THE UNIVERSITY of ADELAIDE

Optimisation & Operations Research

Haide College, Spring Semester

Practical 3 (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.

Problem-based optimisation in MATLAB

Matt Roughan, Mike Chen

The aim of this practical is to introduce the "problem-based" optimisation workflow in MATLAB.

Up until this point we have been entering our optimisation problems into MATLAB using vectors and matrices. This is a reasonable approach for relatively small problems, but for larger problems with many variables and/or constraints it can become problematic.

One difficulty in representing a problem in this way is that it can be difficult to debug and check that a problem has been entered correctly. It would be good if we could enter a problem in a way that avoid the steps of (for example) constructing a matrix for the constraints, and could state the problem in a way that is easy to read.

An alternative way of entering optimisation problems in MATLAB does just that! We will use this to solve a couple of familiar problems.

Similar problem based approaches are found in other software including AMPL (used widely in industry) and JuMP/Julia (modern optimisation package).

---

1. **"Our first problem" as a "problem"**

   We will demonstrate this approach by entering everyone's favourite manufacturing problem.

   To do this, **open a new script in MATLAB** and enter the following.

   ```
   our_first_problem = optimproblem;
   ```

   This create a new optimisation problem object called `our_first_problem`.

   Now that we have a problem, we would like do define some variables. We create the three variables of this problem as follows:

   ```
   x = optimvar('x',3,'LowerBound',0);
   ```

   This creates three variables `x(1)`, `x(2)` and `x(3)`, which can be used to define constraints and the objective. Note that we can set options for these variables, in this case a lower bound of zero so that the variables are non-negative. Some other options are:

   - `'UpperBound'`, put an upper bound on the variables
   - `'Type'`, this can take a value of either `'continuous'` or `'integer'`, as needed.

   We then add the objective function to the problem:

   ```
   our_first_problem.Objective = 13*x(1) + 12*x(2) + 17*x(3);
   our_first_problem.ObjectiveSense = 'maximize';
   ```

   Here we have defined the `Objective` property of `our_first_problem` to be a function of `x(1)`, `x(2)` and `x(3)`. We have also set the `ObjectiveSense` property to `'maximize'` since it is minimise by default.

   Finally, we add the three constraints. Here we use the variables again, but write them as inequalities with <=.

```
our_first_problem.Constraints.labour = 2*x(1) + x(2) + 2*x(3) <= 225;
our_first_problem.Constraints.metal = x(1) + x(2) + x(3) <= 117;
our_first_problem.Constraints.wood = 3*x(1) + 3*x(2) + 4*x(3) <= 420;
```

Note all these constraints are stored in the `Constraints` property of the problem, and have sensible names.

To inspect (and check the problem) enter

```
show(our_first_problem)
```

This give the output

```
Solve for:
       x

maximize :
       13*x(1) + 12*x(2) + 17*x(3)


subject to labour:
       2*x(1) + x(2) + 2*x(3) <= 225

subject to metal:
       x(1) + x(2) + x(3) <= 117

subject to wood:
       3*x(1) + 3*x(2) + 4*x(3) <= 420

variable bounds:
       0 <= x(1)
       0 <= x(2)
       0 <= x(3)
```

This output lets you easily check that you have entered your problem correctly. Each constraint has a name indicating what it represents.

To solve the problem enter:

```
[sol,fval] = solve(our_first_problem);
```

The optimal value of the objective is `fval` and the corresponding value of the variables are stored in `sol.x`.

---

2. **Add integer constraints**

   It is straightforward to modify your script to add integer constraints to the three variables. Change the variable definition line to

   ```
   x = optimvar('x',3,'LowerBound',0,'Type','integer');
   ```

   Rerun your script with this small change. Note how the problem description from the `show` command has slightly changed, that the maximum value of the objective has slightly decreased, and that the variables are now integers!

   Enter this version of the problem (with integer valued variables) into MATLAB Grader for your participation mark for this practical.

---

3. **Knapsack problem as a "problem"**

   We now return to the knapsack problem from the course notes.

   Download the file `knapsack_example.mat` from Cloudcampus and save it to your working directory in MATLAB. Load this file by entering

   ```
   load('knapsack_example.mat')
   ```

   You should see the variable `knapsack_table` in your workspace. Inspect this (double-click on it), and see that it contains the item names, weights and values for this knapsack problem.

   Copy and paste the following code into a script. Compare it to the code from Question 1 to get a feel for what it does.

   ```
   knapsack_example = optimproblem;
   x = optimvar('x',knapsack_table.item,'LowerBound',0,'UpperBound',1,'Type','integer');
   knapsack_example.Objective = knapsack_table.value'*x;
   knapsack_example.ObjectiveSense = 'max';
   knapsack_example.Constraints.weight = knapsack_table.weight'*x <= 2.5;
   show(knapsack_example)
   [sol,fval] = solve(knapsack_example)
   ```

   Note the following:

   - the variables here are called things like `x('chocolate')` and `x('jumper')`, which makes it very clear what each variable is.
   - a number of options are added to the variables
   - the objective is defined by multiplying (inner product) the vectors containing the variables `x` and the vector of the value of each item `knapsack_table.value`.
   - similarly the constraint is defined as the inner product of two vectors.

   Once you have run the code and tried to understand how it works, add an extra item:

   - `'map'`, with weight 0.1 kg and value 2.

   To do this add an extra row to the end of `knapsack_table`, and then rerun the script.

   Storing the parameters of an optimisation problem in this way (or similar) is strongly preferred for larger problems, in particular because it is easy to extend - as we have now seen!

---

4. Recall the following **transportation problem**.

   There are three warehouses at different cities: Sydney, Melbourne and Adelaide. They have 250, 130 and 235 tonnes of paper accordingly. There are four publishers, in Sydney, Melbourne, Brisbane and Hobart. They ordered 75, 230, 240 and 70 tonnes of paper to publish new books. There are the following costs (in tens of dollars) of transportation of one tonne of paper:

From / To	Sydney	Brisbane	Melbourne	Hobart
Sydney	15	20	16	21
Melbourne	25	13	5	11
Adelaide	15	15	7	17

The idea is to find a *transportation plan* such that all orders will be met and the transportation costs will be minimized.

Let x_{ij} = be the number of tonnes of paper shipped from Warehouse i ($i = 1$: Sydney; $i = 2$: Melbourne; $i = 3$: Adelaide), to publisher in City j ($j = 1, 2, 3, 4$ for Sydney, Brisbane, Melbourne and Hobart, respectively). Then constraints are equations, such as $x_{11} + x_{12} + x_{13} + x_{14} = 250$ (amount that can be shipped from Sydney to the group of publishers). Similarly for deliveries from Melbourne and Adelaide. Also, $x_{11} + x_{21} + x_{31} = 75$ (amount that is delivered to the Sydney publisher from the group of three warehouses). Similarly for deliveries to Brisbane, Melbourne and Hobart.

We can write these constraints concisely in the form

$$\begin{aligned} \sum_{i=1}^{n} x_{ij} &= \mathbf{w}, \\ \sum_{j=1}^{m} x_{ij} &= \mathbf{p}, \end{aligned}$$

where **w** and **p** are the demand and supply coefficients, and n and m are the numbers of warehouses and publishers, respectively. The objective function is

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{i,j} x_{i,j}.$$

where c_{ij} is the vector of costs of shipping from i to j.

Solve this integer linear program using problem based optimisation. You are provided with the table of data above as `trasportation_example.mat`.

To set up the problem and variables use the following commands:

```
load('transportation_data.mat')
transportation_example = optimproblem;
x = optimvar('x',3,4,'LowerBound',0,'Type','integer')
```

Then set up the objective and constraints. These can be defined in 3 (or maybe 4) lines of code, using the `sum` function in MATLAB.

Hints:

- Look up MATLAB's documentation on `sum` to see how to do column/row summation.
- Since these are equality constraints they should be written with == instead of <=.
- To access the table data in matrix form use `transportation_table.Variables`

Once you have entered your problem use the `show` command to see that everything has been entered as expected.

Finally, use the `solve` command to solve the optimisation problem.

Look at the output to view the total cost and shipping plan. Write out the shipping plan in words to make sure you understand how to interpret this solution.