knapsack in order to maximize his comfort?

Here is the **mathematical modeling phase**:

Following the algorithm 1.1.2, let us first find the **decision variables**. Since Hlynur only have two things, tins of hákarl and bottles of svarti dauði, in his knapsack, the decision variables are obvious:

$$\begin{aligned} x_1 &= \text{number of tins of hákarl in the knapsack,} \\ x_2 &= \text{number of bottles of svarti dauði in the knapsack.} \end{aligned}$$

With these decision variables, the **objective function**, i.e. Hlynur's comfort is

$$z \quad = \quad 5x_1 + x_2.$$

Next we consider the **constraints**. First we have the obvious constraint that the knapsack can only take 20 liters. Since each tin of hákarl takes 3 liters and each bottle of svarti dauði takes 0.75 liters, we have the constraint

$$3x_1 + 0.75x_2 \quad \leq \quad 20.$$

Next we note that there are no negative hákarls or svarti dauðis, which is quite unfortunate in the next morning — especially in the case of svarti dauði, we have the **sign constraints**

$$x_1, x_2 \quad \geq \quad 0.$$

Finally, remember that Hlynur can only take whole tins and bottles in his knapsack. So, we have the **integrity constraints**

$$x_1, x_2 \quad \text{are integers.}$$

So, we are dealing with the IP

$$\begin{array}{lrcrcll}
\max z & = & 5x_1 & + & x_2 & & & \text{(objective function)} \\
\text{s.t.} & & 3x_1 & + & 0.75x_2 & \leq & 20 & \text{(knapsack constraint)} \\
& & & & x_1, x_2 & \geq & 0, \text{integer} & \text{(sign and integrity constraints)}
\end{array}$$

As for the **implementation phase**, here is the Octave code that solves the Hlynur's knapsack problem 1.2.1 by using the Octave's LP solver `glpk` (which is also an IP solver):

```
octave:1> c = [5, 1]';
octave:2> A = [3, 0.75];
octave:3> b = 20;
octave:4> [x_max,z_max] = glpk(c,A,b,[0,0]',[],"U","II",-1)
x_max =

   6
   2

z_max =  32
```

The answer means that Hlynur should take $x_1 = 6$ tins of hákarl and $x_2 = 2$ bottles of svarti dauði. Then he will get the maximal possible comfort $z = 32$.

If you don't understand the code above, don't worry. We will explain it in detail in later chapters. If you are impatient and you have Octave installed, type `help glpk` in Octave's prompt and all the secrets will be unveiled. The key difference between this code and the code for Giapetto in the previous section is that here we have `"II"` for the two integer decisions and in the previous section we had `"CC"` for the two continuous decisions.

1.3 Portfolio Selection Problem*

1.3.1 Example (Mr. K. invests)**.** Mr. K. wants to invest in two stocks: #1 and #2.

The following parameters have been estimated statistically:

$$\begin{aligned} r_1 &= 10\% \text{ is the return of stock } \#1, \\ r_2 &= 5\% \text{ is the return of stock } \#2, \\ \sigma_1 &= 4 \text{ is the standard deviation of stock } \#1, \\ \sigma_2 &= 3 \text{ is the standard deviation of stock } \#2, \\ \rho &= -0.5 \text{ is the correlation between the stocks } \#1 \text{ and } \#2. \end{aligned}$$

Mr. K. want at least 8% return for his portfolio.

How should Mr. K. distribute his wealth between the two stocks when he wants to minimize his risk?

The **mathematical modeling phase** can be carried out, as before, by using Algorithm 1.1.2:

Let the **decision variables** be w_1 and w_2, the portions of Mr. K.'s wealth put in stocks #1 and #2, respectively. Then Mr. K.'s **objective function** to be minimized is the total risk of his portfolio, which is, according to the bilinearity of the variance function,

$$z = \sqrt{16w_1^2 - 12w_1w_2 + 9w_2^2}$$

The **constraints** are:

$$10w_1 + 5w_2 \geq 8$$

for the return,

$$w_1 + w_2 \leq 1,$$

for the total wealth to be invested, and, if short-selling is not allowed, then there are the **sign constraints**

$$w_1, w_2 \geq 0.$$

At this point we note that this problem is not an LP, since the objective function is not linear. This problem is actually a **quadratic program** (QP). These problems are beyond the scope of this course. If you are interested on how to carry out the **implementation phase** with these kinds of quadratic programs with Octave, type `help qp` in the Octave prompt. That should give you at least a starting point.

1.4 Exercises and Projects

1.1. Model the following problem mathematically:

Arty Artist needs to mix orange color. Arty decides that the color orange has at least 20% red and 20% yellow in it. A 100ml tube of color red costs €2 and a 100ml tube of color yellow costs €3. Arty needs 100ml of the color orange. He has infinity supply of the color red but only 50ml (half a tube) of the color yellow. Arty would like to mix the color orange as cheap as possible.

1.2. Mr. Quine sells gavagais. He will sell one gavagai for 10 Euros. So, one might expect that buying x gavagais from Mr. Quine would cost $10x$ Euros. This linear pricing rule rule may not be true, however. Explain at least three reasons why not.

1.3. Giapetto's problem 1.1.1 was modeled as an LP. This is simplification of reality. Well, all models are. Model Giapetto's problem in a more realistic way by taking into account a connection between price and demand, and any other considerations that may come to your mind.

Is your model linear? Do you think your model can be solved?

1.4. Consider the portfolio selection problem 1.3.1. Suppose Mr K. does not want to minimize his risk, but has as fixed upper bound for it, and wants to maximize the return of his portfolio.

(a) Model this problem.
(b) How is this problem related to the original problem 1.3.1?

1.5. Consider the portfolio selection problem 1.3.1. Suppose Mr. K. measures his risk, not with standard deviation, but with the probability of losing money.

(a) What kind of parameters must Mr. K. estimate and how could he estimate them?
(b) Model this problem.

1.6. * Solve the portfolio selection problem 1.3.1 with Octave's function `qp`.

Chapter 2

Short Introduction to Octave

> How do we convince people that in programming simplicity and clarity — in short: what mathematicians call "elegance" — are not a dispensable luxury, but a crucial matter that decides between success and failure? — Edsger W. Dijkstra
>
> Computers are useless. They can only give you answers. — Pablo Picasso
>
> There are only two kinds of programming languages: those people always bitch about and those nobody uses. — Bjarne Stroustrup
>
> Software is like sex: It's better when it's free. — Linus Torvalds

Octave is a matrix-oriented programming language. It is a free alternative to Matlab©. In these notes we use Octave version 3.2.3. If you are an experienced Matlab© user, you might want to consult
en.wikibooks.org/wiki/MATLAB_Programming/Differences_between_Octave_and_MATLAB.

2.1 Installing Octave

If you use Ubuntu issue the command

```
$ sudo apt-get install octave
```

on a terminal (don't type `$`, that's the prompt). Alternatively, you can use the System → Administration → Synaptic Package Manager, and search for `octave`.

If you use Windows© or Mac©, or some other system than Ubuntu, or simply want to compile Octave yourself, follow the instruction given in octave.sourceforge.net.

Note that classroom computers are "cleaned" every now and then and that the network drives are too slow for any actual use. So, you should **install Octave on a USB stick** if you want to use the classroom computers.

2.2 Octave as Calculator

Basic Arithmetics

Suppose, for example, you want to calculate

$$\frac{1}{8} + 7^2 \times 6 - 1.2 \times 10^{-5}.$$

To do this, type after the Octave prompt (which we denote by `octave:#>`):

```
octave:1> 1/8 + 7^2*6 - 1.2e-5
```

and you get the answer

```
ans = 294.12
```

Here we used the "scientific" notation `1.2e-5` for 1.2×10^{-5}. We could have also written `1.2*10^(-5)` instead of `1.2e-5`.

Suppose then, you want to calculate

$$1 + \frac{2}{7+3}.$$

How to write this in the command line? Is it `1+2/7+3` or `(1+2)/7+3`? It is neither! It is `1+2/(7+3)`. But it is also `1+(2/(7+3))`: It is never wrong to use parentheses to indicate what should be calculated first! E.g., `2*e^2/7+1` is

$$\frac{2e^2}{7} + 1,$$

but to be sure of this you have know the precedence of the operators `^`, `*`, `/`, and `+`. If you are not sure about their precedence, you can always use parentheses: Instead of `2*e^2/7+1` write `(2*(e^2)/7))+1`.

Variables

To assign the value $\frac{1}{3}$ to a variable x, you simply issue

```
octave:1> x = 1/3
x =  0.33333
```

So, if you want to calculate, e.g., $1.2^{1/3}$, you can do it all at once by issuing `1.2^(1/3)`, or you can do it in two phases:

```
octave:1> x = 1/3;
octave:2> 1.2^x
ans = 1.0627
```

Here we used the semicolon (;) to suppress the output in the command `x = 1/3;`. The semicolon (;) is extremely useful in functions where you do not want to have the intermediate steps print their outputs. In the next code we suppress all the outputs of all the commands:

```
octave:1> x = 1/3;
octave:2> y = 1.2^x;
```

Now Octave will give no answer on the screen. The answer is only stored in the variables `x` and `y`. To see the value of a variable one simply calls the variable by stating its name (without suppressing the output with the semicolon):

```
octave:3> x
x =  0.33333
octave:4> y
y =  1.0627
```

If you want to see the variables you have defined, issue the command `who`:

```
octave:1> xx = 2;
octave:2> y = "xx";
octave:3> who
Variables in the current scope:

ans  xx   y
```

The command `who` gives a list of defined variables. In this case we have 3 defined variables: `ans` is the latest unassigned Octave output (if you have one), `xx` is the scalar `2` and `y` is the string of characters `"xx"` (which has nothing to do with the variable `xx`). Note that, depending on your system configuration, you may see other variables too. Also note the double-quote character (") surrounding the string declaration. You can also use the single-quote character (’):

```
octave:1> a = "jippo"; b = 'jippo';
octave:2> strcmp(a,b)
ans =  1
```

Here the function `strcmp` compares character strings. It returns `1` (for true) if the strings are the same, and `0` (for false) otherwise.

Basic Constants and Functions

Octave recognizes some typical mathematical constants like

$$\texttt{pi} = 3.14159\ldots, \quad \texttt{e} = 2.71828\ldots, \quad \texttt{i} = \sqrt{-1}.$$

Any constant can be redefined by the user. If you (accidentally) redefine them, you can always recover the original definition by using the `clear` command. Here

is an example where we first redefine the constant `pi` (for π) and then clear the definition to get the standard built-in definition for `pi` back:

```
octave:1> pi = 42
pi = 42
octave:2> who
Variables in the current scope:

ans  pi

octave:3> clear pi
octave:4> pi
ans =  3.1416
```

Octave has a host of built-in standard mathematical functions like

$$\texttt{exp(x)} = e^x, \quad \texttt{log(x)} = \ln x, \quad \texttt{abs(x)} = |x|.$$

If you (accidentally) redefine built-in functions, you can clear them back to the built-in definitions by using the command `clear` with the switch `-f`, i.e., by typing `clear -f`.

Number Format

Octave standard output uses the precision of 5 significant digits. To get 15 significant digits use the command `format long`. To get back to 5 significant digits use the command `format short`.

```
octave:1> pi
ans = 3.1416
octave:2> format long
octave:3> pi
ans =  3.14159265358979
octave:4> format short; pi
ans =  3.1416
```

The `format` command only changes the output format of the precision. It does not change the internal precision. Octave uses fixed precision, and is thus prone to rounding errors as most programming languages are:

```
octave:1> format short
octave:2> 1 - 0.2 - 0.2 - 0.2 - 0.2 - 0.2
ans =  5.5511e-17
octave:3> format long
octave:4> 1 - 0.2 - 0.2 - 0.2 - 0.2 - 0.2
ans =  5.55111512312578e-17
```

The rounding error in the code above is due to the fact that 0.2 is nasty in base-2 arithmetics: $0.2(\text{base-10}) = 0.0011\,0011\ldots(\text{base-2})$.

2.3 Linear Algebra with Octave

Matrices, vectors, and Their Transposes

> The matrix is a world pulled in front of your eyes to hide the truth from you.
> — Morpheus

The Morpheus's answer above to the Neo's question "What is the matrix?" refers to the Latin language origin of the word matrix: the womb. In mathematics a **matrix** is an array of numbers. We say that $\mathbf{A}$ is an $(m \times n)$-matrix if it has m rows and n columns:

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix}.$$

2.3.1 Example.

$$\mathbf{A} = \begin{bmatrix} 5 & 2 & -3 \\ 6 & 0 & 0.4 \end{bmatrix}$$

is a (2×3)-matrix, and, e.g. $A_{12} = 2$ and $A_{23} = 0.4$; A_{32} does not exist.

In Octave one creates matrices and refers to their elements, in the following way (cf. Example 2.3.1 above):

```
octave:1> A = [ 5, 2, -3; 6, 0, 0.4 ]
A =

   5.00000   2.00000  -3.00000
   6.00000   0.00000   0.40000

octave:2> A(1,2)
ans =  2
octave:3> A(2,3)
ans =  0.40000
octave:4> A(3,2)
error: A(I): Index exceeds matrix dimension.
```

So, elements in a row are separated by a comma (`,`), or simply by a space, and the rows are separated by a semicolon (`;`). To refer to elements of a matrix one uses parentheses instead of sub-indexes: $A_{ij} =$`A(i,j)`.

The **transpose** $\mathbf{A}'$ of a matrix $\mathbf{A}$ is obtained by changing its rows to columns, or vice versa: $A'_{ij} = A_{ji}$. So, if $\mathbf{A}$ is an $(m \times n)$-matrix

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix},$$

then its transpose $\mathbf{A}'$ is the $(n \times m)$-matrix

$$\mathbf{A}' = \begin{bmatrix} A'_{11} & A'_{12} & \cdots & A'_{1m} \\ A'_{21} & A'_{22} & \cdots & A'_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A'_{n1} & A'_{n2} & \cdots & A'_{nm} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{21} & \cdots & A_{m1} \\ A_{12} & A_{22} & \cdots & A_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1n} & A_{2n} & \cdots & A_{mn} \end{bmatrix}.$$

2.3.2 Example. If

$$\mathbf{A} = \begin{bmatrix} 5 & 2 & -3 \\ 6 & 0 & 0.4 \end{bmatrix},$$

then

$$\mathbf{A}' = \begin{bmatrix} 5 & 6 \\ 2 & 0 \\ -3 & 0.4 \end{bmatrix}.$$

So, e.g., $A'_{12} = 6 = A_{21}$.

In Octave the transpose operator is, surprise, surprise, the single-quote (’). So, Example 2.3.2 above with Octave goes like this:

```
octave:1> A = [5 2 -3; 6 0 0.4];
octave:2> A'
ans =

   5.00000   6.00000
   2.00000   0.00000
  -3.00000   0.40000

octave:3> A'(1,2)
ans =  6
octave:4> A(2,1)
ans =  6
```

A **vector** is either an $(n \times 1)$-matrix or a $(1 \times n)$-matrix. We call $(n \times 1)$-matrices **column vectors** and $(1 \times n)$-matrices **row vectors**. We will almost always assume that vectors are column vectors. So, e.g., a 3-dimensional vector $\mathbf{x}$

will be

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \text{not} \quad [x_1 \;\; x_2 \;\; x_3]\,.$$

Block and Sub Matrices

When we want to pick up rows or columns of a matrix $\mathbf{A}$ we use the **dot-notation** (in mathematics) or the **colon-notation** (in Octave):

Let $\mathbf{A}$ be the matrix

$$\mathbf{A} \;=\; \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix}.$$

Then its i^{th} row is the n-dimensional row vector

$$\mathbf{A}_{i\bullet} \;=\; [A_{i1} \;\; A_{i2} \;\; \cdots \;\; A_{in}]\,.$$

Similarly, $\mathbf{A}$'s j^{th} column is the m-dimensional column vector

$$\mathbf{A}_{\bullet j} \;=\; \begin{bmatrix} A_{1j} \\ A_{2j} \\ \vdots \\ A_{mj} \end{bmatrix}.$$

2.3.3 Example. If

$$\mathbf{A} \;=\; \begin{bmatrix} 5 & 2 & -3 \\ 6 & 0 & 0.4 \end{bmatrix},$$

then

$$\mathbf{A}_{2\bullet} \;=\; [6 \;\; 0 \;\; 0.4]$$

and

$$\mathbf{A}_{\bullet 3} \;=\; \begin{bmatrix} -3 \\ 0.4 \end{bmatrix}.$$

With Octave one uses the colon (:) instead of the dot ($\bullet$). So, Example 2.3.3 above with Octave reads:

```
octave:1> A= [ 5 2 -3; 6 0 0.4 ];
octave:2> A(2,:)
ans =

   6.00000   0.00000   0.40000

octave:3> A(:,3)
ans =

  -3.00000
   0.40000
```

When we want to combine matrices we use the **block-notation**:

Let $\mathbf{A}$ be a $(m\times n)$-matrix and let $\mathbf{B}$ be a $(m\times k)$-matrix Then the block matrix $[\mathbf{A}\ \ \mathbf{B}]$ is the $(m\times(n+k))$-matrix

$$[\mathbf{A}\ \ \mathbf{B}] \;=\; \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} & B_{11} & B_{12} & \cdots & B_{1k} \\ A_{21} & A_{22} & \cdots & A_{2n} & B_{21} & B_{22} & \cdots & B_{2k} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} & B_{m1} & B_{m2} & \cdots & B_{mk} \end{bmatrix}.$$

Similarly, if $\mathbf{C}$ is a $(p\times n)$-matrix, then the block matrix $\begin{bmatrix} \mathbf{A} \\ \mathbf{C} \end{bmatrix}$ is defined as

$$\begin{bmatrix} \mathbf{A} \\ \mathbf{C} \end{bmatrix} \;=\; \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \\ C_{11} & C_{12} & \cdots & C_{1n} \\ C_{21} & C_{22} & \cdots & C_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{p1} & C_{p2} & \cdots & C_{pn} \end{bmatrix}.$$

2.3.4 Example. Let

$$\mathbf{A} = \begin{bmatrix} 5.1 & 2.1 \\ 6.5 & -0.5 \\ 0.1 & 10.5 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 20 \\ 30 \end{bmatrix}, \quad \text{and} \quad \mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Then

$$\begin{bmatrix} 1 & -\mathbf{c}' \\ \mathbf{0} & \mathbf{A} \end{bmatrix} \;=\; \begin{bmatrix} 1 & -20 & -30 \\ 0 & 5.1 & 2.1 \\ 0 & 6.5 & -0.5 \\ 0 & 0.1 & 10.5 \end{bmatrix}.$$

With Octave one introduces the block matrices in the obvious way: e.g., Example 2.3.4 can be written as

```
octave:1> A=[5.1 2.1; 6.5 -0.5; 0.1 10.5]; c=[20 30]'; o=[0 0 0]';
octave:2> [ 1 -c'; o A]
ans =

    1.00000  -20.00000  -30.00000
    0.00000    5.10000    2.10000
    0.00000    6.50000   -0.50000
    0.00000    0.10000   10.50000
```

Finally, let us note that by combining the dot and block notation we have:

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{1\bullet} \\ \mathbf{A}_{2\bullet} \\ \vdots \\ \mathbf{A}_{m\bullet} \end{bmatrix} = [\mathbf{A}_{\bullet 1} \ \mathbf{A}_{\bullet 2} \ \cdots \ \mathbf{A}_{\bullet n}].$$

Matrix Sums and Products

Matrix sum and scalar multiplication are defined pointwise:

Let

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1n} \\ B_{21} & B_{22} & \cdots & B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mn} \end{bmatrix}.$$

Then the **matrix sum** $\mathbf{A} + \mathbf{B}$ is defined as

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} A_{11} + B_{11} & A_{12} + B_{12} & \cdots & A_{1n} + B_{1n} \\ A_{21} + B_{21} & A_{22} + B_{22} & \cdots & A_{2n} + B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} + B_{m1} & A_{m2} + B_{m2} & \cdots & A_{mn} + B_{mn} \end{bmatrix}.$$

Let λ be a real number. Then the **scalar multiplication** $\lambda\mathbf{A}$ is defined as

$$\lambda\mathbf{A} = \begin{bmatrix} \lambda A_{11} & \lambda A_{12} & \cdots & \lambda A_{1n} \\ \lambda A_{21} & \lambda A_{22} & \cdots & \lambda A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda A_{m1} & \lambda A_{m2} & \cdots & \lambda A_{mn} \end{bmatrix}.$$

2.3.5 Example. Let

$$\mathbf{A} = \begin{bmatrix} 5 & 2 \\ 33 & 20 \end{bmatrix} \quad \text{and} \quad \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Then

$$\mathbf{A} - 100\mathbf{I} \;=\; \begin{bmatrix} -95 & 2 \\ 33 & -80 \end{bmatrix}.$$

With Octave Example 2.3.5 goes like this:

```
octave:1> A=[5 2; 33 20]; I=[1 0; 0 1];
octave:2> A-100*I
ans =

  -95    2
   33  -80
```

Let $\mathbf{A}$ be a $(m\times n)$-matrix

$$\mathbf{A} \;=\; \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix},$$

and let $\mathbf{B}$ be a $(n\times p)$-matrix

$$\mathbf{B} \;=\; \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{n1} & B_{n2} & \cdots & B_{np} \end{bmatrix}.$$

Then the **product matrix** $\mathbf{C} = \mathbf{AB}$ is the $(m\times p)$-matrix defined by

$$C_{ij} \;=\; \sum_{k=1}^{n} A_{ik}B_{kj}.$$

2.3.6 Example.

$$\begin{bmatrix} 2 & 1 & 5 \\ 0 & 3 & -1 \end{bmatrix} \begin{bmatrix} 1 & 5 & 3 & -1 \\ 7 & -4 & 2 & 5 \\ 0 & 2 & 1 & 6 \end{bmatrix} \;=\; \begin{bmatrix} 9 & 16 & 13 & 33 \\ 21 & -14 & 5 & 9 \end{bmatrix},$$

since, e.g.,

$$\begin{aligned} 9 &= C_{11} \\ &= A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31} \\ &= 2\times 1 + 1\times 7 + 5\times 0. \end{aligned}$$

With Octave one uses star (*) to denote **matrix multiplication**. So, Example 2.3.6 would read

```
octave:1> [2 1 5; 0 3 -1] * [1 5 3 -1; 7 -4 2 5; 0 2 1 6]
ans =

    9   16   13   33
   21  -14    5    9
```

Note that while matrix sum is commutative: $\mathbf{A}+\mathbf{B}=\mathbf{B}+\mathbf{A}$, the matrix product is not: $\mathbf{AB}\neq\mathbf{BA}$. Indeed, it may be that $\mathbf{BA}$ is not even defined even though $\mathbf{AB}$ is. Otherwise the matrix algebra follows the rules of the classical algebra of the real numbers. So, e.g.,

$$\begin{aligned}(\mathbf{A}+\mathbf{B})(\mathbf{C}+\mathbf{D}) &= (\mathbf{A}+\mathbf{B})\mathbf{C}+(\mathbf{A}+\mathbf{B})\mathbf{D}\\ &= \mathbf{AC}+\mathbf{BC}+\mathbf{AD}+\mathbf{BD}\\ &= \mathbf{A}(\mathbf{C}+\mathbf{D})+\mathbf{B}(\mathbf{C}+\mathbf{D}).\end{aligned}$$

Identity Matrix and Matrix Inverse

The **identity matrix** $\mathbf{I}_n$ is an $(n\times n)$-matrix (a square matrix) with 1s on the diagonal and 0s elsewhere:

$$\mathbf{I}_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0\\ 0 & 1 & 0 & \cdots & 0 & 0\\ 0 & 0 & 1 & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & 0 & \cdots & 1 & 0\\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}.$$

We shall usually write shortly $\mathbf{I}$ instead of $\mathbf{I}_n$, since the dimension n of the matrix is usually obvious. To create an identity matrix with Octave one uses the function `eye`:

```
octave:1> I = eye(4)
I =

Diagonal Matrix

   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
```

The **inverse matrix** $\mathbf{A}^{-1}$ of a matrix $\mathbf{A}$, if it exists, is such a matrix that

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I} = \mathbf{A}\mathbf{A}^{-1}.$$

2.3.7 Example. Let

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 1 & 3 \end{bmatrix}.$$

Then

$$\mathbf{A}^{-1} = \begin{bmatrix} 3 & -2 \\ -1 & 1 \end{bmatrix}.$$

Finding the inverse of a matrix manually is tedious. We do not even bother to show how it is done here. With Octave one uses the function `inv` to create the inverse matrix:. So, Example 2.3.7 above would read

```
octave:1> A = [1 2; 1 3];
octave:2> inv(A)
ans =

   3  -2
  -1   1
```

2.3.8 Example. The matrix

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

has no inverse. Indeed, if the inverse $\mathbf{A}^{-1}$ existed then, e.g., the equation

$$\mathbf{Ax} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

would have a solution in $\mathbf{x} = [x_1 \;\; x_2]'$:

$$\mathbf{x} = \mathbf{A}^{-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

But this is impossible since no matter what $\mathbf{x} = [x_1 \; x_2]'$ you choose

$$\mathbf{Ax} = \begin{bmatrix} x_1 + x_2 \\ 0 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Here is what Octave says about Example 2.3.8 above:

```
octave:1> A = [1 1; 0 0];
octave:2> inv(A)
warning: inverse: matrix singular to machine precision,
rcond = 0
ans =

   Inf   Inf
   Inf   Inf
```

Linear Systems of Equations

A **linear system**, or a linear system of equations, is

$$\begin{array}{rcrcccrcl} A_{11}x_1 & + & A_{12}x_2 & + & \cdots & + & A_{1n}x_n & = & b_1, \\ A_{21}x_1 & + & A_{22}x_2 & + & \cdots & + & A_{2n}x_n & = & b_2, \\ & & & & & & & \vdots & \\ A_{m1}x_1 & + & A_{m2}x_2 & + & \cdots & + & A_{mn}x_n & = & b_m. \end{array} \tag{2.3.9}$$

Solving the linear system (2.3.9) means finding the variables $x_1, x_2, \ldots, x_n$ that satisfy all the equations in (2.3.9) simultaneously: the parameters A_{ij} and b_i are given.

The connection between linear systems and matrices is the following: Let

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix},$$

then the linear system (2.3.9) may be rewritten as

$$\mathbf{Ax} = \mathbf{b}.$$

Solving linear systems manually is tedious. We do not even bother to show how it is done here. With Octave one can solve linear systems easily with the **left division** operator (\): The system $\mathbf{Ax} = \mathbf{b}$ is "divided from the left" by $\mathbf{A}$, so that $\mathbf{x} = \mathbf{A}\backslash\mathbf{b}$. The next Octave code shows how to solve the system

$$\begin{array}{rcrcl} x_1 & + & x_2 & = & 2, \\ 2x_1 & + & 4x_2 & = & 7. \end{array}$$

```
octave:1> A = [1 1; 2 4]; b=[2;7];
octave:2> A\b
ans =

   0.50000
   1.50000
```

Pointwise Operators

Octave is matrix-oriented: operators like `+`, `-`, `*`, `/`,`^` operate on matrices. This can cause problems if we want to operate pointwise. For pointwise operations there are the dotted operators `.*` and `./` (for `+` and `-` there is no need for dotted versions). The next example elaborates the difference of `*` and `.*`.

2.3.10 Example. A firm has 2 employees and 3 tasks. Let the matrix **A** tell how many hours each employee (rows) spends on each tasks (columns):

$$\mathbf{A} = \begin{bmatrix} 6 & 0 & 0 \\ 1 & 4 & 1 \end{bmatrix}.$$

This means that employee number 1 spends 6 hours on task 1, and none on other tasks; and employee spends 1 hour for task 1, 4 hours for task 2, and 1 hour for task 3. Suppose then that the number of working hours by the employee number 1 is increased by 20% on the first task, 10% on the second task, and 5% on the third task. Similarly, the number of working hours of the employee number 2 is decreased by 10% on all the tasks 1, 2, and 3. So, the new — relative to the old one — allocations of the tasks are

$$\mathbf{N} = \begin{bmatrix} 1.20 & 1.10 & 1.05 \\ 0.90 & 0.90 & 0.90 \end{bmatrix}.$$

How many hours does each employee spend on each task now?

If **B** is the matrix answering the question, then $B_{ij} = A_{ij}N_{ij}$ ($i = 1, 2$ are the employees and $j = 1, 2, 3$ are the tasks). So, to construct the matrix **B** one does not use the matrix product **AN**, but the **pointwise product A.N**:

```
octave:1> A = [6, 0, 0; 1, 4, 1];
octave:2> N = [1.20, 1.10, 1.05; 0.90, 0.90, 0.90];
octave:3> B = A.*N
B =

   7.20000   0.00000   0.00000
   0.90000   3.60000   0.90000
```

So, the new working-hours for employees (rows) in tasks (rows), are

$$\mathbf{B} = \begin{bmatrix} 7.2 & 0 & 0 \\ 0.9 & 3.6 & 0.9 \end{bmatrix}.$$

If you try the matrix product **AN** you will get

```
octave:4> B= A*N
error: operator *: nonconformant arguments (op1 is 2x3, op2
is 2x3)
```

What he error says is that the dimensions of **A** (`op1`) and **N** (`op2`) are not consistent for matrix multiplication.

Pointwise Vectorized Functions

Most (but not all!) functions in Octave are pointwise vectorized. This means that scalar functions $y = f(x)$ are typically extended (vectorized) as

$$[y_1 \ \cdots \ y_n] \quad = \quad \mathbf{y} \quad = \quad f(\mathbf{x}) \quad = \quad [f(x_1) \ \cdots \ f(x_n)].$$

For example the function `exp` works this way:

```
octave:1> exp([1,-1,0])
ans =

   2.7183   0.36788   1.0000
```

So, upto four decimal places, $e^1 = 2.7183$, $e^{-1} = 0.36788$, and $e^0 = 1.0000$.

2.4 Function and Script Files

Function Files

A **function file** is a text file that defines a function you can call from Octave prompt. The function file should be named `fcn_name.m`, where `fcn_name` is the name of the function.

As an example we present the function `npv` that calculates the net present value (NPV) of an cashflow. For that, recall the mathematical formula:

$$\mathrm{NPV}(\mathbf{c}) \quad = \quad \sum_t \frac{c_t}{(1+r)^t},$$

where $\mathbf{c} = [c_1 \ \cdots \ c_T]'$ is the cashflow and r is the rate of return.

Here is the implementation of the formula above as the function `npv`. The code is written in the file `npv.m`. The line numbers are not part of the code: they are for reference purposes only.

```
 1 function v = npv(cf,r)
 2 ## Function v = npv(cf,r) returns the Net Present Value (npv) of the
 3 ## cashflow cf.  The cash flow cf is received at the end of each
 4 ## period.  The  rate of return over the period is r.  The parameter r
 5 ## is scalar.  The cash flow cf is a (column) vector.
 6
 7 T = length(cf);                 # The number of periods.
 8 pv = zeros(T,1);                # Initialize present values (pv) at zero.
 9 for t=1:T
10   pv(t) = cf(t) / (1+r)^t;      # Set the pv's.
11 endfor
12 v = sum(pv);                   # npv is the sum of pv's.
13 endfunction
```

In the line 1 we have first the keyword `function`. Every function file must start with this keyword. After the keyword there is the return value, or values, of the function. In this case the function will return just one value: `v`, which is the net present value. It is possible to have many return values. For example, if the function `npv` would have 3 return values, `v`, `w`, and `wv`, say, then the line 1 would be

```
1 function [v,w,vw] = npv(cf,r)
```

It is even possible to have variable amount of return values by using the keyword `varargout` (variable arguments out).

After the return value comes always the equal sign (`=`) followed by the name of the function (in this case `npv`) that should be the same as the name of the function file (without the extension `.m`).

After the name of the function come, inside parentheses, the input parameters of the function. In this case we have two input parameters: `cf` for the cash flow and `r` for the rate of return. It is possible to have variable amount of input parameters by using the keyword `varargin` (variable arguments in).

Next, in the lines 2–5 there is a comment block. Octave will interpreter anything that follows the pound signs (`#`) as a comment until the end of the line. In general, for Octave a comment is something to be read by humans, and not to be interpreted by a computer. The comments right after the `function`-line have a special meaning, however: they are the lines to be printed out when a user types `help` for the function:

```
octave:1> help npv
'npv' is a function from the file /home/tsottine/work/teaching/or2010/
m-files/npv.m

 Function v = npv(cf,r) returns the Net Present Value (npv) of the
 cash flow cf.  The cash flow cf is received at the end of each
 period.  The  rate of return over the period is r.  The parameter r
 is scalar.  The cash flow cf is a (column) vector.

Additional help for built-in functions and operators is
available in the on-line version of the manual.  Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at http://www.octave.org and via the help@octave.org
mailing list.
```

The empty line 6 is here to terminate the comment block.

In the line 7 we set the variable `T` to be the number of periods of the cash flow, which is the same as the length of the vector `cf`. Note the semicolon (`;`) at the

end of the expression. The semicolon (`;`) is there to suppress the output: typically one does not want functions to print any results. The functions' results should be in the output variables! The line ends with a comment after the comment sign (`#`). We use these end-of-the-line comments also later, as you can see.

In the line 8 we initialize a vector `pv` for the present values of the individual payments. It is not strictly necessary to initialize your variables, but it is a very good idea to do so. Indeed, initialization will speed up the code, make it more difficult to make mistakes, and make your code easier to debug once you have made a mistake. We initialize all the `pv` values to zero (for no apparent reason) with the function `zeros`. Type `help zeros` for more information of how the function `zeros` work. Indeed, it is always helpful to use `help` whenever you encounter a piece of code you do not understand ☺. Note, again, that the line ends with the semicolon (`;`) so that Octave would not print out a vector of zeros here.

The lines 9–11 constitute a `for`-loop where we set the true present values for `pv` in the line 10, which is indented to emphasize the fact that it is inside a `for`-loop. Note the semicolon in the line 10, that prevents the unnecessary printing of the value of the variable `pv`. The line 9 denotes the beginning of a `for`-loop. Octave's `for`-loop works quite differently from the for loops of, say, C or Java. In Octave, the running index (`t` in this case) runs through the values of the elements of a given vector (`1:T` in this case). In Octave `a:b` means a vector $[a\ (a{+}1)\ \cdots b]$. So, this `for`-loop will run for the values $t = 1, 2, \ldots, T$. The line 11 denotes the end of the `for`-loop.

In the line 12 the return value `v` of the function is set to be the sum of the present values found in the vector `pv`. Note, again, that the output is suppressed by the semicolon (`;`).

Finally, the line 13 denotes the end of the `function`-block that started in the line 1.

Script Files

A **script file** is a text file containing (almost) any sequence of Octave commands. It is read and evaluated just as if you had typed each command at the Octave prompt, and provides a convenient way to perform a sequence of commands. To call the script file from the Octave prompt one simply issues its name (without the filename extension `.m`).

Unlike a function file, a script file must not begin with the keyword `function`. Indeed, that would make it a function file. A script file may, however, contain function definitions: it just may not start with a function definition. If you happen to have a function definition at the beginning of a would-be script file, you can always start your file with the line containing just `1;` . This line does practically nothing. Indeed, the command `1;` evaluates `1` as 1 and then the semicolon suppresses the output. So nothing happens, except that now the file does not

begin with a function definition. That should do the trick of converting a would-be function file into a script file. Note that prepending a comment-block or empty lines before the function definition will not do the trick.

Here is a script file `npv_script.m` that does pretty much the same as the function `npv`, but with fixed parameters `cf=[1 1 1 1]'` and `r=0.05`:

```
cf = [1 1 1 1]';                # The cash flow.
r = 0.05;                       # The rate of return.

T = length(cf);                 # The number of periods.
pv = zeros(T,1);                # Initialize present values (pv) at zero.
for t=1:T
  pv(t) = cf(t) / (1+r)^t;      # Set the pv's.
endfor
v = sum(pv);                    # npv is the sum of pv's.
```

To use this script file one simply types `npv_script` in the Octave prompt.

The key difference between the function `npv` and the script `npv_script` is that after calling the script file `npv_script` the variables `cf`, `r`, `T`, `pv`, and `v` are now (re)defined, while calling the function `npv` does not set any (global) variables. This is why the author avoids script files whenever he can ☺.

Working Directory and Search Path

A typical problem when working with Octave is that Octave does not find your function or script files. Octave will look for script and function files from the so-called **search path**. To see your search path type `path`. Here is the `path` in the author's setting:

```
octave:1> path

Octave's search path contains the following directories:


.
/usr/local/share/octave/site-m
/usr/lib/octave/3.2.3/oct/i486-pc-linux-gnu
/usr/share/octave/3.2.3/m
/usr/share/octave/3.2.3/m/path
/usr/share/octave/3.2.3/m/testfun
/usr/share/octave/3.2.3/m/linear-algebra
/usr/share/octave/3.2.3/m/plot
/usr/share/octave/3.2.3/m/io
/usr/share/octave/3.2.3/m/miscellaneous
/usr/share/octave/3.2.3/m/specfun
/usr/share/octave/3.2.3/m/help
/usr/share/octave/3.2.3/m/image
/usr/share/octave/3.2.3/m/audio
```

```
/usr/share/octave/3.2.3/m/geometry
/usr/share/octave/3.2.3/m/sparse
/usr/share/octave/3.2.3/m/polynomial
/usr/share/octave/3.2.3/m/signal
/usr/share/octave/3.2.3/m/pkg
/usr/share/octave/3.2.3/m/statistics
/usr/share/octave/3.2.3/m/statistics/distributions
/usr/share/octave/3.2.3/m/statistics/base
/usr/share/octave/3.2.3/m/statistics/models
/usr/share/octave/3.2.3/m/statistics/tests
/usr/share/octave/3.2.3/m/deprecated
/usr/share/octave/3.2.3/m/special-matrix
/usr/share/octave/3.2.3/m/optimization
/usr/share/octave/3.2.3/m/time
/usr/share/octave/3.2.3/m/general
/usr/share/octave/3.2.3/m/startup
/usr/share/octave/3.2.3/m/strings
/usr/share/octave/3.2.3/m/elfun
/usr/share/octave/3.2.3/m/set
```

Depending your system configuration you may get a different output. The first path, denoted by dot (`.`) denotes your current **working directory**. The current working directory is always the first directory Octave searches for script or function files. To see your current working directory, use the command `pwd` (Print Working Directory), and to see the contents of your current working directory type `ls` (LiSt). If you want to change the current working directory use the command `cd` (Change Directory). Finally, if you want to add a directory to your search path, use the command `addpath`. For details on how to use these commands use the command `help`.

2.5 Octave Programming: `glpk` Wrapper

We have already seen how to program with Octave in the previous section, and we will see more Octave programming later. In this section we will further illustrate Octave programming by building a simple to use wrapper for the Octave's LP solver `glpk`.

Let us first explain how to use `glpk` (GNU Linear Programming Kit). The function `glpk` solves optimization problems of the type

$$z \;=\; [\max \text{ or } \min]\;\; \mathbf{c}'\mathbf{x}$$

subject to

$$\begin{aligned} \mathbf{A}\mathbf{x} \;\; [= \text{ or } \leq \text{ or } \geq] \;\; \mathbf{b}, \\ \mathbf{lb} \;\leq\; \mathbf{x} \;\leq\; \mathbf{ub}. \end{aligned}$$

The input parameters for `glpk` are:

`c` A column vector containing the objective function coefficients.

`A` A matrix containing the constraints coefficients.

`b` A column vector containing the right-hand side value for each constraint in the constraint matrix.

`lb` A vector containing the lower bound on each of the variables.

`ub` A vector containing the upper bound on each of the variables. If `ub` is not supplied, the default upper bound is assumed to be infinite.

`ctype` An array of characters containing the sense of each constraint in the constraint matrix.

- `"F"` A free (unbounded) constraint (the constraint is ignored).
- `"U"` An inequality constraint with an upper bound ($\mathbf{A}_{i\bullet}\mathbf{x} \leq b_i$).
- `"S"` An equality constraint: $\mathbf{A}_{i\bullet}\mathbf{x} = b_i$.
- `"L"` A lower bound constraint: $\mathbf{A}_{i\bullet}\mathbf{x} \geq b_i$.
- `"D"` Both upper and lower bound constraints: $-b_i \leq \mathbf{A}_{i\bullet}\mathbf{x} \leq b_i$.

`vartype` A column vector containing the types of the variables.

- `"C"` A continuous variable.
- `"I"` An integer variable.

`sense` If `sense` is 1, the problem is a minimization. If `sense` is -1, the problem is a maximization. The default value is 1.

`param` A structure containing parameters used to define the behavior of solver. The most useful fields for us are:

- `msglev` This controls the level of (error) messages `glpk` gives. Setting `param.msglev=0` suppresses all messages and `param.msglev=2` gives a lot of information. The default, `param.msglev=1` will only give error messages.
- `presol` This controls whether `glpk` will use a presolver or not. The default, `param.presol=1`, is to use a presolver. Sometimes it is better not to use a presolver, i.e., to set `param.presol=0`.

The return values of `glpk` are:

`xopt` The value of the decision variables at the optimum.

`fmin` The optimal value of the objective function.

`status` Status of the optimization. E.g.:

- `180` Solution is optimal.
- `182` Solution is infeasible.
- `183` Problem has no feasible solution.
- `184` Problem has no unbounded solution.
- `185` Solution status is undefined.

`extra` A data structure containing, e.g., `lambda` for shadow prices and `redcosts` for reduced costs.

Now the code

```
octave:1> c = [3,2]';
octave:2> A = [2,1; 1,1; 1,0];
octave:3> b = [100,80,40]';
octave:4> [x_max,z_max] = glpk(c,A,b,[0,0]',[],"UUU","CC",-1)
x_max =

   20
   60

z_max =  180
```

for Giapetto's problem 1.1.1 should be clear: The meaning of the parameters `c`, `A`, and `b` are obvious. The column vector `[0 0]'` tells that both of the decision variables must be non-negative. The empty matrix `[]` tells that there are no a priori upper bounds for the decision variables. The string `"UUU"` tells that all the components of the vector `b` are upper bounds. The string `"CC"` tells that both the decision variables are continuous. (In Hlynur's problem 1.2.1 the string was `"II"` indicating that the decision variables are integers.) Finally, the sense parameter -1 indicates that we are dealing with a maximization problem. We have omitted the `param` parameter, so its default values will apply. Note also, that we have omitted the return values for `status` and `extra`. So, those values are lost to us.

As we see, the function `glpk` is quite versatile, maybe even too versatile. Therefore, we create a wrapper for it that is easier to use. The wrapper will only solve problems of the type

$$\begin{array}{lrcl} \max & z & = & \mathbf{c}'\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0}. \end{array}$$

The input parameters of the wrapper will be just `c`, `A`, and `b`. The return values will be `z_max` for the objective at optimum, `x_max` for the optimal decision, and the string `status` telling the status (obviously) of the offered solution.

Here is the wrapper. We call it `stu_lp_solver` (Simple To Use LP solver) and it is written in the file `stu_lp_solver.m`.

```
1 function [z_max, x_max, status] = stu_lp_solver(c, A, b)
2 ## Function [z_max, x_max, status] = stu_lp_solver(c, A, b) solves the LP
3 ##
4 ##     max  c'*x
5 ##     s.t.  A*x <= b
6 ##             x >= 0,
7 ##
8 ## by using glpk.
9 ##
```

```
10 ## Input:
11 ##
12 ## c, A, b
13 ##      The (column) vector defining the objective function, the
14 ##       technology matrix, and the (column) vector of the constraints.
15 ##
16 ## Output:
17 ##
18 ## z_max, x_max
19 ##      The maximal value of the objective and an optimal decision.
20 ##
21 ## status
22 ##      A string indicating the status of the solution:
23 ##     "bounded"
24 ##           A bounded solution was found.
25 ##      "unbounded"
26 ##           The solution is known to be unbounded.
27 ##      "infeasible"
28 ##           Solutions are known not to exist.
29 ##      "unknown"
30 ##           The algorithm failed.
31 ##
32 ## See also:  glpk.
33
```

Here, in the line 1 we have the standard beginning of a function file. The output and input variables are then explained in the following `help`-block in the lines 2–32, and the empty line 33 terminates the `help`-block.

```
34 ## Get m, the number of constraints (excluding the sign constraints), and
35 ## n, is the number of decision variables, from the technology matrix A.
36 [m,n] = size(A);
37
```

The line 36 gives short-hand for the dimensions of `A`. We could have used `rows(A)` for `m` and `columns(A)` for `n` everywhere, but this way the code is maybe easier to read, and a tiny bit faster.

```
38 ## Some input-checking:
39 if ( nargin!=3 )
40   error("stu_lp_solver: The number of input arguments must be 3.\n");
41 elseif ( columns(c)>1 )
42   error("stu_lp_solver: Objective c must be a column vector.\n");
43 elseif (columns(b)>1 )
44   error("stu_lp_solver: Upper bounds b must be a column vector.\n");
45 elseif (rows(c)!=n || rows(b)!=m )
46   error("stu_lp_solver: The dimensions of c, A, and b do not match.\n");
47 endif
48
```

The lines 38–48 check if the input provided by the user is correct. The main tools here are the `if`-block and the `error`-escape.

First, in the line 39 we check if there are 3 input arguments given, when the function is called. If this is not the case (`!=` is Octave for $\neq$) then the `error`-statement in the line 40 is executed: The string in the quotes is printed out, and, because of the newline character (`\n`), the function is exited immediately.

Next, if the condition in the line 39 was not satisfied, we check the condition `columns(c)>1` in the line 41. This condition simply checks if the input parameter `c` is a column vector. If it is not, then the function is exited immediately and the error message in the line 41 is printed out.

The lines 43–44 work just like the lines 41-42.

The lines 45–46 we check that the dimensions of `c`, `A` and `b` are consistent. Here (`||`) denotes the logical (short circuit) "or".

The line 47 marks the end of whole the `if`–`elseif`-block.

```
49 ## Set the parameters for glpk:
50
51 ## Set the decision-wise lower bounds all to zero, i.e. build a zero
52 ## column vector with n zeros.
53 o = zeros(n,1);
54
```

Here we simply set a short-hand notation: `o` is a column vector of zeros.

```
55 ## Set the sense of each constraint as an upper bound.
56 ctype = "";                                 # Start with an empty string.
57 for i=1:m                                   # Loop m times.
58   ctype = [ctype, "U"];                     # Append "U" to the string.
59 endfor
60
61 ## Set the type of each variable as continuous.
62 vtype = "";                                 # Start with an empty string.
63 for i=1:n                                   # Loop n times.
64   vtype = [vtype, "C"];                     # Append "C" to the string.
65 endfor
66
```

In the lines 56–59 we set the `ctype`-string to be `"CCC...C"` (m times): First, in the line 56 we initialize `ctype` to be an empty string. Then in the line 57 we form a `for`-loop that is run m time. Each iteration appends the letter `"C"` to the end of the string `ctype` in the line 58.

The lines 62–65 works the same way as the lines 56–59.

```
67 ## Solve the system by calling glpk.
68 [x_max, z_max, STATUS] = glpk(c, A, b, o, [], ctype, vtype, -1);
69
70 ## Set the STATUS code given by glpk to the appropriate string
71 status = "unknown";                          # Pessimism by default.
72 if ( STATUS==180 )                           # Everything went fine.
73   status="bounded";
74 elseif ( STATUS==183 )                       # LP infeasible detected.
75   status="infeasible";
76 elseif ( STATUS==184 )                       # LP unboundedness detected.
77   status="unbounded";
78 endif
79 endfunction
```

In the line 68 we call the function `glpk` with the appropriate parameters, and in the lines 71–78 we set the `status`-string according to the `STATUS`-code `glpk` gave us. It is worth noticing that, e.g., in the line 72 we ask if `STATUS` is `180` by the double equal sign. Writing `STATUS=180` would be wrong: that would assign the value `180` to the variable `STATUS`. Also, note that Octave is case sensitive: the variables `status` and `STATUS` are not the same.

2.6 Exercises and Projects

2.1. Calculate

(a) e^{2^8-256},
(b) $2^{64}-1$,
(c) $\sqrt{\frac{1}{8}e^3+\log\frac{15}{2}}-\sin\frac{7\pi}{2}$,
(d) $\log(-1)$.

2.2. Let

$$\mathbf{A}=\begin{bmatrix}5 & 2\\ 6 & -1\\ 1 & 5\end{bmatrix},\quad \mathbf{B}=\begin{bmatrix}5 & 0\\ 0 & 1\\ 1 & 1\end{bmatrix},\quad \mathbf{C}=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 1\\ 0 & 0 & 1\end{bmatrix}.$$

Calculate

(a) $\mathbf{C}^{-1}$,
(b) $\mathbf{B}'\mathbf{C}^{-1}$,
(c) $\mathbf{A}\mathbf{B}'\mathbf{C}^{-1}$,
(d) $(\mathbf{B}\mathbf{A}')'\mathbf{C}^{-1}$.

2.3. The function `npv` assumes that the rate of return `r` is constant over time. Modify the function so that is takes as an input variable a vector of rate of returns that change from one period to another.

2.4. Solve the LP

$$\begin{array}{lrcrcl} \max z & = & 4x_1 & + & 3x_2 & \\ \text{s.t.} & & 2x_1 & + & 3x_2 & \leq 6 \\ & & -3x_1 & + & 2x_2 & \leq 3 \\ & & & & 2x_2 & \leq 5 \\ & & 2x_1 & + & x_2 & \leq 4 \\ & & & & x_1, x_2 & \geq 0 \end{array}$$

(a) by using `glpk`,
(b) by using `stu_lp_solver`.

2.5. Solve the LP

$$\begin{array}{lrcrcr} \max z & = & 2x_1 & + & 9x_2 & \\ \text{s.t.} & & x_1 & + & 2x_2 & \leq 500 \\ & & x_1 & + & x_2 & \geq 100 \\ & & & & x_1, x_2 & \geq 0 \end{array}$$

(a) by using `glpk`,
(b) by using `stu_lp_solver`.

2.6. * Translate all the Octave codes in these notes into Matlab codes.

Part II

Linear Programming

Chapter 3

Linear Programs and Their Optima

> I'm sorry to say that the subject I most disliked was mathematics. I have thought about it. I think the reason was that mathematics leaves no room for argument. If you made a mistake, that was all there was to it. — Malcom X

> You have chosen the roughest road, but it leads straight to the hilltops. — John Bunyan

The aim of this chapter is to give a general picture of LPs and to prepare for the simplex method presented in Chapter 4. The key message of this chapter is the fundamental theorem of linear programming 3.2.2 which basically says that

> **The optimal solution of an LP is in one of the corners of the region of all feasible solutions**.

There are some theoretical results, i.e. theorems, in this chapter. The proofs of the theorems are deferred to the last section 3.5 (which can be omitted if time is scarce).

3.1 Form of Linear Program

A **linear optimization problem**, or a **linear program** (LP), is:

$$\begin{array}{lrcl} \max & z & = & \mathbf{c}'\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array} \tag{3.1.1}$$

The function $z = \mathbf{c}'\mathbf{x}$ is the **objective function**, the matrix $\mathbf{A}$ is called the **technology matrix**, the inequalities $\mathbf{A}_{i\bullet}\mathbf{x} \leq b_i$ are the **constraints**, and the

inequalities $x_i \geq 0$ are the **sign constraints**. The fact that the inequalities are inclusive — i.e., we have $\leq$ and $\geq$, and not $<$ and $>$ — is important, but subtle. It has to do with the difference between maximum and supremum, or minimum and infimum. The subtle-minded student should ask google about infimum and supremum. The simple-minded students should ignore the previous sentences.

The LP (3.1.1) is the **standard form** of an LP. In general, an LP may also be a minimization problem, may have no restriction in sign in some (or all) of its decision variables, and may have lower bound or equality constraints. Any LP can however be transformed into a standard form, e.g. by using the following algorithm:

3.1.2 Algorithm (Standard form transformation)**.**

Step 1: Change into maximization If the LP is a minimization problem, change it to a maximization problem by multiplying the objective vector $\mathbf{c}$ by -1:

$$\min\ z = \mathbf{c}'\mathbf{x} \quad \rightsquigarrow \quad \max\ -z = -\mathbf{c}'\mathbf{x}.$$

Step 2: Remove double inequalities If there are both lower and upper bounds in a single constraint, change that constraint into two constraints:

$$\begin{aligned} l_i \leq\ & A_{i1}x_1 + \cdots + A_{in}x_n \ \leq u_i \\ \rightsquigarrow\ & \begin{cases} l_i \leq\ A_{i1}x_1 + \cdots + A_{in}x_n \\ \qquad\quad A_{i1}x_1 + \cdots + A_{in}x_n \ \leq u_i \end{cases} . \end{aligned}$$

Note that equality is actually a double inequality. So, transform

$$\begin{aligned} & A_{i1}x_1 + \cdots + A_{in}x_n \ = \ b_i \\ =\ & b_i \leq\ A_{i1}x_1 + \cdots + A_{in}x_n \leq\ b_i \\ \rightsquigarrow\ & \begin{cases} b_i \leq\ A_{i1}x_1 + \cdots + A_{in}x_n \\ \qquad\quad A_{i1}x_1 + \cdots + A_{in}x_n \ \leq b_i \end{cases} . \end{aligned}$$

Step 3: Remove lower bounds If there is a lower bound constraint l_i, change it to an upper bound constraint by multiplying the corresponding inequality by -1:

$$l_i \ \leq\ A_{i1}x_1 + \cdots + A_{in}x_n \quad \rightsquigarrow \quad -A_{i1}x_1 - \cdots - A_{in}x_n \ \leq\ -l_i.$$

Step 4: Impose sign constraints by splitting the decision variables If the decision variable x_i is not restricted in sign to be positive, then replace it everywhere with $x_i = x_i^+ - x_i^-$ where $x_i^+, x_i^- \geq 0$ are now restricted in sign to be positive.

3.1.3 Example. Let us find the standard form of the LP

$$\begin{array}{rrcrcrcrl} & & \min z & = & -2x_1 + 3x_2 & & & & \\ \text{s.t.} & 1 & \leq & x_1 & + & x_2 & \leq & 9 & (1) \\ & & & 2x_1 & - & x_2 & \leq & 4 & (2) \\ & 2 & \leq & 7x_1 & + & x_2 & \leq & 100 & (3) \\ & & & & & x_1, x_2 & \geq & 0 & (4) \end{array}$$

Step 1: We turn the LP into a maximization problem, and get the objective

$$\max\ -z \ = \ 2x_1 - 3x_2.$$

Step 2: We remove the double inequalities (1) and (3). From the constraint (1) we get the constraints

$$\begin{array}{rcrcrcrl} 1 & \leq & x_1 & + & x_2 & & & (1.a) \\ & & x_1 & + & x_2 & \leq & 9 & (1.b) \end{array}$$

and from the constraint (3) we get the constraints

$$\begin{array}{rcrcrcrl} 2 & \leq & 7x_1 & + & x_2 & & & (3.a) \\ & & 7x_1 & + & x_2 & \leq & 100 & (3.b) \end{array}$$

Before going to Step 3 let us check the status of the LP now:

$$\begin{array}{rrcrcrcrl} & & \max\ -z & = & -2x_1 + 3x_2 & & & & \\ \text{s.t.} & 1 & \leq & x_1 & + & x_2 & & & (1.a) \\ & & & x_1 & + & x_2 & \leq & 9 & (1.b) \\ & & & 2x_1 & - & x_2 & \leq & 4 & (2) \\ & 2 & \leq & 7x_1 & + & x_2 & & & (3.a) \\ & & & 7x_1 & + & x_2 & \leq & 100 & (3.b) \\ & & & & & x_1, x_2 & \geq & 0 & (4) \end{array}$$

Step 3: We remove the lower bounds for the inequalities (1.a) and (3.a). We obtain the standard form

$$\begin{array}{rrcrcrcrl} & & \max\ -z & = & -2x_1 + 3x_2 & & & & \\ \text{s.t.} & & & -x_1 & - & x_2 & \leq & -1 & (1.a) \\ & & & x_1 & + & x_2 & \leq & 9 & (1.b) \\ & & & 2x_1 & - & x_2 & \leq & 4 & (2) \\ & & & -7x_1 & - & x_2 & \leq & -2 & (3.a) \\ & & & 7x_1 & + & x_2 & \leq & 100 & (3.b) \\ & & & & & x_1, x_2 & \geq & 0 & (4) \end{array}$$

There is no need for Step 4, as all the decision variables were constrained in sign.

Assumptions of Linear Programs

Formula (3.1.1) is the mathematical description of an LP. As such it is complete and perfect, as is the nature of mathematical formulas. It is also very laconic, as is also the nature of mathematical formulas. The list below explains the consequences — or assumptions, if you like — of (3.1.1) for the non-Spartans:

Proportionality The contribution to the objective function from each decision variable is proportional to the value of the decision variable: If, say, decision variable x_2 is increased by Δ then the value of objective function is increased by $c_2\Delta$. Similarly, the contribution of each decision variable in restrictions is also proportional to the value of the said variable. So, e.g., if you double the value of the decision variable x_2 the resources consumed by that decision will also double.

Additivity The contribution to the objective function for any variable is independent of the values of the other decision variables. For example, no matter what the value of x_1 is increasing x_2 to $x_2 + \Delta$ will increase the value of the objective function by $c_2\Delta$. Similarly, the resources used by decision x_2 will increase independently of the value of x_1.

Divisibility It is assumed that the decision variables can take fractional values. For example x_1 may be $3.1414936535\ldots$. This assumption is in many practical cases not true, but a reasonably good approximation of the reality. In case this assumption is violated, we have an integer program (IP) or a mixed integer–linear program (MILP).

3.2 Location of Linear Programs' Optima

The **feasible region** of an (standard form) LP

$$\begin{array}{lcrcl} \max z & = & \mathbf{c}'\mathbf{x} & & \\ \text{s.t.} & & \mathbf{A}\mathbf{x} & \leq & \mathbf{b} \\ & & \mathbf{x} & \geq & \mathbf{0} \end{array}$$

is the set of decisions $\mathbf{x}$ that satisfy the constraints $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$. A decision $\mathbf{x}$ in the feasible region is called, of course, a **feasible solution**.

Note that the feasible region is determined by the technology matrix $\mathbf{A}$ and the constraints $\mathbf{b}$. The objective $\mathbf{c}$ has no effect on the feasible region.

Consider a feasible solution, or a decision, $\mathbf{x}$ of an LP. The constraint b_i is **active** at the decision $\mathbf{x}$ if $\mathbf{A}_{i\bullet}\mathbf{x} = b_i$, i.e. the i^{th} inequality constraint $A_{i1}x_1 + \cdots + A_{in}x_n \leq b_i$ turns out to be the equality $A_{i1}x_1 + \cdots + A_{in}x_n = b_i$. Constraint i being active at decision $\mathbf{x}$ means that the resource i is fully consumed, or utilized, with decision $\mathbf{x}$. This means that there is no **slack** in the resource i.

A feasible solution $\mathbf{x} = [x_1 \ \cdots \ x_n]'$, or decision, of an LP is

Inner point if there are no active constraints at that decision,

Boundary point if there is at least one active constraints at that decision,

Corner point if there are at least n linearly independent active constraints at that decision. Corner points are also called **basic feasible solutions** (BFS).

In the above, **linear independence** means that the constraints are genuinely different. For example, the constraints

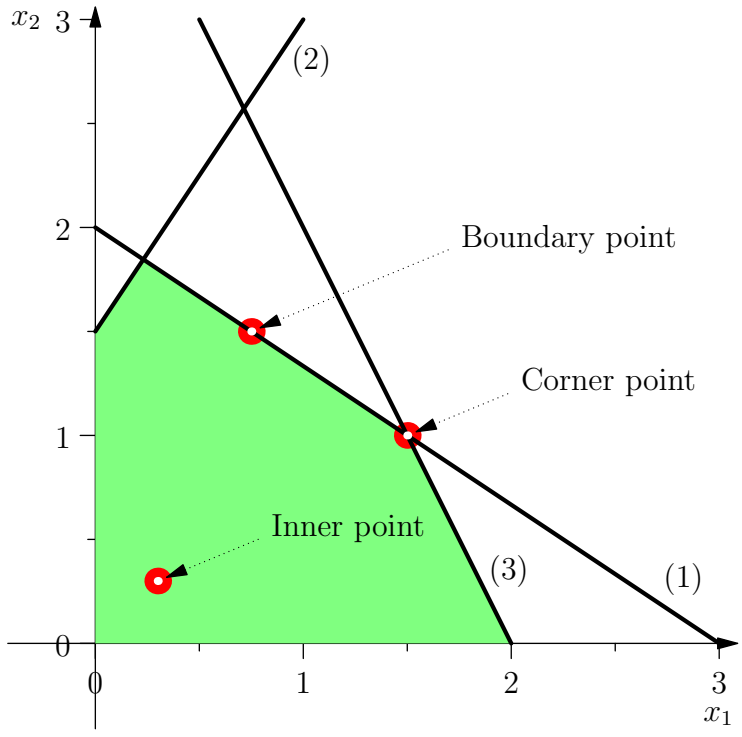
$$\begin{array}{rcrcl} 2x_1 & + & 3x_2 & \leq & 2 \\ x_1 & + & x_2 & \leq & 4 \end{array}$$

are linearly independent, but the constraints

$$\begin{array}{rcrcll} 2x_1 & + & 3x_2 & \leq & 2 & (3.2.1.a) \\ 6x_1 & + & 9x_2 & \leq & 6 & (3.2.1.b) \end{array} \tag{3.2.1}$$

are not. Indeed, the inequality $2x_1 + 3x_2 \leq 2$ is the same as the inequality $6x_1 + 9x_2 \leq 6$: the latter is simply the former multiplied by 3. So, the two constraints in (3.2.1) are actually only a one constraint, i.e. (3.2.1.a) and (3.2.1.b) are the same.

The next picture illustrates the discussion above. In the picture: None of the linearly independent constraints (1), (2), or (3) is active in the "Inner point". In the "Boundary point" one of the constraints, viz. (1), is active. In the "Corner point" two (which is the number of the decision variables) of the constraints, viz. (1) and (3), are active.



Here is the theorem that makes linear programming so easy (compared to non-linear programming):

3.2.2 Theorem (Fundamental theorem of linear programming)**.** *An optimal solution of an LP can be found, if it exists, in one of the corner points of the feasible region, i.e., an optimal solution is a BFS.*

By theorem 3.2.2 it seems that we have a very simple **Brutus Forcius's** (108–44 BC) algorithm for finding an optimum: Just check all the corners! And, indeed, this naïve approach works well with such petty examples we have in this course. The problem with this brute force approach in practice is a manifestation of the **combinatorial curse**: an LP with n decision variables and m constraints has

$$\binom{n+m}{m} = \frac{(n+m)!}{n!\,m!}$$

corners (in the slack form that the simplex algorithm will use). So, an LP with 15 decision variables and 15 constraints has

$$\binom{30}{15} = 155{,}117{,}520$$

corners. Suppose you have a computer that checks $1,000$ corners per second (this is pretty fast for today's computers, and right-out impossible if you program with Java™). Then it would take almost two days for the computer to check all the 155,117,520 corners. You may think this is not a problem: maybe two days is not such a long time, and a problem with 15 decision variables is way bigger than anything you would encounter in the real life anyway. Well, think again! Two days is a long time if you need to update your optimal solution in a changing environment of, say, a stock exchange, and LPs with at 15 decision variables are actually rather small. Indeed, let us be a bit more realistic now: Consider a stock broker who has 50 stocks in her stock (pun intended). Suppose the broker has 50 constraints in selecting the stocks for her portfolio (not unreasonable) and a supercomputer that checks 100 million corners per second (very optimistic, even if one does not program with Java™). Then checking all the corners to optimize the portfolio would take 6.89×10^{44} years. The author doubts that even the universe can wait that long!

The bottom line: **Checking all the corners would take too long**.

The idea of the simplex algorithm is that you do not check **all** the corners. This can be done by using the following rather general idea:

3.2.3 Algorithm (Guess–and–improve optimization)**.**

Meta-step 1 Start with some (feasible) solution candidate.

Meta-step 2 Check if the solution candidate is optimal, or at least good enough. If so, the algorithm terminates. Otherwise go to the next meta-step.

Meta-step 3 Choose a better solution candidate. Go back to meta-step 2.

In simplex algorithm the solution candidates are the corners of the feasible region, since by the fundamental theorem of linear programming 3.2.2, we know that, that is where the optimal solutions are. The better solution candidates are adjacent corners to the previously selected corner. One hopes that in moving from one corner to a next one, one hits an optimal corner pretty soon, so that one does not have to check all the corners.

To use the meta-algorithm 3.2.3 in the simplex case we have to:

- identify the corners analytically,
- know how to tell if a chosen corner is optimal,
- know how to go to the best (or at least to a better) adjacent corner.

Once the points raised above are solved we have a genuine algorithm.

Let us end this section by an example that shows how the brute force algorithm for solving LPs works (in case you are forced to use it). The example also illustrates how the corners of an LP are identified analytically, which is of course much more interesting, and infinitely more important, than the actual brute force algorithm.

Consider the LP

$$\begin{array}{lrcrcrcr} \max z & = & 3x_1 & + & 4x_2 & & & \\ \text{s.t.} & & x_1 & + & x_2 & \leq & 40 & \\ & & x_1 & + & 2x_2 & \leq & 60 & \\ & & & & x_1, x_2 & \geq & 0 & \end{array} \tag{3.2.4}$$

By introducing the so-called **slack variables** s_i, the LP (3.2.4) is equivalent to the **slack form** LP

$$\begin{array}{lrcrcrcrcr} \max z & = & 3x_1 & + & 4x_2 & & & & & \\ \text{s.t.} & & x_1 & + & x_2 & + & s_1 & & = & 40 \\ & & x_1 & + & 2x_2 & & & + \; s_2 & = & 60 \\ & & & & & & x_1, x_2, s_1, s_2 & \geq & & 0 \end{array}$$

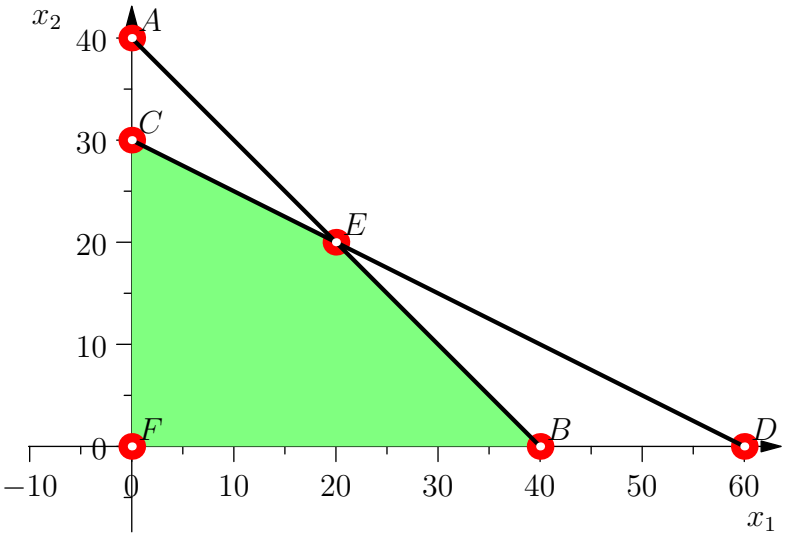
Now, the brute force method (and the simplex method, to some extent) is based on the following observation: Consider the constraints of an LP in the slack form. This is a linear system with m equations and $n+m$ unknowns: n actual decision variables and m slacks. Since $n+m > m$ this linear system is underdetermined. In principle, to solve a system of m equations requires only m variables. The remaining n variables can be set to zero. When solving an LP in slack form with m equations and $n+m$ unknowns, the m chosen variables used to solve the linear system are **basic variables** (BV), and the remaining n variables that are set to zero are called **non-basic variables** (NBV).

We choose successively $2 = m$ of the $4 = n-m$ variables x_1, x_2, s_1, s_2 to be our BVs and set the remaining $2 = n$ variables to be zero, or NBV, and solve the

constraint system. If the solution turns out to be feasible (it may not be since we are omitting the non-negativity constraints here) we check the value of the objective at this solution. Since we this way check all the BFSs of the system we must find the optimal value.

From the next table (and the accompanying picture) we read that the optimal decision is $x_1 = 20,\ x_2 = 20$ with the slacks $s_1 = 0,\ s_2 = 0$. The corresponding optimal value is $z = 140$.

BV	Linear system	x_1	x_2	s_1	s_2	BFS	z	Pt
s_1, s_2	$0 + 0 + s_1 = 40$ $0 + 0 + s_2 = 60$	0	0	40	60	Yes	0	F
x_2, s_2	$0 + x_2 + 0 = 40$ $0 + 2x_2 + s_2 = 60$	0	40	0	−20	No	–	A
x_2, s_1	$0 + x_2 + s_1 = 40$ $0 + 2x_2 + 0 = 60$	0	30	10	0	Yes	120	C
x_1, s_2	$x_1 + 0 + 0 = 40$ $x_1 + 0 + s_2 = 60$	40	0	0	20	Yes	120	B
x_1, s_1	$x_1 + 0 + s_1 = 40$ $x_1 + 0 + 0 = 60$	60	0	−20	0	No	–	D
x_1, x_2	$x_1 + x_2 + 0 = 40$ $x_1 + 2x_2 + 0 = 60$	20	20	0	0	Yes	140	E



3.3 Solution Possibilities of Linear Programs

Logically there are four possible cases that can happen with LPs: (1) there are no solutions, (2) there is a unique bounded solution, (3) there are many bounded solutions, and (4) there are unbounded solutions. There is of course, in principle, the possibility that the case (3) splits into two sub-possibilities: (3.a) there are only finitely many optimal bounded solutions and (2.b) there are infinitely many optimal bounded solutions. The sub-possibility (3.a) can not happen with LPs, however. The reason is that if $\mathbf{x}^*$ and $\mathbf{y}^*$ are any two optimal solutions then any convex combination $\alpha\mathbf{x}^* + (1-\alpha)\mathbf{y}^*$, $0 < \alpha < 1$, is also an optimal solution, i.e. the feasible region is convex.

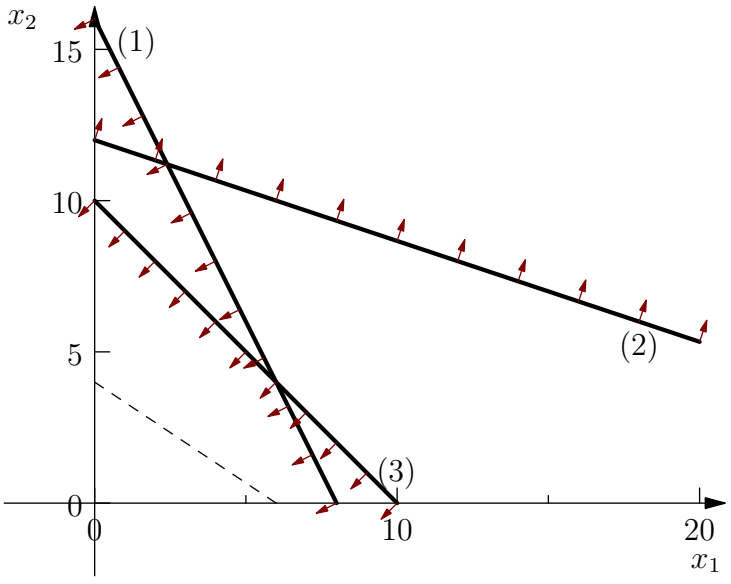
Next we illustrate the four possibilities graphically, and check what Octave's `glpk` says about them.

No Solutions

An LP can fail to admit optimal solutions if (and only if) it has no solutions at all:

$$
\begin{array}{llrcrcrl}
 & \max z & = & 2x_1 & + & 3x_2 & & \\
 & \text{s.t.} & & 0.5x_1 & + & 0.25x_2 & \leq & 4 \qquad (1) \\
(3.3.1) & & & x_1 & + & 3x_2 & \geq & 36 \qquad (2) \\
 & & & x_1 & + & x_2 & \leq & 10 \qquad (3) \\
 & & & & & x_1, x_2 & \geq & 0 \qquad (4)
\end{array}
$$

The graph below shows why there are no feasible solutions of the LP above. The the dashed line is an isoprofit line with $z = 12$.



Here is what `glpk` says about LP (3.3.1):

```
octave:1> c=[2 3]'; A=[0.5 0.25; 1 3; 1 1]; b=[4 36 10]';
octave:2> [x_max,z_max,STATUS] = glpk(c, A, b, [0 0]', [], "ULU", "CC", -1)
x_max =

    NA
    NA

z_max = NA
STATUS =  213
```

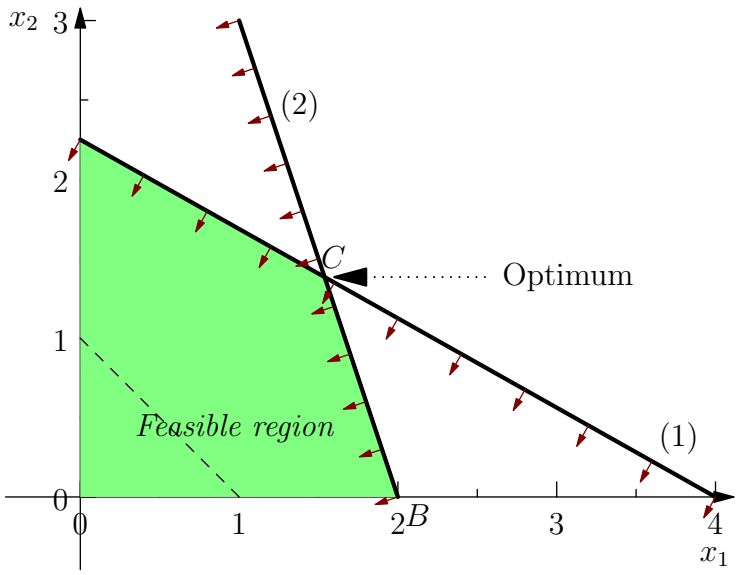
Here `NA` means "missing value". So, something went wrong. The status code `STATUS=213` says "No primal feasible solution (LP presolver)". So, `glpk` indicates that the LP (3.3.1) is infeasible.

Unique Solution

Consider the LP

$$
\begin{array}{lrcrcrcl}
\max z & = & x_1 & + & x_2 & & & \\
\text{s.t.} & & 2x_1 & + & 4x_2 & \leq & 9 & (1) \\
& & 3x_1 & + & x_2 & \leq & 6 & (2) \\
& & & & x_1, x_2 & \geq & 0 & (3)
\end{array}
\tag{3.3.2}
$$

The next picture shows it all: It is obvious, by moving the dashed isoprofit line ($z = 1$ in the picture) further away from the origin that the unique optimal point is C:



Here is what `glpk` says about LP (3.3.2):

```
octave:1> c=[1 1]'; A=[2 4; 3 1]; b=[9 6]';
octave:2> [x_max,z_max,STATUS] = glpk(c, A, b, [0 0]', [], "UU", "CC", -1)
x_max =

   1.5000
   1.5000

z_max =  3
STATUS =  180
```

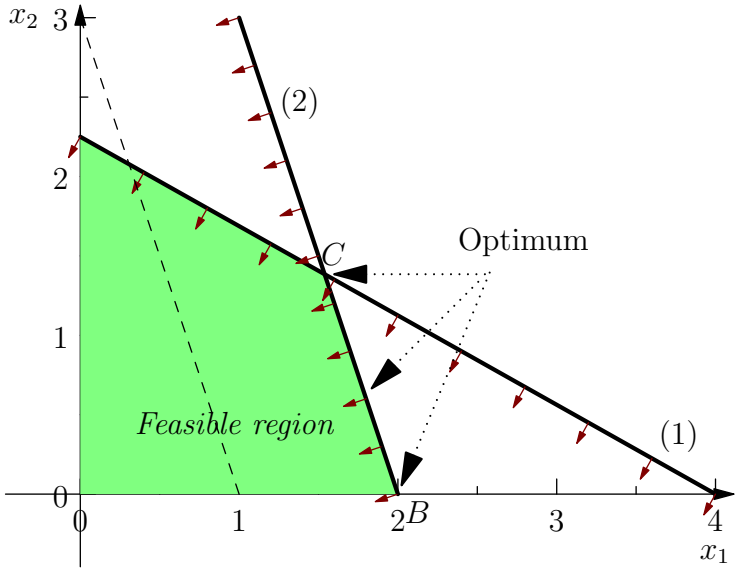
So, the optimal decision is $\mathbf{x} = [x_1\ x_2]' = [1.5\ 1.5]'$ with the optimal objective value $z = 3$. The status code `STATUS=180` means that a bounded solution was found.

Many Bounded Solutions

Change the objective of the unique solution LP (3.3.2) to $z = 6x_1 + 2x_2$. We get the LP

$$
\begin{array}{lrcrcll}
\max z & = & 6x_1 & + & 2x_2 & & \\
\text{s.t.} & & 2x_1 & + & 4x_2 & \leq\ 9 & (1) \\
& & 3x_1 & + & x_2 & \leq\ 6 & (2) \\
& & & & x_1, x_2 & \geq\ 0 & (3)
\end{array}
\tag{3.3.3}
$$

Now the feasible region of LP (3.3.3) and the unique solution LP (3.3.2) are the same, since the only thing that changed was the form of the objective function. But this change made the isoprofit lines parallel to the constraint (2). So, as the next picture illustrates, this makes all the points in the line segment from B to C optimal In the picture the dashed line is the isoprofit line $z = 6$.



Here is what `glpk` says about LP (3.3.3):

```
octave:1> c=[6 2]'; A=[2 4; 3 1]; b=[9 6]';
octave:2> [x_max,z_max,STATUS] = glpk(c, A, b, [0 0]', [], "UU", "CC", -1)
x_max =

   2
   0

z_max =  12
STATUS =  180
```

So, `glpk` gives **an** optimal decision and the optimal value of the objective function. Unfortunately, the `STATUS=180` simply means that the found solution is bounded and optimal: `glpk` does not recognize that there may be other optimal solutions also. The omitted output parameter `EXTRA` in its field `redcosts` for **reduced costs** would give an indication of multiple optimal solutions, however. We will learn about reduced costs later in Chapter 5.

Unbounded Solution

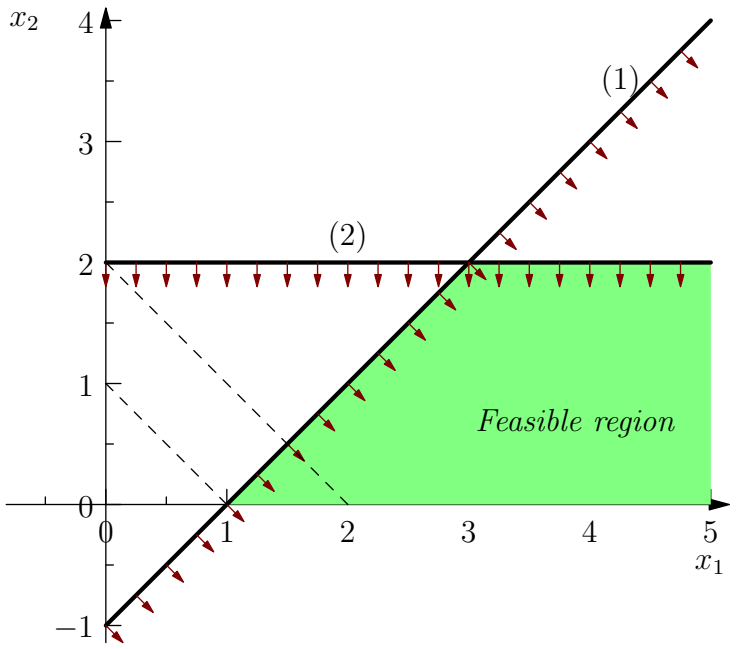
Unbounded solution means that one can always find better and better solutions to the problem so that there is no limit on how good the solution can be. In the case of a maximization problem one can say that the optimal value is $+\infty$, and in the case of a minimization problem one can say that the optimal value is $-\infty$.

The feasible region of an LP is unbounded if there exists at least one decision variable that can take arbitrarily big or small values. If the feasible region is not unbounded then it is bounded ☺. An LP may have unbounded solution (only) if its feasible region is unbounded. So, a bounded feasible region ensures a bounded solution, but an unbounded feasible region does not (necessarily) imply unbounded solution.

Consider the LP

$$
\begin{array}{lrcrcrcll}
\max z & = & x_1 & + & x_2 & & & & \\
\text{s.t.} & & x_1 & - & x_2 & \geq & 1 & & (1) \\
 & & & & 6x_2 & \geq & 2 & & (2) \\
 & & & & x_1, x_2 & \geq & 0 & & (3)
\end{array}
\tag{3.3.4}
$$

In the next picture we see that one finds better and better solutions as one moves the dashed isoprofit lines ($z = 1$ and $z = 2$ in the picture) further away from the origin. Note that one cannot increase the decision x_2 bigger than 2, but the possibility to increase x_1 without a limit is enough to make the solution unbounded.



Here is what `glpk` says about LP (3.3.4):

```
octave:1> c=[1 1]'; A=[1 -1; 0 6]; b=[1 2]';
octave:2> [x_max,z_max,STATUS] = glpk(c, A, b, [0 0]', [], "LL", "CC", -1)
x_max =

    NA
    NA

z_max = NA
STATUS =  214
```

So, `glpk` basically gives up: A lot of "Missing values" and the `status=214` means "No dual feasible solution (LP presolver)". The reason for this giving up is in the presolver that uses dual LPs. Indeed, an unbounded (primal) LP implies an infeasible dual LP. We will learn about primal and dual LPs in Chapter 5.

To fix the problem, one must turn off the presolver `glpk` uses. To turn the presolver off set the field `presol` of the structure `param` to `0`. I.e., issue `param.presol=0`, and then call `glpk` with the non-default parameter `param`. The next code illustrates how to do this:

```
octave:1> c=[1 1]'; A=[1 -1; 0 6]; b=[1 2]';
octave:2> param.presol=0;
octave:3> [x_max,z_max,STATUS] = glpk(c,A,b,[0 0]',[],"LL","CC",-1,param)
Scaling...
 A: min|aij| =  1.000e+00  max|aij| =  6.000e+00  ratio =  6.000e+00
Problem data seem to be well scaled
EQ: min|aij| =  1.000e+00  max|aij| =  1.000e+00  ratio =  1.000e+00
Crashing...
Size of triangular part = 2
x_max =

   1.33333
   0.33333

z_max =  1.6667
STATUS =  184
```

Now, `glpk` gives some error-type messages and the values of `x_max` and `z_max` are wrong. But never mind, the status code `STATUS=184` is correct: it means "Problem has unbounded solution", although there is an embarrassing typo in the `help glpk` in the explanation of the code `184`.

3.4 Karush–Kuhn–Tucker Conditions*

Sometimes one can make an educated guess about the optimal corner of an LP. In that case one asks if the guess is correct. The following Karush–Kuhn–Tucker theorem 3.4.1 provides a way to check the correctness of one's guess. It is also useful for checking the result a computer algorithm gives you, since computers are prone to rounding errors. The Karush–Kuhn–Tucker theorem is also used later in proving the validity optimality criteria of the simplex algorithm.

3.4.1 Theorem (Karush–Kuhn–Tucker theorem)**.** *Consider the LP*

$$\begin{array}{lcrcl} \max z & = & \mathbf{c}'\mathbf{x} & & \\ \text{s.t.} & & \mathbf{A}\mathbf{x} & \leq & \mathbf{b}\ . \\ & & \mathbf{x} & \geq & \mathbf{0} \end{array}$$

Let $\mathbf{x}$ *be a feasible solution to the LP. If there are vectors* $\mathbf{s}, \mathbf{u}, \mathbf{v}$ *such that*

(i) $\mathbf{A}\mathbf{x} + \mathbf{s} = \mathbf{b}$,
(ii) $\mathbf{c} = \mathbf{A}'\mathbf{v} - \mathbf{u}$,
(iii) $\mathbf{u}'\mathbf{x} + \mathbf{v}'\mathbf{s} = 0$,
(iv) $\mathbf{s}, \mathbf{u}, \mathbf{v} \geq \mathbf{0}$

then $\mathbf{x}$ *is an optimal solution to the LP.*

The vectors $\mathbf{s}$, $\mathbf{u}$, $\mathbf{v}$ in the Karush–Kuhn–Tucker theorem 3.4.1 have the following interpretation:

- $\mathbf{s}$ is the **slack** vector: s_i tells how much of the resource i is unused. If $s_i = 0$ then the constraint i is active, i.e., the resource i is completely used. This interpretation is obvious if you look condition (i) of Theorem 3.4.1.
- $\mathbf{u}$ is the **reduced cost** vector. It is connected to the sign constraint $\mathbf{x} \geq 0$ through the **complementary slackness** indicated in the condition (iii): if $x_i > 0$ then the i^{th} sign constraint is not active and $u_i = 0$.
- $\mathbf{v}$ is the **shadow price** vector. It is connected to the resource constraints, or the dual variables. Basically, condition (iii) states the complementary slackness that if there is a slack $s_i > 0$ in the resource i then $v_i = 0$.

The interpretations above are closely related to the sensitivity analysis and the duality of LPs studied in Chapter 5.

3.5 Proofs*

> The good Christian should beware of mathematicians and all those who make empty prophecies. The danger already exists that mathematicians have made a covenant with the devil to darken the spirit and confine man in the bonds of Hell.
>
> —St. Augustine.

Proof of Theorem 3.2.2. Let us first prove that an optimal value is found in the boundary of the feasible region.

This is a proof by contradiction: Suppose there is an optimal point $\mathbf{x}^*$ that is an inner point of the feasible region. Then, for a small enough r, all points that are not further away from $\mathbf{x}^*$ than the distance r belong to the feasible region. In particular, the point

$$\mathbf{w} \quad = \quad \mathbf{x}^* + \frac{r}{2}\frac{\mathbf{c}}{\|\mathbf{c}\|}$$

will belong to the feasible region. Here $\|\mathbf{c}\|$ denotes the Euclidean distance:

$$\|\mathbf{c}\| \quad = \quad \sqrt{\sum_{i=1}^{n} c_i^2},$$

and thus $\mathbf{c}/\|\mathbf{c}\|$ is a unit-length vector pointing at the same direction as $\mathbf{c}$.

Now, at point $\mathbf{w}$ we get for the objective function $\mathbf{c}'\mathbf{x}$ that

$$\mathbf{c}'\mathbf{w} \quad = \quad \mathbf{c}'\mathbf{x}^* + \frac{r}{2}\frac{\mathbf{c}'\mathbf{c}}{\|\mathbf{c}\|} \quad = \quad \mathbf{c}'\mathbf{x}^* + \frac{r}{2}\|\mathbf{c}\| \quad > \quad \mathbf{c}'\mathbf{x}^*, \tag{3.5.1}$$

since $\mathbf{c}'\mathbf{c} = \|\mathbf{c}\|^2$. But inequality (3.5.1) is a contradiction, since $\mathbf{x}^*$ was optimal. So the assumption that $\mathbf{x}^*$ was an inner point must be wrong.

Let us then prove that an optimal value is actually found in a corner of the feasible region. This part of the proof requires rather deep knowledge of linear algebra, and of linear spaces, although the idea itself is not so complicated if you can visualize n-dimensional spaces. (Taking $n = 3$ should give you the idea.)

Let $\mathbf{x}^*$ be an optimal solution, and let $z^* = \mathbf{c}'\mathbf{x}^*$ be the optimal value. We already know that $\mathbf{x}^*$ is in the boundary of the feasible region. So, at least one constraint is active. Let now V be the subspace of $\mathbb{R}^n$ spanned by the active constraints at the point $\mathbf{x}^*$. Let k be the dimension of V. If $k = n$, then $\mathbf{x}^*$ is already a corner point, and we are done. Suppose then that $k < n$. Then V is a proper subspace or $\mathbb{R}^n$ and any vector in $\mathbb{R}^n$ can be written as an orthogonal sum of a vector from the subspace V and a vector from the orthogonal complement $V^\perp$. Let us write the vector $\mathbf{c}$ this way: $\mathbf{c} = \mathbf{c}_V + \mathbf{c}_{V^\perp}$.

Next we show that $\mathbf{c}$ belongs to the subspace V, i.e., $\mathbf{c}_{V^\perp} = \mathbf{0}$. (This claim is the n-dimensional analogue of the 2-dimensional case, where the isoprofit line must be parallel to a constraint line, if the optimal solution lies in the said line.) Suppose the contrary: $\mathbf{c}_{V^\perp} \neq \mathbf{0}$. Now, there is a small $\epsilon > 0$ such that $\mathbf{x}^+ = \mathbf{x}^* + \epsilon\mathbf{c}_{V^\perp}$ is a feasible solution. Indeed, $\mathbf{x}^+$ is obtained by moving a small amount along an active constraint line (or surface). This is possible in principle, since we are not in a corner point. But now

$$\begin{aligned} z^+ &= \mathbf{c}'\mathbf{x}^+ \\ &= \mathbf{c}'\mathbf{x}^* + \epsilon\mathbf{c}'\mathbf{c}_{V^\perp} \\ &= z^* + \epsilon\mathbf{c}_V'\mathbf{c}_{V^\perp} + \epsilon\mathbf{c}_{V^\perp}'\mathbf{c}_{V^\perp} \\ &= z^* + \epsilon\|\mathbf{c}_{V^\perp}\|^2 \\ &> z^*, \end{aligned}$$

which is a contradiction, since z^* was the optimal value.

Since $k < n$ there is a non-zero point $\mathbf{w}$ in $V^\perp$ such that $\tilde{\mathbf{x}} = \mathbf{x}^* + \alpha\mathbf{w}$ is feasible when $\alpha > 0$ is small enought, and not feasible when $\alpha > 0$ is too large. Now, let α be just small enough for $\mathbf{x}$ to be feasible. Then at point $\tilde{\mathbf{x}}$ at least one more constraint will become active. So, the space $\tilde{V}$ associated to the point $\tilde{\mathbf{x}}$ has at least the dimension $k+1$. Moreover, the point $\tilde{\mathbf{x}}$ is at least as good as the point $\mathbf{x}^*$, since

$$\begin{aligned} \tilde{z} &= \mathbf{c}'\tilde{\mathbf{x}} \\ &= \mathbf{c}'(\mathbf{x}^* + \alpha\mathbf{w}) \\ &= z^* + \alpha\mathbf{c}'\mathbf{w} \\ &= z^*. \end{aligned}$$

Here we used the fact that $\mathbf{c}$ belongs to V, i.e. $\mathbf{c}'\mathbf{w} = 0$ for all $\mathbf{w} \in V^\perp$.

Now, $\tilde{\mathbf{x}}$ is "closer to a corner" than $\mathbf{x}^*$, since it has $k+1$ active constraints. By taking $\tilde{\mathbf{x}}$ to be the new $\mathbf{x}^*$ and repeating the procedure described above $n-k-1$ times we will find an optimal solution in a corner. □

Proof of Theorem 3.4.1. Let $\mathbf{y}$ be some BFS of the LP. Theorem 3.4.1 is proved if we can show that $\mathbf{c}'\mathbf{y} \leq \mathbf{c}'\mathbf{x}$.

Now, since $\mathbf{y}$ is feasible there is $\mathbf{t} \geq \mathbf{0}$ such that $\mathbf{A}\mathbf{y} + \mathbf{t} = \mathbf{b}$. Denote $\mathbf{w} = \mathbf{x} - \mathbf{y}$. Then $\mathbf{A}\mathbf{w} = \mathbf{s} - \mathbf{t}$, and

$$\begin{aligned}
\mathbf{c}'\mathbf{y} &= \mathbf{c}'(\mathbf{x} + \mathbf{w}) \\
&= \mathbf{c}'\mathbf{x} + \mathbf{c}'\mathbf{w} \\
&= \mathbf{c}'\mathbf{x} + \mathbf{v}'\mathbf{A}\mathbf{w} - \mathbf{u}'\mathbf{w} \\
&= \mathbf{c}'\mathbf{x} + \mathbf{v}'\mathbf{s} - \mathbf{v}'\mathbf{t} - \mathbf{u}'\mathbf{w} \\
&= \mathbf{c}'\mathbf{x} + \mathbf{v}'\mathbf{s} - \mathbf{v}'\mathbf{t} - \mathbf{u}'\mathbf{y} + \mathbf{u}'\mathbf{x} \\
&= \mathbf{c}'\mathbf{x} - \mathbf{v}'\mathbf{t} - \mathbf{u}'\mathbf{y} \\
&\leq \mathbf{c}'\mathbf{x}.
\end{aligned}$$

So, $\mathbf{x}$ was indeed optimal. □

3.6 Exercises and Projects

3.1. Find the

(a) standard form,
(b) slack form

of the LP

$$\begin{array}{rrrcrcrcr}
 & \min z & = & -2x_1 & + & 3x_2 & & & \\
\text{s.t.} & 1 & \leq & x_1 & + & x_2 & \leq & 9 \\
 & & & 2x_1 & - & x_2 & \leq & 4 \\
 & 2 & \leq & 7x_1 & + & x_2 & \leq & 100 \\
 & & & & & x_2 & \geq & 0
\end{array}$$

You can also make an Octave program that constructs the standard and slack forms to you automatically.

3.2. Solve the LP of Exercise 3.1 by checking all the corners (of its slack form). Checking all the corners may be hard work. So, if you are lazy and clever, you can make an Octave program that checks the corners for you.

3.3. Solve the LP of Exercise 3.1 with

(a) `stu_lp_solver`,
(b) `glpk`.

3.4. Find **all** the optima of the LP

$$\begin{aligned} \max z &= -3x_1 + 6x_2 \\ \text{s.t.} \quad 5x_1 + 7x_2 &\leq 35 \\ -x_1 + 2x_2 &\leq 2 \\ x_1, x_2 &\geq 0 \end{aligned}$$

3.5. * Make an Octave function that takes in a general form LP

$$\begin{aligned} \max \text{ or } \min z &= \mathbf{c}'\mathbf{x} \\ \text{s.t.} \quad \mathbf{l} \leq \mathbf{A}\mathbf{x} &\leq \mathbf{u} \\ x_i &\geq 0 \text{ for all } i \in I, \end{aligned}$$

where I is a given set of the decision variables, and transforms it into a standard form. So, the function returns the parameters $\mathbf{c}$, $\mathbf{A}$, and $\mathbf{b}$ of the standard form LP. The input parameters could be t for type (min or max), $\mathbf{l}$, $\mathbf{A}$, $\mathbf{u}$, and I.

Chapter 4

Simplex Algorithm

> A simple test of if you understand something is to try to teach it to someone. The real test is to try to teach it to a computer, for computers cannot be intimidated.
> — Anonymous (of course)

> The question of whether computers can think is like the question of whether submarines can swim.
> — Edsger W. Dijkstra

> Programmers are in a race with the Universe to create bigger and better idiot-proof programs, while the Universe is trying to create bigger and better idiots. So far the Universe is winning.
> — Rich Cook

We will explain the simplex algorithm by implementing it with Octave. Of course, there is no practical need for this, since the simplex algorithm is already implemented, in a much better way, in the Octave function `glpk`. Our implementation consists of the main function `simplex_lp_solver` and the auxiliary functions `first_simplex_tableau`, `new_simplex_tableau`, and `is_optimal_simplex_tableau`. We will also elaborate the simplex algorithm with the following manual example:

4.0.1 Example (Manuel's Problem)**.** Manuel Eixample produces and sells three different labels of ice cream: Aragon, Castile and Catalonia. To produce one liter of each label Manuel needs egg, milk and sugar as follows:

	Product		
Resource	Aragon	Castile	Catalonia
Egg	8 liters	6 liters	1 liter
Milk	4 liters	2 liters	1.5 liters
Sugar	2 liters	1.5 liters	0.5 liters

Manuel has bought 48 liters of egg, 20 liters of milk, and 8 liters of sugar. A liter of Aragon sells for €60, a liter of Castile for €30, and a liter of Catalonia for €20. The demand for Aragon and Catalonia is unlimited, but at most 5 liters of Castile can be sold.

Manuel wants to maximize total revenue.

As a modeling problem Manuel's problem 4.0.1 is very similar to Giapetto'sproblem 1.1.1. After making some comparisons on how we modeled Giapetto's problem 1.1.1 we notice that we should set the decision variables to

$$\begin{aligned} x_1 &= \text{liters of Aragon produced} \\ x_2 &= \text{liters of Castile produced} \\ x_3 &= \text{liters of Catalonia produced} \end{aligned}$$

and that Manuel should solve the following LP:

$$\begin{array}{lrcrcrcrl} \max z & = & 60x_1 & + & 30x_2 & + & 20x_3 & & \text{(revenue)} \\ \text{s.t.} & & 8x_1 & + & 6x_2 & + & x_3 & \leq & 48 \quad \text{(egg)} \\ & & 4x_1 & + & 2x_2 & + & 1.5x_3 & \leq & 20 \quad \text{(milk)} \\ & & 2x_1 & + & 1.5x_2 & + & 0.5x_3 & \leq & 8 \quad \text{(sugar)} \\ & & & & x_2 & & & \leq & 5 \quad \text{(Castile demand)} \\ & & & & & & x_1, x_2, x_3 & \geq & 0 \end{array} \tag{4.0.2}$$

Before going any further let us admit honestly that there will be a huge hiccup with our simplex algorithm: It will only solve LPs of the form

$$\begin{array}{lrcl} \max & z & = & \mathbf{c}'\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{4.0.3}$$

where $\mathbf{b} \geq \mathbf{0}$: There are LPs our algorithm cannot handle! The sign constraint $\mathbf{x} \geq \mathbf{0}$ or the max-assumption are no problems because any LP can be transformed into a standard form. The assumption $\mathbf{b} \geq \mathbf{0}$ is the hiccup! Indeed, e.g. the LP

$$\begin{array}{lrcrcrl} \max z & = & 3x_1 & + & 2x_2 & & \\ \text{s.t.} & & 2x_1 & + & x_2 & \leq & -10 \\ & & x_1 & + & x_2 & \geq & -80 \\ & & & & x_1, x_2 & \geq & 0 \end{array}$$

cannot be transformed into the form (4.0.3). The problem is the first inequality. The second inequality is fine, since it can be multiplied by -1 on the both sides to get the desired form

$$-x_1 - x_2 \leq 80.$$

The general case, where $\mathbf{b}$ may have negative components requires more sophisticated algorithms, e.g. like the ones used by `glpk`.

4.1 Simplex tableaux and General Idea

The simplex algorithm expresses the LP as a matrix called the **simplex tableau**. The first simplex tableau of the LP 4.0.3 is the augmented matrix

$$\mathbf{T} = \left[\begin{array}{ccc|c} 1 & -\mathbf{c}' & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{A} & \mathbf{I} & \mathbf{b} \end{array}\right]. \tag{4.1.1}$$

The idea is that the standard form (4.0.3) is equivalent to the **slack form**

$$\begin{array}{lrcl} \max & z & & \\ \text{s.t.} & z - \mathbf{c}'\mathbf{x} & = & 0, \\ & \mathbf{A}\mathbf{x} + \mathbf{s} & = & \mathbf{b}, \\ & \mathbf{x}, \mathbf{s} & \geq & \mathbf{0}. \end{array} \tag{4.1.2}$$

Here $\mathbf{s}$ is the vector of the **slack variables**: s_i tells how much there is slack in the i^{th} constraint $\mathbf{A}_{i\bullet}\mathbf{x} \leq b_i$. In other words, s_i tells how much of the resource i is unused. Since we know, by Theorem 3.2.2, that an optimum of an LP is in a corner of the feasible region, it follows that a solution of the LP (4.1.2) must be a BFS of the linear system

$$\begin{bmatrix} 1 & -\mathbf{c}' & \mathbf{0} \\ \mathbf{0} & \mathbf{A} & \mathbf{I} \end{bmatrix} \begin{bmatrix} z \\ \mathbf{x} \\ \mathbf{s} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{b} \end{bmatrix}.$$

So, the first simplex tableau comes from this system.

The first simplex tableau (4.1.1) is already solved if you take the slacks to be the BV. This is so because there is an identity matrix $\mathbf{I}$ corresponding to the slack columns. So, in the first simplex tableau the decision variables are all set to zero, which is hardly an optimal BFS, since you would just be slacking off ☺.

Let us build the first simplex tableau for our manual example 4.0.1. We start by transforming the Manuel's LP (4.0.2) into a slack form:

$$\begin{array}{lrcrcrcrcrcrcl} \max z = & 60x_1 & + & 30x_2 & + & 20x_3 & & & & & & & & \\ \text{s.t.} & 8x_1 & + & 6x_2 & + & x_3 & + & s_1 & & & & & = & 48 \\ & 4x_1 & + & 2x_2 & + & 1.5x_3 & & & + & s_2 & & & = & 20 \\ & 2x_1 & + & 1.5x_2 & + & 0.5x_3 & & & & & + & s_3 & = & 8 \\ & & & x_2 & & & & & & & + & s_4 & = & 5 \\ & & & & & & & & x_1, x_2, x_3, s_1, s_2, s_3, s_4 & & & & \geq & 0 \end{array}$$

Taking s_1, s_2, s_3, s_4 to be our first BV our first Simplex tableau for Manuel is:

Row	z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	BV	RHS
1	1	−60	−30	−20	0	0	0	0	$z =$	0
2	0	8	6	1	1	0	0	0	$s_1 =$	48
3	0	4	2	1.5	0	1	0	0	$s_2 =$	20
4	0	2	1.5	0.5	0	0	1	0	$s_3 =$	8
5	0	0	1	0	0	0	0	1	$s_4 =$	5

In a very small nut-shell the simplex algorithm works then as follows:

If the simplex algorithm identifies the current simplex tableau to be solved with an optimal BFS there is nothing to be done any more, and the optimal solution

can be read from the tableau with the given BV corresponding the BFS. If the current BFS is not optimal the simplex algorithm starts to look for a better BFS by changing the BV and then solving the tableau with respect to the new, hopefully better, BV. This is repeated until an optimal solution is found (if ever).

Let us explain the simplex method in a bit more detail with some matrix gymnastics:

We write the (first) simplex tableau (4.1.1) so that the variables (decisions or slacks) that are BV are before the variables (decisions or slacks) that are NBV. This may require interchanging some columns of the augmented matrix, but that is not a problem: Augmented matrices remain equivalent under column-switches, if you just keep track which column means which variable. (The equivalence of augmented matrices mean that they represent the same system of linear equations. I.e. augmented matrix $[\mathbf{A}|\mathbf{b}]$ is equivalent to the augmented matrix $[\tilde{\mathbf{A}}|\tilde{\mathbf{b}}]$ if the linear systems $\mathbf{A}\mathbf{x} = \mathbf{b}$ and $\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ have the same solutions: $\mathbf{x} = \tilde{\mathbf{x}}$.) We also augment the objective coefficients for the slacks to be zero. Similarly, we augment $\mathbf{A}$ to $[\mathbf{A}\ \mathbf{I}]$. We denote by $\mathbf{c_{BV}}$ the coefficients corresponding to BV and by $\mathbf{c_{NBV}}$ the coefficients corresponding to NBV. Similarly, we denote by $\mathbf{B}$ the columns of the augmented $\mathbf{A}$ corresponding to BV and by $\mathbf{N}$ the columns of the augmented $\mathbf{A}$ corresponding to NBV. Finally, we shall use the obvious notations $\mathbf{x_{BV}}$ and $\mathbf{x_{NBV}}$ for the BV and NBV parts of the augmented decision-and-slacks vector $\mathbf{x}$. With this notation we can write (4.1.1) as

$$\mathbf{T} = \left[\begin{array}{c|cc|c} 1 & -\mathbf{c}'_{\mathbf{BV}} & -\mathbf{c}'_{\mathbf{NBV}} & 0 \\ \mathbf{0} & \mathbf{B} & \mathbf{N} & \mathbf{b} \end{array}\right]. \tag{4.1.3}$$

So, if this is the first simplex tableau, we have just switched the decision variables and the slacks around. Also, the tableau is solved, since $\mathbf{B} = \mathbf{I}$ and $\mathbf{c_{BV}} = \mathbf{0}$.

Now, the simplex algorithm starts changing simplex tableaux into new simplex tableaux so that all the tableaux are always equivalent. Suppose we change the BV to a new one. In our notation it means that the columns of (4.1.3) have changed. So, $\mathbf{B}$ is no longer the identity matrix $\mathbf{I}$. In order to solve the tableau (4.1.3) we simply multiply the lower block of the tableau from the left with the inverse of $\mathbf{B}$. This way we get the identity matrix $\mathbf{B}^{-1}\mathbf{B} = \mathbf{I}$. We obtain the tableau

$$\mathbf{T} = \left[\begin{array}{c|cc|c} 1 & -\mathbf{c}'_{\mathbf{BV}} & -\mathbf{c}'_{\mathbf{NBV}} & 0 \\ \mathbf{0} & \mathbf{I} & \mathbf{B}^{-1}\mathbf{N} & \mathbf{B}^{-1}\mathbf{b} \end{array}\right]. \tag{4.1.4}$$

But (4.1.4) is not solved for the new BV. Indeed, there may be non-zero coefficients in $-\mathbf{c_{BV}}$. So, we must eliminate $-\mathbf{c_{BV}}$ from the first row. This can be done by adding the lower block multiplied by $\mathbf{c}'_{\mathbf{BV}}$ to the the first row. After a bit of reasonably tedious matrix algebra we obtain

$$\mathbf{T} = \left[\begin{array}{cc|c|c} 1 & \mathbf{0} & \mathbf{c}'_{\mathbf{BV}}\mathbf{B}^{-1}\mathbf{N} - \mathbf{c}'_{\mathbf{NBV}} & \mathbf{c}'_{\mathbf{BV}}\mathbf{B}^{-1}\mathbf{b} \\ \mathbf{0} & \mathbf{I} & \mathbf{B}^{-1}\mathbf{N} & \mathbf{B}^{-1}\mathbf{b} \end{array}\right]. \tag{4.1.5}$$

Now it is important to note that given any BV the form of the solved simplex tableau is always given by the formula (4.1.5). So, we are left with two problems:

1. How to identify an optimal simplex tableau?
2. How to change a non-optimal BV into a better one?

How to solve the problems 1–2 and to carry out the simplex idea explained above with Octave is the content of the rest of this chapter.

4.2 Top-Level Algorithm

In the previous section we already saw how the simplex algorithm works in the general level, or top-level, if you like. This general level is implemented in the Octave function `simplex_lp_solver`. In this section we go through this top-level code line-by-line. The code is in the file `simplex_lp_solver.m`. The line numbers are not part of the code (they never are). They are for reference purposes only.

```
1 function [z_max, x_max, status] = simplex_lp_solver(c, A, b, maxiter=100)
```

The file starts by the keyword `function` following with the return values `z_max`, `x_max`, and `status` of the function. Then comes the name `simplex_lp_solver` of the function, and the input parameters `c`, `A`, `b`, and `maxiter` of the function. The input value `maxiter` is given a default value `100`, that is used if the user does not provide a different value for it. The meanings of the input and output variables are given in the help block in the lines 11–34.

```
 2 ## Function [z_max, x_max, status] =
 3 ## simplex_lp_solver (c, A, b, maxiter=100) solves the LP
 4 ##
 5 ##     max  c'*x
 6 ##     s.t.  A*x <= b
 7 ##             x >= 0,
 8 ##
 9 ## where b>=0, by using the standard simplex algorithm.
10 ##
11 ## Input:
12 ##
13 ## c, A, b
14 ##      The (column) vector defining the objective function, the
15 ##       technology matrix, and the (column) vector of the constraints.
16 ##
17 ## maxiter
18 ##      The maximum iterations used in finding a solution (default: 100).
19 ##
```

```
20 ## Output:
21 ##
22 ## z_max, x_max
23 ##      The maximal value of the objective and an optimal decision.
24 ##
25 ## status
26 ##      A string indicating the status of the solution:
27 ##     "bounded"
28 ##           A bounded solution was found.
29 ##      "unbounded"
30 ##           The solution is known to be unbounded.
31 ##      "infeasible"
32 ##           Solutions are known not to exist (not applicable for b>=0).
33 ##      "unknown"
34 ##           The algorithm failed.
35 ##
36 ## simplex_lp_solver uses the functions  first_simplex_tableau,
37 ## new_simplex_tableau, and is_best_simplex_tableau.
38 ##
39 ## See also:  glpk.
40 ##
```

Lines 2–39 is a comment block that will be printed if the user asks for help by typing `help simplex_lp_solver`. The empty line 40 terminates the help block so that the following line 41 will not be printed when `help` is called for.

```
41 ## Get the first simplex tableau.
42 [T,BV] = first_simplex_tableau(c,A,b);
43
```

The lines 41–43 call the external function `first_simplex_tableau` that will build the first simplex tableau from the coefficient vector `c` determining objective function, the technology matrix `A`, and the upper bounds `b`. The simplex tableaux will be identified by the pair `[T,BV]`, where the matrix `T` is the simplex tableau without the knowledge of which of the variables are basic, and the vector `BV` consists of the indexes (not the values) of the basic variables. The function `first_simplex_tableau` will be defined later.

```
44 ## Look for better simplex tableaux, but avoid the infinite loop.
45 status = "unknown";                          # Status still unknown :)
46 iter = 0;                                    # No iterations yet.
47 while ( iter<maxiter && ...
48         !is_optimal_simplex_tableau(T,BV) && ...
49         !strcmp(status,"unbounded") )
50   [T,BV,status] = new_simplex_tableau(T,BV); # New [T,BV], unbounded (?).
51   iter = iter + 1;                          # We just had an iteration.
52 endwhile
53
```

The lines 44–52 are **the Beef** of the top-level algorithm!

In the line 45 the `status` of the LP is set to `"unknown"`, since we do not know its status yet ☺.

The variable `iter` introduced in the line 46 is simply a counter that is increased each time after a new simplex tableau is generated. If `iter` exceeds the input variable `maxiter` new simplex tableaux will not be generated. The variables `iter` and `maxiter` are needed to avoid a possibly infinite loop in the `while`-block in lines 47–52.

In the lines 47–49 an iteration `while`-loop starts if the number of iterations `iter` has not exceeded the number `maxiter` **and** we haven't found an optimal simplex tableau yet **and** we haven't found out that the problem has an unbounded solution. The external function `is_optimal_simplex_tableau` is used to check if the simplex tableau `[T,BV]` is optimal. We don't care at this point how this is done. The implementation of the function `is_optimal_simplex_tableau` will be explained later. There is a funny-looking statement `!strcmp(status,"unbounded")` in the line 49. It would be more intuitive to use the statement `status!="unbounded"`. That would be wrong, however. In Octave, to check if two strings are same one uses the function `strcmp` (string compare). The statement `status!="unbounded"` makes character-wise comparisons. It took a long time for the author to realize this ☹.

In the line 50 a new simplex tableau is generated from the old one by calling the external function `new_simplex_tableau`. Again, at this point we do not care how the function `new_simplex_tableau` generates a new simplex tableau. We only care that it will give us a new simplex tableau with associated BV, and change the status of the problem to `"unbounded"` if it finds out that the problem has unbounded solutions.

Finally, the line 51 increments the counter `iter` and the line 52 marks the end of the `while`-loop.

```
54 ## Exit (with zeros), if a solution was not found or found unbounded.
55 if ( iter>=maxiter || strcmp(status,"unbounded") )
56   z_max = 0;
57   x_max = zeros(length(c),1);
58   return;
59 endif
60
```

The lines 54–60 check if something went "wrong". Basically this means that either the status is still unknown, so that the algorithm was unable to find a solution in the `maxiter` amount of iterations, or the solution is known to be unbounded (the third alternative, infeasible, cannot happen in the simple simplex case). The `if`-block in the lines 55–59 is entered if either one of these situations occurred. If the `if`-block is entered, the the return values for `z_max` and `x_max` are set to zeros

(for no apparent reason ☺), and we exit the function at the `return`-statement in the line 58. The line 59 marks the end of the `if`-block.

```
61 ## Collect the results from the last simplex tableau.
62 status = "bounded";                          # We know this now.
63 z_max = T(1,columns(T));                     # z_max is in the NE corner.
64 x_max = zeros(length(c)+length(b),1);       # Zeros for now.
65 x_max(BV) = T(2:(length(b)+1),columns(T));  # Put BV values to x_max.
66 x_max = x_max(1:length(c));                 # Cut the slacks away.
67 endfunction
```

If we ever get to the line 62, i.e. we haven't exited the function in the line 58, we must have a bounded solution.

The value of the objective function is in the top-right corner of the simplex tableau `T`. It is retrieved to the output variable `z_max` in the line 63.

Next we retrieve the values of the decision variables. This is a bit tricky, since we only have the values of the BV in the last column of the simplex tableau `T`. So, in the line 64 we set the column vector `x_max` big enough for both the decisions and the slacks. The vector is initialized to zero. In the line 65 we set the BV values to their right places in the vector `x_max`. The values of the BV are found from the last column of the tableau `T` from the row 2 and the following rows. Since only the BV are non-zero the vector `x_max` now contains the correct values for all the decision and slack variables. In the line 66 we crop the slack variables away from the vector `x_max` so that only the decision variables remain.

Finally, the line 67 ends the `function`-block.

Although we do not yet have detailed knowledge of how the function `simplex_lp_solver` works, we use it to solve the Manuel's problem 4.0.1:

```
octave:1> c = [60; 30; 20];
octave:2> A = [ 8, 6, 1;
> 4, 2, 1.5;
> 2, 1.5, .5;
> 0, 1, 0];
octave:3> b = [48; 20; 8; 5];
octave:4> [z_max,x_max,status] = simplex_lp_solver(c,A,b)
z_max =  280
x_max =

   2
   0
   8

status = bounded
```

So, according to `simplex_lp_solver`, Manuel should produce 2 liters of Aragon, no Castile at all, and 8 liters of Catalonia. This way his revenue would be €280.

4.3 Initialization Algorithm

The simplex algorithm needs the first simplex tableau to start with. The idea how to construct it was already given in Section 4.1. The top-level function `simplex_lp_solver` explained in the previous Section 4.2 used the function `first_simplex_tableau` to construct it. Here we show the details of this function. The function is written in the m-file `first_simplex_tableau.m` (surprise, surprise).

```
1 function [T,BV] = first_simplex_tableau(c,A,b)
```

As always, the first line of an Octave function file starts with the keyword `function` followed by the output variables, the name, and the input variables of the function. The output and input variables should be familiar to you by now.

```
2 ## Function [T, BV] = first_simplex_tableau (c, A, b) builds the first
3 ## simplex tableau for the function simplex_lp_solver.  T is the tableau,
4 ## BV are the indexes of the basic variables (the slacks in this case).
```

In the lines 2–4 following the `function`-line there is, as usual, the comment block that will be printed if the user asks for help for the function. In this function we do not give much help. The reason is that this function is an auxiliary function used by the top-level function `simplex_lp_solver`. The idea is that the user would not call this function directly, although it is perfectly possible for the user to do so.

```
5
6 ## n is the number of decision variables, m is the number of constraints.
7 [m,n] = size(A);
8
```

The empty line 5 prevents the comment line 6 to be printed with the `help`-command. In the line 7 we get the number of decision variables and the number constraints from the dimensions of the technology matrix `A`. Indeed, the number of constraints is the number of rows of `A`, and the number of decision variables is the number of columns of `A`.

```
 9 ## The simplex tableau without the BV information.
10 T = [          1  -c'  zeros(1,m)  0;
11       zeros(m,1)    A      eye(m)  b];
12
13 ## The indexes of the BV's (the slacks in this case).
14 BV = ( (n+1):(n+m) )';
15 endfunction
```

Finally, the simplex tableau `T` is set in the lines 10–11. Compare this with the equation (4.1.1).

In the line 14 the indexes of the first BV are collected to the vector `BV`. Since the first BV are the slacks, and the m slacks are listed after the n decisions, we set `BV` to be $[n{+}1\ n{+}2\ \cdots\ n{+}m]'$.

Finally the line 15 ends the `function`-block.

If you wish, you can check the first simplex tableau in the manual example 4.0.1:

```
octave:1> c = [60; 30; 20];
octave:2> A = [ 8, 6, 1;
> 4, 2, 1.5;
> 2, 1.5, .5;
> 0, 1, 0];
octave:3> b = [48; 20; 8; 5];
octave:4> [T,BV] = first_simplex_tableau(c,A,b)
T =

 Columns 1 through 5:

    1.00000  -60.00000  -30.00000  -20.00000    0.00000
    0.00000    8.00000    6.00000    1.00000    1.00000
    0.00000    4.00000    2.00000    1.50000    0.00000
    0.00000    2.00000    1.50000    0.50000    0.00000
    0.00000    0.00000    1.00000    0.00000    0.00000

 Columns 6 through 9:

    0.00000    0.00000    0.00000    0.00000
    0.00000    0.00000    0.00000   48.00000
    1.00000    0.00000    0.00000   20.00000
    0.00000    1.00000    0.00000    8.00000
    0.00000    0.00000    1.00000    5.00000

BV =

   4
   5
   6
   7
```

So this is the Octave's version of the tableau

Row	z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	BV	RHS
1	1	−60	−30	−20	0	0	0	0	$z =$	0
2	0	8	6	1	1	0	0	0	$s_1 =$	48
3	0	4	2	1.5	0	1	0	0	$s_2 =$	20
4	0	2	1.5	0.5	0	0	1	0	$s_3 =$	8
5	0	0	1	0	0	0	0	1	$s_4 =$	5

4.4 Optimality-Checking Algorithm

The top-level function `simplex_lp_solver` explained in Section 4.2 used the function `is_optimal_simplex_tableau` to check whether a given tableau `[T,BV]` is optimal or not. The criterion for optimality is:

> **The simplex tableau $[\mathbf{T}, \mathbf{BV}]$ is optimal if there are no negative coefficients in the first row corresponding to the NBV-variables.**

We will not prove the validity of the optimality criterion here. The interested reader finds the proof in the end of this section.

Before going into the Octave implementation, let us consider the manual example 4.0.1 first. In Manuel's first simplex tableau we have $\mathbf{BV} = [s_1\ s_2\ s_3\ s_4]'$ and the first row of the first simplex tableau is

Row	z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	BV	RHS
1	1	−60	−30	−20	0	0	0	0	$z =$	0

So, according to the optimality criterion, this tableau is not optimal. Indeed, there is e.g. -60 for the NBV x_1. Indeed, all the NBV have negative coefficients.

Let us then consider `is_optimal_simplex_tableau` defined in the m-file `is_optimal_simplex_tableau.m`.

```
1 function optimal = is_optimal_simplex_tableau(T,BV)
2 ## Function optimal = is_optimal_simplex_tableau (T,BV) tells (to the
3 ## function simplex_lp_solver) if the simplex tableau [T, BV] is optimal.
4
```

As always, the m-file begins with the declaration of the output variables, name, and the input variables with the functions in the line 1, after which there is a comment block for the `help`-command in the lines 2–3. The empty line 4 is there to break the comment block.

```
 5 ## The tableau [T,BV] is optimal if in the first row all the
 6 ## coefficients are non-negative for all Non-Basic Variables (NBV).
 7 NBV = get_NBV(BV,columns(T)-2);  # Get NBV (cf. the function below).
 8 optimal = 0;                    # A priori assume non-optimality.
 9 NBVval = T(1,NBV+1);            # First column is for z, hence NBV+1.
10 if ( all(NBVval>=0) )           # Check for optimality.
11  optimal = 1;
12 endif
13 endfunction
14
```

We need to check the sign of all the NBV in the first row of the tableau `T`.

In the line 7 the indexes of the NBV are retrieved from the BV and the dimension of the decisions-and-slacks vector. This is done by using the auxiliary function `get_NBV` that is defined below.

In the line 8 we assume that the simplex tableau is not optimal: We will change the optimality status later, if optimality is detected.

In the line 9, the vector `NBVval` is set to contain the values (not indexes) of the NBV.

The `if`-block in the lines 10–12 checks the actual optimality: The condition `all(NBVval>=0)` is true if and only if all the values of the NBV are non-negative. If this is the case the `if`-block is entered, where the optimality status is set to 1 (i.e. true).

Finally, the line 13 ends the main `function`-block.

```
15 ##############################################################################
16 ## Auxiliary function to get NBV indexes from BV indexes.
17 ##############################################################################
18
19 function NBV = get_NBV(BV,nvar)
20 vars = ones(nvar,1);            # Set "true" to all indexes.
21 vars(BV) = 0;                   # Set "false" to BV indexes.
22 NBV = find(vars);               # Get the "true" indexes.
23 endfunction
```

The lines 19–23 define an internal auxiliary function for the main function `is_optimal_simplex_tableau`. Since it is defined in the same m-file as the main function, a user cannot call it directly. Indeed, only the main function can call it.

The line 19 marks the beginning of the `function`-block in the normal way. The input parameters for the function `get_NBV` are `BV`, the indexes of the BV, and `nvar`, the total number of decision variables and slacks.

The line 20 initializes the vector `vars` that will be a vector of zeros (for false) and ones (for true) indicating if a given variable is NBV or not. It is set to be vectors of ones to begin with. So, at first we assume that all the variables are NBV.

In the line 21 the indexes corresponding to BV in the vector `vars` are set to zero. So, now the vector `vars` has 1 at NBV indexes and 0 at BV indexes.

In the line 22 the vector of NBV indexes is set to be the indexes of the non-zero (true) values of the vector `vars`. This is done by the Octave function `find`. Now the vector `NBV` contains the indexes of the NBV and we are done. So, the `function`-block ends in the following line 23.

Proof of Optimality Criterion*

We prove the optimality criterion by using the Karush–Kuhn–Tucker theorem 3.4.1. The proof requires some matrix-gymnastics that can be rather tedious. But it can also be rewarding if you have the stamina for it.

Let $\mathbf{x}(d)$ and $\mathbf{x}(s)$ denote the decision part and the slack part, respectively, of the augmented decision vector $\mathbf{x} = [\mathbf{x}'_{\mathbf{BV}}\ \mathbf{x}'_{\mathbf{NBV}}]'$. Let $\mathbf{c}(d)$ be the vector of the original objective coefficient. We choose $\mathbf{u}$ to be the coefficients of the decision variables in the first row in the (last) simplex tableau, and $\mathbf{v}$ to be the coefficients of the slack variables in the first row in the (last) simplex tableau.

According to Karush–Kuhn–Tucker theorem 3.4.1 we need to show that

(i) $\mathbf{A}\mathbf{x}(d) + \mathbf{x}(s) = \mathbf{b}$,
(ii) $\mathbf{c}(d) = \mathbf{A}'\mathbf{v} - \mathbf{u}$,
(iii) $\mathbf{u}'\mathbf{x}(d) + \mathbf{v}'\mathbf{x}(s) = 0$,
(iv) $\mathbf{x}(s), \mathbf{u}, \mathbf{v} \geq \mathbf{0}$.

when $\mathbf{c}'_{\mathbf{BV}}\mathbf{B}^{-1}\mathbf{N} - \mathbf{c}'_{\mathbf{NBV}} \geq \mathbf{0}$, i.e. all the NBV have non-negative coefficients in the first row of the generic tableau

$$\mathbf{T} = \left[\begin{array}{cc|c|c} 1 & \mathbf{0} & \mathbf{c}'_{\mathbf{BV}}\mathbf{B}^{-1}\mathbf{N} - \mathbf{c}'_{\mathbf{NBV}} & \mathbf{c}'_{\mathbf{BV}}\mathbf{B}^{-1}\mathbf{b} \\ \mathbf{0} & \mathbf{I} & \mathbf{B}^{-1}\mathbf{N} & \mathbf{B}^{-1}\mathbf{b} \end{array}\right]. \tag{4.4.1}$$

Condition (i) is obviously satisfied.

Condition (iv) is almost obviously satisfied. The only thing that needs justification is that $\mathbf{u}, \mathbf{v} \geq \mathbf{0}$. But this our optimality criterion together with the fact the BV coefficients are zero in the first row. So, (iv) is indeed satisfied.

Condition (iii) also holds. Indeed, if $x(d)_j$ is BV then $u_j = 0$, since BV must have zero coefficient in the first row in a solved simplex tableau. On the other hand, if $x(d)_j$ is NBV it is zero. So, in any case, $u_j x(d)_j = 0$ for all j. Consequently $\mathbf{u}'\mathbf{x}(d) = 0$. The same argument shows that $\mathbf{v}'\mathbf{x}(s) = 0$.

Condition (ii) can be proven by showing that

$$(\mathbf{A}'\mathbf{v} - \mathbf{u})'\mathbf{x}(d) = \mathbf{c}(d)'\mathbf{x}(d) \tag{4.4.2}$$

for **all** decisions $\mathbf{x}(d)$. Now, by (i) and (iii), which hold for **all** decisions $\mathbf{x}(d)$ and slacks $\mathbf{x}(s)$, the LHS of (4.4.2) is

$$\begin{aligned}(\mathbf{A}'\mathbf{v}-\mathbf{u})'\mathbf{x}(d) &= \mathbf{v}'\mathbf{A}\mathbf{x}(d)-\mathbf{u}'\mathbf{x}(d)\\ &= \mathbf{v}'(\mathbf{b}-\mathbf{x}(s))-\mathbf{u}'\mathbf{x}(d)\\ &= \mathbf{v}'\mathbf{b}-\mathbf{v}'\mathbf{x}(s)-\mathbf{u}'\mathbf{x}(d)\\ &= \mathbf{v}'\mathbf{b}.\end{aligned}$$

On the other hand, the RHS of (4.4.2) is, by the first row of the generic simplex tableau (4.4.1),

$$\mathbf{c}(d)'\mathbf{x}(d) \;=\; z \;=\; \mathbf{c}'_{\mathbf{BV}}\mathbf{B}^{-1}\mathbf{b}.$$

So, our problem becomes showing that

$$\mathbf{v}'\mathbf{b} \;=\; \mathbf{c}'_{\mathbf{BV}}\mathbf{B}^{-1}\mathbf{b},$$

which is true, since (after suitable augmentation or deaugmentation)

$$\mathbf{v}' \;=\; \mathbf{c}'_{\mathbf{BV}}\mathbf{B}^{-1}, \tag{4.4.3}$$

a fact that may not be immediately obvious, but we shall come back to it in the next chapter

We have proven the optimality criterion to be valid. □

4.5 Tableau Improvement Algorithm

The improvement routing is a very critical part of the simplex algorithm. Indeed, so far the algorithm just starts with one BFS and checks its optimality. If the improvement routine that gives the next BFS is not good, we might end up looking for new BFSs for a very long time, or even forever! (This may still happen with the improvement routine we present here, but the probability of such a bad luck is tiny.)

The general idea of the improvement routine is to find the "best" variable in the NBV and the "worst" variable in the BV and then make a switch.

Let us explain how this work with the manual example 4.0.1.

Recall that we have the first simplex tableau

Row	z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	BV	RHS
1	1	−60	−30	−20	0	0	0	0	$z=$	0
2	0	8	6	1	1	0	0	0	$s_1=$	48
3	0	4	2	1.5	0	1	0	0	$s_2=$	20
4	0	2	1.5	0.5	0	0	1	0	$s_3=$	8
5	0	0	1	0	0	0	0	1	$s_4=$	5

which we found to be non-optimal since there are negative coefficients in the first row for an NBV. So, we have to change the BV.

> **The entering variable is the one with the smallest coefficient in the first row.**

The idea is that this way (we hope) to increase the RHS of the first row as much and as fast as possible. So, we find that the variable x_1 will enter the BV since it has the smallest coefficient -60.

> **The leaving BV will the one associated to the row that wins the ratio test** (the smallest finite positive value is the winner)
>
> $$\frac{\text{RHS of row}}{\text{Coefficient of entering variable in row}}.$$

The idea of the ratio test is, that we shall increase the entering variable as much as possible. At some point the increasing of the entering variable will force one of the BVs to become zero. This BV will then leave. The ratio test picks up the row associated to the leaving variable.

The ratio test gives Manuel

$$\begin{array}{lrcll} \text{Row 2 limit in on } x_1 & = & 48/8 & = & 6 \\ \text{Row 3 limit in on } x_1 & = & 20/4 & = & 5 \\ \text{Row 4 limit in on } x_1 & = & 8/2 & = & 4 \\ \text{Row 5 limit in on } x_1 & = & 5/0 & = & \infty \quad \text{No limit.} \end{array}$$

So, Row 4 wins the ratio test. Hence s_3 is the leaving BV.

Now we have new BV: s_1, s_2, x_1, s_4 and an unsolved simplex tableau

Row	z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	BV	RHS
1	1	-60	-30	-20	0	0	0	0	$z =$	0
2	0	8	6	1	1	0	0	0	$s_1 =$	48
3	0	4	2	1.5	0	1	0	0	$s_2 =$	20
4	0	2	1.5	0.5	0	0	1	0	$x_1 =$	8
5	0	0	1	0	0	0	0	1	$s_4 =$	5

Now we have to solve this Simplex tableau in terms of the BV. This means that each row must have coefficient 1 for its BV, and that BV must have coefficient 0 on the other rows. This can be done by pivoting with respect to the BV that just entered. After some tedious pivoting we get the solved simplex tableau

Row	z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	BV	RHS
1	1	0	15	-5	0	0	30	0	$z =$	240
2	0	0	0	-1	1	0	-4	0	$s_1 =$	16
3	0	0	-1	0.5	0	1	-2	0	$s_2 =$	4
4	0	1	0.75	0.25	0	0	0.5	0	$x_1 =$	4
5	0	0	1	0	0	0	0	1	$s_4 =$	5

This tableau is not yet optimal, since the NBV x_3 has negative coefficient -5. It is obvious that x_3 enters the BV. To find out the leaving variable we perform the ratio test:

$$\begin{array}{lrcll} \text{Row 2 limit in on } x_3 & = & 16/(-1) & = & -16 \quad \text{No limit} \\ \text{Row 3 limit in on } x_3 & = & 4/0.5 & = & 8 \\ \text{Row 4 limit in on } x_3 & = & 4/0.25 & = & 16 \\ \text{Row 5 limit in on } x_3 & = & 5/0 & = & \infty \quad \text{No limit} \end{array}$$

So, row 3 wins the ratio test. Since s_2 is the BV of row 3, s_2 will leave. So, we have the following unsolved simplex tableau

Row	z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	BV	RHS
1	1	0	15	-5	0	0	30	0	$z =$	240
2	0	0	0	-1	1	0	-4	0	$s_1 =$	16
3	0	0	-1	0.5	0	1	-2	0	$x_3 =$	4
4	0	1	0.75	0.25	0	0	0.5	0	$x_1 =$	4
5	0	0	1	0	0	0	0	1	$s_4 =$	5

which can be solved easily (but tediously) enough by pivoting. We obtain

Row	z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	BV	RHS
1	1	0	5	0	0	10	10	0	$z =$	280
2	0	0	-2	0	1	2	-8	0	$s_1 =$	24
3	0	0	-2	1	0	2	-4	0	$x_3 =$	8
4	0	1	1.25	0	0	-0.5	1.5	0	$x_1 =$	2
5	0	0	1	0	0	0	0	1	$s_4 =$	5

and note that the tableau is indeed optimal! No need to continue table-dancing!

Finally, let us interpret the result: The liters of Aragon, Castile, and Catalonia Manuel should produce are 2, 0, and 8. With this decision Manuel's revenue is €280.

Let us then consider the Octave implementation of the improvement routine, i.e. the function file `new_simplex_tableau.m`:

```
 1 function [Tnew,BVnew,status] = new_simplex_tableau(T,BV)
 2 ## Function [Tnew, BVnew, status] = new_simplex_tableau (T, BV) builds
 3 ## a new, hopefully better, simplex tableau [Tnew, BVnew] from the
 4 ## tableau [T, BV], or detects the solution to be unbounded.
 5 ## This function is used by the function simplex_lp_solver.
 6
```

The line 1 defines, as always, the output variables, the name, and the input variables of the function. The input variables are the old simplex tableau `[T,BV]`, and the output variables are the new simplex tableau `[Tnew,BVnew]`, and the string `status`. The string `status` will be `"unknown"` by default, but may change to `"unbounded"`, if an unbounded solution is detected.

The lines 2–5 consist of a very brief comment block for the `help`-command, and the empty line 6 terminates the comment block.

```
 7 ## Variable initializations and short-hand notations.
 8 status = "unknown";               # Paranoia!
 9 Tnew = T; BVnew = BV;             # New tableau is old tableau (for now).
10 [m,n] = size(T);                  # Almost the dimensions of A.
11
```

The line 8 should not be needed, as the status is unknown by default. But it's better to be paranoid than sorry. The line 9 sets the new simplex tableau to be the old one at this point. Later changes are made to the new tableau. The line 10 is simply a short-hand notation.

```
12 ## Get the entering BV.
13 coeff = T(1,2:(n-1));             # Coeffs. of the decisions and slacks.
14 [tmp,in] = min(coeff);            # Index of the entering coefficient.
15
```

The line 13 gets the coefficients of the decision and slack variables from the first row of the simplex tableau. The line 14 finds the value and the place of the minimal coefficient from the coefficient vector `coeff`. The value (for which we have no need) is stored in `tmp` and the index (for which we are interested in) is stored in `in`, which is the entering variable.

```
16 ## Get the leaving BV, or detect unboundedness and leave the function.
17 rhs = T(2:m,n);                   # RHS of the constraint rows.
18 cin = T(2:m,in+1);                # Coeffs. of the entering variables.
19 ratio = rhs./cin;                 # Pointwise ratios.
20
```

```
21 ## Set all the "no-limit" ratios to -infinity
22 for i=1:length(ratio)
23   if ( ratio(i)==Inf || ratio(i)<0 )
24     ratio(i) = -Inf;
25   endif
26 endfor
27
28 ## Check boundedness. Exit if unbounded.
29 if ( all(ratio<0) )
30   status = "unbounded";
31   return;
32 endif
33
34 [tmp,out] = min(abs(ratio));      # Get the winner index.
35
```

Here we find the leaving variable `out` by using the ratio test, and detect possible unboundedness.

In the line 17 the RHS of the constraint rows of the tableau are stored in the vector `rhs`. In the line 18 the coefficients of the entering variable `in` in the constraint rows are stored in the variable `cin` (coefficient of in). In the line 19 the pointwise ratio vector `ratio` is build. Note that the pointwise division operator is dot-slash (`./`), not slash (`/`).

The lines 22–26 are an ugly hack. Here we set the "no-limit" ratios to $-\infty$. The `for`-loop in the lines 22–26 goes through all the elements of the vector `ratio` and sets negative and $+\infty$ elements to $-\infty$.

The `if`-block in the lines 29–32 checks if the problem is unbounded, which happens if all the ratios are negative (actually $-\infty$ because of our ugly hack). If this happens the status of the problem is set unbounded in the line 30, and the function exits in the line 31.

In the line 34 the leaving variable `out` is set to be the index of the minimal (in absolute value) coefficient in the ratio vector. The variable `tmp` contains the actual (absolute) value, for which we have absolutely no use at all.

```
36 ## Solve T the new BV.
37 BVnew(out) = in;                  # The new BV.
38 Tnew = pivot(T,out+1,in+1);       # Solve T for the new BV by pivoting.
39 endfunction
```

The line 37 switches the entering and the leaving variable in the BV. The line 38 solves the new simplex tableau with respect to the new BV by using the auxiliary function `pivot` defined below. Finally, the line 39 ends the `function`-block.

```
40
41 ##############################################################################
42 ## The auxiliary function pivot
43 ##############################################################################
44
45 function Ap = pivot(A, r, c)
46 ## Function Ap = pivot (A, r, c) pivots A with row r and column c.
47
48 A(r,:) = A(r,:)/A(r,c);           # Get 1 for the pivot in the pivot row.
49 for i = 1:rows(A)                 # Remove the pivot from other rows.
50   if (i != r)
51     A(i,:) = A(i,:) - A(i,c)*A(r,:);
52   endif
53 endfor
54 Ap = A;                           # Return the pivoted matrix Ap.
55 endfunction
```

The lines 41–55 define the internal auxiliary function `pivot`. Since pivoting should be known by everyone from the elementary linear algebra courses, we do not waste valuable paper by explaining how the function works.

4.6 Exercises and Projects

4.1. Solve the Giapetto's problem 1.1.1 by using `simplex_lp_solver`.

4.2. Solve the LP

$$\begin{array}{lrcrcrcr} \max z & = & x_1 & + & 2x_2 & & & \\ \text{s.t.} & & x_1 & + & x_2 & \leq & 6 \\ & & x_1 & + & 4x_2 & \leq & 20 \\ & & & & x_1, x_2 & \geq & 0 \end{array}$$

(a) by using `simplex_lp_solver`,
(b) by using `glpk`.

4.3. Consider the LP

$$\begin{array}{lrcrcrcr} \max z & = & 4x_1 & + & 7x_2 & & & \\ \text{s.t.} & & -x_1 & + & 3x_2 & \leq & 5 \\ & & -6x_1 & - & 6x_2 & \leq & -3 \\ & & & & x_1, x_2 & \geq & 0 \end{array}$$

(a) Can this LP be solved by `simplex_lp_solver`?
(b) Solve the LP by using `glpk`. Note that you may have to switch of the `glpk`'s presolver. Type `help glpk` to find out how to do this.

4.4. Don Guido has peculiar taste: He likes a mixture of Monte Conero and Pinot Grigio (yuck). His "rosé" must have at least 20% of both grapes. Monte Conero

costs €2 per liter and Pinot Grigio costs €3 per liter. Don Guido wants to mix 100 liters of his rosé for the upcoming wedding of his daughter (poor guests). He has infinite supply of Monte Conero, but only 50 liters of Pinot Grigio at his disposal. Also, he has promised to buy at least 10 liters of Monte Conero from his cousin Don Luigi.

How should Don Guido mix his rosé to make it as cheap as possible?

You can find the answer any way you want, e.g. by using `glpk` or `simplex_lp_solver` (if it works), or by simply deducing the obvious (and miss the joy of modeling the problem as an LP).

4.5. `simplex_lp_solver` assumes that the LP is a maximization problem. Modify it so that it takes an additional input value indicating whether the problem is maximization or minimization, and then solves the problem.

4.6. Find out an LP that is in principle within the scope of the simplex algorithm implemented in `simplex_lp_solver`, i.e. $\mathbf{b} \geq \mathbf{0}$, but for which `simplex_lp_solver` fails anyway. Can you find the problem with `simplex_lp_solver` and fix it?

Note that there are many different solutions to this exercise. The main point of this exercise is to show that the function `simlplex_lp_solver` does not meet "industry standards".

4.7. * The function `simplex_lp_solver` has no input-checking whatsoever. Modify the function, and if necessary its auxiliary functions, to check that the input parameters are correct. If the input parameters are not correct the function should return informative error messages.

Chapter 5

Sensitivity and Duality

> Measure with a micrometer. Mark with chalk. Cut with an axe.
> — Ray's rule of precision

> In the physical world, one cannot increase the size or quantity of anything without changing its quality. Similar figures exist only in pure geometry. — Paul Valéry

> The qualitative vs. quantitative dichotomy is bollocks: Consider yourself in a cage with a cat. The quantitative difference is that the cat weighs either 3 kg or 300 kg.
> — Anonymous

5.1 Sensitivity Analysis

What and Why is Sensitivity Analysis

A significant problem with LPs (or with any other models, for that matter) is the assumption that the parameters $\mathbf{c}$, $\mathbf{A}$, and $\mathbf{b}$ in

$$\begin{array}{lrcl} \max & z & = & \mathbf{c}'\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array}$$

are known without any margin of error. In practice we only have a (good or bad) guess about the true values of the parameters. **Sensitivity analysis** is a systematic study of how, well, sensitive, the solutions of the LP are to **small changes** in the data. The key questions are

1. If the objective function $\mathbf{c}$ changes in its parameter c_i, how does the solution change?
2. If the resources available change, i.e., the constraint vector $\mathbf{b}$ change in its parameter b_i, how does the solution change?

Question 1 is related to the concept of **reduced cost** and question 2 is related to the concept of **shadow price**.

The brute force approach to these questions is to solve lots and lots of LPs: One LP for each change in the parameters. For example, in Giapetto's problem 1.1.1 there might be uncertainty in what is the actual market demand for soldiers. It was assumed to be 40, but it could be anything between, say, 30 and 50. We could then solve the Giapetto's LP separately for market demands $30, 31, \ldots, 49, 50$. So, we would solve 20 different LPs (21, actually, but who's counting). If it is also assumed that the the profit for soldiers might not be exactly €3 but could be anything between €2.5 and €3.5, then we could also solve the LP separately for profits €2.5, €2.6, ..., €3.4, €3.5. Combining this with the different LPs we got from the uncertainty in the market demand we would then have $20 \times 10 = 200$ different LPs to solve (well, $21 \times 11 = 231$ if you count correctly). This "checking the scenarios" method works, and it is indeed widely used in practice.

The brute force method explained above has at least three problems: (1) It is inelegant, (2) it would involve a large amount of calculations, and (3) it is hard to see what happens when more than two parameters change. These problems are, however, not critical. Indeed, (3) understanding high-dimensional spaces is always difficult, (2) solving hundreds of LPs is not that time-consuming with modern computers and efficient algorithms like the simplex, and (1) who cares about elegance these days? Nevertheless, we shall try to be at least a little bit elegant in this chapter.

Shadow Prices

> **The shadow price π_i of a constraint b_i is the amount that the objective function's value z at the optimum would change if the constraint b_i is changed by one unit — given that the optimal BV does not change.**

Note the clause "given that the optimal BV does not change". This means that the shadow price is valid for small changes in the constraints. If the optimal corner changes when a constraint is changed, then the interpretation of the shadow price is no longer valid.

Shadow prices are sometimes called **dual variables** or **marginal prices**. The name "marginal price" actually a much more informative name than the nebulous shadow price (or dual variable). Indeed, suppose you have a constraint that limits, say, the amount of labor available to 40 hours per week. Then the shadow price will tell you how much you would be willing to pay for an additional hour of labor. If your shadow price is €10 for the labor constraint, for instance, you should pay no more than €10 an hour for additional labor. Labor costs of less than €10 per hour will increase the objective value; labor costs of more than €10 per hour will

decrease the objective value. Labor costs of exactly €10 will cause the objective function value to remain the same.

If you like mathematical formulas — and even if you don't — the shadow prices can be defined as follows: Consider the LP

$$\begin{array}{lrcl} \max & z & = & \mathbf{c}'\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array}$$

The optimal solution z^* of this LP is a function of the objective vector $\mathbf{c}$, the technology matrix $\mathbf{A}$, and the constraint vector $\mathbf{b}$:

$$z^* \quad = \quad z^*(\mathbf{c}, \mathbf{A}, \mathbf{b}).$$

Then the shadow price π_i associated to the constraint b_i is the partial derivative

$$\pi_i \quad = \quad \frac{\partial z^*}{\partial b_i},$$

or, in vector form,

$$\pi \quad = \quad \frac{\partial z^*}{\partial \mathbf{b}},$$

where

$$\pi = \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_m \end{bmatrix} \quad \text{and} \quad \frac{\partial z^*}{\partial \mathbf{b}} = \begin{bmatrix} \frac{\partial z^*}{\partial b_1} \\ \vdots \\ \frac{\partial z^*}{\partial b_m} \end{bmatrix}.$$

Suppose now that $\epsilon = [\epsilon_1 \ \cdots \ \epsilon_m]'$ is a small vector, and

$$z^*_\epsilon \quad = \quad z^*(\mathbf{c}, \mathbf{A}, \mathbf{b} + \epsilon)$$

is the optimal value, when the constraints $\mathbf{b}$ are changed by ϵ. Then the first order approximation for the new optimal value is

$$\begin{aligned} z^*_\epsilon \quad &= \quad z^* + \epsilon' \frac{\partial z^*}{\partial \mathbf{b}} \\ &= \quad z^* + \epsilon' \pi. \end{aligned} \tag{5.1.1}$$

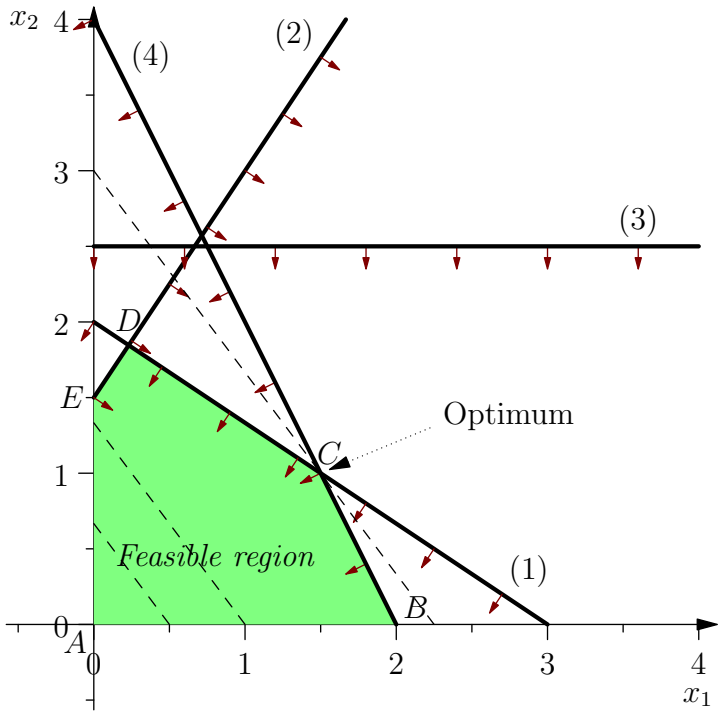
The equality (5.1.1) is valid as long as the elements ϵ_i of ϵ are small enough in absolute value. If some of the elements of ϵ are too big, then the equality (5.1.1) may fail to be true.

Let us see now how to use formula (5.1.1) in sensitivity analysis.

5.1.2 Example. Consider the LP

$$\begin{array}{lrcrcrcll} \max z & = & 4x_1 & + & 3x_2 & & & & \\ \text{s.t.} & & 2x_1 & + & 3x_2 & \leq & 6 & & (1) \\ & & -3x_1 & + & 2x_2 & \leq & 3 & & (2) \\ & & & & 2x_2 & \leq & 5 & & (3) \\ & & 2x_1 & + & x_2 & \leq & 4 & & (4) \\ & & & & x_1, x_2 & \geq & 0 & & \end{array}$$

Here is a picture representing the LP:



From the picture we read — by moving the isoprofit line (the dashed lines) away from the origin — that the optimal decision is at the point $C = (1.5, 1)$. Therefore, the optimal value is $z = 4{\times}1.5 + 3{\times}1 = 9$. We also see that the constraints (1) and (4) are active at the optimum. So, changing them should change the optimal value. Indeed, they should have positive shadow prices. In contrast, the constraints (2) and (3) are not active at the optimum. So, changing them — slightly — should have no effect on the optimum. So, both of them should have 0 as their shadow price.

Let us calculate the shadow prices of Example 5.1.2 with `glpk`. For this we use the output structure `extra` and its field `lambda` (for some reason `glpk` call shadow prices λ instead of π):

```
octave:1> c=[4;3]; A=[2 3;-3 2;0 2;2 1]; b=[6;3;5;4];
octave:2> [x_max,z_max,status,extra]=glpk(c,A,b,[0;0],[],"UUUU","CC",-1)
x_max =

   1.5000
   1.0000

z_max =  9
status =  180
extra =
{
  lambda =

     0.50000
     0.00000
    -0.00000
     1.50000

  redcosts =

     0
     0

  time = 0
  mem = 0
}
```

So, the shadow prices for the constraints (1) and (4) are 0.5 and 1.5, respectively. All other shadow prices are 0. So, the shadow price vector is

$$\pi = \begin{bmatrix} 0.5 \\ 0 \\ 0 \\ 1.5 \end{bmatrix}.$$

Let us then try to use formula (5.1.1) to see what happens if the constraints (1)–(4) change. Suppose each constraint is relaxed by 0.5. That means that in formula (5.1.1) we have $\epsilon = [0.5\ 0.5\ 0.5\ 0.5]'$. So, the new optimum should be

$$\begin{aligned} z_\epsilon^* &= z^* + \epsilon'\pi \\ &= 9 + 0.5\times 0.5 + 0.5\times 0 + 0.5\times 0 + 0.5\times 1.5 \\ &= 10. \end{aligned}$$

Is this correct? Let us see:

```
octave:1> c=[4;3]; A=[2 3;-3 2;0 2;2 1]; b=[6;3;5;4]+0.5;
octave:2> [x_max,z_max,status,extra]=glpk(c,A,b,[0;0],[],"UUUU","CC",-1)
x_max =

   1.7500
   1.0000

z_max =  10
status =  180
extra =
{
  lambda =

     0.50000
     0.00000
    -0.00000
     1.50000

  redcosts =

     0
     0

  time = 0
  mem = 0
}
```

So, we see that formula (5.1.1) is correct, when $\epsilon \leq [0.5\ 0.5\ 0.5\ 0.5]'$.

Let us then consider a big change. Let $\epsilon = [10\ 10\ 10\ 0]'$. Then formula (5.1.1) would give us

$$\begin{aligned} z_\epsilon^* &= z^* + \epsilon'\pi \\ &= 9 + 10\times 0.5 + 10\times 0 + 10\times 0 + 0\times 1.5 \\ &= 14. \end{aligned}$$

Let's see what really happens:

```
octave:1> c=[4;3]; A=[2 3;-3 2;0 2;2 1];
octave:2> b=[6;3;5;4]+[10;10;10;0];
octave:3> [x_max,z_max,status,extra]=glpk(c,A,b,[0;0],[],"UUUU","CC",-1)
x_max =

   0
   4

z_max =  12
status =  180
extra =
{
  lambda =

     0
     0
    -0
     3

  redcosts =

    -2
     0

  time = 0
  mem = 0
}
```

We see that `glpk` gives 12 a the optimum. The formula (5.1.1) gave us the optimum 14. So, the change $[10\ 10\ 10\ 0]'$ is not small enough for the formula (5.1.1) to be valid. What actually happened here was that the optimal point jumped corners.

Reduced Costs

Let us then consider the reduced costs. Remember that the shadow prices were associated to the constraints, or — if you like simplex language — to the slacks. The reduced costs are associated to the decision variables.

> **The reduced cost u_i for a (NBV) decision variable x_i is the amount the objective value z at the optimum would decrease if x_i would be forced to be 1, and thus a BV — given that the change from $x_i = 0$ to $x_i = 1$ is small.**

Here are some interpretations and remarks of reduced costs that should help you to understand the concept:

- The clause "given that the change from $x_i = 0$ to $x_i = 1$ is small" is a similar clause that the clause "given that the optimal BVs don't change" was

in the definition of the shadow price. Indeed, it may be, e.g., that forcing $x_i \geq 1$ will make the LP infeasible. Remember: In sensitivity analysis we are talking about **small changes** — whatever that means. The analysis may, and most often will, fail for big changes.

- Decision variables that are BV have zero reduced costs.
- The reduced cost is also known as **opportunity cost**. Indeed, suppose we are given the **forced** opportunity (there are no problems — only opportunities) to produce one unit of x_i that we would not otherwise manufacture at all. This opportunity would cost us, since our optimized objective would decrease to a suboptimal value. Indeed, we have now one more constraint — the forced opportunity — in our optimization problem. So, the optimal solution can only get worse. The decrease of the objective value is the opportunity cost.
- The reduced cost u_i of x_i is the amount by which the objective coefficient c_i for x_i needs to change before x_i will become non-zero at the optimum.
- As an example of the point above, consider that you are producing $x_1, \ldots, x_n$ that will give you profits $c_1, \ldots, c_n$. You have some constraints, but the actual form of them does not matter here. Now, you form the LP to optimize your profit, and you solve it. You get optimal solution for productions: $x_1^*, x_2^*, \ldots, x_n^*$, and you get your optimal profit z^*. You notice that, say, $x_2^* = 0$. So, obviously the profit c_2 for x_2 is not big enough. Then you ask yourself: How big should the profit c_2 for x_2 be so that it becomes more profitable to produce x_2, at least a little, rather than not to produce x_2 at all? The answer is $c_2 - u_2$ (or $c_2 + u_2$, depending on how you interpret the sign). This means that the profit must increase at least by the reduced cost before it becomes more profitable to produce a product you would not produce otherwise.

Let us now consider the reduced cost with `glpk` with:

5.1.3 Example.

$$
\begin{array}{lrcrcrl}
\max z & = & 4x_1 & + & 3x_2 & & \\
\text{s.t.} & & 2x_1 & + & 3x_2 & \leq\ 16 & (1)\\
 & & -3x_1 & + & 2x_2 & \leq\ 13 & (2)\\
 & & & & 2x_2 & \leq\ 15 & (3)\\
 & & 2x_1 & + & x_2 & \leq\ 4 & (4)\\
 & & & & x_1, x_2 & \geq\ 0 &
\end{array}
$$

`glpk` returns the reduced costs in the output structure `extra` in its field `redcosts`:

```
octave:1> c=[4 3]'; A=[2 3; -3 2; 0 2; 2 1]; b=[16 13 15 4]';
octave:2> [x_max,z_max,status,extra]=glpk(c,A,b,[0 0]',[],"UUUU","CC",-1)
x_max =

   0
   4

z_max =  12
status =  180
extra =
{
  lambda =

     0
     0
    -0
     3

  redcosts =

    -2
     0

  time = 0
  mem = 0
}
```

So, we see that the redcosts are $u_1 = -2$ for the NBV decision x_1 and $u_2 = 0$ for the BV decision x_2

Let us then test the interpretation

> "reduced cost is the decrease in the value of the objective if we are forced to produce one unit where we otherwise would produce none".

We test the interpretation with the following LP:

$$\begin{array}{lrcrcrcl}
\max z & = & 4x_1 & + & 3x_2 & & & \\
\text{s.t.} & & 2x_1 & + & 3x_2 & \leq & 16 & (1) \\
 & & -3x_1 & + & 2x_2 & \leq & 13 & (2) \\
 & & & & 2x_2 & \leq & 15 & (3) \\
 & & 2x_1 & + & x_2 & \leq & 4 & (4) \\
 & & x_1 & & & \geq & 1 & (5) \\
 & & & & x_1, x_2 & \geq & 0 &
\end{array}$$

So, we have added to the LP of Example 5.1.3 the requirement that we must have at least one x_1 in the solution. This is the constraint (5). Remember that without this requirement we would have zero x_1's in the solution.

So, here is the Octave code for this problem:

```
octave:1> c=[4 3]'; A=[2 3; -3 2; 0 2; 2 1; 0 1]; b=[16 13 15 4 1]';
octave:2> [x_max,z_max,status,extra]=glpk(c,A,b,[0 0]',[],"UUUUL","CC",-1)
x_max =

   1
   2

z_max =  10
status =  180
extra =
{
  lambda =

     0
    -0
    -0
     3
    -2

  redcosts =

    -0
     0

  time = 0
  mem = 0
}
```

We see that the interpretation is indeed correct: The previous optimal value 12 dropped by 2 into 10.

Sensitivity Analysis with Simplex with Matrices

We have learned how to perform sensitivity analysis with `glpk`. Let us then consider how sensitivity analysis works with the simplex tableaux. The short answer is

> **The shadow prices are the coefficients of the slacks in the first row of the optimal simplex tableau and the reduced costs are (minus) the coefficients of the decision variables in the first for of the optimal simplex tableau.**

Consider the Giapetto's LP 1.1.1 as an example. In the slack form Giapetto's LP (1.1.3) reads

$$
\begin{array}{lrcrcrcrcrcr}
 & \max & z & = & 3x_1 & + & 2x_2 & & & & & \\
 & \text{s.t.} & & & 2x_1 & + & x_2 & + & s_1 & & & = & 100 \\
(5.1.4) & & & & x_1 & + & x_2 & & & + & s_2 & = & 80 \\
 & & & & x_1 & & & & & & + \; s_3 & = & 40 \\
 & & & & & & & & x_1, x_2, s_1, s_2, s_3 & & & \geq & 0
\end{array}
$$

So, the first Simplex tableau for (5.1.4) is

Row	z	x_1	x_2	s_1	s_2	s_3	BV	RHS
1	1	-3	-2	0	0	0	$z =$	0
2	0	2	1	1	0	0	$s_1 =$	100
3	0	1	1	0	1	0	$s_2 =$	80
4	0	1	0	0	0	1	$s_3 =$	40

and the last (optimal) simplex tableau is

Row	z	x_1	x_2	s_1	s_2	s_3	BV	RHS
0	1	0	0	1	1	0	$z =$	180
1	0	0	1	-1	2	0	$x_2 =$	60
2	0	0	0	-1	1	1	$s_3 =$	20
3	0	1	0	0	-1	0	$x_1 =$	20

Let $\mathbf{u}$ denote the vector of reduced costs and π the vector of shadow prices. From the 1st row we read that

$$\mathbf{u} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \pi = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}.$$

So, the shadow prices for the NBV slacks s_1 and s_2 are 1 for both. So, additional carpentry and finishing hours are both worth €1 per hour for Giapetto. Since s_3 is BV additional unit of market demand for soldiers is worthless to Giapetto. Finally, we see that the reduced costs are zero, since all the decision variables are BV.

Here is another example:

5.1.5 Example.

$$\begin{array}{lrcrcrcrl} \max & z & = & 60x_1 & + & 30x_2 & + & 20x_3 & \\ \text{s.t.} & & & 8x_1 & + & 6x_2 & + & x_3 & \leq 48 \\ & & & 4x_1 & + & 2x_2 & + & 1.5x_3 & \leq 20 \\ & & & 2x_1 & + & 3x_2 & + & 0.5x_3 & \leq 8 \\ & & & & & x_1, x_2, s_1, s_2, s_3 & & & \geq 0 \end{array}$$

Our first simplex tableau is

Row	z	x_1	x_2	x_3	s_1	s_2	s_3	BV	RHS
1	1	-60	-30	-20	0	0	0	$z =$	0
2	0	8	6	1	1	0	0	$s_1 =$	48
3	0	4	2	1.5	0	1	0	$s_2 =$	20
4	0	2	3	0.5	0	0	1	$s_3 =$	8

and after long and tedious table-dancing we get the optimal simplex tableau

Row	z	x_1	x_2	x_3	s_1	s_2	s_3	BV	RHS
1	1	0	5	0	0	10	10	$z =$	280
2	0	0	-2	0	1	2	-8	$s_1 =$	24
3	0	0	-2	1	0	2	-4	$x_3 =$	8
4	0	1	1.25	0	0	-0.5	1.5	$x_1 =$	2

Now we can read sensitivity information from the 1st row:

$$\mathbf{u} = \begin{bmatrix} 0 \\ -5 \\ 0 \end{bmatrix} \quad \text{and} \quad \pi = \begin{bmatrix} 0 \\ 10 \\ 10 \end{bmatrix}.$$

We see that the reduced cost for x_2 is -5. This means that the profit for x_2 should increase at least by 5 before it makes sense to produce them. Or, if you like, producing one x_1 would decrease the optimal profit 280 by 5. The reduced costs for x_1 and x_3 are zero, since they are BV. The shadow prices are: 0 for the slack s_1, since it is not active, and 10 for both the active constraints s_2 and s_3. So, additional resources for the second and the third constraints are both worth 10, and additional resources for the first constraint are worthless.

5.2 Dual Problem

Finding Dual

Associated with any LP there is another LP, called the **dual** — and then the original LP is called the **primal**. In general, if the primal LP is a maximization problem, the dual is a minimization problem — and vice versa. Also, the constraints of the primal LP are the coefficients of the objective of the dual problem — and vice versa. If the constraints of the primal LP are of type $\leq$ then the constraints of the dual LP are of type $\geq$ — and vice versa.

Let us now give the formal definition of the dual. We assume that the primal LP is in standard form. Since all LPs can be transformed into a standard form this assumption does not restrict the generality of the duality. The assumption is made only for the sake of convenience.

The (linear) **dual** of the LP

$$\begin{array}{lrcl} \max & z & = & \mathbf{c}'\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array}$$

is

$$\begin{array}{lrcl} \min & w & = & \mathbf{b}'\mathbf{y} \\ \text{s.t.} & \mathbf{A}'\mathbf{y} & \geq & \mathbf{c} \\ & \mathbf{y} & \geq & \mathbf{0} \end{array}$$

5.2.1 Example. Consider the LP

$$\max z = [1\ 2\ 3] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\text{s.t.} \quad \begin{bmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 10 \\ 11 \end{bmatrix}.$$

The dual LP is

$$\min w = [10\ 11] \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$\text{s.t.} \quad \begin{bmatrix} 4 & 7 \\ 5 & 8 \\ 6 & 9 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \geq \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

Let us discuss briefly concept of duality in general and the linear duality in particular.

- In general, dual is a transformation with the following property: **transforming twice you get back**. This is the abstract definition of duality. In mathematics a function is f is called **involution** if it is its own inverse, i.e., $f^{-1} = f$. So, duality is a **meta-mathematical involution**.
- Looking at the definition of the linear dual one sees the dual is LP itself. So, it can be transformed into a standard form, and the one can construct the dual of the dual. When one does so one gets back to the original primal LP, i.e., **the dual of the dual is primal**. So, the linear dual deserves its name.
- We have already seen one duality between LPs before: A minimization problem is in duality with a maximization problem with the transform where the objective function is multiplied by -1. The usefulness of this simple duality was that we only need to consider maximization problems, and the solution of the minimization problem is -1 times the solution of the corresponding maximization problem. Also, the optimal decisions in the maximization and minimization problems are the same.
- The linear duality is more complicated than the simple "multiply by -1 duality" of the previous point. This makes the linear duality in some sense more useful. Indeed, since the transformation is more complicated, our change

of perspective is more radical, and thus this transformation gives us better intuition of the original problem.

- The linear duality is very useful because of the following theorems: The **weak duality theorem** states that the objective function value w of the dual at any feasible solution $\mathbf{y}$ is always greater than or equal to the objective function value z of the primal at any feasible solution $\mathbf{x}$:

$$w = \mathbf{b}'\mathbf{y} \quad \geq \quad \mathbf{c}'\mathbf{x} = z.$$

The weak duality theorem can be used to get upper bounds to the primal LP. The **strong duality theorem** states that if the primal has an optimal solution, $\mathbf{x}^*$, then the dual also has an optimal solution, $\mathbf{y}^*$, such that

$$z^* = \mathbf{c}'\mathbf{x}^* \quad = \quad \mathbf{b}'\mathbf{y}^* = w^*.$$

The strong duality theorem can be used to solve the primal LP. Finally, the **complementary slackness theorem** states that if a constraint in either the primal or the dual is non-active, then the corresponding variable in the other — complementary — problem must be zero.

Let us find a dual of an LP that is not in standard form.

5.2.2 Example. Consider the LP

$$\begin{array}{llrcrcrcr} \min & z & = & 50x_1 & + & 20x_2 & + & 30x_3 & \\ \text{s.t.} & & & 2x_1 & + & 3x_2 & + & 4x_3 & \geq & 11 \\ & & & 12x_1 & + & 13x_2 & + & 14x_3 & \leq & 111 \\ & & & x_1 & + & x_2 & + & x_3 & = & 1 \\ & & & & & & & x_1, x_2, x_3 & \geq & 0 \end{array}$$

The LP of Example 5.2.2 is not in standard form. So, before constructing its dual, we transform it into standard form. This is not necessary. Sometimes we can be clever, and find the dual without first transforming the primal into standard form. But we don't feel clever now. So, here is the standard form:

$$\begin{array}{llrcrcrcr} \max & -z & = & -50x_1 & - & 20x_2 & - & 30x_3 & \\ \text{s.t.} & & & -2x_1 & - & 3x_2 & - & 4x_3 & \leq & -11 \\ & & & 12x_1 & + & 13x_2 & + & 14x_3 & \leq & 111 \\ & & & x_1 & + & x_2 & + & x_3 & \leq & 1 \\ & & & -x_1 & - & x_2 & - & x_3 & \leq & -1 \\ & & & & & & & x_1, x_2, x_3 & \geq & 0 \end{array}$$

Now we are ready to present the dual:

$$\begin{array}{llrcrcrcrcr} \min & -w & = & -11y_1 & + & 111y_2 & + & y_3 & - & y_4 & & \\ \text{s.t.} & & & -2y_1 & + & 12y_2 & + & y_3 & - & y_4 & \geq & -50 \\ & & & -3y_1 & + & 13y_2 & + & y_3 & - & y_4 & \geq & -20 \\ & & & -4y_1 & + & 14y_2 & + & y_3 & - & y_4 & \geq & -30 \\ & & & & & & & y_1, y_2, y_3, y_4 & & & \geq & 0 \end{array} \tag{5.2.3}$$

(we used variable $-w$ in the dual because there was variable $-z$ in the standard form primal). Note now that the dual LP (5.2.3) in in "dual standard form": It is a minimization problem with only inequalities of type $\geq$. The original primal LP was a minimization problem. So, it is natural to express the dual LP as a maximization problem. Also, inequalities of type $\leq$ are more natural to maximization problems than the opposite type inequalities $\geq$. So, let us transform the dual LP (5.2.3) into a maximization problem with $\leq$ type inequalities. In fact, let us transform the dual LP (5.2.3) into a standard form. We obtain

$$\begin{array}{lrcrcrcrcr} \max & w & = & 11y_1 & - & 111y_2 & - & y_3 & + & y_4 & \\ \text{s.t.} & & & 2y_1 & - & 12y_2 & - & y_3 & + & y_4 & \leq & 50 \\ & & & 3y_1 & - & 13y_2 & - & y_3 & + & y_4 & \leq & 20 \\ & & & 4y_1 & - & 14y_2 & - & y_3 & + & y_4 & \leq & 30 \\ & & & & & & & y_1, y_2, y_3, y_4 & & & \geq & 0 \end{array}$$

Economic Interpretation of Dual

5.2.4 Example (Dakota's fractional furniture)**.** The Dakota Furniture Company manufactures desks, tables, and chairs. The manufacture of each type of furniture requires lumber and two types of skilled labor: finishing labor and carpentry labor. The amount of each resource needed to make each type of furniture is given in the table below:

	Product		
Resource	Desk	Table	Chair
Lumber	8 units	6 units	1 unit
Finishing	4 hours	2 hours	1.5 hours
Carpentry	2 hours	1.5 hours	0.5 hours

At present, 48 units of lumber, 20 finishing hours, and 8 carpentry hours are available. A desk sells for €60, a table for €30, and a chair for €20.

Since the available resources have already been purchased, Dakota wants to maximize total revenue.

Dakota's fractional furniture problem is, given that it is O.K. to produce fractional amount of desks, tables, and chairs, a typical product selection problem. So, following Algorithm 1.1.2 in the same way as in the Giapetto's problem 1.1.1, we find that the LP for Dakota's fractional furniture example 5.2.4 is

$$\begin{array}{lrcrcrcrcrl} \max & z & = & 60x_1 & + & 30x_2 & + & 20x_3 & & & \\ \text{s.t.} & & & 8x_1 & + & 6x_2 & + & x_3 & \leq & 48 & \text{(lumber)} \\ & & & 4x_1 & + & 2x_2 & + & 1.5x_3 & \leq & 20 & \text{(finishing)} \\ & & & 2x_1 & + & 1.5x_2 & + & 0.5x_3 & \leq & 8 & \text{(carpentry)} \\ & & & & & & & x_1, x_2, x_3 & \geq & 0 & \end{array}$$

where

$$\begin{aligned} x_1 &= \text{number of desks manufactured} \\ x_2 &= \text{number of tables manufactured} \\ x_3 &= \text{number of chairs manufactured} \end{aligned}$$

Now, the dual of this problem is

$$\begin{array}{llrcrcrcrl} & \min & w & = & 48y_1 & + & 20y_2 & + & 8y_3 & \\ & \text{s.t.} & & & 8y_1 & + & 4y_2 & + & 2y_3 & \geq 60 \quad \text{(desk)} \\ (5.2.5) & & & & 6y_1 & + & 2y_2 & + & 1.5y_3 & \geq 30 \quad \text{(table)} \\ & & & & x_1 & + & 1.5y_2 & + & 0.5y_3 & \geq 20 \quad \text{(chair)} \\ & & & & & & & & y_1, y_2, y_3 & \geq 0 \end{array}$$

We have given the constraints the names (desk), (table), and (chair). Those were the decision variables x_1, x_2 and x_3 in the primal LP. By symmetry, or duality, we could say that y_1 is associated with lumber, y_2 with finishing, and y_3 with carpentry. What is going on here? It is instructive to represent all the data of the Dakota's problem in a single table:

	Product			
Resource	Desk	Table	Chair	*Availability*
Lumber	8 units	6 units	1 unit	48 units
Finishing	4 hours	2 hours	1.5 hours	20 hours
Carpentry	2 hours	1.5 hours	0.5 hours	8 hours
Price	€60	€30	€20	

Now, the table above can be read either horizontally or vertically. You should already know how the read the table above horizontally. That is the Dakota's point of view. But what does it mean to read the table vertically? Here is the explanation, that is also the **economic interpretation** of the dual LP:

Suppose you are an entrepreneur who wants to purchase all of Dakota's resources — maybe you are a competing furniture manufacturer, or maybe you need the resources to produce soldiers and trains like Giapetto. Then you must determine the price you are willing to pay for a unit of each of Dakota's resources. But what are the Dakota's resources? Well they are lumber, finishing hours, and carpentry hours, that Dakota uses to make its products. So, the **decision variables** for the entrepreneur who wants to buy Dakota's resources are:

$$\begin{aligned} y_1 &= \text{price to pay for one unit of lumber,} \\ y_2 &= \text{price to pay for one hour of finishing labor,} \\ y_3 &= \text{price to pay for one hour of carpentry labor.} \end{aligned}$$

Now we argue that the resource prices y_1, y_2, y_3 should be determined by solving the Dakota's dual (5.2.5).

First note that you are buying **all** of Dakota's resources. Also, note that this is a minimization problem: You want to pay as little as possible. So, the **objective function** is

$$\min w \quad = \quad 48y_1 + 20y_2 + 8y_3.$$

Indeed, Dakota has 48 units of lumber, 20 hours of finishing labor, and 8 hours of carpentry labor.

Now we have the decision variables and the objective. How about the **constraints**? In setting the resource prices y_1, y_2, and y_3, what kind of constraints do you face? You must set the resource prices high enough so that Dakota would sell them to you. Now Dakota can either use the resources itself, or sell them to you. How is Dakota using its resources? Dakota manufactures desks, tables, and chair. Take desks first. With 8 units of lumber, 4 hours of finishing labor, and 2 hours of carpentry labor Dakota can make a desk that will sell for €60. So, you have to offer more than €60 for this particular combination of resources. So, you have the constraint

$$8y_1 + 4y_2 + 2y_3 \quad \geq \quad 60.$$

But this is just the first constraint in the Dakota's dual, denoted by (desk). Similar reasoning shows that you must pay at least €30 for the resources Dakota uses to produce one table. So, you get the second constraint, denoted by (table), of the Dakota's dual:

$$6y_1 + 2y_2 + 1.5y_3 \quad \geq \quad 30.$$

Similarly, you must offer more than €20 for the resources Dakota can use itself to produce one chair. That way you get the last constraint, labeled as (chair), of the Dakota's dual:

$$y_1 + 1.5y_2 + 0.5y_3 \quad \geq \quad 20.$$

We have just interpreted economically the dual of a maximization problem. Let us then change our point of view to the opposite and interpret economically the dual of a minimization problem.

5.2.6 Example (Ursus Maritimus's diet)**.** My diet requires that I only eat brownies, chocolate ice cream, cola, and pineapple cheesecake. The costs of the foods (in Cents) and my daily nutritional requirements together with their calorie, chocolate, sugar, and fat contents are

	Nutrient				
Product	Calories	Chocolate	Sugar	Fat	*Price*
Brownie	400	3	2	2	50
Chocolate ice cream	200	2	2	4	20
Cola	150	0	4	1	30
Pineapple cheesecake	500	0	4	5	80
Requirement	500	6	10	8	

I want to minimize the cost of my diet. How should I eat?

Let us then find the LP for the Diet problem of Example 5.2.6. As always, we use Algorithm 1.1.2. So, first we determine the **decision variables**. The decision to be made is: how much each type of food should be eaten daily. So, I have the decision variables

$$\begin{aligned} x_1 &= \text{number of brownies eaten daily,} \\ x_2 &= \text{number (of scoops) of chocolate ice creams eaten daily,} \\ x_3 &= \text{number (of bottles) of cola drunk daily,} \\ x_4 &= \text{number (of pieces) of pineapple cheesecake eaten daily.} \end{aligned}$$

I want to minimize the cost of the diet. So, the **objective function** is

$$\min z = 50x_1 + 20x_2 + 30x_3 + 80x_4.$$

Finally, we define the **constraints**. The daily calorie intake requirement gives

$$400x_1 + 200x_2 + 150x_3 + 500x_4 \geq 500.$$

The daily chocolate requirement gives

$$3x_1 + 2x_2 \geq 6.$$

The daily sugar requirement gives

$$2x_1 + 2x_2 + 4x_3 + 4x_4 \geq 10,$$

and the daily fat requirement gives

$$2x_1 + 4x_2 + x_3 + 5x_4 \geq 8.$$

So, we see that the Diet problem of Example 5.2.6 is the LP

$$
\begin{array}{lrcrcrcrcll}
\min z = & 50x_1 & + & 20x_2 & + & 30x_3 & + & 80x_4 & & & \\
\text{s.t.} & 400x_1 & + & 200x_2 & + & 150x_3 & + & 500x_4 & \geq & 500 & \text{(calorie)} \\
& 3x_1 & + & 2x_2 & & & & & \geq & 6 & \text{(chocolate)} \\
& 2x_1 & + & 2x_2 & + & 4x_3 & + & 4x_4 & \geq & 10 & \text{(sugar)} \\
& 2x_1 & + & 4x_2 & + & x_3 & + & 5x_4 & \geq & 8 & \text{(fat)} \\
& & & & & x_1, x_2, x_3, x_4 & & & \geq & 0 &
\end{array}
\tag{5.2.7}
$$

What about the dual of (5.2.7) then. Now, the LP (5.2.7) is not in standard form. So, in principle we should first transform it into standard form, and then construct the dual. We shall not do that, however. Instead, we remember that the dual of the dual is primal. So, we read the definition of the dual backwards, and obtain immediately the dual of (5.2.7):

$$
\begin{array}{lrcrcrcrcll}
\max w = & 500y_1 & + & 6y_2 & + & 10y_3 & + & 8y_4 & & & \\
\text{s.t.} & 400y_1 & + & 3y_2 & + & 2y_3 & + & 2y_4 & \leq & 50 & \text{(brownie)} \\
& 200y_1 & + & 2y_2 & + & 2y_3 & + & 4y_4 & \leq & 20 & \text{(ice cream)} \\
& 150y_1 & & & + & 4y_3 & + & y_4 & \leq & 30 & \text{(soda)} \\
& 500y_1 & & & + & 4y_3 & + & 5y_4 & \leq & 80 & \text{(cheesecake)} \\
& & & & & y_1, y_2, y_3, y_4 & & & \geq & 0 &
\end{array}
\tag{5.2.8}
$$

What is then the economic interpretation of this dual? Well, reading the table

	Nutrient				
Product	Calories	Chocolate	Sugar	Fat	*Price*
Brownie	400	3	2	2	50
Chocolate ice cream	200	2	2	4	20
Cola	150	0	4	1	30
Pineapple cheesecake	500	0	4	5	80
Requirement	500	6	10	8	

vertically, instead of horizontally, we see that we can consider a "nutrient" salesperson who sells calories, chocolate, sugar, and fat. The salesperson wishes to ensure that a dieter will meet all of his daily requirements by purchasing calories, sugar, fat, and chocolate from the the salesperson. So, the salesperson must determine the prices of her products:

$$
\begin{aligned}
y_1 &= \text{price of a calorie,} \\
y_2 &= \text{price of a unit of chocolate,} \\
y_3 &= \text{price of a unit of sugar,} \\
y_4 &= \text{price of a unit of fat.}
\end{aligned}
$$

The salesperson wants to maximize her profit. So, what is the salesperson selling? She is selling daily diets. So, the objective is

$$\max w \quad = \quad 500y_1 + 6y_2 + 10y_3 + 8y_4.$$

What are the constraints for the salesperson? In setting the nutrient prices she must set the prices low enough so that it will be in the dieter's economic interest to purchase all his nutrients from her. For example, by purchasing a brownie for €0.50, the dieter can obtain 400 calories, 3 units of chocolate, 2 units of sugar, and 2 units of fat. So, the salesperson cannot charge more than €0.50 for this combination of nutrients. This gives her the brownie constraint

$$400y_1 + 3y_2 + 2y_3 + 2y_4 \quad \leq \quad 50$$

(remember, we counted in Cents). In the similar way the salesperson will have the ice cream, soda, and cheesecake constraints listed in the dual LP (5.2.8).

5.3 Duality Theorems*

Weak Duality Theorem

5.3.1 Theorem (Weak duality theorem)**.** *Let* $\mathbf{x}$ *be any BFS of the primal LP and let* $\mathbf{y}$ *be any BFS of the dual LP. Then*

$$z = \mathbf{c}'\mathbf{x} \quad \leq \quad \mathbf{b}'\mathbf{y} = w.$$

Proof. Consider any of the dual decision variable y_i, $i = 1, \ldots, m$. Since $y_i \geq 0$ we can multiply the ith primal constraint by y_i without changing the direction of the constraint number i. (Moreover, the system remains equivalent, but that's not important here). We obtain

$$y_i A_{i1} x_1 + \cdots + y_i A_{in} x_n \quad \leq \quad b_i y_i \quad \text{for all } i = 1, \ldots, m. \tag{5.3.2}$$

Adding up all the m inequalities (5.3.2), we find that

$$\sum_{i=1}^{m} \sum_{j=1}^{n} y_i A_{ij} x_j \quad \leq \quad \sum_{i=1}^{m} b_i y_i. \tag{5.3.3}$$

Similarly, if we consider any of the primal decision variables x_j, $j = 1, \ldots, n$, we have that $x_j \geq 0$. So, we can multiply the jth dual constraint by the decision x_j without changing the direction of the constraint. We obtain

$$x_j A_{1j} y_1 + \cdots + x_j A_{mj} y_m \quad \geq \quad c_j x_j. \tag{5.3.4}$$

Adding up all the n inequalities (5.3.4), we find that

$$\sum_{i=1}^{m}\sum_{j=1}^{n} y_i A_{ij} x_j \quad \geq \quad \sum_{j=1}^{n} c_j x_j. \tag{5.3.5}$$

Combining (5.3.3) and (5.3.5), we obtain double-inequality

$$\sum_{j=1}^{n} c_j x_j \quad \leq \quad \sum_{i=1}^{m}\sum_{j=1}^{n} y_i A_{ij} x_j \quad \leq \quad \sum_{i=1}^{m} b_i y_i.$$

But, that's it! □

Let us then consider the consequences — or corollaries — of the weak duality theorem 5.3.1.

5.3.6 Corollary. *If the primal LP and dual LP both are feasible, then their optimal solutions are bounded.*

Proof. Let $\mathbf{y}$ be a BFS for the dual problem. Then the weak duality theorem 5.3.1 shows that $\mathbf{b}'\mathbf{y}$ is an upper bound for any objective value $\mathbf{c}'\mathbf{x}$ associated with any BFS $\mathbf{x}$ of the primal LP:

$$\mathbf{c}'\mathbf{x} \quad \leq \quad \mathbf{b}'\mathbf{y}.$$

Since this is true for **any** primal decision $\mathbf{x}$, we have that

$$\begin{aligned} z^* &= \max\left\{\mathbf{c}'\mathbf{x}\,;\, \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\right\} \\ &\leq \max\left\{\mathbf{b}'\mathbf{y}\,;\, \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\right\} \\ &= \mathbf{b}'\mathbf{y} \\ &< \infty \end{aligned}$$

is bounded. Now, change the rôles of the primal and the dual, and you see that the claim of Corollary 5.3.6 is true. □

5.3.7 Corollary. *Suppose* $\mathbf{x}^*$ *is a BFS for the primal and* $\mathbf{y}^*$ *is a BFS for the dual. Suppose further that* $\mathbf{c}'\mathbf{x}^* = \mathbf{b}'\mathbf{y}^*$. *Then both* $\mathbf{x}^*$ *and* $\mathbf{y}^*$ *are optimal for their respective problems.*

Proof. If $\mathbf{x}$ is any BFS for the primal, then the weak duality theorem 5.3.1 tells us that

$$\mathbf{c}'\mathbf{x} \quad \leq \quad \mathbf{b}'\mathbf{y}^* \quad = \quad \mathbf{c}'\mathbf{x}^*.$$

But this means that $\mathbf{x}^*$ is primal optimal. Now, change the rôles of the primal and dual, and you see that the claim of the Corollary 5.3.7 is true. □

Strong Duality Theorem

Here is the **duality theorem**, or the **strong duality theorem**:

5.3.8 Theorem (Duality theorem)**.** *Let*

$$\mathbf{x_{BV}} = \mathbf{B}^{-1}\mathbf{b}$$

be the BV-part of an optimal BFS to the primal with the corresponding optimal objective value

$$z^* = \mathbf{c}'_{\mathbf{BV}}\mathbf{B}^{-1}\mathbf{b} = \pi'\mathbf{b}.$$

Then π is the decision part of an optimal BFS for the dual. Also, the values of the objectives at the optimum are the same:

$$w^* \quad = \quad \pi'\mathbf{b} \quad = \quad \mathbf{c}'_{\mathbf{BV}}\mathbf{x_{BV}}.$$

We shall not prove Theorem 5.3.8 in these notes. Instead, we leave it as an exercise. The author is well aware that this is a very demanding exercise, but not all of the exercises have to be easy!

Complementary Slackness

It is possible to obtain an optimal solution to the dual when only an optimal solution to the primal is known using the **complementary slackness theorem**. To state this theorem, we assume that the primal is in standard form with non-negative RHSs and objective coefficients. The primal decision variables are $\mathbf{x} = [x_1 \ \cdots \ x_n]'$ and the primal slacks are $\mathbf{s} = [s_1 \ \cdots \ s_m]'$. The dual is then a minimization problem with decision variables $\mathbf{y} = [y_1 \ \cdots \ y_m]'$, and with n constraints of type $\geq$ with non-negative RHSs. Let $\mathbf{e} = [e_1 \ \cdots \ e_n]'$ be the excesses of the dual problem associated with the constraints.

So, in slack form the primal LP is

$$\begin{array}{lrcccccl}
\max z & = & c_1x_1 & +\cdots+ & c_nx_n & & & \\
\text{s.t.} & & A_{11}x_1 & +\cdots+ & A_{1n}x_n & +s_1 & & = b_1 \\
& & A_{21}x_1 & +\cdots+ & A_{2n}x_n & +s_2 & & = b_2 \\
& & \vdots & \vdots & \vdots & \ddots & & \vdots \\
& & A_{m1}x_1 & +\cdots+ & A_{mn}x_n & & +s_m & = b_m \\
& & & & & x_1,\ldots,x_n,s_1,\ldots,s_m & & \geq 0
\end{array}$$

Similarly, the dual LP in slack — or rather excess — form is

$$\begin{array}{lrcccccl}
\min w & = & b_1y_1 & +\cdots+ & b_my_m & & & \\
\text{s.t.} & & A_{11}y_1 & +\cdots+ & A_{m1}y_m & -e_1 & & = c_1 \\
& & A_{12}y_1 & +\cdots+ & A_{m2}y_m & -e_2 & & = c_2 \\
& & \vdots & \vdots & \vdots & \ddots & & \vdots \\
& & A_{1n}y_1 & +\cdots+ & A_{mn}y_m & & -e_n & = c_n \\
& & & & & y_1,\ldots,y_m,e_1,\ldots,e_n & & \geq 0
\end{array}$$

5.3.9 Theorem (Complementary slackness theorem)**.** *Let* $\mathbf{x}$ *be a primal BFS, and let* $\mathbf{y}$ *be a dual BFS. Then* $\mathbf{x}$ *is primal optimal and* $\mathbf{y}$ *is dual optimal if and only if*

$$\begin{aligned} s_i y_i &= 0 \quad \text{for all } i = 1, \ldots, m, \\ e_j x_j &= 0 \quad \text{for all } j = 1, \ldots, n. \end{aligned}$$

Before going into the proof of the Complementary Slackness Theorem 5.3.9 let us note that it actually says that if a constraint in either the primal or the dual is non-active, then the corresponding variable in the other — complementary — problem must be zero. Hence the name **complementary** slackness.

Proof. The theorem 5.3.9 is of the type "if and only if". So, there are two parts in the proof: the "if part" and the "only if part". Before going to those parts let us note that

$$s_i = 0 \quad \text{if and only if} \quad \sum_{j=1}^{m} A_{ij} x_j^* = b_i, \tag{5.3.10}$$

$$e_j = 0 \quad \text{if and only if} \quad \sum_{i=1}^{n} A_{ij} y_i^* = c_j. \tag{5.3.11}$$

Proof of the if part: By (5.3.10) we see that

$$\begin{aligned} \sum_{i=1}^{m} b_i y_i^* &= \sum_{i=1}^{m} y_i^* \sum_{j=1}^{n} A_{ij} x_j^* \\ &= \sum_{i=1}^{m} \sum_{j=1}^{n} y_i^* A_{ij} x_j^* \end{aligned}$$

In the same way, by using (5.3.11) we see that

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} \sum_{j=1}^{n} y_i^* A_{ij} x_j^*.$$

So, the conclusion is that

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*,$$

and the "if part" follows from the Weak Duality Theorem 5.3.1.

Proof of the only if part: Like in the proof of the Weak Duality Theorem 5.3.1 we obtain

$$\sum_{j=1}^{n} c_j x_j^* \leq \sum_{i=1}^{m} \sum_{j=1}^{n} y_i^* A_{ij} x_j^* \leq \sum_{i=1}^{m} b_i y_i^*. \tag{5.3.12}$$

Now, by the Strong Duality Theorem 5.3.8, if $\mathbf{x}^*$ and $\mathbf{y}^*$ are optimal, then the LHS of (5.3.12) is equal to the RHS of (5.3.12). But this means that

$$\sum_{j=1}^{n}\left(c_j - \sum_{i=1}^{m} y_i^* A_{ij}\right) x_j^* = 0. \tag{5.3.13}$$

Now, both $\mathbf{x}^*$ and $\mathbf{y}^*$ are feasible. This means that the terms in (5.3.13) are all non-negative. This implies that the terms are all zeros. But this means that that $e_j x_j = 0$. The validity of $s_i y_i = 0$ can be seen in the same way by considering the equality

$$\sum_{i=1}^{m}\left(b_i - \sum_{j=1}^{m} A_{ij} x_j^*\right) y_i^* = 0.$$

This finishes the proof of the complementary slackness theorem 5.3.9. □

As an example of the use of the complementary slackness theorem 5.3.9, let us consider solving the following LP:

5.3.14 Example. You want to solve the LP

$$\begin{array}{lrcrcrcrl} \min & w & = & 4y_1 & + & 12y_2 & + & y_3 & \\ \text{s.t.} & & & y_1 & + & 4y_2 & - & y_3 & \geq 1 \\ & & & 2y_1 & + & 2y_2 & + & y_3 & \geq 1 \\ & & & & & & & y_1, y_2, y_3 & \geq 0 \end{array}$$

Now, suppose you have already solved, e.g. graphically — which is challenging for the LP in 5.3.14 — the much easier LP

$$\begin{array}{lrcrcrcl} \max & z & = & x_1 & + & x_2 & & \\ \text{s.t.} & & & x_1 & + & 2x_2 & \leq & 4 \\ & & & 4x_1 & + & 2x_2 & \leq & 12 \\ & & & -x_1 & + & x_2 & \leq & 1 \\ & & & & & x_1, x_2 & \geq & 0 \end{array}$$

The solution to this dual is

$$\begin{aligned} x_1^* &= 8/3 \\ x_2^* &= 2/3 \end{aligned}$$

with the optimal value

$$\begin{aligned} z^* &= x_1^* + x_2^* \\ &= 10/3. \end{aligned}$$

This means that you have already solved the dual — or primal, if you take the opposite point of view — of the Example 5.3.14.

Now, how can the solution above, combined with the Complementary Slackness Theorem 5.3.9, help you to solve the LP of Example 5.3.14?

Here is how: First note that $x_1^* > 0$ and $x_2^* > 0$. So, the Complementary Slackness Theorem 5.3.9 tells us that the optimal solution $\mathbf{y}^* = [y_1^* \; y_2^* \; y_3^*]'$ of the LP in Example 5.3.14 must have zero excesses. So, the inequalities in 5.3.14 are actually equalities at the optimum. Also, if we check the optimum $\mathbf{x}^*$ in the first three constraints of the maximum problem, we find equalities in the first two of them, and a strict inequality in the third one. So, the Complementary Slackness Theorem 5.3.9 tells us that $y_3^* = 0$. So, in the optimum $\mathbf{y}^*$ of the LP in Example 5.3.14 we must have

$$\begin{array}{rcrcl} y_1^* & + & 4y_2^* & = & 1 \\ 2y_1^* & + & 2y_2^* & = & 1 \\ & & y_3^* & = & 0 \end{array}$$

But this is a very easy system to solve. We obtain

$$\begin{aligned} y_1^* &= 1/3, \\ y_2^* &= 1/6, \\ y_3^* &= 0 \end{aligned}$$

with the optimal value

$$\begin{aligned} w^* &= 4y_1^* + 12y_2^* + y_3 \\ &= 10/3. \end{aligned}$$

5.4 Primal and Dual Sensitivity

For a standard form LP and its dual we have:

$$\begin{aligned} \mathbf{x}^*_{\text{primal}} &= \pi_{\text{dual}}, \\ \pi_{\text{primal}} &= \mathbf{y}^*_{\text{dual}}, \\ \mathbf{s}^*_{\text{primal}} &= \mathbf{u}_{\text{dual}}, \\ \mathbf{u}_{\text{primal}} &= \mathbf{e}^*_{\text{dual}}, \\ z^*_{\text{primal}} &= w^*_{\text{dual}}. \end{aligned}$$

Here $\mathbf{u}_{\text{primal}}$ and $\mathbf{u}_{\text{dual}}$ denote the vectors of reduced costs in the primal and dual, respectively. Also, $\mathbf{s}^*_{\text{primal}}$ and $\mathbf{e}^*_{\text{dual}}$ denote the slacks and excesses of the primal and dual at optimum, respectively. All the other notations should be clear.

Note that the equality

$$\pi_{\text{primal}} \quad = \quad \mathbf{y}^*_{\text{dual}}$$

explains the name "shadow prices". Indeed, the dual is a "shadow problem". So, the shadow prices of the constraints at the primal optimum are the prices of the dual variables (that are related to the constraints) at the dual optimum. Sometimes the shadow prices are called the dual prices.

5.5 Exercises and Projects

5.1. Consider the parameters $\mathbf{c}$, $\mathbf{A}$, and $\mathbf{b}$ of the Giapetto's problem 1.1.1. Which of them are reasonably certain, and which of them contain a lot of uncertainty? Can any of the parameters be taken to be absolutely certain?

5.2. Consider Manuel Eixample's 4.0.1 optimal simplex tableau from section 4.5 and answer to the following questions

(a) What are the shadow prices and reduced costs in Manuel's problem?
(b) Suppose Manuel had 21 instead of 20 liters of milk. What would be Manuel's new optimal revenue?
(c) Suppose Manuel had promised to sell at least one liter of Castile to José. How much will Manuel's optimal revenue decrease?
(d) How much should Manuel ask for a liter of Castile so that producing it would make sense for him (assuming that the demand remains unchanged)?

5.3. The function `simplex_lp_solver` does not provide shadow prices or reduced costs as outputs. Modify the function so that it does.

5.4. Find the duals for the following LPs, and solve them with your favorite method.

(a)

$$\begin{array}{lrcrcrcl} \max z & = & 2x_1 & + & x_2 & & & \\ \text{s.t.} & & -x_1 & + & x_2 & \leq & 1 \\ & & x_1 & + & x_2 & \leq & 3 \\ & & x_1 & - & 2x_2 & \leq & 4 \\ & & & & x_1, x_2 & \geq & 0 \end{array}$$

(b)

$$\begin{array}{lrcrcrcl} \min w & = & y_1 & - & y_2 & & & \\ \text{s.t.} & & 2y_1 & + & y_2 & \geq & 4 \\ & & y_1 & + & y_2 & \geq & 1 \\ & & y_1 & + & 2y_2 & \geq & 3 \\ & & & & y_1, y_2 & \geq & 0 \end{array}$$

5.5. A student is deciding what to purchase from a bakery for a tasty afternoon snack after a long and tedious lecture of Operations Research. There are two choices of food: Brownies, which cost 50 cents each, and mini-cheesecakes, which cost 80 cents. The bakery is service-oriented and is happy to let the student purchase a fraction of an item if she wishes. The bakery requires 30 g chocolate to make each brownie (no chocolate is needed in the cheesecakes). 20 g of sugar are needed for each brownie and 40 g of sugar for each cheesecake. Finally, 20 g of cream cheese are needed for each brownie and 50 g for each cheesecake. Being health-conscious, the student has decided that she needs at least 60 g of chocolate in her snack, along with 100 g of sugar and 80 g of cream cheese. She wishes to optimize her purchase by finding the least expensive combination of brownies and cheesecakes that meet these requirements.

(a) Model the problem as an LP,
(b) find the dual of the LP,
(c) interpret the dual LP economically, and finally
(d) solve the dual LP (by any method you like).

5.6. Prove the strong duality theorem 5.3.8.

5.7. * Make Octave function(s) that

(a) Visualizes the changes in the optimal value of an LP when any one or two parameters a varied.
(b) For an LP with two decision variables visualizes the changes of the location of an optimal decision when any one of its parameters are varied.

Here Octave functions `surf` and `plot` may turn out to be useful.

Part III

Linear Models

Chapter 6

Data Envelopment Analysis

> Efficiency is doing things right; effectiveness is doing the right things.
> — Peter F. Drucker

> In theory there is no difference between theory and practice, but in practice there is.
> — Anonymous

We discuss how to apply LP in the problem of **evaluating the relative efficiency of different units, relative only to themselves**. This is a nice application because of three reasons:

1. it is not at all obvious that LP can be used in this problem,
2. the application gives valuable insight to the LP duality,
3. the application itself is extremely useful.

Data envelopment analysis (DEA), occasionally called **frontier analysis**, was introduced by Charnes, Cooper and Rhodes in 1978. DEA is a performance measurement technique which can be used for **evaluating the relative efficiency** of **decision-making units** (DMUs). Here DMU is an abstract term for an entity that transforms inputs into outputs. The term is abstract on purpose: Typically one thinks of DMUs as manufacturers of some goods (outputs) who use some resources (inputs). This way of thinking, while correct, is very narrow-minded: DMU is a much more general concept, and DEA can be applied in very diverse situations. Indeed, basically the DMUs can be pretty much anything. The only restrictions are:

1. the DMUs have the same inputs and outputs,
2. the DMUs' inputs and outputs can be measured numerically.

There is one point of DEA that must be emphasized: DEA is a **data oriented extreme point method**. This means that it will only use the data related to the inputs and outputs of the DMUs under consideration. It does not use any

extra theoretical — or practical, or philosophical — knowledge. In this respect, it differs from classical comparison methods where DMUs are compared either to a "representative" DMU or to some "theoretically best" DMU. DEA compares the DMUs to the "best" DMUs under consideration.

6.1 Graphical Introduction*

One Input — One Output

6.1.1 Example (Kaupþing I)**.** Consider a number of Kaupþing Bank's branches. For each branch we have a single output measure: Number of personal transactions completed per week. Also, we have a single input measure: number of staff. The data we have is:

Branch	**Outputs and inputs**	
	Personal transactions	Number of staff
Reykjavík	125	18
Akureyri	44	16
Kópavogur	80	17
Hafnarfjörður	23	11

How then can we compare these branches — or DMUs — and measure their performance using this data? A commonly used method is **ratios**, which means that we will compare **efficiencies**.

For our Kaupþing Bank branch example 6.1.1 we have a single input measure, the number of staff, and a single output measure, the number of personal transactions. Hence the meta-mathematical formula

$$\text{efficiency} \quad = \quad \frac{\text{outputs}}{\text{inputs}} \quad = \quad \frac{\text{output}}{\text{input}}$$

is a well-defined mathematical formula — no metas involved. We have:

Branch	Personal transactions per staff member
Reykjavík	6.94
Akureyri	2.75
Kópavogur	4.71
Hafnarfjörður	2.09

Here we can see that Reykjavík has the highest ratio of personal transactions per staff member, whereas Hafnarfjörður has the lowest ratio of personal transactions per staff member. So, relative to each others, Reykjavík branch is the best (most efficient), and the Hafnarfjörður branch is the worst (least efficient).

As Reykjavík branch is the most efficient branch with the highest ratio of 6.94, it makes sense to compare all the other branches to it. To do this we calculate their **relative efficiency** with respect to Reykjavík branch: We divide the ratio for any branch by the Reykjavík's efficiency 6.94, and multiply by 100% (which is one) to convert to a percentage. This gives:

Branch	Relative Efficiency		
Reykjavík	$100\% \times (6.94/6.94)$	=	100%
Akureyri	$100\% \times (2.75/6.94)$	=	40%
Kópavogur	$100\% \times (4.71/6.94)$	=	68%
Hafnarfjörður	$100\% \times (2.09/6.94)$	=	30%

The other branches do not compare well with Reykjavík. That is, they are **relatively less efficient** at using their given input resource (staff members) to produce output (number of personal transactions).

One Input — Two Outputs

Typically we have more than one input and one output. In this subsection we consider the case of one input and two outputs. While the case of one input and one output was almost trivial, the case of two outputs and one input is still simple enough to allow for graphical analysis. The analog with LPs would be: LPs with one decision variable are trivial, and LPs with two decision variables are still simple enough to allow for graphical analysis.

Let us extend the Kaupþing Bank branch example 6.1.1:

6.1.2 Example (Kaupþing II)**.** Consider a number of Kaupþing Bank's branches. For each branch we have a two output measures: Number of personal transactions completed per week, and number of business transaction completed per week. We have a single input measure: number of staff. The data is:

Branch	**Outputs and inputs**		
	Personal transactions	Business transactions	Number of staff
Reykjavík	125	50	18
Akureyri	44	20	16
Kópavogur	80	55	17
Hafnarfjörður	23	12	11

How now can we compare these branches and measure their performance using this data? As before, a commonly used method is ratios, just as in the case considered before of a single output and a single input. Typically we take one of the output measures and divide it by one of the input measures.

For our bank branch example 6.1.2 the input measure is plainly the number of staff (as before) and the two output measures are number of personal transactions and number of business transactions. Hence we have the two ratios:

Branch	**Ratios**	
	Personal transactions per staff member	Business transactions per staff member
Reykjavík	6.94	2.78
Akureyri	2.75	1.25
Kópavogur	4.71	3.24
Hafnarfjörður	2.09	1.09

Here we can see that Reykjavík has the highest ratio of personal transactions per staff member, whereas Kópavogur has the highest ratio of business transactions per staff member. So, it seems that Reykjavík and Kópavogur are the best performers. Akureyri and Hafnarfjörður do not compare so well with Reykjavík and Kópavogur. That is, they are relatively less efficient at using their given input resource (staff members) to produce outputs (personal and business transactions).

One problem with comparison via ratios is that different ratios can give a different picture and it is difficult to combine the entire set of ratios into a single numeric judgment. For example, consider Akureyri and Hafnarfjörður:

- Akureyri is $2.75/2.09 = 1.32$ times as efficient as Hafnarfjörður at personal transactions,
- Akureyri is $1.25/1.09 = 1.15$ times as efficient as Hafnarfjöður at business transactions.

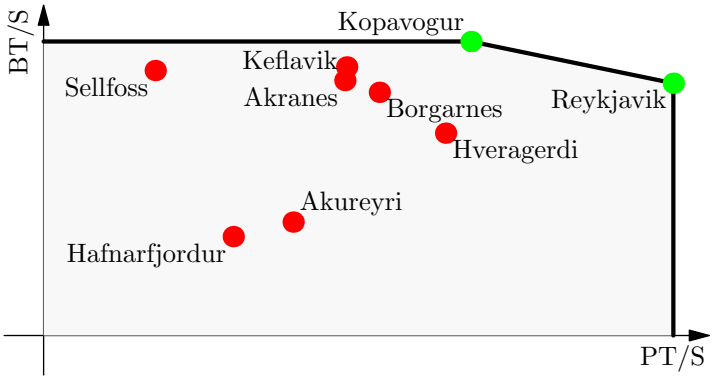
How would you combine these figures — 1.32 and 1.15 — into a single judgment? This problem of different ratios giving different pictures would be especially true if we were to increase the number of branches or increase the number of inputs or outputs.

6.1.3 Example (Kaupþing III)**.** We ad five extra branches, Sellfoss, Hveragerði, Akranes, Borgarnes, and Keflavík, to Example 6.1.2. The data is now:

Branch	**Ratios**	
	Personal transactions per staff member	Business transactions per staff member
Reykjavík	6.94	2.78
Akureyri	2.75	1.25
Kópavogur	4.71	3.24
Hafnarfjörður	2.09	1.09
Sellfoss	1.23	2.92
Hveragerði	4.43	2.23
Akranes	3.32	2.81
Borgarnes	3.70	2.68
Keflavík	3.34	2.96

What can be now said about the efficiencies of the branches?

One way around the problem of interpreting different ratios, at least for problems involving just two outputs and a single input, is a simple **graphical analysis**. Suppose we plot the two ratios for each branch as shown below. In the picture we have no ticks to express the scale. The ticks are left out on purpose: DEA is about **relative** efficiency. So, the scales do not matter.



The positions of Reykjavík and Kópavogur in the graph demonstrate that they are superior to all other branches: They are the extreme points, other DMUs are inferior to them. The line drawn in the picture is called the **efficient frontier**. It was drawn by taking the extreme points, and then connecting them to each others and to the axes. That was a very vague drawing algorithm, but I hope you got the picture. The name "data envelopment Analysis" arises from the efficient frontier that envelopes, or encloses, all the data we have.

Any DMU on the efficient frontier is 100% **DEA efficient**.

In our Kaupþing Bank branch examples 6.1.2 and 6.1.3, Reykjavík and Kópavogur branches are 100% efficient. This is not to say that the performance of Reykjavík or Kópavogur could not be improved. It may, or may not, be possible to do that. However, we can say that, based on the evidence provided by the different branches, we have no idea of the extent to which their performance can be improved. Also it is important to note here that:

- DEA only gives relative efficiencies, i.e., efficiencies relative to the data considered. It does not — and cannot — give absolute efficiencies.
- No extra information or theory was used in determining the relative efficiencies of the DMUs. What happened was that we merely took data on inputs and outputs of the DMUs we considered, and presented the data in a particular way.
- The statement that a DMU is 100% efficient simply means that we have no other DMU that can be said to be better than it.

Now we know when a DMU is 100% efficient: A DMU is 100% efficient if it is on the efficient frontier. How about the non-efficient DMUs? Can we associate the DMUs that are not in the efficient frontier with a number representing their efficiency? We can. How to do this, is explained below. So, we will now discuss about quantifying efficiency scores for inefficient DMUs.

Let us take Hafnarfjörður as an example of a non-efficient branch. We can see that, with respect to both of the ratios Reykjavík — and Kópavogur, too — dominates Hafnarfjörður. Plainly, Hafnarfjörður is less than 100% efficient. But how much? Now, for Hafnarfjörður we have the ratio

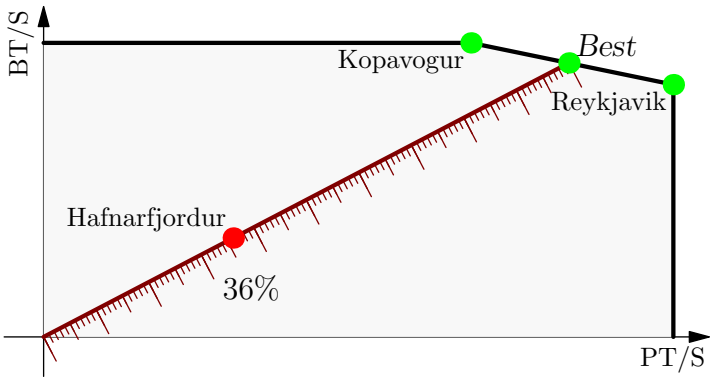
$$\frac{\text{personal transactions}}{\text{business transactions}} = \frac{23}{12} = 1.92.$$

This means that there are 1.92 personal transactions for every business transaction. This figure of 1.92 is also the ratio

$$\frac{\text{personal transactions per staff member}}{\text{business transactions per staff member}}.$$

This number, 1.92, is the **business mix** of the Hafnarfjörður branch. It can be also be interpreted that Hafnarfjörður branch weighs its outputs, personal transactions and business transactions, so that personal transactions get weight 1.92 and business transactions get weight 1.

Consider now the diagram below. In the diagram we have removed all the inefficient branches, except Hafnarfjörður. The line with the percentage ruler attached drawn through Hafnarfjörður represent all the possible — or virtual, if you like — branches having the same business mix, 1.92, as Hafnarfjörður.



Note the virtual branch *Best* in the picture above. *Best* represents a branch that, were it to exist, would have the same business mix as Hafnarfjörður and would have an efficiency of 100%. Now, since *Best* and Hafnarfjörður have the same business mix, it makes sense to compare them numerically. Here is how to do it: Hafnarfjörður's relative position in the ruler line from the worst branch with the same business mix (the origin) to the best branch with the same business mix (*Best*) is 36%. In other words, 36% of the ruler line is before Hafnarfjörður, and 64% of the ruler line is after Hafnarfjörður. So, it makes sense to say that Hafnarfjörður is, relative to the best branches, 36% efficient — or 64% inefficient, if you like.

So, given the graphical consideration above we have the following definition for the (relative) efficiency of a DMU with two outputs and one input:

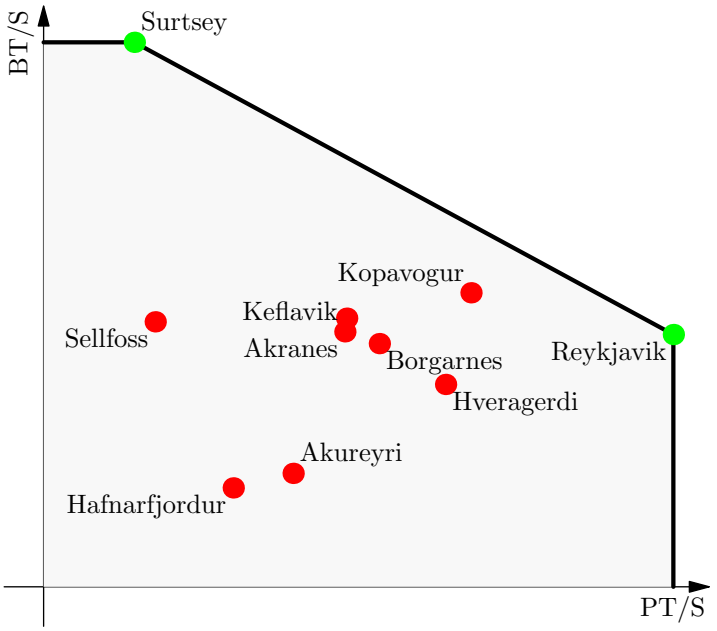
> Draw a line segment from the origin through the DMU in question until you hit the efficient frontier. The **DEA efficiency** of the DMU is
>
> $$\frac{\text{length of the line segment from the origin to the DMU}}{\text{total length of the line segment}} \times 100\%.$$

The picture — and the definition — above is relative: You can change the scale of either the PT/S or the BT/S axis, or even switch the axes, but the relative efficiency of the Hafnarfjörður branch — or any other branch — won't change.

Let us ten illustrate the relative nature of the DEA efficiencies. We shall ad two extra branches to Example 6.1.3 — Surtsey and Flatey — and see what happens.

6.1.4 Example (Kaupþing IV)**.** Suppose we have an extra branch, Surtsey, added to the branches of Example 6.1.3. Assume that Surtsey has 1 personal transactions per staff member, and 6 business transactions per staff member. What changes in the efficiency analysis as a result of including the extra branch Surtsey?

(There are actually no Kaupþing Bank branches in Surtsey. There are no people in Surtsey: People are not allowed in the Fire-Demon's island. There are only puffins is Surtsey.) The effect of including Surtsey to the graphical DEA can be seen in the next picture:

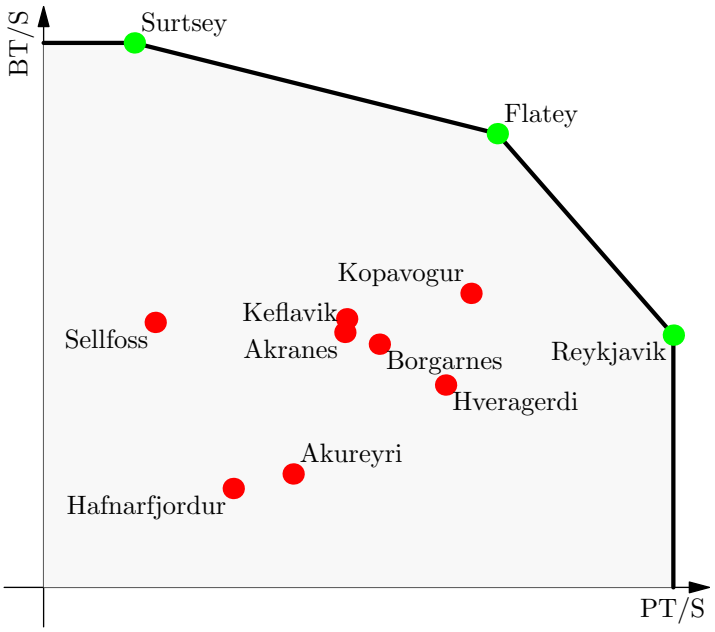


Note that the efficient frontier now excludes Kópavogur. Also, note that the relative efficiencies of most (if not all) of the inefficient branches have changed.

In the above it is clear why Reykjavík and Surtsey have a relative efficiency of 100% (i.e. are efficient): Both are the top performers with respect to one of the two ratios we are considering. The example below, where we have added the Flatey branch, illustrates that a branch can be efficient even if it is not a top performer. In the diagram below Flatey is efficient since under DEA it is judged to have "strength with respect to both ratios", even though it is not the top performer in either.

6.1.5 Example (Kaupþing V)**.** Suppose we have an extra branch, Flatey, added to the branches of Example 6.1.3. Assume that Flatey has 5 personal transactions per staff member, and 5 business transactions per staff member. What changes as a result of this extra branch being included in the analysis?

Here is the new picture with Flatey added. Note that Flatey is on the efficient frontier, i.e., 100% efficient, but it is not at top performer in either of the criteria "personal transactions per staff" (PT/S) or "business transactions per staff" (BT/S).



Multiple Input — Multiple Output

In our simple examples 6.1.2, 6.1.3, 6.1.4, and 6.1.5 of the Kaupþing Bank branches we had just one input and two outputs. This is ideal for a simple graphical analysis. If we have more inputs or outputs then drawing simple pictures is not possible without sculptures. However, it is still possible to carry out exactly the same analysis as before, but using mathematics rather than pictures.

In words DEA, in evaluating any number of DMUs, with any number of inputs and outputs:

1. requires the inputs and outputs for each DMU to be specified,
2. defines efficiency for each DMU as a weighted sum of outputs divided by a weighted sum of inputs, where
3. all efficiencies are restricted to lie between zero and one (i.e. between 0% and 100%),
4. in calculating the numerical value for the efficiency of a particular DMU weights are chosen so as to maximize its efficiency, thereby presenting the DMU in the best possible light.

How to carry out the vague four-point list presented above is the topic of the next section 6.2.

6.2 Charnes–Cooper–Rhodes Model

Now we consider mathematically what we have considered graphically in Section 6.1. We consider n Decision Making Units (DMUs). We call them unimaginatively as DMU_1, DMU_2, DMU_3, and so on upto DMU_n. We are interested in assigning a measure of relative efficiency for each DMU without resorting to any other data than the one provided by the inputs and output of the DMUs themselves.

Data Envelopment Analysis with Matrices

We assume that each DMU has m inputs and s outputs. So, the m inputs of the DMU_k are

$$\mathbf{X}_{\bullet k} = \begin{bmatrix} X_{1k} \\ \vdots \\ X_{mk} \end{bmatrix}.$$

In the same way the s outputs of the DMU_k are

$$\mathbf{Y}_{\bullet k} = \begin{bmatrix} Y_{1k} \\ \vdots \\ Y_{sk} \end{bmatrix}.$$

If we collect the inputs and the outputs into single matrices we have the **input matrix**

$$\mathbf{X} = [X_{jk}] = [\mathbf{X}_{\bullet 1} \cdots \mathbf{X}_{\bullet n}] = \begin{bmatrix} X_{11} & \cdots & X_{1n} \\ \vdots & \ddots & \vdots \\ X_{m1} & \cdots & X_{mn}, \end{bmatrix}$$

and the **output matrix**

$$\mathbf{Y} = [Y_{ik}] = [\mathbf{Y}_{\bullet 1} \cdots \mathbf{Y}_{\bullet n}] = \begin{bmatrix} Y_{11} & \cdots & Y_{1n} \\ \vdots & \ddots & \vdots \\ Y_{s1} & \cdots & Y_{sn} \end{bmatrix}.$$

So,

$$\begin{aligned} X_{jk} &= \text{the input } j \text{ of the } \mathrm{DMU}_k, \\ Y_{ik} &= \text{the output } i \text{ of the } \mathrm{DMU}_k. \end{aligned}$$

Charnes–Cooper–Rhodes Fractional Program

Given what we have learned it seems reasonable to measure the (relative) efficiency of the DMUs as weighted sums. So, let

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_s \end{bmatrix}$$

be the weights associated with the s outputs of the DMUs. Similarly, let

$$\mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix}$$

be the weights associated with the inputs of the DMUs. Then the **weighted efficiency**, with weights $\mathbf{u}$ and $\mathbf{v}$, of any DMU, say DMU_o (o for DMU under observation) is

$$\begin{aligned} h_o(\mathbf{u},\mathbf{v}) &= \text{the } (\mathbf{u},\mathbf{v}) \text{ weighted efficiency of } \text{DMU}_o \\ &= \frac{\mathbf{u} \text{ weighted outputs of } \text{DMU}_o}{\mathbf{v} \text{ weighted inputs of } \text{DMU}_o} \\ &= \frac{\sum_{j=1}^{s} u_j Y_{jo}}{\sum_{i=1}^{m} v_i X_{io}} \\ &= \frac{\mathbf{u}'\mathbf{Y}_{\bullet o}}{\mathbf{v}'\mathbf{X}_{\bullet o}}. \end{aligned} \tag{6.2.1}$$

6.2.2 Example. Consider the Kaupþing Bank's branches of Example 6.1.2 of the previous section:

Branch	**Outputs and inputs**		
	Personal transactions	Business transactions	Number of staff
Reykjavík	125	50	18
Akureyri	44	20	16
Kópavogur	80	55	17
Hafnarfjörður	23	12	11

Denote the data of Example 6.2.2 by

$$\begin{aligned} \mathbf{X}_{1\bullet} &= \text{number of staff,} \\ \mathbf{Y}_{1\bullet} &= \text{number of personal transactions,} \\ \mathbf{Y}_{2\bullet} &= \text{number of business transactions.} \end{aligned}$$

So, e.g., $\mathbf{X}_{1\bullet}$ is the 4-dimensional row vector consisting of the number of staff data for the DMUs Reykjavík, Akureyri, Kópavogur, and Hafnarfjörður. Similarly, $\mathbf{Y}_{1\bullet}$ and $\mathbf{Y}_{2\bullet}$ are the 4-dimensional row vectors indicating the number of personal and business transactions for each of the four DMUs: Reykjavík (1), Akureyri (2),

Kópavogur (3), and Hafnarfjörður (4). The output matrix for this example is:

$$\begin{aligned}\mathbf{Y} &= \begin{bmatrix} \mathbf{Y}_{1\bullet} \\ \mathbf{Y}_{2\bullet} \end{bmatrix} \\ &= \begin{bmatrix} Y_{11} & Y_{12} & Y_{13} & Y_{14} \\ Y_{21} & Y_{22} & Y_{23} & Y_{24} \end{bmatrix} \\ &= \begin{bmatrix} 125 & 44 & 80 & 23 \\ 50 & 20 & 55 & 12 \end{bmatrix}.\end{aligned}$$

The input matrix is

$$\begin{aligned}\mathbf{X} &= \mathbf{X}_{1\bullet} \\ &= [X_{11}\ X_{12}\ X_{13}\ X_{14}] \\ &= [18\ 16\ 17\ 11].\end{aligned}$$

Let us then take the Hafnarfjörður branch under consideration. So, $\mathrm{DMU}_o = \mathrm{DMU}_4$ is Hafnarfjörður. With our vector notation Hafnarfjörður would have the (weighted) efficiency

$$h_o(\mathbf{u},\mathbf{v}) \quad = \quad \frac{u_1 Y_{1o} + u_2 Y_{2o}}{v_1 X_{1o}} \quad = \quad \frac{u_1 \times 23 + u_2 \times 12}{v_1 \times 11}.$$

Now there is the problem of fixing the weight $\mathbf{u}$ and $\mathbf{v}$ of the outputs and the inputs. Each DMU would — of course — want to fix the weights $\mathbf{u}$ and $\mathbf{v}$ in such a way that they would look best in comparison with the other DMUs. So, it is in the interests of each and every one of the DMUs to maximize the weighted efficiency $h_o(\mathbf{u},\mathbf{v})$. In particular, this means that Hafnarfjörður faces an optimization problem

$$\max_{\mathbf{u},\mathbf{v}} h_o(\mathbf{u},\mathbf{v}) \quad = \quad \max_{u_1,u_2,v_1} \frac{u_1 \times 23 + u_2 \times 12}{v_1 \times 11}. \tag{6.2.3}$$

Obviously there must be constraints to the decision variables $\mathbf{u}$ and $\mathbf{v}$. Indeed, otherwise the optimization problem (6.2.3) would yield an unbounded optimum. So, what are the constraints? Well, obviously we have the sign constraints

$$\mathbf{u},\mathbf{v} \geq \mathbf{0}.$$

This does not help too much yet, though. The optimum of (6.2.3) is still unbounded. Now, we remember that we are dealing with efficiencies. But, an efficiency is a number between 0% and 100%. So, we have the constraint

$$0 \quad \leq \quad h_o(\mathbf{u},\mathbf{v}) \quad \leq \quad 1.$$

This does not help too much either. Indeed, now the optimum for (6.2.3) would be 1, or 100%. But we are close now. Remember that the efficiency is **always** a

number between 0% and 100%. So, the efficiencies of the other DMUs must also be between 0% and 100%. So, we let Hafnarfjörður set the weights $\mathbf{u}$ and $\mathbf{v}$, and the other DMUs are then measured in the same way. So, the constraints are

$$0 \quad \leq \quad h_k(\mathbf{u}, \mathbf{v}) \quad \leq \quad 1 \qquad \text{for all DMU}_k,\ k = 1, \ldots, n.$$

Collecting what we have found above we have found the **fractional form** of the Charnes–Cooper–Rhodes (CCR) model:

The **CCR fractional program** for DMU_o relative to $\text{DMU}_1, \ldots, \text{DMU}_n$ is

$$\begin{aligned} \max \theta &= \frac{\mathbf{u}'\mathbf{Y}_{\bullet o}}{\mathbf{v}'\mathbf{X}_{\bullet o}} \\ \text{s.t.} \quad & \frac{\mathbf{u}'\mathbf{Y}_{\bullet k}}{\mathbf{v}'\mathbf{X}_{\bullet k}} \leq 1 \qquad \text{for all } k = 1, \ldots, n \\ & \mathbf{u}, \mathbf{v} \geq 0. \end{aligned} \tag{6.2.4}$$

The figure θ is the DMU_o's **DEA efficiency**.

We have dropped the zero lower bounds for the efficiencies in (6.2.4). The reason is, that they are not necessarily needed. It would not, however, be wrong to keep them.

Charnes–Cooper–Rhodes Linear Program

Consider the CCR fractional program (6.2.4) in the previous subsection. This is not an LP. But the name of this part of the chapters is "Linear Models". So, it seems that we have a misnomer! Also, we do not know how to solve fractional programs like (6.2.4). Fortunately there is a way of transforming the fractional program (6.2.4) into an LP.

Before going to the LP let us note that while the efficiency score θ of the CCR fractional program (6.2.4) is unique, there are many different weights $\mathbf{u}, \mathbf{v}$ that give the same efficiency. Indeed, if the weights $\mathbf{u}, \mathbf{v}$ give the optimal efficiency, then so do the weights $\alpha\mathbf{u}, \alpha\mathbf{v}$ for any $\alpha > 0$. This is due to the fact that we are dealing with ratios. Indeed, for any $\alpha > 0$

$$\begin{aligned} h_o(\mathbf{u}, \mathbf{v}) &= \frac{\mathbf{u}'\mathbf{Y}_{\bullet o}}{\mathbf{v}'\mathbf{X}_{\bullet o}} \\ &= \frac{\alpha\mathbf{u}'\mathbf{Y}_{\bullet o}}{\alpha\mathbf{v}'\mathbf{X}_{\bullet o}} \\ &= h_o(\alpha\mathbf{u}, \alpha\mathbf{v}). \end{aligned}$$

There is an easy way out, however. We just normalize the denominator in the ratio, i.e., we insist that

$$\mathbf{v}'\mathbf{X}_{\bullet o} \quad = \quad 1.$$

Now we are ready to give the LP formulation of the fractional program 6.2.4:

The **CCR LP** for DMU_o relative to $\mathrm{DMU}_1, \ldots, \mathrm{DMU}_n$ is

$$\begin{aligned} & \max \theta = \mathbf{u}'\mathbf{Y}_{\bullet o} \\ & \text{s.t.} \quad \mathbf{v}'\mathbf{X}_{\bullet o} = 1, \\ & \qquad \mathbf{u}'\mathbf{Y} \leq \mathbf{v}'\mathbf{X} \\ & \qquad \mathbf{u}, \mathbf{v} \geq 0. \end{aligned} \tag{6.2.5}$$

The figure θ is the DMU_o's **DEA efficiency**.

It may not be immediately clear why the LP (6.2.5) is the same optimization problem as the fractional program (6.2.4). So, we explain a little why this is so. Consider the fractional program (6.2.4). First, note that the extra assumption $\mathbf{v}'\mathbf{X}_{\bullet o} = 1$ does not change the optimal value of θ in the fractional program. Indeed, we have already seen that this restriction merely chooses one optimal choice among many. Next note that in the LP (6.2.5) we have

$$\theta \quad = \quad \mathbf{u}'\mathbf{Y}_{\bullet o},$$

while in the fractional program (6.2.4) we have

$$\theta \quad = \quad \frac{\mathbf{u}'\mathbf{Y}_{\bullet o}}{\mathbf{v}'\mathbf{X}_{\bullet o}}.$$

Remember, that we have now the normalizing assumption $\mathbf{v}'\mathbf{X}_0 = 1$. So, we see that the fractional and linear objectives are actually the same. Finally, let us look the constraints

$$\frac{\mathbf{u}'\mathbf{Y}_{\bullet k}}{\mathbf{v}'\mathbf{X}_{\bullet k}} \quad \leq \quad 1$$

of the fractional program (6.2.4) and compare them to the constraints

$$\mathbf{u}'\mathbf{Y}_{\bullet k} \quad \leq \quad \mathbf{v}'\mathbf{X}_{\bullet k}$$

of the linear program (6.2.5) (written in the matrix form there). If you multiply both sides of the fractional programs constraints by $\mathbf{v}'\mathbf{X}_{\bullet k}$ you notice that these constraints are actually the same. So, we see that the fractional program (6.2.4) and the linear program (6.2.5) are the same.

DEA Efficiency for Hafnarfjörður

Let us calculate mathematically, as opposed to graphically, Hafnarfjörður's DEA efficiency in Example 6.2.2 by using the CCR LP (6.2.5). Recall the data

Branch	**Outputs and inputs**		
	Personal transactions	Business transactions	Number of staff
Reykjavík	125	50	18
Akureyri	44	20	16
Kópavogur	80	55	17
Hafnarfjörður	23	12	11

and the notation

$$\begin{aligned} \mathbf{X}_{1\bullet} &= \text{number of staff,} \\ \mathbf{Y}_{1\bullet} &= \text{number of personal transactions,} \\ \mathbf{Y}_{2\bullet} &= \text{number of business transactions.} \end{aligned}$$

Note that $\mathbf{X}_{1\bullet}, \mathbf{Y}_{1\bullet}, \mathbf{Y}_{2\bullet}$ are **not** the decision variables. They are the data. The decision variables are the weights v_1, u_1, u_2.

Here is the CCR LP for Hafnarfjörður

$$\begin{array}{lrcrcrcrl} \max \theta & = & 23u_1 & + & 12u_2 & & & & \text{(DEA Efficiency)} \\ \text{s.t.} & & & & & 11v_1 & = & 1 & \text{(Normalization)} \\ & & 125u_1 & + & 50u_2 & & \leq & 18v_1 & \text{(DMU Reykjavik)} \\ & & 44u_1 & + & 20u_2 & & \leq & 16v_1 & \text{(DMU Akureyri)} \\ & & 80u_1 & + & 55u_2 & & \leq & 17v_1 & \text{(DMU Kopavogur)} \\ & & 23u_1 & + & 12u_2 & & \leq & 11v_1 & \text{(DMU Hafnarfjordur)} \\ & & & & & u_1, u_2, v_1 & \geq & 0 & \end{array}$$

Now, this LP is certainly not in standard form. To use `glpk` we need at least to remove the variables v_1 from the RHSs. But this is easy enough: We just subtract them and we obtain the LP

$$\begin{array}{lrcrcrcrl} \max \theta & = & 23u_1 & + & 12u_2 & & & & \text{(DEA Efficiency)} \\ \text{s.t.} & & & & & 11v_1 & = & 1 & \text{(Normalization)} \\ & & 125u_1 & + & 50u_2 & - \; 18v_1 & \leq & 0 & \text{(DMU Reykjavik)} \\ & & 44u_1 & + & 20u_2 & - \; 16v_1 & \leq & 0 & \text{(DMU Akureyri)} \\ & & 80u_1 & + & 55u_2 & - \; 17v_1 & \leq & 0 & \text{(DMU Kopavogur)} \\ & & 23u_1 & + & 12u_2 & - \; 11v_1 & \leq & 0 & \text{(DMU Hafnarfjordur)} \\ & & & & & u_1, u_2, v_1 & \geq & 0 & \end{array}$$

This LP is pretty close to standard form and the following Octave code solves it:

```
octave:1> theta=[23 12 0]';
octave:2> A=[
> 0 0 11;
> 125 50 -18;
> 44 20 -16;
> 80 55 -17;
> 23 12 -11];
octave:3> b=[1 0 0 0 0]';
octave:4> [coeff_max,theta_max]=glpk(theta,A,b,[0 0 0]',[],"SUUUU","CCC",-1)
coeff_max =

   0.0044269
   0.0216601
   0.0909091

theta_max =  0.36174
```

So, we see that Hafnarfjörðurs DEA efficiency is 36%. Also, we see that Hafnarfjörður weighs the personal transactions with 4.4%, business transactions with 2.2% (the outputs) and the number of staff with 0.9% (the input). Sometimes these weights may be interesting, but in general they are not.

6.3 Charnes–Cooper–Rhodes Model's Dual

Finding the DEA efficiency of a DMU_o in the CCR model is an LP (6.2.5). So, it must have a dual LP associated with it. In this section we explain how to construct the dual, and how to interpret it. Also, in the last subsection we illustrate how to find the DEA efficiency for Hafnarfjörður branch in Example 6.2.2 by using the the dual of the CCR model.

Finding Dual

In this subsection we will find the dual of the CCR LP (6.2.5) by using brute force matrix calculus. We throw all intuition to the wind, and simply follow mathematical algorithms. In the next subsection we shall interpret the dual we find in this subsection.

Recall the LP form of the CCR model:

$$\begin{aligned} &\max \theta = \mathbf{u}'\mathbf{Y}_{\bullet o} \\ &\text{s.t.} \quad \mathbf{v}'\mathbf{X}_{\bullet o} = 1, \\ &\qquad \mathbf{u}'\mathbf{Y} \leq \mathbf{v}'\mathbf{X} \\ &\qquad \mathbf{u}, \mathbf{v} \geq 0. \end{aligned} \tag{6.3.1}$$

To find the dual, we write the LP (6.3.1) in standard form. We could use the block matrix notation, but the derivation is probably easier to understand if we do not use the concise matrix notation. So, we abandon matrices in the derivation.

Without matrices the CCR LP (6.3.1) can be written as

$$\begin{array}{lrcl} \max\ \theta\ = & u_1Y_{1o} + \cdots + u_sY_{so} & & \\ \text{s.t.} & v_1X_{1o} + \cdots + v_mX_{mo} & = & 1 \\ & u_1Y_{11} + \cdots + u_sY_{s1} & \leq & v_1X_{11} + \cdots + v_mX_{mo} \\ & \vdots & \vdots & \vdots \\ & u_1Y_{1n} + \cdots + u_sY_{sn} & \leq & v_1X_{1n} + \cdots + v_mX_{mn} \\ & u_1, \ldots, u_s, v_1, \ldots, v_m & \geq & 0. \end{array} \tag{6.3.2}$$

The LP (6.3.2) is certainly not in standard form. Actually, it is pretty far from it. As a first step in transforming (6.3.2) into a standard form let us write it so that the decision variables $\mathbf{u} = [u_1 \ \cdots \ u_s]'$ and $\mathbf{v} = [v_1 \ \cdots \ v_m]'$ are in the right places, i.e., coefficients are in front of the decision variables, all the decision

variables are represented everywhere, and there are no decision variables in the RHSs. We obtain:

$$
\begin{array}{lrcrcrcrcl}
\max\theta = & Y_{1o}u_1 & + \cdots + & Y_{so}u_s & +0v_1 & + \cdots + & 0v_m & & \\
\text{s.t.} & 0u_1 & + \cdots + & 0u_s & +X_{1o}v_1 & + \cdots + & X_{mo}v_m & = & 1 \\
& Y_{11}u_1 & + \cdots + & Y_{s1}u_s & -X_{11}v_1 & - \cdots - & X_{m1}v_m & \leq & 0 \\
& \vdots & & \vdots & \vdots & & \vdots & \vdots & \vdots \\
& Y_{1n}u_1 & + \cdots + & Y_{sn}u_s & -X_{1n}v_1 & - \cdots - & X_{mn}v_m & \leq & 0 \\
& u_1 & \ldots & u_s & v_1 & \ldots & v_m & \geq & 0
\end{array}
\tag{6.3.3}
$$

Next, we split the equality constraint in (6.3.3) into two $\leq$ inequalities. We obtain:

(6.3.4)

$$
\begin{array}{lrcrcrcrcl}
\max\theta = & Y_{1o}u_1 & + \cdots + & Y_{so}u_s & +0v_1 & + \cdots + & 0v_m & & \\
\text{s.t.} & 0u_1 & + \cdots + & 0u_s & +X_{1o}v_1 & + \cdots + & X_{mo}v_m & \leq & 1 \\
& -0u_1 & - \cdots - & -0u_s & -X_{1o}v_1 & - \cdots - & -X_{mo}v_m & \leq & -1 \\
& Y_{11}u_1 & + \cdots + & Y_{s1}u_s & -X_{11}v_1 & - \cdots - & X_{m1}v_m & \leq & 0 \\
& \vdots & & \vdots & \vdots & & \vdots & \vdots & \vdots \\
& Y_{1n}u_1 & + \cdots + & Y_{sn}u_s & -X_{1n}v_1 & - \cdots - & X_{mn}v_m & \leq & 0 \\
& u_1 & \ldots & u_s & v_1 & \ldots & v_m & \geq & 0
\end{array}
$$

Now it is pretty straightforward to transform the LP (6.3.3) into the dual. Let ϑ be the objective, and let $\mu = [\mu_1\ \mu_2\ \mu_3\ \cdots \mu_{n+2}]'$ be the decision variables. We obtain:

$$
\begin{array}{lrcrcrcl}
\min\vartheta = & \mu_1 - \mu_2 & & & & & & \\
\text{s.t.} & 0\mu_1 - 0\mu_2 & +Y_{11}\mu_3 & + \cdots + & Y_{1n}\mu_{n+2} & \geq & Y_{1o} \\
& \vdots & & \vdots & & \vdots & \vdots \\
& 0\mu_1 - 0\mu_2 & +Y_{s1}\mu_3 & + \cdots + & Y_{sn}\mu_{n+2} & \geq & Y_{so} \\
& X_{1o}\mu_1 - X_{1o}\mu_2 & -X_{11}\mu_3 & - \cdots - & X_{1n}\mu_{n+2} & \geq & 0 \\
& \vdots & & \vdots & & \vdots & \vdots \\
& X_{mo}\mu_1 - X_{mo}\mu_2 & -X_{m1}\mu_3 & - \cdots - & X_{mn}\mu_{n+2} & \geq & 0 \\
& & & & \mu_1, \ldots, \mu_{n+2} & \geq & 0
\end{array}
\tag{6.3.5}
$$

We have found the dual (6.3.5) of the CCR LP (6.2.5). Unfortunately, this dual is not easy to interpret. So, we have to transform it slightly in order to understand what is going on. This is what we do in the next subsection.

Interpreting Dual

Let us substitute the objective

$$\vartheta \quad = \quad \mu_1 - \mu_2$$

into the constraints of (6.3.5). In doing so, we actually eliminate all the occurrences on μ_1 and μ_2 in the system. We obtain:

$$
\begin{array}{lrcrcl}
\min \vartheta & & & & & \\
\text{s.t.} & & Y_{11}\mu_3 & +\cdots+ & Y_{1n}\mu_{n+2} & \geq & Y_{1o} \\
 & & & \vdots & & \vdots & \vdots \\
 & & Y_{s1}\mu_3 & +\cdots+ & Y_{sn}\mu_{n+2} & \geq & Y_{so} \\
 & X_{1o}\vartheta & -X_{11}\mu_3 & -\cdots- & X_{1n}\mu_{n+2} & \geq & 0 \\
 & \vdots & & \vdots & & \vdots & \vdots \\
 & X_{mo}\vartheta & -X_{m1}\mu_3 & -\cdots- & X_{mn}\mu_{n+2} & \geq & 0 \\
 & & & & \vartheta, \mu_3, \ldots, \mu_{n+2} & \geq & 0
\end{array}
\tag{6.3.6}
$$

Next, we shall renumber the remaining decision variables. The new decision variables will be ϑ and $\lambda = [\lambda_1 \ \cdots \ \lambda_n]'$, where $\lambda_1 = \mu_3$, $\lambda_2 = \mu_4$, $\ldots$, $\lambda_n = \mu_{n+2}$. So, the LP (6.3.6) becomes

$$
\begin{array}{lrcrcl}
\min \vartheta & & & & & \\
\text{s.t.} & & +Y_{11}\lambda_1 & +\cdots+ & Y_{1n}\lambda_n & \geq & Y_{1o} \\
 & & & \vdots & & \vdots & \vdots \\
 & & +Y_{s1}\lambda_1 & +\cdots+ & Y_{sn}\lambda_n & \geq & Y_{so} \\
 & X_{1o}\vartheta & -X_{11}\lambda_1 & -\cdots- & X_{1n}\lambda_n & \geq & 0 \\
 & \vdots & & \vdots & & \vdots & \vdots \\
 & X_{mo}\vartheta & -X_{m1}\lambda_1 & -\cdots- & X_{mn}\lambda_n & \geq & 0 \\
 & & & & \vartheta, \lambda_1, \ldots, \lambda_n & \geq & 0
\end{array}
\tag{6.3.7}
$$

Finally, we reorganize the $\geq$ inequalities, for a reason that will become apparent later when we get to the interpretation. We obtain:

$$
\begin{array}{lrcrcl}
\min \vartheta & & & & & \\
\text{s.t.} & Y_{11}\lambda_1 & +\cdots+ & Y_{1n}\lambda_n & \geq & Y_{1o} \\
 & & \vdots & & \vdots & \vdots \\
 & Y_{s1}\lambda_1 & +\cdots+ & Y_{sn}\lambda_n & \geq & Y_{so} \\
 & X_{11}\lambda_1 & +\cdots+ & X_{1n}\lambda_n & \leq & X_{1o}\vartheta \\
 & & \vdots & & \vdots & \vdots \\
 & X_{m1}\lambda_1 & +\cdots+ & X_{mn}\lambda_n & \leq & X_{mo}\vartheta \\
 & & & \vartheta, \lambda_1, \ldots, \lambda_n & \geq & 0
\end{array}
\tag{6.3.8}
$$

We have found out a formulation of the dual of the CCR LP (6.2.5) that we can understand in a meaningful way: The dual variables

$$
\lambda \ = \ \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix}
$$

are the weights for a **virtual DMU**, denoted by $\mathrm{DMU}_{\mathrm{virtual}}$. The virtual DMU is the reference point for the DMU_o, the DMU under observation. Indeed, the virtual DMU will have the same **business mix** as DMU_o, but the virtual DMU is 100% DEA efficient. The virtual DMU is constructed from the actual DMUs by weighting them with the vector λ:

$$\mathrm{DMU}_{\mathrm{virtual}} \quad = \quad \sum_{k=1}^{n} \lambda_k \mathrm{DMU}_k.$$

Note that the virtual DMU depends on the DMU under observation. Different DMUs have different virtual DMUs associated with them.

Now, the restrictions

$$\begin{array}{ccccc} Y_{11}\lambda_1 & +\cdots+ & Y_{1n}\lambda_n & \geq & Y_{1o} \\ & \vdots & & \vdots & \vdots \\ Y_{s1}\lambda_1 & +\cdots+ & Y_{sn}\lambda_n & \geq & Y_{so} \end{array}$$

can be interpreted as

> All the outputs of the associated virtual DMU are at least as great as the corresponding outputs of the DMU under observation,

and the restrictions

$$\begin{array}{ccccc} X_{11}\lambda_1 & +\cdots+ & X_{1n}\lambda_n & \leq & X_{1o}\vartheta \\ & \vdots & & \vdots & \vdots \\ X_{m1}\lambda_1 & +\cdots+ & X_{mn}\lambda_n & \leq & X_{mo}\vartheta \end{array}$$

can be interpreted as

> If the inputs of the DMU under observation are scaled down by ϑ, then all the inputs are at least as great as the corresponding inputs of the associated virtual DMU.

Finally, here is the CCR dual LP (6.3.8) in matrix form:

> The **CCR dual LP** for DMU_o relative to $\mathrm{DMU}_1, \ldots, \mathrm{DMU}_n$ is

(6.3.9)
$$\begin{aligned} &\min \vartheta \\ &\text{s.t.} \quad \vartheta \mathbf{X}_{\bullet o} \geq \mathbf{X}\lambda, \\ &\qquad \mathbf{Y}\lambda \geq \mathbf{Y}_{\bullet o} \\ &\qquad \vartheta, \lambda \geq 0. \end{aligned}$$

> The figure ϑ is the DMU_o's **DEA efficiency**. The decision variables λ are the coefficients of the virtual DMU associated with DMU_o.

Dual DEA Efficiency for Hafnarfjörður

Let us see what are the (dual) DEA efficiencies for the Hafnarfjörður in Example 6.2.2. We have already found out the solution to be 36% in the previous sections by using the graphical approach and the primal CCR approach. So, we shall now check if this third approach, the dual CCR approach, will give the same results, as it should.

The CCR dual LP for Hafnarfjörður is

$$
\begin{array}{lrrrrcr}
& \min \vartheta & & & & & \\
\text{s.t.} & 18\lambda_1 & +16\lambda_2 & +17\lambda_3 & +11\lambda_4 & \leq & 11\vartheta \\
& 125\lambda_1 & +44\lambda_2 & +80\lambda_3 & +23\lambda_4 & \geq & 23 \\
& 50\lambda_1 & +20\lambda_2 & +55\lambda_3 & +12\lambda_4 & \geq & 12 \\
& & & \vartheta, \lambda_1, \lambda_2, \lambda_3, \lambda_4 & & \geq & 0
\end{array}
\tag{6.3.10}
$$

Once again, the LP (6.3.10) is pretty far from a standard form. So, we transform it into a more standard from. Only thing that needs to be done, is to move the variable ϑ to the LHS of the first inequality, and to note that we may well assume that $\vartheta \geq 0$:

$$
\begin{array}{lrrrrrcr}
& \min \vartheta & & & & & & \\
\text{s.t.} & -11\vartheta & +18\lambda_1 & +16\lambda_2 & +17\lambda_3 & +11\lambda_4 & \leq & 0 \\
& & 125\lambda_1 & +44\lambda_2 & +80\lambda_3 & +23\lambda_4 & \geq & 23 \\
& & 50\lambda_1 & +20\lambda_2 & +55\lambda_3 & +12\lambda_4 & \geq & 12 \\
& & & & \vartheta, \lambda_1, \lambda_2, \lambda_3, \lambda_4 & & \geq & 0
\end{array}
$$

Now the LP in in a form that can be fed to Octave's `glpk`. Here is the code:

```
octave:1> c=[1 0 0 0 0]';
octave:2> b=[0 23 12]';
octave:3> A=[
> -11 18 16 17 11;
>  0 125 44 80 23;
>  0  50 20 55 12;
>  ];
octave:4> [x_max,z_max]=glpk(c,A,b,[0 0 0 0 0]',[],"ULL","CCCCC",1)
x_max =

   0.36174
   0.10609
   0.00000
   0.12174
   0.00000

z_max =  0.36174
```

We see that the (dual) DEA efficiency for Hafnarfjörður is 36%, as it should. We

can also read the composition of the virtual DMU associated with Hafnarfjörður:

$$\begin{aligned} \text{DMU}_{\text{virtual}} &= \lambda_1 \text{DMU}_1 + \lambda_2 \text{DMU}_2 + \lambda_3 \text{DMU}_3 + \lambda_4 \text{DMU}_4 \\ &= 10.6\% \times \text{DMU}_{\text{Reykjavik}} + 12.2\% \times \text{DMU}_{\text{Kopavogur}}. \end{aligned}$$

So, one way to interpret the result is:

> Consider the virtual DMU that is composed of 10.6% of Reykjavík and 12.2% of Kópavogur. This virtual DMU has the same business mix as Hafnarfjörður, it uses only 36% of the inputs Hafnarfjörður uses.

6.4 Strengths and Weaknesses of Data Envelopment Analysis

Data Envelopment Analysis is a very general framework that draws conclusions from the data available and makes very few, if any, assumptions of the context where the data came from. This is its main strength and this its main weakness.

Strengths

1. DEA is simple enough to be modeled with LPs.
2. DEA can handle multiple input and multiple outputs.
3. DEA does not require an assumption of a functional form relating inputs to outputs. In particular, one does not have to think that the outputs are consequences of the inputs.
4. DMUs are directly compared against a peer or (virtual) combination of peers.
5. Inputs and outputs can have very different units. For example, output Y_1 could be in units of lives saved and input X_1 could be in units of Euros without requiring an a priori trade-off between the two.

Weaknesses

1. Since a standard formulation of DEA creates a separate LP for each DMU, large problems can be computationally intensive.
2. Since DEA is an extreme point technique, noise (even symmetrical noise with zero mean) such as measurement error can cause significant problems.
3. DEA is good at estimating relative efficiency of a DMU but it converges very slowly to "absolute" efficiency. In other words, it can tell you how well you are doing compared to your peers but not compared to a "theoretical maximum".
4. Since DEA is a nonparametric technique, statistical hypothesis tests are difficult to apply in the DEA context.

5. DEA is very generous: If a DMU excels in just one output (or input) it is likely to get 100% efficiency, even if it performs very badly in all the other outputs (or inputs). So, if there are many outputs (or inputs) one is likely to get 100% efficiency for all the DMUs, which means that DEA cannot differentiate between the DMUs (it does not mean that all the DMUs are doing well in any absolute sense).

6.5 Exercises and Projects

6.1. Calculate the DEA efficiencies for all the branches in the examples

(a) 6.1.2,
(b) 6.1.3,
(c) 6.1.4,
(d) 6.1.5.

6.2. The S faculty of SFU University has four departments: MS, P, C, and CS. We have the following data indicating the number of professors (P), lecturers (L), and assistants (A) for each department together with the number of bachelor (B), master (M), and doctor (D) degrees granted by each of the departments last year, and the number of refereed journal articles (J) produced by the staff of each of the department last year

	Inputs and outputs						
Dept.	P	L	A	B	M	D	J
MS	3	3	2	10	5	5	24
P	1	5	3	14	9	1	4
C	6	4	20	10	15	11	1
CS	7	6	4	21	11	10	7

Calculate the relative DEA efficiencies of the departments.

6.3. Consider the departments of the SFU University's S faculty of Exercise 6.2.

(a) The SFU University wants to move the MS department from the S faculty over to the B faculty. How would this change the relative DEA efficiencies of the remaining S faculty departments?
(b) Suppose the SFU University decides it does not care about research. So, the output refereed journal articles (J) becomes obsolete. How does this effect the relative DEA efficiencies of the departments MS, P, C, and CS?

6.4. * Make an Octave function

```
[eff, lambda] = dea_eff(X, Y)
```

that returns DEA efficiencies `eff` and the coefficients `lambda` of their corresponding virtual best DEA. The input parameters for `dea_eff` are the input matrix `X` and the output matrix `Y`.

Chapter 7

Transportation-Type Models

> We willingly pay 30,000–40,000 fatalities per year for the advantages of individual transportation by automobile. — John von Neumann

> Bypasses are devices that allow some people to dash from point A to point B very fast while other people dash from point B to point A very fast. People living at point C, being a point directly in between, are often given to wonder what's so great about point A that so many people from point B are so keen to get there and what's so great about point B that so many people from point A are so keen to get there. They often wish that people would just once and for all work out where the hell they wanted to be. — Douglas Adams

We consider a set of so-called transportation problems that can be modeled as LPs, and thus solved with, say, the `glpk` LP solver. There are also specialized algorithms for solving these problems that work much faster than the general simplex-type algorithms, but of course work only in the special type problems.

7.1 Transportation Problem

In a **transportation problem** one has to ship products from ports to markets so that the total shipping cost is as small as possible while still meeting the demands of each market.

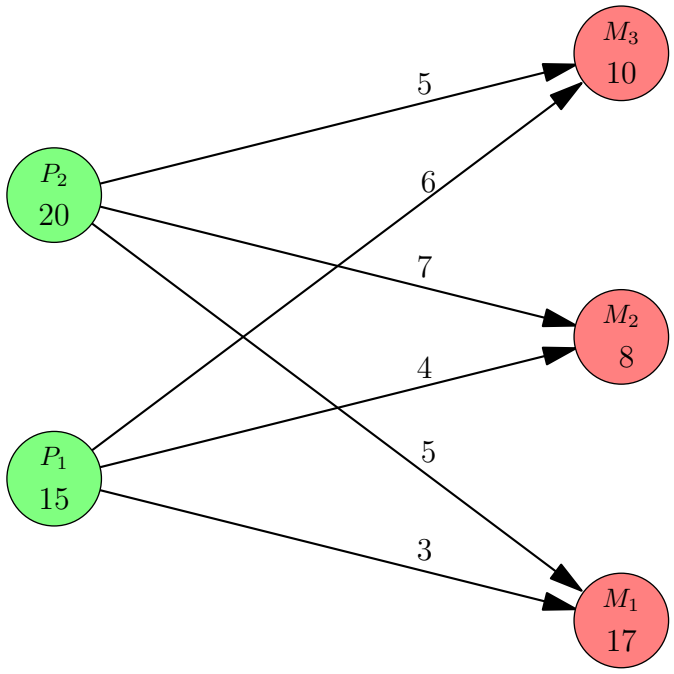
7.1.1 Example (Frábært skyr)**.** Frábært ehf. produces skyr in plants P_1 and P_2. Plant P_1 produces 15 tons of skyr per day, and plant P_2 produces 20 tons of skyr per day. The skyr is shipped to the markets M_1, M_2, and M_3. The market M_1 demands 17 tons of skyr, the market M_2 demands 8 tons of skyr, and the market M_3 demands 10 tons of skyr.

To ship one ton of skyr from plant P_1 to markets M_1, M_2, and M_3 costs €3, €4, and €6, respectively. To ship one ton of skyr from plant P_2 to markets M_1, M_2, and M_3 costs €5, €7, and €5, respectively. Frábært ehf. wants to deliver (all or some of) its skyr to the markets with the minimal possible shipping cost while still meeting the demand of all the markets. How should Frábært ehf. ship its skyr, i.e. what is the optimal transportation schedule for Frábært?

The shipping costs and the supplies and the demands of Frábært's example 7.1.1 can be expressed in the tabular form as

	Market			
Port	M_1	M_2	M_3	*Supply*
P_1	3	4	6	15
P_2	5	7	5	20
Demand	17	8	10	

Frábært ehf.'s problem can also be expressed as a diagram, or a network (which the author finds a much more better way, but your taste may be different even though the tabular form is somewhat more natural for the LP setting):



Transportation Problems as Linear Programs

The Frábært's problem is a typical **transportation problem**, and it, as all transportation problems, can be modeled as an LP. Let us follow Algorithm 1.1.2 to see how to do this:

First we set

$$X_{ij} \quad = \quad \text{tons of skyr shipped from plant } i \text{ to market } j.$$

These are obviously the **decision variables** for Frábært's problem, or for any transportation problem for that matter, save the skyr. Everything else is fixed, and the only thing that is left open is the actual amounts transported from ports to markets. (Note the clever use of indexes here: we consider the decision variables in a matrix form. There is a small price we have to pay for this cleverness later, however.)

The **objective** of any transportation problem is to minimize the total shipping cost. So, Frábært's objective is

$$\min z \quad = \quad \sum_i \sum_j C_{ij} X_{ij} \tag{7.1.2}$$

where

$$C_{ij} \quad = \quad \text{the cost of shipping one ton of skyr from plant } i \text{ to market } j.$$

(Note again the clever use of indexes: the objective is also represented as a matrix. Again, there is a small price we have to pay for this cleverness later.) The objective (7.1.2) is a linear one. Indeed, sums are linear, and double-sums doubly so.

What about the **constraints** then? There are of course the **sign constraints**

$$X_{ij} \quad \geq \quad 0 \qquad \text{for all plants } i \text{ and markets } j.$$

Indeed, it would be pretty hard to transport negative amounts of skyr. There are also the supply and demand constraints: Each plant P_i has only so many tons of skyr it can supply. So, if s_i is the supply limit for plant P_i then we have the **supply constraints**

$$\sum_j X_{ij} \quad \leq \quad s_i \qquad \text{for all plants } i.$$

(Please, do not confuse s_i with a slack variable here. We are sorry for the clashing notation!) Each market also demands so many tons of skyr, and according to the problem we are committed to meet the market demands. So, if d_j is the demand for skyr in the market M_j then we have the **demand constraints**

$$\sum_i X_{ij} \quad \geq \quad d_j \qquad \text{for all markets } j.$$

Here is the LP for Frábært ehf.:

$$
\begin{array}{lrrrrrrrcr}
 & \min z = & 3X_{11} & +4X_{12} & +6X_{13} & +5X_{21} & +7X_{22} & +5X_{23} & & \\
 & \text{s.t.} & X_{11} & +X_{12} & +X_{13} & & & & \leq & 15 \\
 & & & & & X_{21} & +X_{22} & +X_{23} & \leq & 20 \\
(7.1.3) & & X_{11} & & & +X_{21} & & & \geq & 17 \\
 & & & X_{12} & & & +X_{22} & & \geq & 8 \\
 & & & & X_{13} & & & +X_{23} & \geq & 10 \\
 & & & & & & & X_{ij} & \geq & 0
\end{array}
$$

Here is a quick and dirty Octave code that solves the LP (7.1.3):

```
octave:1> C = [3 4 6; 5 7 5];
octave:2> s = [15 20]';
octave:3> d = [17 8 10]';
octave:4> A = [
> 1 1 1 0 0 0;
> 0 0 0 1 1 1;
> 1 0 0 1 0 0;
> 0 1 0 0 1 0;
> 0 0 1 0 0 1];
octave:5> c = C'(:);
octave:10> [schedule,cost] = glpk(
c,A,[s;d],[0 0 0 0 0 0]',[],"UULLL","CCCCCC",1);
octave:11> schedule = reshape(schedule,3,2)';
octave:12> cost
cost =  153
octave:13> schedule
schedule =

    7    8    0
   10    0   10
```

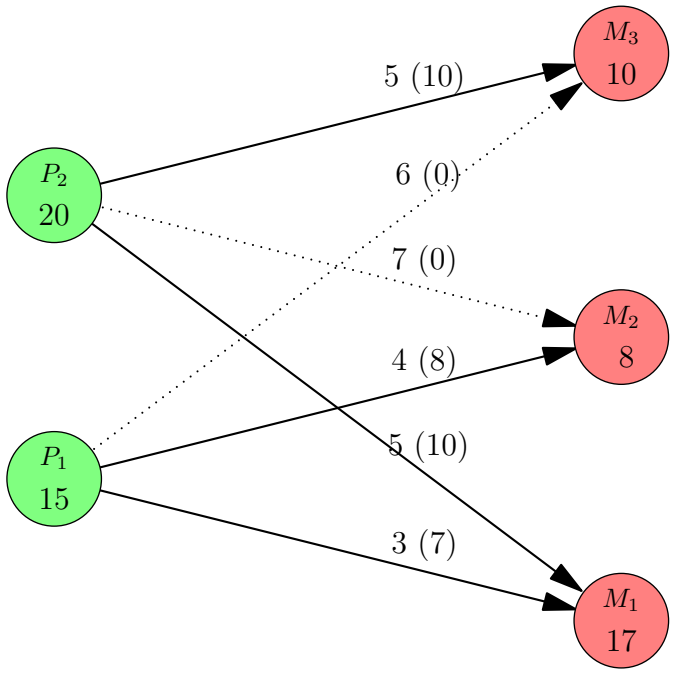
The author guesses that at least the command lines 5 and 11 require some explanation. These lines are related to the small price we have to pay for our clever indexing, i.e. using matrices instead of vectors: In the command line 5 the vector $\mathbf{c}$ is constructed from the matrix $\mathbf{C}$ by gluing its rows together. The command `C(:)` would do this, except that it glues columns. This is why we first transpose the matrix $\mathbf{C}$. The command line 11 works the other way around: the command `reshape` forms a matrix out of a vector by splitting the vectors column-wise. Since we want to make the splitting row-wise, we reshape it "the wrong way around" and then transpose the result.

The Octave code above tells us that the minimal cost for Frábært ehf. is €153 and the optimal shipping schedule that obtains this cost is

$$\mathbf{X} = \begin{bmatrix} 7 & 8 & 0 \\ 10 & 0 & 10 \end{bmatrix}.$$

With a diagram, the optimal shipping schedule can be represented as follows. Here

the optimal shippings are in the parenthesis after their shipping costs per unit and the shipping routes that are not used are dotted to emphasize the fact.



Balancing Transportation Problems

Example 7.1.1 is **balanced**: Total supply equals total demand, i.e.,

$$\sum_i s_i \quad = \quad \sum_j d_j.$$

In balanced problems all the products will be shipped and the market demands will be met exactly. So, the supply and demand constraints will realize as equalities rather than inequalities:

The **balanced transportation problem** is the LP

$$\min z \quad = \quad \sum_i \sum_j C_{ij} X_{ij} \qquad \text{(total shipping cost)}$$

subject to

$$\begin{aligned} \sum_j X_{ij} &= s_i && \text{(supply)} \\ \sum_i X_{ij} &= d_j && \text{(demand)} \\ X_{ij} &\geq 0 \end{aligned}$$

Here i is the index for ports and j is the index for markets.

What about non-balanced transportation problems then? There are two possible cases:

More supply than demand In this case we can **introduce an imaginary market called the dump**. The dump has just enough demand so that you can dump you excess supply there. Shipping to dump is costless (no matter what the Greens may say). This solution may seem silly to you, but remember that the objective was to minimize the transportation cost while meeting the market demands. So, you are allowed to lose your shipments, if it helps.

More demand than supply In this case the problem is infeasible: **you cannot possibly meet the market demands if you do not have enough to supply**. One can of course generalize the transportation problem so that if the supply does not meet the demand there is a (linear) **penalty** associated with each market whose demand is not satisfied. This can be accomplished by using negative supply: **introduce an imaginary port called the shortage**. The shortage supply has the supply of the total demand minus the total supply. The cost of shipping shortage to the markets is the penalty for not meeting the respective markets demands. Now the problem is balanced.

Transportation Solver `trans_solver`

To further illustrate the LP nature of transportation problems we consider an Octave function `trans_solver` that solves transportation problems by using the LP solver `glpk`.

The function `trans_solver` assumes that the transportation problem is balanced, i.e., the supply and the demand are the same. If you want to use the function `trans_solver` for unbalanced problems, you have to introduce the dump or the shortage and shortage penalties manually.

As always, the line numbers are here for reference purposes only, and the function is written in the file `trans_solver.m`.

```
1 function [cost,schedule] = trans_solver(C,s,d)
```

The file `trans_solver.m` starts as all m-files defining a function should: by the keyword `function` following the definition of the output variables, name, and the input variables of the said function.

```
 2 ## Function [cost, schedule] = trans_solver(C, s, d) solves the
 3 ## balanced transportation problem
 4 ##
 5 ##         -----   -----
 6 ##         \       \
 7 ##    min   >       >     C(i,j)*X(i,j)
 8 ##         /       /
 9 ##         -----   -----
10 ##           i       j
11 ##
12 ##           -----
13 ##           \
14 ##    s.t.    >     X(i,j) = s(i)  for all ports i
15 ##           /
16 ##           -----
17 ##             j
18 ##
19 ##           -----
20 ##           \
21 ##            >     X(i,j) = d(j)  for all markets j
22 ##           /
23 ##            -----
24 ##              i
25 ##
26 ## Input:
27 ##
28 ## C, s, d
29 ##      The matrix of the transportation costs, the column vector
30 ##      of the supplies, and the column vector of the demands.
31 ##
32 ## Output:
33 ##
34 ## cost, schedule
35 ##      The minimal transportation cost and schedule.
36 ##
37 ## See also:  glpk.
38
```

The lines 2–37 consist of the comment block that is printed out if you ask `help` for the function. The empty line 38 terminates the comment block.

```
39 ## Some short-hands.
40 m1 = length(s);                                  # Number of ports.
41 m2 = length(d);                                  # Number of markets.
42 n = m1*m2;                                       # Number of decisions.
43 o = zeros(n,1);                                  # For glpk lower bounds.
44
```

The lines 39–44 simply set up some short-hand notations: nothing complicated here.

```
45 ## Build the technology matrix A
46 A = zeros(m1+m2,n);                                  # Initialization.
47 for i = 1:m1
48   A(i, (1:m2)+(i-1)*m2) = ones(1,m2);               # Supply side.
49   A((1:m2)+m1, (1:m2)+(i-1)*m2) = eye(m2);          # Demand side.
50 endfor
51
```

Here a lot happens and fast! The technology matrix of any transportation LP is of the form

$$\mathbf{A} = \left[\begin{array}{cccccc} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \begin{array}{llll} \text{supply:} & P_1 & \text{with bound} & s_1 \\ \text{supply:} & P_2 & \text{with bound} & s_2 \\ \hline \text{demand:} & M_1 & \text{with bound} & d_1 \\ \text{demand:} & M_2 & \text{with bound} & d_2 \\ \text{demand:} & M_3 & \text{with bound} & d_3 \end{array}, \tag{7.1.4}$$

where we first have blocks of ones going to the right (the supply side) and then blocks of identity matrices going to the right (the demand side). (The particular matrix (7.1.4) is the technology matrix for Frábært ehf. of Example 7.1.1, by the way.) The lines 46–50 build this kind of zero-one matrices by using the Octave's block-indexing feature. (This is pretty advanced Octave programming: the author apologizes!) To understand the Octave's block-indexing concept, consider the following code:

```
octave:1> A = zeros(4,16)
A =

   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

octave:2> rw = 2:4
rw =

   2   3   4

octave:3> clmn = [1 10 11 12]
clmn =

    1   10   11   12

octave:4> A(rw,clmn) = ones(3,4)
A =

   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   1   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0
   1   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0
   1   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0
```

Now, play around with this code, e.g., by changing the vectors `clmn` and `rw`. Then remember that for Octave the vector `1:m2` is the vector $[1\ 2\ \cdots\ m2]$. After this the lines 46–50 should start make sense to you.

```
52 ## Set the sense of bounds (equality for balanced).
53 ctype = "";
54 for i=1:(m1+m2)
55   ctype = [ctype, "S"];
56 endfor
57
58 ## Set the type of each decision variable as continuous.
59 vtype = "";
60 for i=1:n
61   vtype = [vtype, "C"];
62 endfor
63
```

The lines 52–63 set the `ctype` and `vtype` strings for the function `glpk` pretty much the same way as the lines 56–66 of the function `stu_lp_solver` did. Note here that we use the identifier `"S"` for the bounds, since we are dealing with a **balanced** transportation problem here.

```
64 ## Solve the system by calling glpk.
65 [schedule, cost] = glpk(C'(:), A, [s; d], o, [], ctype, vtype, 1);
66
```

Now that we have all the parameters right, we call the `glpk` LP solve. The first parameter `C'(:)` is the vector defining the objective function. We use the "total colon", as opposed to the colon-notation that only operates on rows or columns, here. This way the matrix `C` is transformed into a vector. The transpose is needed, since the total colon operator works column-wise, and wee need a row-wise gluing of the matrix `C`. The technology matrix `A` was set in the lines 46–40. The bounds are twofold: first we have the supply side `s` and then we have the demand side `d`. We combine them together as the (column) vector `[s; d]`. Then we have the lower bounds of zeros: `o`. There are no a priori upper bounds for the decisions: hence the empty matrix `[]`. Then we have the `ctype` and `vtype` vectors for the types of the bounds and the variables. These were set in the lines 53–62. Finally, we have the sense parameter set as `1` indicating that this is a minimization problem. (We could have omitted the `1` for the sense, since it is the default value.)

```
67 ## Reshape the schedule vector to a matrix.
68 schedule = reshape(schedule, m2, m1)';
69 endfunction
```

Finally, in the line 68 we reshape the vector `schedule` into the appropriate matrix

form. The best way to understand what is happening here is to read the `help` for the `reshape` function and then play around. A good playmate is the matrix `A = [11 12 13; 21 22 23]`.

Transportation Algorithm*

The transportation algorithm is a specialized algorithm for solving transportation LPs. It is a cousin of the simplex algorithm: it is a tableau-dance with transportation tableaux. The first transportation tableau is an empty transportation tableau that is a tabular representation of the transportation problem's data:

	M_1	M_2	$\cdots$	M_n	
P_1	C_{11}	C_{12}		C_{1n}	s_1
P_2	C_{21}	C_{22}		C_{2n}	s_2
$\vdots$					$\vdots$
P_m	C_{m1}	C_{m2}		C_{mn}	s_m
	d_1	d_2	$\cdots$	d_n	

The Frábært's transportation problem's 7.1.1 tabular representation is

	M_1	M_2	M_3	
P_1	3	4	6	15
P_2	5	7	5	20
	17	8	10	

The transportation algorithm fills the empty transportation tableau with the shipping schedule X_{ij}:

	M_1	M_2	$\cdots$	M_n	
P_1	C_{11} X_{11}	C_{12} X_{12}		C_{1n} X_{1n}	s_1
P_2	C_{21} X_{21}	C_{22} X_{22}		C_{2n} X_{2n}	s_2
$\vdots$					$\vdots$
P_m	C_{m1} X_{m1}	C_{m2} X_{m2}		C_{mn} X_{mn}	s_m
	d_1	d_2	$\cdots$	d_n	

The general idea of the **transportation algorithm** is the same as in the simplex algorithm, viz.

7.1.5 Algorithm (Transportation meta-algorithm)**.** To solve a transportation problem, take the following steps:

Meta-step 1 Find a BFS.

Meta-step 2 Check for optimality. If solution is optimal, the algorithm terminates. Otherwise move to step 3.

Meta-step 3 Find a new, improved, BFS. Go back to Meta-step 2.

Next we explain the metas away from the Algorithm 7.1.5 above.

Finding a BFS The first BFS can be found, e.g., by using the so-called **NW corner method**. The method is called such since the traditional way of choosing available squares goes from NW to SE. The method goes as follows:

1. Choose any available square, say (i_0, j_0). Specify the shipment X_{i_0,j_0} as large as possible subject to the supply and demand constraints, and mark this variable.
2. Delete from consideration whichever row or column has its constraint satisfied, but not both. If there is a choice, do not delete a row (column) if it is the last row (resp. column) undeleted.
3. Repeat 1. and 2. until the last available square is filled with a marked variable, and then delete from consideration both row and column.

Now we construct the first BFS for Example 7.1.1. The marked variables will be in parentheses, and the deleted squares will have 0.

We start with the NW corner. So $(i_0, j_0) = (1, 1)$. We put there as a big number as possible. This means that we ship all the plant P_1's supply to market M_1, i.e., $X_{11} = 15$. Now there is nothing left in P_1. So, we must have $X_{12} = X_{13} = 0$. So, squares $(1, 2)$ and $(1, 3)$ get deleted — or, if you like — row 1 gets deleted.

	M_1	M_2	M_3	
P_1	3 (15)	4 0	6 0	15
P_2	5	7	5	20
	17	8	10	

Next we move south, since east is deleted. The biggest number we can now put to square $(2, 1)$ is $X_{21} = 2$, since there is already 15 tons of skyr shipped to the market M_1 that demands 17 tons. No rows or columns will get deleted because of this operation.

	M_1	M_2	M_3	
P_1	3 (15)	4 0	6 0	15
P_2	5 (2)	7	5	20
	17	8	10	

Next we move east to square $(2,2)$. There the biggest number we can put is $X_{22} = 8$ since that is the market demand for M_2. No rows or columns will be deleted because of this operation.

	M_1	M_2	M_3	
P_1	3 (15)	4 0	6 0	15
P_2	5 (2)	7 (8)	5	20
	17	8	10	

Finally, we move to the last free square $(2,3)$. It is obvious that $X_{23} = 10$. We get the first BFS:

	M_1	M_2	M_3	
P_1	3 (15)	4 0	6 0	15
P_2	5 (2)	7 (8)	5 (10)	20
	17	8	10	

Checking Optimality Given a BFS, i.e., a feasible shipping schedule X_{ij}, we shall use the complementary slackness theorem 5.3.9 to check whether the BFS is optimal. This means finding dual variables u_i and v_j that satisfy

$$X_{ij} > 0 \quad \text{implies that} \quad v_j - u_i = C_{ij}.$$

One method of finding the dual variables u_i and v_j is to solve the equations

$$v_j - u_i \;=\; C_{ij}$$

for all (i,j)-squares containing marked variables. There are $m+n-1$ marked variables, and so we have $m+n-1$ equations with $m+n$ unknowns. This means that one of the variables u_i, v_j can be fixed to 0, say. Some of the u_i or v_j may

turn out to be negative, which as such is not allowed in the dual problem, but this is not a problem. Indeed, one can always add a big enough constant to all the u_is and v_js without changing the values of $v_j - u_i$.

Once the dual variables u_i and v_j are found we can check the optimality of the BFS by using the following algorithm:

1. Set one of the v_j or u_i to zero, and use the condition
$$v_j - u_i \quad = \quad C_{ij}$$
for **squares containing marked variables** to find all the v_j and u_i.
2. Check feasibility,
$$v_j - u_i \quad \leq \quad C_{ij},$$
for the **remaining squares**. If the BFS is feasible, it is optimal for the problem and its dual, due to the Complementary Slackness Theorem 5.3.9

Let us then find the dual variables u_i and v_j for the BFS Frábært's problem 7.1.1. The next tableau is the BFS we found with the, yet unknown, dual variables in their appropriate places.

	v_1	v_2	v_3	
u_1	3 (15)	4 0	6 0	15
u_2	5 (2)	7 (8)	5 (10)	20
	17	8	10	

To solve u_i and v_j for the marked variables we put $u_2 = 0$. Remember we can choose any one of the u_i or v_j be zero. We chose u_2 because then we see immediately that $v_1 = 5$, $v_2 = 7$, and $v_3 = 5$. As for the last unknown u_1, we have

$$u_1 \quad = \quad v_1 - C_{11} \quad = \quad 5 - 3 \quad = \quad 2.$$

So, we have the following BFS with the duals

	5	7	5	
2	3 (15)	4 0	6 0	15
0	5 (2)	7 (8)	5 (10)	20
	17	8	10	

Now we check the remaining squares. For the BFS to be optimal we must have $v_j - u_i \leq C_{ij}$. We see that this is not the case. The culprit is the square $(1,2)$:

$$v_2 - u_1 \;=\; 7-2 \;=\; 5 \;>\; 4 \;=\; C_{12}.$$

This means that we must improve our BFS.

Improvement Routine Now we have a BFS that is not optimal. So, we must have a square (i_0, j_0), say, for which

$$v_{j_0} - u_{i_0} \;>\; C_{i_0 j_0}.$$

We would like to ship some amount of skyr, say, from the port P_{i_0} to the market M_{j_0}. The current amount $X_{i_0 j_0} = 0$ will be changed to a new amount denoted by Δ. But if we change $X_{i_0 j_0}$ to Δ we must subtract and add Δ to other squares containing marked variables. This means that we are looking forward to a new BFS

	M_1	M_2	M_3	
P_1	3 $-\Delta$ (15)	4 $+\Delta$ 0	6 0	15
P_2	5 $+\Delta$ (2)	7 $-\Delta$ (8)	5 (10)	20
	17	8	10	

Now we choose the change Δ to be as big as possible bearing in mind that the shipments cannot be negative. This means that Δ will be the minimum of the X_{ij}s in the squares we are subtracting Δ. We see that the biggest possible change is $\Delta = 8$, which makes the the shipment X_{22} zero.

Note that it may turn out that that we have $\Delta = 0$. This means that the value of the objective won't change. However, the shipping schedule and the marked variables will change. While the new shipping schedule is no better than the old one, one can hope that from this new shipping schedule one can improve to a better one.

Now we have the new BFS

	M_1	M_2	M_3	
P_1	3 (7)	4 (8)	6 0	15
P_2	5 (10)	7 0	5 (10)	20
	17	8	10	

We have to go back to the previous step and check optimality for this BFS. So, we have to solve the dual variables u_i and v_j. We set now $u_2 = 0$ which gives us immediately that $v_1 = 5$ and $v_3 = 5$. So, we find out that

$$u_1 \;=\; v_1 - C_{11} \;=\; 2,$$

and that

$$v_2 \;=\; u_1 + C_{12} \;=\; 6.$$

So, we have the BFS with dual variables

	5	6	5	
2	3 (7)	4 (8)	6 0	15
0	5 (10)	7 0	5 (10)	20
	17	8	10	

This solutions passes the optimality test:

$$v_j - u_i \;\leq\; C_{ij}$$

for all i and j. So, we have found an optimal shipping schedule. The cost associated with this shipping schedule can now be easily read from the tableau above:

$$\begin{aligned} z \;&=\; \sum_{i=1}^{2}\sum_{j=1}^{3} C_{ij}X_{ij}^{*} \\ &=\; 3\times 7 + 4\times 8 + 5\times 10 + 5\times 10 \\ &=\; 153. \end{aligned}$$

The improvement algorithm can now be stated as

1. Choose any square (i, j) with $v_j - u_i > C_{ij}$. Set $X_{ij} = \Delta$, but keep the constraints satisfied by subtracting and adding Δ to appropriate marked variables.
2. Choose Δ to be the minimum of the variables in the squares in which Δ is subtracted.
3. Mark the new variable X_{ij} and remove from the marked variables one of the variables from which Δ was subtracted that is now zero.

7.2 Assignment Problem

In this section we consider assignment problems that are — although it may not seem so at first sight — special cases of transportation problems. In an assignment problem one has to assign each worker to each task in a most efficient way.

A typical assignment problem is:

7.2.1 Example (Machines-to-jobs assignment)**.** Machineco has four machines and four jobs to be completed. Each machine must be assigned to complete one job. The times required to set up each machine for completing each job are:

	Jobs			
Machines	Job 1	Job 2	Job 3	Job 4
Machine 1	14	5	8	7
Machine 2	2	12	6	5
Machine 3	7	8	3	9
Machine 4	2	4	6	10

Machineco want to minimize the total setup time.

If you think about it a while, you see that:

> An **assignment problem** is a transportation problem with equal amount of ports and markets, where the demands and supplies for each port and market are equal to one.

Assignment Problem as Linear Program

Let us build the LP representing Example 7.2.1.

The key point in modeling the Machineco's problem 7.2.1 is to find out the decision variables — everything else is easy after that. So what are the decisions Machineco must make? Machineco must choose which machine is assigned to which job. Now, how could we write this analytically with variables? A common trick here is to use **binary variables**, i.e., variables that can take only two possible values: 0 or 1. So, we set binary variables X_{ij}, $i = 1, \ldots, 4$, $j = 1, \ldots, 4$, for each machine and each job to be

$$X_{ij} = \begin{cases} 1 & \text{if machine } i \text{ is assigned to meet the demands of job } j, \\ 0 & \text{if machine } i \text{ is } \textbf{not} \text{ assigned to meet the demands of job } j. \end{cases}$$

In other words, the variable X_{ij} is an **indicator** of the claim

> "Machine i is assigned to job j".

Now it is fairly easy to formulate a program, i.e., an optimization problem, for Machineco's problem 7.2.1. Indeed, the objective is to minimize the total setup time. With our binary variables we can write the total setup time as

$$\begin{aligned} z \;=\; & 14X_{11} + 5X_{12} + 8X_{13} + 7X_{14} \\ & +2X_{21} + 12X_{22} + 6X_{23} + 5X_{24} \\ & +7X_{31} + 8X_{32} + 3X_{33} + 9X_{34} \\ & +2X_{41} + 4X_{42} + 6X_{43} + 10X_{44} \end{aligned}$$

Note that there will be a lot of zeros in the objective function above.

What about the constraints for Machineco? First, we have to ensure that each machine is assigned to a job. This will give us the supply constraints

$$\begin{array}{rcrcrcrcl} X_{11} & + & X_{12} & + & X_{13} & + & X_{14} & = & 1 \\ X_{21} & + & X_{22} & + & X_{23} & + & X_{24} & = & 1 \\ X_{31} & + & X_{32} & + & X_{33} & + & X_{34} & = & 1 \\ X_{41} & + & X_{42} & + & X_{43} & + & X_{44} & = & 1 \end{array}$$

Second, we have to ensure that each job is completed, i.e., each job has a machine assigned to it. This will give us the demand constraints

$$\begin{array}{rcrcrcrcl} X_{11} & + & X_{21} & + & X_{31} & + & X_{41} & = & 1 \\ X_{12} & + & X_{22} & + & X_{32} & + & X_{42} & = & 1 \\ X_{13} & + & X_{23} & + & X_{33} & + & X_{43} & = & 1 \\ X_{14} & + & X_{24} & + & X_{34} & + & X_{44} & = & 1 \end{array}$$

So, putting the objective and the constraints we have just found together, and not forgetting the binary nature of the decisions, we have obtained the following program for Machineco's problem 7.2.1:

$$\begin{array}{lrcrcrcrcll} \min z = & 14X_{11} & + & 5X_{12} & + & 8X_{13} & + & 7X_{14} & & \\ & +2X_{21} & + & 12X_{22} & + & 6X_{23} & + & 5X_{24} & & \\ & +7X_{31} & + & 8X_{32} & + & 3X_{33} & + & 9X_{34} & & \\ & +2X_{41} & + & 4X_{42} & + & 6X_{43} & + & 10X_{44} & & \\ \text{s.t.} & X_{11} & + & X_{12} & + & X_{13} & + & X_{14} & = 1 & \text{(Machine)} \\ & X_{21} & + & X_{22} & + & X_{23} & + & X_{24} & = 1 & \\ & X_{31} & + & X_{32} & + & X_{33} & + & X_{34} & = 1 & \\ & X_{41} & + & X_{42} & + & X_{43} & + & X_{44} & = 1 & \\ & X_{11} & + & X_{21} & + & X_{31} & + & X_{41} & = 1 & \text{(Job)} \\ & X_{11} & + & X_{21} & + & X_{31} & + & X_{41} & = 1 & \\ & X_{12} & + & X_{22} & + & X_{32} & + & X_{42} & = 1 & \\ & X_{13} & + & X_{23} & + & X_{33} & + & X_{43} & = 1 & \\ & X_{14} & + & X_{24} & + & X_{34} & + & X_{44} & = 1 & \\ & & & X_{ij} = 0 & & \text{or} & & X_{ij} = 1 & & \end{array} \tag{7.2.2}$$

In (7.2.2) we have **binary constraints** $X_{ij} = 0$ or $X_{ij} = 1$ for the decision variables. So, at first sight it seems that the program (7.2.2) is not a linear one. Indeed, it is a special case of an IP called **binary integer program** (BIP). However, the structure of the assignment problems is such that if one omits the assumption $X_{ij} = 0$ or $X_{ij} = 1$, and simply assumes that $X_{ij} \geq 0$, one will get on optimal solution where the decisions X_{ij}^* are either 0 or 1. Hence, the program (7.2.2) is a linear one, i.e., it is an LP. Or, to be more precise, the program (7.2.2) and its linear relaxation, or **LP relaxation**,

$$
\begin{array}{lrcl}
\min z = & \begin{array}{l} 14X_{11} + 5X_{12} + 8X_{13} + 7X_{14} \\ +2X_{21} + 12X_{22} + 6X_{23} + 5X_{24} \\ +7X_{31} + 8X_{32} + 3X_{33} + 9X_{34} \\ +2X_{41} + 4X_{42} + 6X_{43} + 10X_{44} \end{array} & & \\
\text{s.t.} & X_{11} + X_{12} + X_{13} + X_{14} = 1 & & \text{(Machine)} \\
& X_{21} + X_{22} + X_{23} + X_{24} = 1 & & \\
& X_{31} + X_{32} + X_{33} + X_{34} = 1 & & \\
& X_{41} + X_{42} + X_{43} + X_{44} = 1 & & \\
& X_{11} + X_{21} + X_{31} + X_{41} = 1 & & \text{(Job)} \\
& X_{11} + X_{21} + X_{31} + X_{41} = 1 & & \\
& X_{12} + X_{22} + X_{32} + X_{42} = 1 & & \\
& X_{13} + X_{23} + X_{33} + X_{43} = 1 & & \\
& X_{14} + X_{24} + X_{34} + X_{44} = 1 & & \\
& X_{ij} \geq 0 & &
\end{array}
\tag{7.2.3}
$$

are equivalent. Equivalence of programs means that they have the same optimal decision and the same optimal objective value.

Assignment Problem as Transportation Problem

Recall that

> An **assignment problem** is a transportation problem with equal amount of ports and markets, where the demands and supplies for each port and market are equal to one.

The next code solves Example 7.2.1 by using the function `trans_solver` designed to solve transportation LPs:

```
octave:1> C = [
> 14  5  8  7;
>  2 12  6  5;
>  7  8  3  8;
>  2  4  6 10];
octave:2> s = [1 1 1 1]'; d = [1 1 1 1]';
octave:3> [cost, schedule] = trans_solver(C, s, d)
cost =  15
schedule =

   0   1   0   0
   0   0   0   1
   0   0   1   0
   1   0   0   0
```

So, the minimal setup cost is 15 and it is achieved by assigning machine 1 to job 2, machine 2 to job 4, machine 3 to job 3, and machine 4 to job 1.

Hungarian Method*

Assignment problems, being both transportation problems and LPs, can be solved with the general simplex method or with the specialized transportation algorithm. There is, however, an even more specialized algorithm for solving assignment problems: the Hungarian method.

In the **Hungarian method**, the data of the assignment problem is presented in an $(n \times n)$-table, where n is the number of ports, or markets, which is the same. For Example 7.2.1 the "Hungarian tableau" is

14	5	8	7
2	12	6	5
7	8	3	9
2	4	6	10

The Hungarian method is based on the following two observation:

1. The problem is solved if we can choose n squares from the "Hungarian tableau" so that:
   (a) Exactly one square is chosen from each row.
   (b) Exactly one square is chosen from each column.
   (c) The sum of the costs in the chosen n squares is the smallest possible.
2. If a same number is subtracted from all squares in a row, then the optimal selection of squares does not change. The same is true, if a same number is subtracted from all squares in a column.

7.2.4 Algorithm (Hungarian method)**.**

Step 1 For each row, subtract the row minimum from each element in the row.

Step 2 For each column, subtract the column minimum from each element in the column.

Step 3 Draw the minimum number of lines — horizontal, vertical, or both — that are needed to cover all the zeros in the tableau. If n lines were required then the optimal assignment can be found among the covered zeros in the tableau, and the optimal cost can be read from the first tableau by summing up the number in the squares corresponding to the lined-out zeros.

Step 4 Find the smallest non-zero element that is uncovered by lines. Subtract this element from each uncovered element, and add this element to each square that is covered with two lines. Return to Step 3.

Here are steps 1 and 2 for Example 7.2.1 starting from the initial table:

14	5	8	7
2	12	6	5
7	8	3	9
2	4	6	10

9	0	3	2
0	10	4	3
4	5	0	6
0	2	4	8

9	0	3	0
0	10	4	1
4	5	0	4
0	2	4	6

Here are the remaining steps 3 and 4 for Example 7.2.1:

9	0	3	0
0	10	4	1
4	5	0	4
0	2	4	6

10	0	4	0
0	9	4	0
4	4	0	3
0	1	4	5

Now, we go back to Step 3, and line-out the zeros:

10	0	4	0
0	9	4	0
4	4	0	3
0	1	4	5

Now we can read the optimal assignment from the covered zeros. First we note that the only covered zero in column 3 is in square $(3,3)$. So, we must have assignment $X_{33} = 1$. Also, in column 2 the only available zero is in square $(1,2)$. Consequently, $X_{12} = 1$. Now, as we can no longer use row 1 the only available zero in column 4 is in square $(2,4)$. So, $X_{24} = 1$. Finally, we choose $X_{41} = 1$. Next we read the corresponding setup cost from the first Hungarian tableau. We obtain

$$\text{setup cost} \quad = \quad 5+5+3+2 \quad = \quad 15.$$

7.3 Transshipment Problem

> In a **transshipment problem** one has to ship products from ports to markets as in a transportation problem, but in addition to the ports and the markets one has transshipment points at one's disposal. So, one can ship products directly from ports to markets, or one can use transshipment points in the shipping.

At first sight the transshipment problems are network problems. At least, they seem to be much more complicated than transportation problems. It turns out, however, that we can express transshipment problems as transportation problems. So the generalization from transportation problems to transshipment problems is mathematically no generalization at all.

Transshipment Problems as Transportation Problems

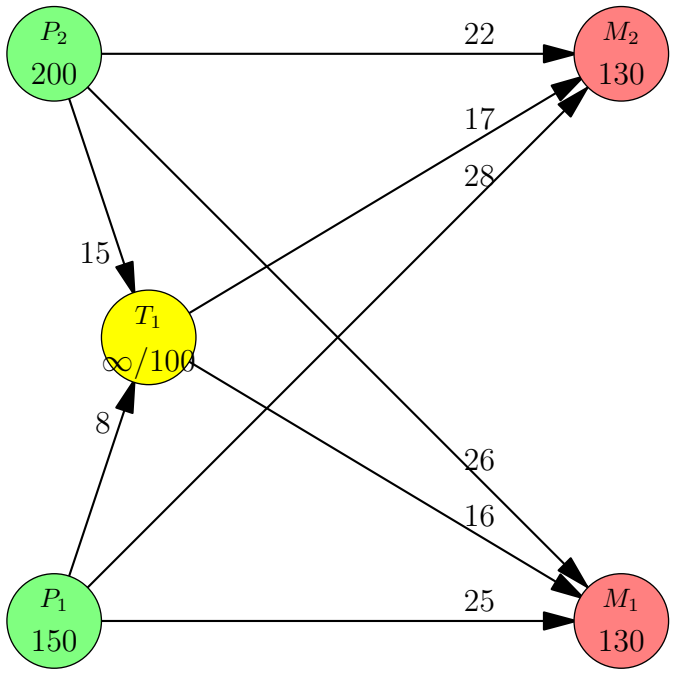
7.3.1 Example (Generic transshipment)**.** The Generic Company produces generic products and ships them to the markets M_1 and M_2 from the ports P_1 and P_2. The products can be either shipped directly to the markets, or the Generic Company can use a transshipment point T_1.

The ports P_1 and P_2 supply 150 and 200 units, respectively. The markets M_1 and M_2 demand both 130 units. The shipment costs from P_1 to M_1 is 25, from P_1 to M_2 it is 28, from P_2 to M_1 it is 26, and from P_2 to M_2 it is 22. The shipping cost from P_1 to the transshipment point T_1 is 8 and the shipping cost from P_2 to the transshipment point T_1 is 15. The shipping costs from the transshipment point T_1 to the markets M_1 and M_2 are 16 and 17, respectively.

The Generic Company wants to minimize its shipping costs while meeting the market demands. How should the Generic Company ship its products, if

(a) the transshipment point T_1 has infinite capacity,
(b) only 100 products can be transshipped through T_1?

Here is the data of Example 7.3.1 in a graph form.



The general idea in modeling transshipment problems is to model them as transportation problems where the transshipment points are both ports and markets. To be more precise:

- A **transshipment problem** is a transportation problem, where each transshipment point is both a port and a market. The shipping cost from a transshipment point to itself is, of course, zero.
- If a transshipment point has capacity N, then N will be both demand and supply for that transshipment point. If the capacity is unlimited set the demand and supply to be the total supply of the system for that transshipment point.
- Cost for shipping from transshipment points to the balancing dump is a very very very very big number M. This ensures that the excess supply to be dumped is dumped immediately.

Here is the tabular expression of the transshipment problems of Example 7.3.1 as transportation problems (D denotes the dump):

	Market				
Port	M_1	M_2	T_1	D	*Supply*
P_1	25	28	8	0	150
P_2	26	22	15	0	200
T_1	16	17	0	M	350/100
Demand	130	130	350/100	90	

Solving Transshipment Problems with Transportation Algorithm*

Solution to Variant (a) of Example 7.3.1

In Example 7.3.1 the total supply is 350 and the total demand is 260. So, in order to balance the problem we need the dump market with demand 90. Let us denote the dump by D. The supply and demand for the transshipment point will be 350, so that everything can be, if necessary, transported via T_1. We also do not want to transship anything to the dump via the transshipment point T_1. So, we make the cost of shipping from T_1 to D a very very very very very big number M. The direct dumping from P_1 or P_2 to D of costless, of course. So, we have the following initial transportation tableau

	T_1	M_1	M_2	D	
P_1	8	25	28	0	150
P_2	15	26	22	0	200
T_1	0	16	17	M	350
	350	130	130	90	

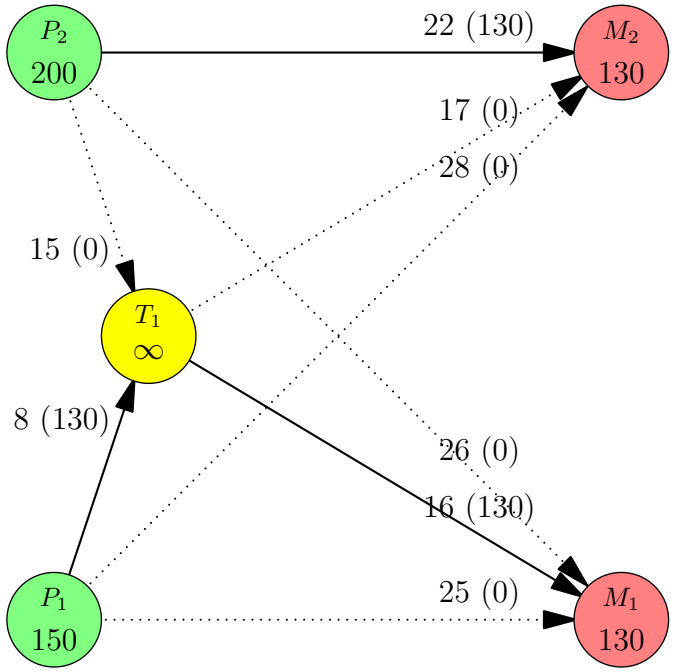
Now we can apply the transportation algorithm to this tableau. Here is a (greedy minimal cost) initial BFS

	T_1	M_1	M_2	D	
P_1	8 (150)	25 0	28 0	0 0	150
P_2	15 (110)	26 0	22 0	0 (90)	200
T_1	0 (90)	16 (130)	17 (130)	M 0	350
	350	130	130	90	

We leave it as an exercise to carry out the transportation algorithm for this BFS. We note the optimal transportation tableau:

	T_1	M_1	M_2	D	
P_1	8 (130)	25 0	28 0	0 (20)	150
P_2	15 0	26 0	22 (130)	0 (70)	200
T_1	0 (220)	16 (130)	17 0	M 0	350
	350	130	130	90	

Here is a graphical representation of the optimal transportation tableau above (the transportation schedule is in the parentheses):



From the picture we see that the total shipping cost is

$$130 \times 8 + 130 \times 16 + 130 \times 22 \quad = \quad 5980.$$

We cannot read the transshipment and dump data of the last Transportation Tableau from the picture above, but that data was auxiliary anyway, i.e. we might well not be interested in it.

Solution to Variant (b) of Example 7.3.1

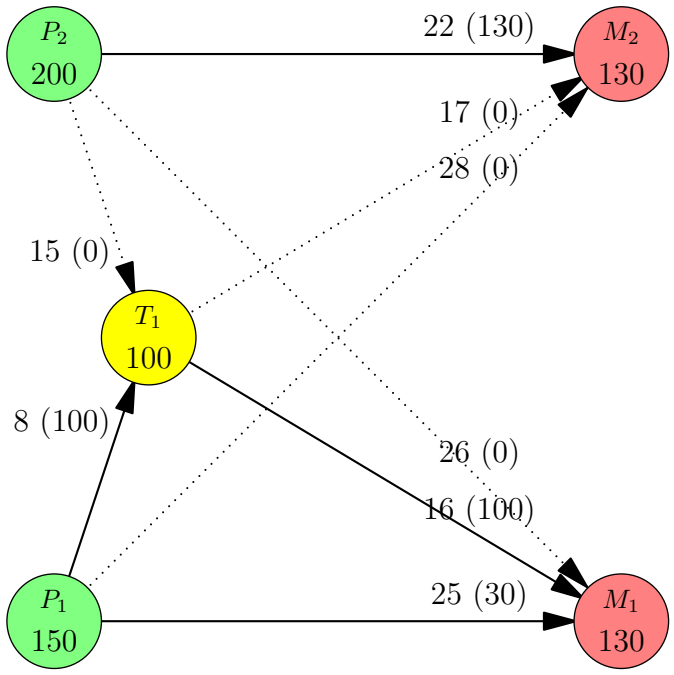
In Example 7.3.1 the total supply is 350 and the total demand is 260. So, in order to balance the problem we need the dump market with demand 90. Let us denote the dump by D. The supply and demand for the transshipment point will be 100, so that at most 100 units can be transported via T_1. We also do not want to transship anything to the dump via the transshipment point T_1. So, we make the cost of shipping from T_1 to D a very very very very very big number M. The direct dumping from P_1 or P_2 to D of costless, of course. So, we have the following initial transportation tableau

	T_1	M_1	M_2	D	
P_1	8	25	28	0	150
P_2	15	26	22	0	200
T_1	0	16	17	M	100
	100	130	130	90	

We leave it for the students to carry out the transportation algorithm, and simply note the solution:

	T_1	M_1	M_2	D	
P_1	8 (100)	25 (30)	28 0	0 (20)	150
P_2	15 0	26 0	22 (130)	0 (70)	200
T_1	0 0	16 (100)	17 0	M 0	100
	100	130	130	90	

Here is a graphical representation of the optimal transportation tableau above:



From the picture we see that the total shipping cost is

$$100 \times 8 + 100 \times 16 + 30 \times 25 + 130 \times 22 \quad = \quad 6010.$$

We cannot read the transshipment and dump data of the last Transportation Tableau from the picture above, but that data was auxiliary anyway, i.e. we might well not be interested in it.

7.4 Exercises and Projects

7.1. Powerco has three electric power plants that supply the needs of four cities. The power plants can supply 35 GWh, 50 GWh, and 40 GWh, respectively. The demands (peak power) of the four cities are 40 GWh, 20 GWh, 30 GWh, and 30 GWh, respectively.

The transportation costs of 1 GWh from power plant 1 to the cities 1, 2, 3, and 4, are €8, €6, €10, and €9, respectively. The transportation costs of 1 GWh from power plant 2 to the cities 1, 2, 3, and 4, are €9, €12, €13, and €7, respectively. The transportation costs of 1 GWh from power plant 2 to the cities 1, 2, 3, and 4, are €14, €9, €16, and €5, respectively.

Powerco wants to minimize the total transportation cost while meeting the peak power demands of the cities. How should Powerco transport its power from the plants to the cities?

7.2. * The function `[cost, schedule] = trans_solver(C, s, d)` works only for balanced transportation problems. Make a function

```
[cost, schedule, dump] = transu_solver(C, s, d, p)
```

that works for unbalanced transportation problems also. Here the new output parameter `dump` tells the amount each port dumps and the new input parameter `p` tells how much each market penalizes the short-fall in meeting the demand.

7.3. The city of Vaasa is taking bids on four construction jobs. Three companies have placed bids on the jobs. Their bids in millions of Euros are

	Jobs			
Company	Job 1	Job 2	Job 3	Job 4
AAA	50	46	42	40
AAB	51	48	44	–
ABB	–	47	45	45

In the above '–' means that there was no bid.

Company AAA can do only one job. Companies AAB and ABB can do as many as two jobs. The city of Vaasa wants to settle the matter all at once, i.e., all the construction jobs are to be distributed between the companies AAA, AAB, and ABB immediately.

(a) How should the city of Vaasa distribute the construction jobs between the companies?
(b) Suppose that the companies that have bid for a construction job are willing to share the said job. How should the city of Vaasa distribute the construction jobs between the companies AAA, AAB and ABB now?

7.4. Solve the transshipment problems of Example 7.3.1 by using the function `trans_solver`.

7.5. General Ford produces road-boats at L.A. and Detroit and has a warehouse in Atlanta. The customers are in Houston and Tampa. The costs of shipping a road-boat are in Dollars

	City				
City	L.A.	Detroit	Atlanta	Houston	Tampa
L.A.	0	140	100	90	225
Detroit	145	0	111	110	119
Atlanta	105	115	0	113	78
Houston	89	109	121	0	–
Tampa	210	117	82	–	0

In the above '–' means that shipping is not allowed.

L.A. can produce 1,100 road-boats, and Detroit can produce 2,900 road-boats. Houston must receive 2,400 road-boats and Tampa must receive 1,500 road-boats. There is no limit in the Atlanta warehouse.

(a) Solve the General Ford's transshipment problem, i.e., find the transshipment schedule with the minimum cost.
(b) General Ford's grandfather Major Nepotist is a senator who can make the transportation between Houston and Tampa cost 10 Dollars both ways. How much should General Ford pay for his grandfather to make the transportation between Houston and Tampa possible?

7.6. * Implement the specialized transportation algorithm with Octave.

7.7. * Implement the Hungarian method with Octave.

Part IV

Mixed Integer Linear Programming and Models

Chapter 8

Mixed Integer Linear Programming

> God is Real, unless declared Integer. — J. Allan Toogood

In this chapter we lift the **divisibility** assumption of LPs. This means that we are looking at LPs where some (or all) of the decision variables are required to be integers.

8.1 Mixed Integer Linear Programming Terminology

Although the name integer program does not state it explicitly, it is assumed that

> Integer and mixed integer programs are LPs with the additional requirement that all (or some) of the decision variables are integers.

If the additional requirement that some of the decision variables are integers is lifted, then the resulting LP is called the **LP relaxation** of the prgram in question.

Pure Integer Program

An IP in which **all** the decision variables are required to be integers is called a **pure integer program**, or simply an **integer program** (IP).

For example,

$$\begin{array}{lrcrcrcl} & \max z = & 3x_1 & + & 2x_2 & & & \\ \text{(8.1.1)} & \text{s.t.} & x_1 & + & x_2 & \leq & 6 & \\ & & & & x_1, x_2 & \geq & 0, & \text{integer} \end{array}$$

is a pure integer program.

Binary Integer Program

A **binary integer program** (BIP) is a IP where all the decision variables a required to be either 0 or 1.

For example,

$$\begin{array}{lrcrcrcl} \max z = & 3.2x_1 & + & 2.4x_2 & + & 1.5x_3 & & \\ \text{s.t.} & 0.1x_1 & + & 0.9x_2 & - & 0.5x_3 & \leq & 2.5 \\ & & & -1.7x_2 & + & 9.5x_3 & \geq & 1.5 \\ & & & & & x_1, x_2, x_3 & = & 0 \text{ or } 1. \end{array} \tag{8.1.2}$$

Assignment problems are BIPs, although they can be solved as LPs.

Any BIP can be transformed into an IP by simply noting that the constraint

$$x_i = 0 \text{ or } 1$$

is the same as the constraints

$$\begin{aligned} x_i &\leq 1 \\ x_i &\geq 0, \text{ integer} \end{aligned}$$

Mixed Integer Program

An LP in which **some but not all** of the decision variables are required to be integers is called a **mixed integer linear program** (MILP). Sometimes the name mixed integer program (MIP) is also used.

For example,

$$\begin{array}{lrcrcl} \max z = & 3x_1 & + & 2x_2 & & \\ \text{s.t.} & x_1 & + & x_2 & \leq & 6 \\ & & & x_1, x_2 & \geq & 0, x_1 \text{ integer} \end{array} \tag{8.1.3}$$

is a MILP, since x_1 is required to be an integer, but x_2 is not.

8.2 Branch-And-Bound Method

In this section we provide a relatively fast algorithm for solving IPs and MILPs. The general idea of the algorithm is to solve LP relaxations of the IP or MILP and to look for an integer solution by branching and bounding on the decision variables provided by the the LP relaxations.

Branch-And-Bound by Example

8.2.1 Example (Furniture with integrity)**.** The Integrity Furniture Corporation manufactures tables and chairs. A table requires 1 hour of labor and 9 units of wood. A chair requires 1 hour of labor and 5 units of wood. Currently 6 hours of labor and 45 units of wood are available. Each table contributes €8 to profit, and each chair contributes €5 to profit.

Integrity Furniture's want to maximize its profit.

This problem would be a classical product selection problem, but we insist that only integral furniture is produced. So, the problem is no longer an LP but an IP.

Let us formulate and solve the (pure) IP for Example 8.2.1. Let

$$\begin{aligned} x_1 &= \text{number of tables manufactured,} \\ x_2 &= \text{number of chairs manufactured.} \end{aligned}$$

Since x_1 and x_2 must be integers, Integrity Furniture wishes to solve the following (pure) IP:

$$\begin{array}{lrcrcrll} \max z = & 8x_1 & + & 5x_2 & & & & \\ \text{s.t.} & x_1 & + & x_2 & \leq & 6 & & \text{(labor)} \\ & 9x_1 & + & 5x_2 & \leq & 45 & & \text{(wood)} \\ & & & x_1, x_2 & \geq & 0, & \text{integer} & \end{array} \tag{8.2.2}$$

The first step in the branch-and-bound method is to solve the LP relaxation of the IP (8.2.2):

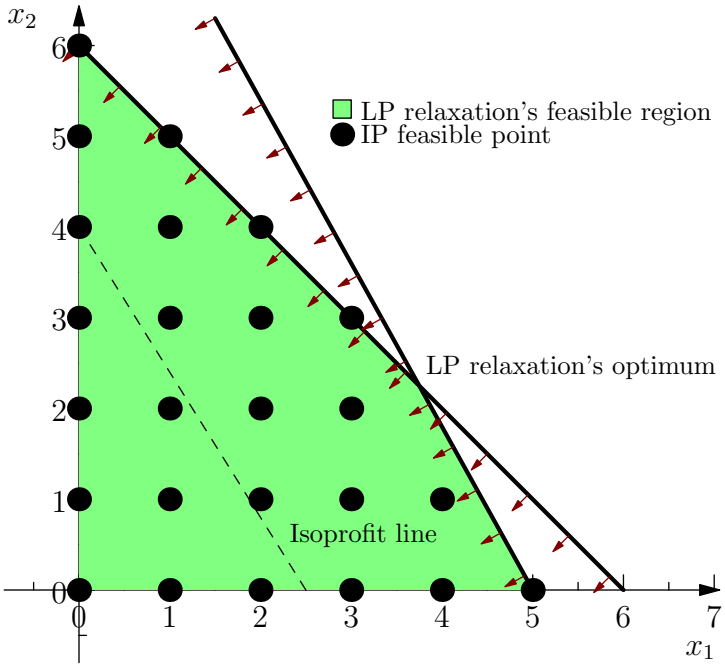
$$\begin{array}{lrcrcrl} \max z = & 8x_1 & + & 5x_2 & & & \\ \text{s.t.} & x_1 & + & x_2 & \leq & 6 & \text{(labor)} \\ & 9x_1 & + & 5x_2 & \leq & 45 & \text{(wood)} \\ & & & x_1, x_2 & \geq & 0 & \end{array} \tag{8.2.3}$$

If we are lucky, all the decision variables (x_1 for tables and x_2 for chairs in the example we are considering) in the LP relaxation's optimum turn out to be integers. In this lucky case the optimal solution of the LP relaxation is also the optimal solution to the original IP.

In the branch-and-bound algorithm we use next, we call the LP relaxation (8.2.3) of the IP (8.2.2) **subproblem 1** (SP 1). After solving SP 1 we find the solution to be

$$\begin{aligned} z &= 165/4 \\ x_1 &= 15/4 \\ x_2 &= 9/4 \end{aligned}$$

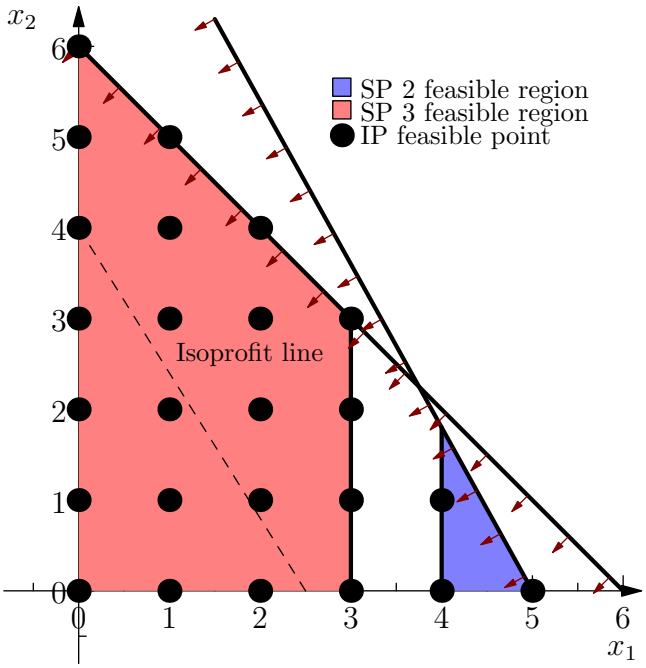
This means that we were not lucky: the decision variables turned out to be fractional. So, the LP relaxation (8.2.3) has (possibly) a better optimum than the original IP (8.2.2). In any case, we have found an **upper bound** to the original IP: Integrity Furniture Corporation of Example 8.2.1 cannot possibly have better profit than €165/4.



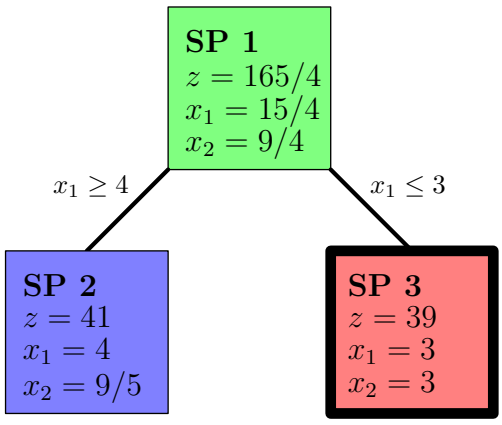
Next we split the feasible region (painted light green in the picture above) of the LP relaxation (8.2.3) in hope to find a solution that is an integer one. We arbitrarily choose a variable that is fractional at the optimal solution of the LP SP 1 (the first LP relaxation). We choose x_1. Now, x_1 was $15/4 = 3.75$. Obviously, at the optimal solution to the IP we have either $x_1 \geq 4$ or $x_1 \leq 3$, since the third alternative $3 < x_1 < 4$ is out of the question for IPs. So, we consider the two possible cases $x_1 \geq 4$ and $x_1 \leq 3$ as separate subproblems. We denote these subproblems as SP 2 and SP 3. So,

$$\begin{aligned} SP\ 2 &= SP\ 1 + "x_1 \geq 4", \\ SP\ 3 &= SP\ 1 + "x_1 \leq 3". \end{aligned}$$

In the next picture we see that every possible feasible solution of the Integrity Furniture's IP (8.2.2) (the bullet points) is included in the feasible region of either SP 2 or SP 3. Since SP 2 and SP 3 were created by adding constraints involving the fractional solution x_1, we say that SP 2 and SP 3 were created by **branching** on x_1.



Below we have a tree-representation of of the picture above, and of our branching-and-bounding so far. The color coding refer to the two previous pictures.

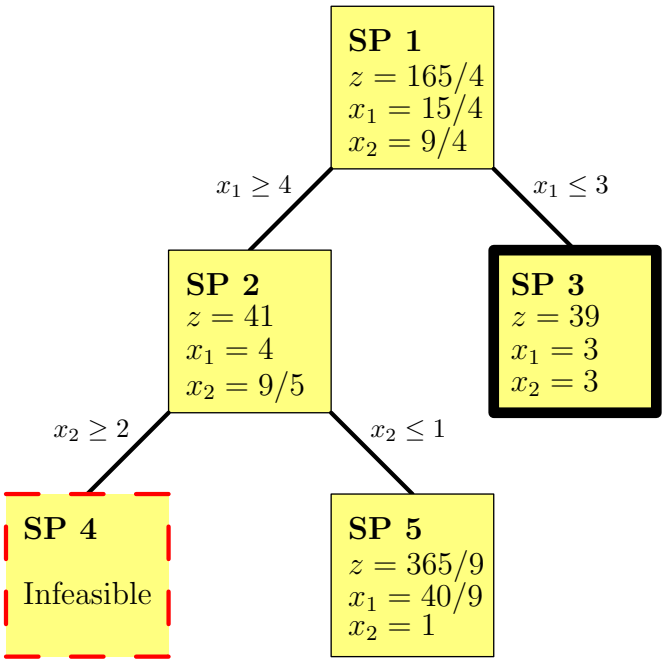


We see that SP 3 has an integer solution. So, we do not have to branch SP 2 any more. Unfortunately, the integer solution of SP 3, $z = 39$, is suboptimal when compared to the non-integer solution, $z = 41$, of SP 2. So, it may turn out that the SP 2 has a further subproblem that has better integer solution than SP 3. So, we have to branch SP 2 to find out what happens. Since $x_1 = 4$ is an integer, we branch on $x_2 = 9/5 = 1.8$. So, we have the new subproblems of SP 2:

$$\begin{aligned} SP\ 4 &= SP\ 2\ +\ "x_2 \geq 2", \\ SP\ 5 &= SP\ 2\ +\ "x_2 \leq 1". \end{aligned}$$

When the solutions to these subproblems are added to the tree above we get the following tree. There is no color coding in the boxes in the tree. There is border coding, however. A thick borderline expresses a feasible IP solution, and a dashing red borderline expresses an infeasible case.

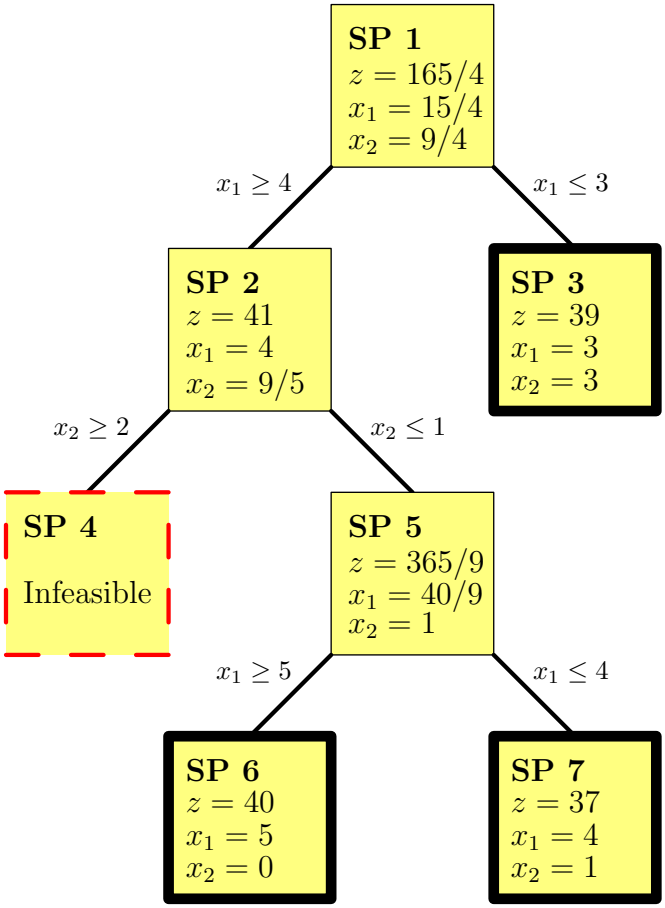
(the color coding is dropped):



We see that SP 4 is infeasible. So, the optimal solution is not there. However, SP 5 gives us a non-integer solution that is better than the integer solution of SP 3. So, we have to branch on SP 5. Since $x_2 = 1$ is already an integer, we branch on $x_1 = 40/9 = 4.444$. So, we get two new subproblems of SP 5:

$$\begin{aligned} SP\ 6 &= SP\ 5\ +\ "x_1 \geq 5", \\ SP\ 7 &= SP\ 5\ +\ "x_1 \leq 4". \end{aligned}$$

After this branching we finally arrive at the final solution where all the subproblems are either infeasible, have integer solutions, or are suboptimal to some subproblem that have integer solution, i.e. we have integer bounds for the subproblems. This is the bound part of the branch–and–bound.



From the tree above can read the solution to the IP: The SP 6 is the optimal subproblem with integer solution. So, the solution to the IP (8.2.2) is

$$\begin{aligned} z &= 40, \\ x_1 &= 5, \\ x_2 &= 0. \end{aligned}$$

General Branch-And-Bound Algorithm

We have solved a pure IP with branch and bound. To solve MILP with branch-and-bound one follows the same steps as in the pure IP case **except** one only branches on decision variables that are required to be integers. So, solving MILPs is actually somewhat easier than solving pure IPs!

8.2.4 Algorithm (Branch-and-bound algorithm)**.**

Step 1 Solve the LP relaxation of the problem. If the solution is integer where required, then we are done. Otherwise create two new subproblems by **branching** any fractional variable that is required to be integer.

Step 2 A subproblem is **not active** when any of the following occurs:

(a) You used the subproblem to branch on, i.e., the subproblem is not a leaf in the tree.

(b) All variables in the solution that are required to be integers, are integers.

(c) The subproblem is infeasible.

(d) You can fathom the subproblem by a **bounding** argument.

Choose an **active** subproblem and branch on a fractional variable that should be integer in the final solution. Repeat until there are no active subproblems.

Step 3 Solution to the IP/MILP is the best IP/MILP solution of the subproblems you have created. It is found in one of the leaves of the tree representing the subproblems.

8.3 Exercises and Projects

8.1. Solve the optimization problems (8.1.1), (8.1.2), and (8.1.3) and their corresponding LP relaxations.

8.2. How can MILP be used to ensure that a variable, x_1, say, can assume only values 1, 2, 3, 4?

8.3. How can MILP be used to ensure that a variable, x_1, say, can assume only values from a given finite set of values?

8.4. * Make an Octave function

```
[zmax, xmax, status] = rlp_solver(c,A,b, rvar, rvarsets)
```

(restricted LP solver) that solves a given standard form LP, given by the input variables `c`, `A`, and `b`, with the additional restrictions that the decision variables indicated by the input variable `rvar` belong to the sets defined by the input variable `rvarsets`.

8.5. * Implement the branch-and-bound algorithm with Octave.

Chapter 9

Mixed Integer Linear Models

> An approximate answer to the right problem is worth a good deal more than an exact answer to an approximate problem. — John Tukey

9.1 Traveling Salesman Problem

MILPs can be use to model and solve many combinatorial problems. Arguably, the most famous, or notorious, combinatorial problem is the **traveling salesman problem**. Many ways of solving it are known. Here we present the MILP way. Unfortunately, no-one knows any fast ways to solve the traveling salesman problem.

> In the **traveling salesman problem** one has to find a tour around N points so that the the total distance traveled is as short as possible.

An example of the traveling salesman problem is:

9.1.1 Example (Maya travels)**.** Maya the Traveler wants to take a Kyiv–Paris–Rome–Vaasa tour (in some order). The distances (in kilometers) between the cities are

	City			
City	Kyiv	Paris	Rome	Vaasa
Kyiv	0	2,023	1,674	1,497
Paris	2,023	0	1,105	1,967
Rome	1,674	1,105	0	2,429
Vaasa	1,497	1,967	2,429	0

In which order should Maya take her tour?

An obvious brute force method to solve the traveling salesman problem is to check all the possible routes and pick the shortest one. The obvious problem with this obvious method is that with N cities we have to check $N!$ routes. E.g., with 20 cities we would have $20! = 2,432,902,008,176,640,000$ routes to check. So we should check over 2 quintillion routes if we are Americans. The Europeans have to check over 2 trillion routes ☺. This is our old Nemesis, the **combinatorial curse**.

Here is a MILP formulation of the traveling salesman problem. It will work much faster than the brute force method. (I wouldn't call it fast, though.)

First we consider the **data** associated with the problem. The data is the cities and their distances. Let C_{ij} be the distance between cities i and j. We can generalize the problem right away by **not** assuming that $C_{ij} = C_{ji}$. So, can we allow that, for some strange reason, the distance from the city i to the city j can be different than the distance from the city j to the city i. However, the distance from the city i to itself is set to be a very very very very big number M_i: $C_{ii} = M_i$. This may look silly (shouldn't it be 0?), but this is to ensure that the traveling salesman does not go to the city i immediately after visiting the same city i (that would be silly).

Now the data of the problem is formulated as the matrix $\mathbf{C} = [C_{ij}]$. Then we can start the LP, or MILP, modeling by using Algorithm 1.1.2.

First, we have to find the **decision variables**. We use indicator decisions:

$$X_{ij} \quad = \quad \begin{cases} 1, & \text{if the salesman goes from the city } i \text{ to the city } j \\ 0, & \text{otherwise} \end{cases}$$

Second, we have to find the **objective function**. But this is to minimize the total traveling distance:

$$\min z \quad = \quad \sum_i \sum_j C_{ij} X_{ij}.$$

Third, we have to find the **constraints**. Now, we have to arrive once at every city. So, for all cities j we must have

$$\sum_i X_{ij} \quad = \quad 1.$$

Also, we must leave every city i once. So, for all cities i we must have

$$\sum_j X_{ij} \quad = \quad 1.$$

Finally, there is a very tricky set of constraints. We must visit **all** the cities by a single continuous path going from one city to another. This is ensured by the constraints: For all $i, j \geq 2$, $i \neq j$, and for some $u_i, u_j \geq 0$,

$$u_i - u_j + N X_{ij} \quad \leq \quad N - 1.$$

These constraints are a bit difficult to understand. What they do is that they prevent subtours, i.e., tours that form a loop that do not contain all the cities. Indeed, suppose that there is a subtour starting from the city 1, say, that does not visit all the cities. Then there must be another subtour that does not visit the city 1. Then the number of cities of this subtour is $R < N$. If we now add together the constraint above corresponding to the X_{ij}s of the latter subtour we get $NR \leq (N-1)R$, since all the u_is cancel. So, the constraint set is violated. On the other hand, interpreting the variables u_i as the first time the city i is visited, we see (at least after some tedious case-checking) that a complete tour will satisfy the constraints above.

Putting what we have found out together we find that the traveling salesman problem with N cities is the MILP

$$
\begin{array}{lrcll}
\min z = & \sum_i \sum_j C_{ij} X_{ij}, & & & \\
\text{s.t.} & \sum_i X_{ij} & = & 1, & \text{for all } j \\
& \sum_j X_{ij} & = & 1, & \text{for all } i \\
& u_i - u_j + N X_{ij} & \leq & N-1 & \text{for all } i \neq j \geq 2, \\
& X_{ij} & = & 0 \text{ or } 1 & \text{for all } i \text{ and } j, \\
& u_i & \geq & 0 & \text{for all } i \geq 2.
\end{array}
\tag{9.1.2}
$$

Note that, in addition to the actual decision variables X_{ij} determining the optimal route, we have the auxiliary variables u_i also.

The following script file solves Maya's problem 9.1.1 with Octaves's `glpk`. It should also be useful in solving any traveling salesman problem with Octave. You might want to copy–paste it to your editor and save it as `maya_travels.m`. To make the copy–pasting easier, we have not included line numbers here. Consequently, we do not explain the script in detail here. Anyway, the in-script comments should help to understand the code.

```
##############################################################################
## SCRIPT FILE:   Maya the Traveler:  Kyiv, Paris, Rome, Vaasa.
##############################################################################

## The distance matrix C, and its vector form c augmented with u:
M = 10000;
C = [   M   2023   1674   1497;
     2023      M   1105   1967;
     1674   1105      M   2429;
     1497   1967   2429      M];
c = [C'(:); 0; 0; 0];

## The first 4*4 columns are for the tour matrix X in the vector form
## and the last 3 columns are for the auxiliary vector u:
A = [ 1  0  0  0  1  0  0  0  1  0  0  0  1  0  0  0    0  0  0;
      0  1  0  0  0  1  0  0  0  1  0  0  0  1  0  0    0  0  0;
      0  0  1  0  0  0  1  0  0  0  1  0  0  0  1  0    0  0  0;
      0  0  0  1  0  0  0  1  0  0  0  1  0  0  0  1    0  0  0;

      1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0    0  0  0;
      0  0  0  0  1  1  1  1  0  0  0  0  0  0  0  0    0  0  0;
      0  0  0  0  0  0  0  0  1  1  1  1  0  0  0  0    0  0  0;
      0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1    0  0  0;

      0  0  0  0  0  0  4  0  0  0  0  0  0  0  0  0    1 -1  0;
      0  0  0  0  0  0  0  4  0  0  0  0  0  0  0  0    1  0 -1;
      0  0  0  0  0  0  0  0  0  4  0  0  0  0  0  0   -1  1  0;
      0  0  0  0  0  0  0  0  0  0  0  4  0  0  0  0    0  1 -1;
      0  0  0  0  0  0  0  0  0  0  0  0  0  4  0  0   -1  0  1;
      0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0    0 -1  1];
##   11 12 13 14 21 22 23 24 31 32 33 34 41 42 43 44

## The bounds and other glpk parameters, and finally call glpk:
L = 50;                                          # For u upper bounds.
b = [1 1 1 1 1 1 1 1 3 3 3 3 3 3]';
lb = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';
ub = [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 L L L]';
ctype = "SSSSSSSSUUUUUU";
vtype = "IIIIIIIIIIIIIIICCC";
param.msglev=0;                                  # Suppress glpk messages.
[x_min,tot_cost,stat] = glpk(c,A,b,lb,ub,ctype,vtype,1,param);

## Get tour.  A tour always starts from the city 1 (why not?).
X = reshape(x_min(1:16),4,4);                    # x_min to matrix form.
tour = [1];                                      # Start from city 1.
for k=1:3                                        # Find the next 3 cities.
  tour = [tour; find(X(tour(k),:))];             # Append city from row k.
endfor

## Print the outputs
tot_cost
tour
```

If you call the script `maya_travels` you get

```
maya_travels
tot_cost =  6243
tour =

   1
   4
   2
   3
```

So, the optimal tour is Kyiv→Vaasa→Paris→Rome(→Kyiv).

9.2 Fixed-Charge Problem

A **fixed-charge problem** is a product selection problem where there are an additional fixed charge to be paid for each product produced.

The examples 9.2.1 and 9.2.3 below illustrate what is going on with the definition above.

9.2.1 Example (Dr. Strangelove's handmade fluids)**.** Dr. Strangelove produces handmade fluids A, B, and C. To manufacture the fluids, he needs labor and raw fluid. Dr. Strangelove wants to maximize his total profit, when the product-wise profits and available resources per week are

	Resource		
Fluid	Labor (hours)	Raw fluid (liters)	*Profit* (Euros)
A	30	4	6
B	20	3	4
C	60	4	7
Availability	150	160	

Dr. Strangelove's problem is a classical product selection problem. Indeed, set

$$\begin{aligned} x_1 &= \text{the amount of fluid A produced} \\ x_2 &= \text{the amount of fluid B produced} \\ x_3 &= \text{the amount of fluid C produced} \end{aligned}$$

Then the LP for Example 9.2.1 is

$$\begin{array}{lrcrcrcrl} \max z = & 6x_1 & + & 4x_2 & + & 7x_3 & & & \\ \text{s.t.} & 30x_1 & + & 20x_2 & + & 60x_3 & \leq & 150 & \text{(labor)} \\ & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 160 & \text{(raw fluid)} \\ & & & & & x_1, x_2, x_3 & \geq & 0 & \end{array} \tag{9.2.2}$$

and here is how you solve it with Octave's `glpk`:

```
octave:1> c=[6 4 7]'; A=[30 20 60; 4 3 4]; b=[150 160]';
octave:2> [x_max,z_max]=glpk(c,A,b,[0 0 0]',[],"UU","CCC",-1)
x_max =

    5
    0
    0

z_max =  30
```

Now we introduce fluid machines. This turns Example 9.2.1 into a **fixed-charge problem**.

9.2.3 Example (Dr. Strangelove's mass-produced fluids)**.** Dr. Strangelove decided to upgrade from handmade fluids to mass produced ones, and the labor hours needed in production dropped to 10%. So, the business data is:

	Resource		
Fluid	Labor (hours)	Raw fluid (liters)	*Profit* (Euros)
A	3	4	6
B	2	3	4
C	6	4	7
Availability	150	160	

But now Dr. Strangelove needs to rent machinery to manufacture his fluids. The renting costs per week are

Machinery	**Rent (Euros)**
Fluid A machine	200
Fluid B machine	150
Fluid C machine	100

What should Dr. Strangelove do now in order to maximize his profit?

Dr. Strangelove's mass-production example 9.2.3 can no longer be modeled as an LP because of the fixed machinery charges. It can, however, be modeled as an MILP. First we use the **indicator trick**: we introduce binary indicator variables

indicating the use of the machineries:

$$
\begin{aligned}
y_1 &= \begin{cases} 1, & \text{if fluid A is produced,} \\ 0, & \text{if fluid A is not produced,} \end{cases} \\
y_2 &= \begin{cases} 1, & \text{if fluid B is produced,} \\ 0, & \text{if fluid B is not produced,} \end{cases} \\
y_3 &= \begin{cases} 1, & \text{if fluid C is produced,} \\ 0, & \text{if fluid C is not produced.} \end{cases}
\end{aligned}
$$

Then the objective function becomes

$$
\begin{array}{rcrcrcrl}
z & = & 6x_1 & + & 4x_2 & + & 7x_3 & \text{(linear profit)} \\
 & & -200y_1 & - & 150y_2 & - & 100y_3 & \text{(fixed charge)}
\end{array}
\tag{9.2.4}
$$

What about the constraints then? First, of course, we have the constraints for labor and raw fluid as in the no-fixed-charge case (9.2.2):

$$
\begin{array}{rcrcrcrl}
3x_1 & + & 2x_2 & + & 6x_3 & \leq & 150 & \text{(labor)} \\
4x_1 & + & 3x_2 & + & 4x_3 & \leq & 160 & \text{(raw fluid)}
\end{array}
\tag{9.2.5}
$$

Then comes the **big M trick** (and this is really ingenious): in order to ensure that the binary variables y_i are 1 whenever the corresponding production variables x_i are non-zero, we impose

$$
\begin{array}{rcl}
x_1 & \leq & M_1 y_1 \\
x_2 & \leq & M_2 y_2 \\
x_3 & \leq & M_3 y_3
\end{array}
\tag{9.2.6}
$$

where M_1, M_2, and M_3 are very, very, very, very big numbers. Indeed, if y_i is not 1, then it is 0, and, consequently, by the constraint $x_i \leq M_1 y_i$, x_i must be zero also. On the other hand, if M_i is big enough, the constraint $x_i \leq M_i y_i$ does not unnecessarily restrict the size of x_i when $y_i = 1$. In practice, one should take M_i to be **at least** the maximum value that x_i may attain. Finally, we have the sign and the binarity constraints

$$
\begin{array}{rcl}
x_1, x_2, x_3 & \geq & 0 \\
y_1, y_2, y_3 & = & 0 \text{ or } 1
\end{array}
\tag{9.2.7}
$$

Putting (9.2.4), (9.2.5), (9.2.6), and (9.2.7) together we obtain, by using Algorithm 3.1.2, the following MILP for Example 9.2.3:

$$
\begin{array}{lrrrrrrcr}
\max z = & 6x_1 & +4x_2 & +7x_3 & -200y_1 & -150y_2 & -100y_3 & & \\
\text{s.t.} & 3x_1 & +2x_2 & +6x_3 & & & & \leq & 150 \\
 & 4x_1 & +3x_2 & +4x_3 & & & & \leq & 160 \\
 & x_1 & & & -M_1 y_1 & & & \leq & 0 \\
 & & x_2 & & & -M_2 y_2 & & \leq & 0 \\
 & & & x_3 & & & -M_3 y_3 & \leq & 0 \\
 & & & & & & x_1, x_2, x_3 & \geq & 0 \\
 & & & & & & y_1, y_2, y_3 & = & 0 \text{ or } 1
\end{array}
\tag{9.2.8}
$$

There is still a slight problem with LP (9.2.8): `glpk` does not understand binary variables. The problem is easily solved by noting that the constraints $y_i = 0$ or 1 is the same as the constraints $y_i \geq 0$, $y_i \leq 1$, y_i integer. So, we get the form that `glpk` can understand:

$$
\begin{array}{lrrrrrrrrr}
\max z = & 6x_1 & +4x_2 & +7x_3 & -200y_1 & -150y_2 & -100y_3 & & \\
\text{s.t.} & 3x_1 & +2x_2 & +6x_3 & & & & \leq & 150 \\
& 4x_1 & +3x_2 & +4x_3 & & & & \leq & 160 \\
& x_1 & & & -M_1 y_1 & & & \leq & 0 \\
& & x_2 & & & -M_2 y_2 & & \leq & 0 \\
& & & x_3 & & & -M_3 y_3 & \leq & 0 \\
& & & & y_1 & & & \leq & 1 \\
& & & & & y_2 & & \leq & 1 \\
& & & & & & y_3 & \leq & 1 \\
& & & & x_1, x_2, x_3, & y_1, & y_2, y_3 & \geq & 0 \\
& & & & & & y_1, y_2, y_3 & \text{integer} &
\end{array}
\tag{9.2.9}
$$

Now, the LP (9.2.9) of Dr. Strangelove's mass-production Example (9.2.3) can be solved, e.g., with the Octave's function `glpk`. In the code we set all the M_is to be 100. Of course, 100 is not a very very very very big number, but a very very very very big number may cause problems rounding errors in Octave. In any case, 100 is big enough.

```
octave:1> M = 100;
octave:2> c = [6 4 7 -200 -150 -100]';
octave:3> A = [
> 3 2 6  0  0  0;
> 4 3 4  0  0  0;
> 1 0 0 -M  0  0;
> 0 1 0  0 -M  0;
> 0 0 1  0  0 -M;
> 0 0 0  1  0  0;
> 0 0 0  0  1  0;
> 0 0 0  0  0  1];
octave:4> b = [150 160 0 0 0 1 1 1]';
octave:7> [x_max,z_max] = glpk(c,A,b,[0 0 0 0 0 0]',[],"UUUUUUUU","CCCIII",-1)
x_max =

    0
    0
   25
    0
    0
    1

z_max =  75
```

So, Dr. Strangelove should produce only fluid C.

9.3 Set-Covering Problem

> In a **set-covering problem**, each element of a given set must be covered by elements from, possibly, another set. The objective is to use as few elements from the other set as possible.

The following rather typical example of a set-covering problem illustrates the rather nebulous definition above.

9.3.1 Example (Count the time)**.** There are six villages in the county of Brutopia. The count has to decide where to place knights in order to keep the peasants from revolting. If the peasants in a village are revolting a knight should get there in at least 15 hours: otherwise the count's liege-lord, The Holy Emperor, gets angry. The knights must be placed into villages. The time (in hours) it takes a knight to travel from one village to another are:

	To					
From	1	2	3	4	5	6
1	0	10	20	30	30	20
2	10	0	25	35	20	10
3	20	25	0	15	30	20
4	30	35	15	0	15	25
5	30	20	30	15	0	14
6	20	10	20	25	14	0

The count wants to hire as few knight as possible. What should the count do?

In Example 9.3.1 the set to be covered is the villages and the covering set is the villages, also. Since there is the 15 hour limit we have the following table for the coverage:

Village	**Covers villages**
1	1,2
2	1,2,6
3	3,4
4	3,4,5
5	4,5,6
6	2,5,6

Let us now turn into the LP formulation of the set-covering problem 9.3.1. According to Algorithm 1.1.2 we first decide the **decision variables**. So, what

does the count has to decide? He has to decide whether to place a knight in a given village or not. So, the decision variables are indicators

$$x_i \;=\; \begin{cases} 1, & \text{if a knight is placed in the Village } i, \\ 0, & \text{otherwise.} \end{cases}$$

Now the **objective function** to be minimized is the total number of knights placed in the villages. Since the decision variables are indicators, this is simply the sum of the decision variables:

$$z \;=\; x_1 + x_2 + x_3 + x_4 + x_5 + x_6.$$

What about the **constraints** then? For each village we have the constraint that there must be a knight within 15 hours. So, for example, Village 1 constraint is

$$x_1 + x_2 \;\geq\; 1,$$

i.e., there must be a knight either in Village 1 or in Village 2 (or in both). This is because Village 2 covers Village 1 and Village 2. (Actually, we should think the other way around: Village 1 is covered by Village 1 and Village 2. There is symmetry here, however. So, it does not matter which way you think. Be careful, though! In some set-covering problems it may matter.) Now, from the coverage table above we get the constraints for all the 6 villages as

$$\begin{array}{rcrcrcrcrcrcl}
x_1 & + & x_2 & & & & & & & & & \geq & 1 \\
x_1 & + & x_2 & & & & & & & + & x_6 & \geq & 1 \\
 & & & x_3 & + & x_4 & & & & & & \geq & 1 \\
 & & & x_3 & + & x_4 & + & x_5 & & & & \geq & 1 \\
 & & & & & x_4 & + & x_5 & + & x_6 & & \geq & 1 \\
 & & x_2 & & & & + & x_5 & + & x_6 & & \geq & 1
\end{array}$$

Finally, remember that we have binarity constraints: $x_i = 0$ or 1. (Actually, integrity constraints would be enough.)

Putting it all together we have the following BILP for Example 9.3.1:

$$\begin{array}{lrcrcrcrcrcrcl}
\min z = & x_1 & + & x_2 & + & x_3 & + & x_4 & + & x_5 & + & x_6 & & \\
\text{s.t.} & x_1 & + & x_2 & & & & & & & & & \geq & 1 \\
 & x_1 & + & x_2 & & & & & & & + & x_6 & \geq & 1 \\
 & & & & & x_3 & + & x_4 & & & & & \geq & 1 \\
 & & & & & x_3 & + & x_4 & + & x_5 & & & \geq & 1 \\
 & & & & & & & x_4 & + & x_5 & + & x_6 & \geq & 1 \\
 & & & x_2 & & & & & + & x_5 & + & x_6 & \geq & 1 \\
 & & & & & & & x_1, x_2, x_3, x_4, x_5, x_6 & & & & & = & 0 \text{ or } 1
\end{array}$$

9.4 Exercises and Projects

9.1. Mr. Kulukumulukku is a traveling salesman selling kalakukkos. Each week he travels to the five biggest cities of Finland to sell his kalakukkos. Mr. Kulukumulukku wants to minimize his total traveling distance. What is his optimal tour, when

(a) he just wants to make the tour,
(b) he starts and ends his tour from his home town of Kuopio?

(If you don't know the five biggest cities of Finland or their distances to each others and to Kuopio, you might want to consult Google.)

9.2. * Make an Octave function

```
[tot_dist, tour] = tsp_solver(C)
```

that solves the traveling salesman problem. The input parameter `C` is the distance matrix. The output parameters are `tot_dist` for the total distance traveled and `tour` for the optimal tour: e.g., if `tour = [1 3 2 4]'`, then the optimal tour connecting the cities $1, 2, 3, 4$ is $1 \to 3 \to 2 \to 4 (\to 1)$.

The script `maya_travels` can be helpful here.

9.3. How low should the rent for the fluid A machine in Example 9.2.3 be so that it makes sense for Dr. Strangelove to produce fluid A? What about fluid B?

9.4. Consider examples 9.2.1 and 9.2.3. Suppose Dr. Strangelove has an option to rent (some of) the fluid machines. What should he do?

Here, the following a big M trick for the "logical or" and the "logical if–then" could be helpful: The logical constraint

$$f(x_1, \ldots, x_n) \le 0 \quad \text{or} \quad g(x_1, \ldots, x_n) \le 0$$

can be written as

$$\begin{aligned} f(x_1, \ldots, x_n) &\le My, \\ g(x_1, \ldots, x_n) &\le M(1-y), \end{aligned}$$

where $y = 0$ or 1 and M is a very very very very very very big number. In a similar way, since "if A then B" is "(not A) or B", the logical constraint

$$\text{if} \quad f(x_1, \ldots, x_n) > 0 \quad \text{then} \quad g(x_1, \ldots, x_n) \ge 0$$

can be written as

$$\begin{aligned} -g(x_1, \ldots, x_n) &\le My, \\ f(x_1, \ldots, x_n) &\le M(1-y), \end{aligned}$$

where $y = 0$ or 1 and M is a very very very very very very big number.

9.5. Solve the set-covering problem of Example 9.3.1.

9.6. Consider the set-covering example 9.3.1. Suppose Village 2 is such a dreadful place that no knight would want to live there. What should the count do now?

9.7. * Make an Octave function

```
cover_points = scp_solver(C)
```

That solves a general set-covering problem. The output variable `cover_points` are the points in the minimal covering and the input variable `C` is the covering matrix: $C_{ij} = 1$ if the point i covers the point j, $C_{ij} = 0$ otherwise.

9.8. The Holy Emperor is waging a desperate war against The Supreme Sultan. The count of Brutopia, who is a vassal of The Holy Emperor, fathoms that the war will be lost in exactly 3 years, and after the war there will be no county, nor count, of Brutopia. So, the count decides to capitalize his possessions. After the capitalization, the count intends to disappear into the New World. The count's income comes from the 6 villages under his yoke. The average monthly income of the villages are (in Holy Imperial Ducats):

Village	**Taxes**
1	8
2	19
3	18
4	18
5	19
6	102

To collect the taxes, the count must hire imperial tax-collectors. The imperial tax-collectors are placed in the villages and their salary is 2 Holy Imperial Ducats per week. The time it takes for a tax-collector based in the village i to collect the taxes from the village j, and then to deliver them to the count are (in days):

	Taxed village						
Tax-collectors village	1	2	3	4	5	6	**Delivery**
1	0	100	100	107	101	192	12
2	100	0	128	135	197	197	71
3	100	128	0	115	130	189	87
4	107	135	115	0	192	185	88
5	101	197	130	192	0	184	89
6	192	197	189	185	184	0	97

Now, the Holy Emperor finances his war by diluting the gold content of the Holy Roman Ducats by 1% each day.

How much should The Supreme Sultan offer to the count of Brutopia to disappear immediately?

Appendix A

Octave Codes

Infinite loop See: *Loop, infinite*
Loop, infinite See: *Infinite loop*
Recursion See: *Recursion*

—The Contradictionary

The contents of the function m-files used in these notes are listed here. The m-files are also available from www.uwasa.fi/~tsottine/or_with_octave/. The line numbers are not a part of the code: they are for reference purposes only. The functions are listed here in the alphabetical order, since the author was unable to decide what would be the logical order.

A.1 first_simplex_tableau

```
 1 function [T,BV] = first_simplex_tableau(c,A,b)
 2 ## Function [T, BV] = first_simplex_tableau (c, A, b) builds the first
 3 ## simplex tableau for the function simplex_lp_solver.  T is the tableau,
 4 ## BV are the indexes of the basic variables (the slacks in this case).
 5
 6 ## n is the number of decision variables, m is the number of constraints.
 7 [m,n] = size(A);
 8
 9 ## The simplex tableau without the BV information.
10 T = [          1  -c'  zeros(1,m)  0;
11       zeros(m,1)    A      eye(m)  b];
12
13 ## The indexes of the BV's (the slacks in this case).
14 BV = ( (n+1):(n+m) )';
15 endfunction
```

A.2 is_optimal_simplex_tableau

```
1 function optimal = is_optimal_simplex_tableau(T,BV)
```

```
 2 ## Function optimal = is_optimal_simplex_tableau (T,BV) tells (to the
 3 ## function simplex_lp_solver) if the simplex tableau [T, BV] is optimal.
 4
 5 ## The tableau [T,BV] is optimal if in the first row all the
 6 ## coefficients are non-negative for all Non-Basic Variables (NBV).
 7 NBV = get_NBV(BV,columns(T)-2);  # Get NBV (cf. the function below).
 8 optimal = 0;                     # A priori assume non-optimality.
 9 NBVval = T(1,NBV+1);             # First column is for z, hence NBV+1.
10 if ( all(NBVval>=0) )            # Check for optimality.
11   optimal = 1;
12 endif
13 endfunction
14
15 ##############################################################################
16 ## Auxiliary function to get NBV indexes from BV indexes.
17 ##############################################################################
18
19 function NBV = get_NBV(BV,nvar)
20 vars = ones(nvar,1);            # Set "true" to all indexes.
21 vars(BV) = 0;                   # Set "false" to BV indexes.
22 NBV = find(vars);               # Get the "true" indexes.
23 endfunction
```

A.3 new_simplex_tableau

```
 1 function [Tnew,BVnew,status] = new_simplex_tableau(T,BV)
 2 ## Function [Tnew, BVnew, status] = new_simplex_tableau (T, BV) builds
 3 ## a new, hopefully better, simplex tableau [Tnew, BVnew] from the
 4 ## tableau [T, BV], or detects the solution to be unbounded.
 5 ## This function is used by the function simplex_lp_solver.
 6
 7 ## Variable initializations and short-hand notations.
 8 status = "unknown";               # Paranoia!
 9 Tnew = T; BVnew = BV;             # New tableau is old tableau (for now).
10 [m,n] = size(T);                  # Almost the dimensions of A.
11
12 ## Get the entering BV.
13 coeff = T(1,2:(n-1));             # Coeffs. of the decisions and slacks.
14 [tmp,in] = min(coeff);            # Index of the entering coefficient.
15
16 ## Get the leaving BV, or detect unboundedness and leave the function.
17 rhs = T(2:m,n);                   # RHS of the constraint rows.
18 cin = T(2:m,in+1);                # Coeffs. of the entering variables.
19 ratio = rhs./cin;                 # Pointwise ratios.
20
21 ## Set all the "no-limit" ratios to -infinity
22 for i=1:length(ratio)
23   if ( ratio(i)==Inf || ratio(i)<0 )
24     ratio(i) = -Inf;
25   endif
26 endfor
27
```

```
28 ## Check boundedness. Exit if unbounded.
29 if ( all(ratio<0) )
30   status = "unbounded";
31   return;
32 endif
33
34 [tmp,out] = min(abs(ratio));      # Get the winner index.
35
36 ## Solve T the new BV.
37 BVnew(out) = in;                  # The new BV.
38 Tnew = pivot(T,out+1,in+1);       # Solve T for the new BV by pivoting.
39 endfunction
40
41 ##############################################################################
42 ## The auxiliary function pivot
43 ##############################################################################
44
45 function Ap = pivot(A, r, c)
46 ## Function Ap = pivot (A, r, c) pivots A with row r and column c.
47
48 A(r,:) = A(r,:)/A(r,c);           # Get 1 for the pivot in the pivot row.
49 for i = 1:rows(A)                 # Remove the pivot from other rows.
50   if (i != r)
51     A(i,:) = A(i,:) - A(i,c)*A(r,:);
52   endif
53 endfor
54 Ap = A;                           # Return the pivoted matrix Ap.
55 endfunction
```

A.4 npv

```
 1 function v = npv(cf,r)
 2 ## Function v = npv(cf,r) returns the Net Present Value (npv) of the
 3 ## cash flow cf.  The cash flow cf is received at the end of each
 4 ## period.  The  rate of return over the period is r.  The parameter r
 5 ## is scalar.  The cash flow cf is a (column) vector.
 6
 7 T = length(cf);                 # The number of periods.
 8 pv = zeros(T,1);                # Initialize present values (pv) at zero.
 9 for t=1:T
10   pv(t) = cf(t) / (1+r)^t;      # Set the pv's.
11 endfor
12 v = sum(pv);                    # npv is the sum of pv's.
13 endfunction
```

A.5 simplex_lp_solver

```
 1 function [z_max, x_max, status] = simplex_lp_solver(c, A, b, maxiter=100)
 2 ## Function [z_max, x_max, status] =
 3 ## simplex_lp_solver (c, A, b, maxiter=100) solves the LP
```

```
 4 ##
 5 ##     max  c'*x
 6 ##     s.t.  A*x <= b
 7 ##             x >= 0,
 8 ##
 9 ## where b>=0, by using the standard simplex algorithm.
10 ##
11 ## Input:
12 ##
13 ## c, A, b
14 ##      The (column) vector defining the objective function, the
15 ##       technology matrix, and the (column) vector of the constraints.
16 ##
17 ## maxiter
18 ##      The maximum iterations used in finding a solution (default: 100).
19 ##
20 ## Output:
21 ##
22 ## z_max, x_max
23 ##      The maximal value of the objective and an optimal decision.
24 ##
25 ## status
26 ##      A string indicating the status of the solution:
27 ##     "bounded"
28 ##           A bounded solution was found.
29 ##      "unbounded"
30 ##           The solution is known to be unbounded.
31 ##      "infeasible"
32 ##           Solutions are known not to exist (not applicable for b>=0).
33 ##      "unknown"
34 ##           The algorithm failed.
35 ##
36 ## simplex_lp_solver uses the functions  first_simplex_tableau,
37 ## new_simplex_tableau, and is_best_simplex_tableau.
38 ##
39 ## See also:  glpk.
40
41 ## Get the first simplex tableau.
42 [T,BV] = first_simplex_tableau(c,A,b);
43
44 ## Look for better simplex tableaux, but avoid the infinite loop.
45 status = "unknown";                          # Status still unknown :)
46 iter = 0;                                    # No iterations yet.
47 while ( iter<maxiter && ...
48         !is_optimal_simplex_tableau(T,BV) && ...
49         !strcmp(status,"unbounded") )
50   [T,BV,status] = new_simplex_tableau(T,BV); # New [T,BV], unbounded (?).
51   iter = iter + 1;                          # We just had an iteration.
52 endwhile
53
54 ## Exit (with zeroes), if a solution was not found or found unbounded.
55 if ( iter>=maxiter || strcmp(status,"unbounded") )
56   z_max = 0;
57   x_max = zeros(length(c),1);
```

```
58   return;
59 endif
60
61 ## Collect the results from the last simplex tableau.
62 status = "bounded";                          # We know this now.
63 z_max = T(1,columns(T));                     # z_max is in the NE corner.
64 x_max = zeros(length(c)+length(b),1);       # Zeros for now.
65 x_max(BV) = T(2:(length(b)+1),columns(T));  # Put BV values to x_max.
66 x_max = x_max(1:length(c));                 # Cut the slacks away.
67 endfunction
```

A.6 stu_lp_solver

```
 1 function [z_max, x_max, status] = stu_lp_solver(c, A, b)
 2 ## Function [z_max, x_max, status] = stu_lp_solver(c, A, b) solves the LP
 3 ##
 4 ##     max  c'*x
 5 ##     s.t.  A*x <= b
 6 ##             x >= 0,
 7 ##
 8 ## by using glpk.
 9 ##
10 ## Input:
11 ##
12 ## c, A, b
13 ##      The (column) vector defining the objective function, the
14 ##       technology matrix, and the (column) vector of the constraints.
15 ##
16 ## Output:
17 ##
18 ## z_max, x_max
19 ##      The maximal value of the objective and an optimal decision.
20 ##
21 ## status
22 ##      A string indicating the status of the solution:
23 ##     "bounded"
24 ##           A bounded solution was found.
25 ##      "unbounded"
26 ##           The solution is known to be unbounded.
27 ##      "infeasible"
28 ##           Solutions are known not to exist.
29 ##      "unknown"
30 ##           The algorithm failed.
31 ##
32 ## See also:  glpk.
33
34 ## Get m, the number of constraints (excluding the sign constraints), and
35 ## n, is the number of decision variables, from the technology matrix A.
36 [m,n] = size(A);
37
38 ## Some input-checking:
39 if ( nargin!=3 )
```

```
40   error("stu_lp_solver: The number of input arguments must be 3.\n");
41 elseif ( columns(c)>1 )
42   error("stu_lp_solver: Objective c must be a column vector.\n");
43 elseif (columns(b)>1 )
44   error("stu_lp_solver: Upper bounds b must be a column vector.\n");
45 elseif (rows(c)!=n || rows(b)!=m )
46   error("stu_lp_solver: The dimensions of c, A, and b do not match.\n");
47 endif
48
49 ## Set the parameters for glpk:
50
51 ## Set the decision-wise lower bounds all to zero, i.e. build a zero
52 ## column vector with n zeros.
53 o = zeros(n,1);
54
55 ## Set the sense of each constraint as an upper bound.
56 ctype = "";                                 # Start with an empty string.
57 for i=1:m                                   # Loop m times.
58   ctype = [ctype, "U"];                     # Append "U" to the string.
59 endfor
60
61 ## Set the type of each variable as continuous.
62 vtype = "";                                 # Start with an empty string.
63 for i=1:n                                   # Loop n times.
64   vtype = [vtype, "C"];                     # Append "C" to the string.
65 endfor
66
67 ## Solve the system by calling glpk.
68 [x_max, z_max, STATUS] = glpk(c, A, b, o, [], ctype, vtype, -1);
69
70 ## Set the STATUS code given by glpk to the appropriate string
71 status = "unknown";                         # Pessimism by default.
72 if ( STATUS==180 )                          # Everything went fine.
73   status="bounded";
74 elseif ( STATUS==183 )                      # LP infeasible detected.
75   status="infeasible";
76 elseif ( STATUS==184 )                      # LP unboundedness detected.
77   status="unbounded";
78 endif
79 endfunction
```

A.7 trans_solver

```
1 function [cost,schedule] = trans_solver(C,s,d)
2 ## Function [cost, schedule] = trans_solver(C, s, d) solves the
3 ## balanced transportation problem
4 ##
5 ##         -----   -----
6 ##         \       \
7 ##    min   >       >     C(i,j)*X(i,j)
8 ##         /       /
9 ##         -----   -----
```

```
10 ##           i       j
11 ##
12 ##           -----
13 ##           \
14 ##    s.t.    >     X(i,j) = s(i)  for all ports i
15 ##           /
16 ##           -----
17 ##             j
18 ##
19 ##           -----
20 ##           \
21 ##            >     X(i,j) = d(j)  for all markets j
22 ##           /
23 ##            -----
24 ##              i
25 ##
26 ## Input:
27 ##
28 ## C, s, d
29 ##      The matrix of the transportation costs, the column vector
30 ##      of the supplies, and the column vector of the demands.
31 ##
32 ## Output:
33 ##
34 ## cost, schedule
35 ##      The minimal transportation cost and schedule.
36 ##
37 ## See also:  glpk.
38
39 ## Some short-hands.
40 m1 = length(s);                                  # Number of ports.
41 m2 = length(d);                                  # Number of markets.
42 n = m1*m2;                                       # Number of decisions.
43 o = zeros(n,1);                                  # For glpk lower bounds.
44
45 ## Build the technology matrix A
46 A = zeros(m1+m2,n);                              # Initialization.
47 for i = 1:m1
48   A(i, (1:m2)+(i-1)*m2) = ones(1,m2);           # Supply side.
49   A((1:m2)+m1, (1:m2)+(i-1)*m2) = eye(m2);      # Demand side.
50 endfor
51
52 ## Set the sense of bounds (equality for balanced).
53 ctype = "";
54 for i=1:(m1+m2)
55   ctype = [ctype, "S"];
56 endfor
57
58 ## Set the type of each decision variable as continuous.
59 vtype = "";
60 for i=1:n
61   vtype = [vtype, "C"];
62 endfor
63
```

```
64 ## Solve the system by calling glpk.
65 [schedule, cost] = glpk(C'(:), A, [s; d], o, [], ctype, vtype, 1);
66
67 ## Reshape the schedule vector to a matrix.
68 schedule = reshape(schedule, m2, m1)';
69 endfunction
```

Index