

## Optimisation & Operations Research

Haide College, Spring Semester

### Practical 1 (1%)

See website for practical and due dates

Work through the below instructions in MATLAB and MATLAB Grader as indicated. Submit to MATLAB Grader by the due date.

### Built-in solvers and reading data

*Matt Roughan, Mike Chen*

In this practical you will learn about how to solve linear programs using MATLAB's built-in `linprog` solver. This will give you a taste of the powerful optimisation tools available in MATLAB.

You will also learn about the following:

- reading in data from a file
- using MATLAB documentation

#### 1. Check the Optimization Toolbox is installed

There are many built-in functions in MATLAB for solving optimisation problems, these are mostly part of the **Optimization Toolbox**. Before proceeding make sure you have the **Optimization Toolbox** installed. To check this enter the following in the MATLAB command window on your computer.

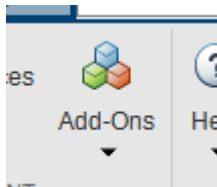
```
>> ver
```

If you have the Optimization Toolbox installed you should see something like this:

MATLAB	Version 9.12	(R2022a)
Deep Learning Toolbox	Version 14.4	(R2022a)
Image Processing Toolbox	Version 11.5	(R2022a)
Optimization Toolbox	Version 9.3	(R2022a)
Symbolic Math Toolbox	Version 9.1	(R2022a)

If this list include the Optimization Toolbox then continue to the next question.

If the Optimization Toolbox is not in your list, then install it by pressing the 'Add-Ons' button:



Find the Optimization Toolbox in the window that pops up, the easiest way to do is to enter "optimization" in the search box, and then install it!

#### 2. Learn the basics of `linprog`

The function for solving linear programs in MATLAB is called `linprog`. This function works similarly to the Simplex routine you are currently writing for Assignment 2.

The inputs of `linprog` are:

- a vector defining the objective;
- a matrix for the left-hand side of the constraints;
- a vector for the right-hand side of the constraints;

Some of the details of using `linprog` are a little different to what we typically have been using in particular:

- by default the problem is in standard **inequality** form, but you can specify a mix of inequality and equality constraints
- the objective is **minimised** rather than maximised

Get started by looking up **and carefully reading** the documentation for `linprog`. This can be found at: <https://www.mathworks.com/help/optim/ug/linprog.html>

a) The documentation includes the following simple example:

```
A = [1 1
     1 1/4
     1 -1
     -1/4 -1
     -1 -1
     -1 1];
```

```
b = [2 1 2 1 -1 2];
```

```
f = [-1 -1/3];
```

```
x = linprog(f,A,b)
```

Run this code on MATLAB on your computer and make sure you understand what it is doing.

b) Rewrite the problem in equality form (with slack variables) and solve it using `linprog`.  
**Hint:** Define `Aeq` for the equality constraints, a new objective function with enough variables and include an assumption that the variables are all non-negative.

The command to call the solver will look like:

```
x = linprog(f, [], [], Aeq, beq, lb, ub)
```

where the `[]` indicate empty input (since you will have only equality constraints).

3. Code up and solve 'Our First Problem' using `linprog`.

4. Recall from Tutorial 1 the **portfolio problem**.

A bank has \$1 million to invest in variety of bonds offered by the government and other agencies. Each bond has a *rated quality*, and an *after-tax yield*, and a *years to maturity* (how long the investment is committed). The portfolio manager must try to maximise the return on investment, but must also meet other criteria:

- the average quality of bonds cannot be worse than 1.5 (note that for quality, a low number corresponds to high-quality)

- the average years to maturity should not exceed 4 years.

Assuming we have  $N$  bonds (rather than the 4 discussed in the tutorial). The linear program for this problem is:

$$\max \sum_{i=1}^N y_i x_i,$$

subject to

$$\sum_{i=1}^N (q_i - Q) x_i \leq 0,$$

$$\sum_{i=1}^N (m_i - M) x_i \leq 0,$$

$$\sum_{i=1}^N c_i x_i = C,$$

with all  $x_i \geq 0$ . Here  $x_i$  is the number of each bond purchased,  $y_i$  is the yield,  $q_i$  is the quality (lower is better),  $m_i$  is the years to maturity and  $c_i$  is the cost.

We will look at a (relatively) large number of bonds with  $N = 50$ . The values of  $y_i$ ,  $q_i$  and  $m_i$  are given in the file `portfolio.csv`, which can be found on Cloudcampus (and is also available in MATLAB Grader). It is assumed that the cost  $c_i$  for all bonds is \$1.

- Write a script that uses the MATLAB function `readtable` to read in the data file. You could open this in a text editor to see what the contents look like. Note there is a command for you to adapt given in the MATLAB Grader template.

Look up the documentation to find out what this command does and then work out how you can use it. You can either go to the documentation website or enter `help readtable` in the command window.

- Continuing in your script. Use the data you have read in to form the various matrices and vectors of the linear program. When ready solve this using `linprog`.
  - Copy your script to MATLAB Grader to make sure it passes all the tests. **You must submit your script to get the 1% course mark.**
  - What proportion of all bonds were included in your portfolio?
  - What overall yield do you achieve?
  - What average yield do you achieve (as all of the bonds cost the same, the average yield will be the total divided by the number of bonds)?
-