7 Optimization Problems

7.1 Solving Linear Programming Problems using `linprog`

We can use the function `linprog` to solve minimization problems in linear programming in MATLAB. One of the possible syntaxes is

```
linprog(f, A, b, Aeq, beq, lb, ub)
```

We illustrate its usage in the following examples. More info can be found using the command `help`.

Example 7.1.1 *Suppose we would like to minimize the function*

$$f(x_1, x_2) = -135x_1 - 75x_2$$

subjecting to the constraints

$$\begin{aligned} x_1 + 3x_2 &\leq 18, \\ x_1 + x_2 &\leq 10, \\ 3x_1 + x_2 &\leq 24, \end{aligned}$$

where

$$x_1 \geq 0 \text{ and } x_2 \geq 0.$$

It is customary to denote these constraints in matrix notation as follows:

$$\begin{aligned} Ax &\leq b, \\ lb &\leq x, \end{aligned}$$

where

$$A = \begin{pmatrix} 1 & 3 \\ 1 & 1 \\ 3 & 1 \end{pmatrix}, \; b = \begin{pmatrix} 18 \\ 10 \\ 24 \end{pmatrix}, \text{ and } lb = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

are the coefficient matrix, the right-hand side, and the vector of lower bounds, while

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

We can solve this minimization problem in MATLAB by first setting up these matrices:

```
>> f = [-135; -75];
>> A = [1 3; 1 1; 3 1];
>> b = [18; 10; 24];
>> lb = [0; 0];
```

The solution of the minimization problem is then obtained by typing

```
>> [x, fval] = linprog(f, A, b, [], [], lb, [])
```

which returns

```
Optimization terminated.

x =

    7.0000
    3.0000


fval =

 -1.1700e+003
```

The variable `x` *contains the values of* x_1 *and* x_2 *which minimize* f*, while the variable* `fval` *contains the corresponding minimum value of* f*.*

Example 7.1.2 *Suppose we would like to maximize the function*

$$g(x_1, x_2) = 5x_1 + 4x_2$$

subjecting to the constraints

$$\begin{aligned} 6x_1 + 4x_2 &\leq 24, \\ x_1 + 2x_2 &\leq 6, \\ -x_1 + x_2 &\leq 1, \end{aligned}$$

where

$$x_1 \geq 0 \text{ and } 0 \leq x_2 \leq 2.$$

Note that the function `linprog` *"only" solves minimization problems. Hence to maximize* $g(x_1, g_2)$*, we minimize* $f(x_1, x_2) = -g(x_1, x_2) = -5x_1 - 4x_2$ *instead. To do this in MATLAB, we set up the corresponding matrices:*

```
>> f = [-5; -4];
>> A = [6 4; 1 2; -1 1];
>> b= [24; 6; 1];
>> lb = [0; 0];
>> ub = [Inf; 2];
```

The vector `ub` *is the vector of upper bounds. Then the mininum of* f *is obtained by typing*

```
>> [x, fval] = linprog(f, A, b, [], [], lb, ub)
```

which returns

```
Optimization terminated.
```

```
x =

    3.0000
    1.5000


fval =

  -21.0000
```

The minimum value of $f(x_1, x_2)$ *is therefore* $f(3, 1.5) = -21$, *while the maximum value of* $g(x_1, x_2)$ *is* $g(3, 1.5) = -f(3, 1.5) = 21$.

7.2 Solving Binary Integer Programming Problems using `bintprog`

We can use the function `bintprog` to solve minimization problems in binary integer programming in MATLAB. One of the possible syntaxes is

```
bintprog(f, A, b, Aeq, beq)
```

The usage of `bintprog` is similar to that of `linprog`. More info can be found using the command `help`.

Example 7.2.1 *Suppose we would like to maximize*

$$g(x_1, x_2, x_3, x_4) = 0.2x_1 + 0.3x_2 + 0.5x_3 + 0.1x_4$$

subjecting to the constraints

$$\begin{aligned} 0.5x_1 + 1.0x_2 + 1.5x_3 + 0.1x_4 &\leq 3.1, \\ 0.3x_1 + 0.8x_2 + 1.5x_3 + 0.4x_4 &\leq 2.5, \\ 0.2x_1 + 0.2x_2 + 0.3x_3 + 0.1x_4 &\leq 0.4 \end{aligned}$$

where

$$x_1, x_2, x_3, x_4 \in \{0, 1\}.$$

As `linprog`, `bintprog` *"only" solves minimization problems. Hence instead of maximizing* $g(x_1, x_2, x_3, x_4)$ *directly, we minimize* $f(x_1, x_2, x_3) = -g(x_1, x_2, x_3) = -0.2x_1 - 0.3x_2 - 0.5x_3 - 0.1x_4$. *To do this in MATLAB, we type*

```
>> f = [-0.2; -0.3; -0.5; -0.1];
>> A = [0.5 1.0 1.5 0.1; 0.3 0.8 1.5 0.4; 0.2 0.2 0.3 0.1];
>> b = [3.1; 2.5; 0.4];
```

Then

```
>> [x fval] = bintprog(f, A, b)
```

yields

```
Optimization terminated.

x =

     0
     0
     1
     1


fval =

   -0.6000
```

indicating that the desired maximum value of $g(x_1, x_2, x_3, x_4)$ *is* $g(0, 0, 1, 1) = -f(0, 0, 1, 1) = 0.6$.

Example 7.2.2 *More interesting is to see the effects on the maxima of the objective function*

$$g(x_1, x_2, x_3, x_4) = 0.2x_1 + 0.3x_2 + 0.5x_3 + 0.1x_4$$

as we vary the constraints. For instance, suppose we have now the contraints

$$\begin{aligned} 0.5x_1 + 1.0x_2 + 1.5x_3 + 0.1x_4 &\leq c, \\ 0.3x_1 + 0.8x_2 + 1.5x_3 + 0.4x_4 &\leq 2.5, \\ 0.2x_1 + 0.2x_2 + 0.3x_3 + 0.1x_4 &\leq 0.4 \end{aligned}$$

where

$$x_1, x_2, x_3, x_4 \in \{0, 1\},$$

while c *is to vary between* 0 *and* 5 *inclusively. It is then natural to ask how the maxima of* $g(x_1, x_2, x_3, x_4)$ *change with respect to the variation in* c. *This we do by the following script m-file:*

```
%  Script file: eg_bintprog.m
%
%  Purpose:
%    To consider the effects of varying the constraints in
%    maximizing the objective function
%
%    g(x_1,x_2,x_3,x_4) = 0.2x_1+0.3x_2+0.5x_3+0.1x_4
%
%    subjecting to the constraints
%
%    0.5x_1+1.0x_2+1.5x_3+0.1x_4 <= c,
%    0.3x_1+0.8x_2+1.5x_3+0.4x_4 <= 2.5,
%    0.2x_1+0.2x_2+0.3x_3+0.1x_4 <= 0.4
%
%    where each x_1, x_2, x_3, and x_4 is either 0 or 1, while
%    c is to vary in the range 0 to 5 inclusively with a stepsize
%    of 0.1.
%
```

```
%  Record of revisions:
%    Date          Programmer          Description of change
%    ====          ==========          =====================
%    Sep 23, 13    K. H. Kwa           Original code
%
% Define variables:
%   f          -- (Negative) of the objective function, i.e., -g
%   fval       -- The minium values of f
%   gval       -- The maximum values of g
%   x          -- Points minimizing f
%   A          -- The coefficient matrix in the inequalities
%   b          -- RHS in the inequalities
%   crange     -- The range of c, i.e., 0:0.1:5

% Initialize variables.
gval = [];
f = -[0.2; 0.3; 0.5; 0.1];
A = [0.5 1.0 1.5 0.1; 0.3 0.8 1.5 0.4; 0.2 0.2 0.3 0.1];
b = [0; 2.5; 0.4];
crange = 0:0.1:5;

% Optimize g as c varies.
for c = crange
    b(1) = c;
    [x fval] = bintprog(f, A, b);
    gval = [gval -fval];
end

% Plot c-values versus maxima of g.
plot(crange, gval);
xlabel('c');
ylabel('Maxima of g');
title('Variation of Maximum of g')
```

The variation in the maxima of g relative to c-values is shown in Figure 7.2.1.

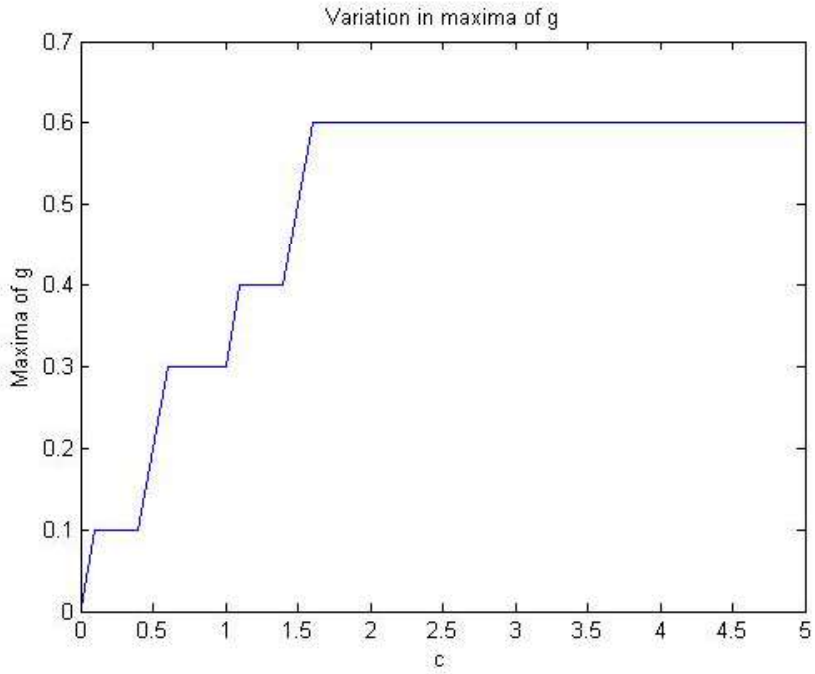

Figure 7.2.1: An instance of the effects of variation in constraints on the maxima of the objective function in a binary integer programming problem.

7.3 Finding Minimum/Maximum on a Fixed Interval using `fminbnd`

We can use the function `fminbnd` to find the minimum of a single-variable function on fixed interval, one of the possible syntaxes of which is

```
[x, fmin] = fminbnd(fun, x1, x2)
```

where `fun` is the handle to the function, `x1` and `x2` are endpoints of the interval, and, finally, `x` and `fmin` contain the corresponding critical point at which the minimum occurs and the minimum, respectively.

Example 7.3.1 *We can calculate the maximum value of* $g(x) = -5x^2 - 2x + 3$ *on the interval* $[-1, 1]$ *by finding the minimum of* $f(x) = -g(x) = 5x^2 + 2x - 3$ *on the same interval. To do this, we type*

```
>> f = @(x) 5*x.^2+2*x-3;
>> [x, fmin] = fminbnd(f, -1, 1)
```

This yields

```
x =

   -0.2000


fmin =

   -3.2000
```

Thus the maximum of g on the interval $[-1,1]$ is $g(-0.2) = -(-3.2) = 3.2$ and the corresponding critical value is -0.2.

Example 7.3.2 *Alternatively, we can do the last example by using the command* `max`*. To do this, type*

```
>> x = linspace(-1,1,100);
>> y = -5*x.^2-2*x+3;
>> max(y)
```

This yields an approximation to the maximum of $g(x) = -5x^2 - 2x + 3$ on the interval $[-1,1]$:

```
ans =

    3.1997
```

This can be bettered by increasing the number of points at which $g(x)$ is evaluated, i.e.,

```
>> x = linspace(-1,1,500);
>> y = -5*x.^2-2*x+3;
>> max(y)
```

which yields

```
ans =

    3.2000
```

The critical point can be approximated with the aid of logical indexing:

```
>> x(y==max(y))

ans =

   -0.1984
```

This can also be improved by increasing the data points:

```
>> x = linspace(-1,1,5000);
>> y = -5*x.^2-2*x+3;
>> x(y==max(y))

ans =

   -0.1998
```

8 Finding Zeros

8.1 Zeros of Nonlinear Functions

We can find a zero of a nonlinear function in MATLAB using the function `fzero` whenever the function changes signs at this zero. One of the syntaxes is

```
x = fzero(fun, x0)
```

where `fun` is the handle of the function, `x0` is an initial guess of the zero or the endpoints of an interval containing the zero, and finally `x` will contain the zero.

Warning: `fzero` cannot find a zero of a function such as x^2.

Example 8.1.1 *We can calculate the value of* $\frac{\pi}{2}$ *as a zero of* $f(x) = \cos(x)$ *near* $x_0 = 1$. *To do this, we type*

```
>> f = @(x) cos(x);
>> x = fzero(f, 1)
```

and MATLAB returns

```
x =

    1.5708
```

More simply, we can type

```
>> fzero(@cos, 1)
```

to obtain

```
ans =

    1.5708
```

Example 8.1.2 *We can calculate the zero of a function within an interval at the endpoints of which the function differs in signs. For instance, we can calculate* $\frac{3\pi}{2}$ *as a zero of* $f(x) = \cos(x)$ *within the interval* $[2, 5]$. *(Note that* $\cos(2) \approx -0.4161 < 0$ *and* $\cos(5) \approx 0.2837 > 0$.*) To do this, we type*

```
>> fzero(@cos, [2 5])
```

to get

```
ans =

    4.7124
```

Example 8.1.3 *more interestingly, we can consider the effects on the location of a zero as a parameter is varied. For instance, consider the function*

$$f(t, c) = \cos^3(t) - ct,$$

where c is a parameter taking values in $[0, 10]$. Then the zeros of $f(t, c)$ near $t_0 = 1$ as c varies can be found using the following script m-file, the result of which is shown in Figure 8.1.1:

```
%  Script file: eg_fzero.m
%
%  Purpose:
%    To consider the effects of varying a parameter c on
%    the location of the zero of
%
%    f(t) = cos(t).^3 - c * t
%
%    near t0 = 1 for c between 0 and 10 inclusively with
%    a stepsize of 0.01.
%
%  Record of revisions:
%    Date          Programmer          Description of change
%    ====          ==========          =====================
%    Sep 23, 13    K. H. Kwa           Original code
%
% Define variables:
%   fun       -- The parametrized nonlinear function
%   f         -- The nonlinear function with a fixed parameter
%   zf        -- The zeros of f
%   crange    -- The range of c, i.e., 0:0.1:5

% Initialize variables.
fun = @(t, c) cos(t).^3 - c * t;
zf = [];
crange = 0:.01:10;

% Locate the zero.
for c = crange
    f = @(t) fun(t, c);
    zf = [zf fzero(f, 1)];
end

% Plot the c-values versus the zeros.
plot(crange, zf);
xlabel('c');
ylabel('Zeros');
title('Variation in the Location of Zero');
```

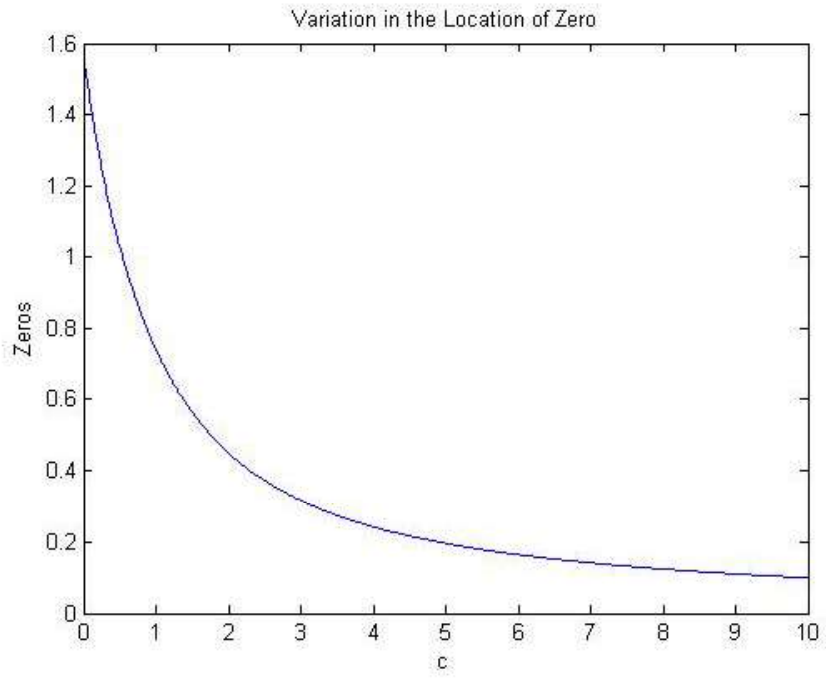

Figure 8.1.1: An instance of the effects of variation in a parameter on the location of a zero of $f(t, c) = \cos^3(t) - ct$.

9 File Input-Output

MATLAB has a family of built-in commands to input and/or output data. Some of these commands are listed below. Use `help <command>` to obtain more info. Alternatively, visit

`http://www.mathworks.com/help/matlab/data-import-and-export.html`.

9.1 Delimited or Formatted I/O to Text Files

Commands	Description
`csvread('<file name>')`	Reads data into MATLAB from the named file, one row per line of input; the numbers in each line must be comma-separated. Note that the number of elements in each row need not be equal. The undefined numbers are set to `0`.
`csvwrite('<file name>', A)`	Writes out the elements of the matrix `A` to the named file using the same format as `csvread`.
`load('<file name>')`	Reads data into MATLAB from the named, file one row per line of input; the numbers can either be space-separated or comma--separated. Note that the number of elements in each row needs to be equal. The name of the resulting matrix is `<file name>`.
`importdata('<file name>')`	Similar to `load`. However, the number of elements in each row need not be equal. The undefined elements are set to `NaN`.

Table 9.1.1: Some common commands to input data from and/or output data to delimited or formmated text files. Here `<file name>` is a string of characters.

The commands `csvread`, `csvwrite`, `load`, and `importdata` read and/or write files in CSV[8] format which can also be imported into and exported from spreadsheet programs such as Microsoft EXCEL, LibreOffice Calc, OpenOffice.org Calc, and Gnumeric.

[8]CSV stands for comma-separated values or character-separated values format because the separator character does not have to be a comma. A CSV file stores tabular data (numbers and text) in plain-text form. Plain text means that the file is a sequence of characters, with no data that has to be interpreted instead, as binary numbers.

9.2 Spreadsheets

More directly, the following commands allow MATLAB to handle spreadsheets generated by spreadsheet programs.

Commands	**Description**
`xlsfinfo('<file name>')`	Determine if file contains Microsoft Excel spreadsheet. It returns the string `'Microsoft Excel Spreadsheet'` if the specified file is in a format that `xlsread` can read. Otherwise, it returns the empty string, `''`.
`xlsread('<file name>')`	Read Microsoft Excel spreadsheet file. It can also read spreadsheet file generated by other spreadsheet programs.
`xlswrite('<file name>', A)`	Write the matrix `A` to an Excel spreadsheet file as named.

Table 9.2.1: Some common commands to handle spreadsheets generated by spreadsheet programs directly. Here `<file name>` is a string of characters.

9.3 Low-Level File I/O

Commands	Description
`fid = fopen('<file name>', <permission string>)`	Opens the named file with the permission string determining how the filed is to be accessed. Here `fid` is a unique nonnegative integer attached to the file. If errors occur, `fid` is set to -1. Some permission strings are: `'r'`: read only from the file. `'w'`: write only to the file. Previous contents are overwritten. If necessary, the file is created. `'a'`: append to the end of the file. Previous contents are retained. `'r+'`: Read from and write to the file. But do not create the file. `'w+'`: Read from and write to the file. If necessary, create the file. `'a+'`: Read from and write to the file by appending new data to the end of the file.
`fclose(fid)`	Closes the file with the handle `fid`.
`fprintf(fid, <format string>, <variable 1>, ...)`	Writes data to the file with handle `fid` using desired format. Behaves very similarly to the corresponding C command.
`fscanf(fid, <format string>).`	Reads data to the file with handle `fid` using desired format. Behaves very similarly to the corresponding C command.

Table 9.3.1: Some common commands to for low-level file I/O. Here `<file name>` is a string of characters.