Appendix B

SOLVING SHORT-TERM PLANNING PROBLEMS

In this compendium we focus on formulating short-term planning problems, but in practise it is of course also necessary to actually solve them. In a real short-term planning problem there will be hundreds or thousands of optimisation variables and then it is of course impossible to solve be problem by hand calculations. The solution is to use computers and today there is a variety of software to solve both linear programming problems and other kinds of optimisation problems.

The challenge when using computer software to solve optimisation problems is to ascertain that the intended problem has been solved—a small programming error may easily result in an incorrect solution or an infeasible problem. It is therefore important to be careful both when formulating the optimisation problem and then entering it into the computer. In this appendix we will provide an example of how to set about.

In example 5.6 we formulated a short-term planning problem for two hydro power plants. Below we will show how this problem can be solved using different programming languages. However, let us start with a compilation of the original problem formulation:

Indices of power plants

Degerforsen 1, Edensforsen 2.

Parameters

The following parameters are given:

$$\overline{H}_i = \text{installed capacity in power plant } i = \begin{cases} 62 & i = 1, \\ 63 & i = 2, \end{cases}$$

$$\overline{Q}_i = \text{maximal discharge in power plant } i = \begin{cases} 300 & i = 1, \\ 270 & i = 2, \end{cases}$$

$$\overline{M}_i = \text{maximal contents of reservoir } i = \begin{cases} 5\,000\,000/3\,600 & i = 1, \\ 4\,000\,000/3\,600 & i = 2, \end{cases}$$

$$\lambda_f = \text{expected future electricity price} = 185,$$

$$w_i = \text{mean annual flow} = \begin{cases} 163 & i = 1, \\ 164 & i = 2, \end{cases}$$

$$D_t = \text{contracted load for hour } t = \begin{cases} 90 & t = 1, \\ 98 & t = 2, \\ 104 & t = 3, \\ 112 & t = 4, \\ 100 & t = 5, \\ 80 & t = 6. \end{cases}$$

The following parameters are calculated based on the given parameters:

$M_{i,0}$ = start contents of reservoir $i = 0.5\overline{M}_i$, $i = 1, 2$,

$$V_i = \text{local inflow to reservoir } i = \begin{cases} w_1 & i = 1, \\ w_2 - w_1 & i = 2, \end{cases}$$

$\overline{Q}_{i,1}$ = maximal discharge in power plant i, segment $1 = 0.75\overline{Q}_i$, $i = 1, 2$,

$\overline{Q}_{i,2}$ = maximal discharge in power plant i, segment $2 = \overline{Q}_i - \overline{Q}_{i,1}$, $i = 1, 2$,

$\mu_{i,1}$ = marginal production equivalent of power plant i, segment 1 =

$$= \frac{\overline{H}_i}{\overline{Q}_{i,1} + 0.95\overline{Q}_{i,2}}, \; i = 1, 2,$$

$\mu_{i,2}$ = marginal production equivalent of power plant i, segment $2 = 0.95\mu_{i,1}$.

Optimisation variables

$Q_{i,j,t}$ = discharge in power plant i, segment j, during hour t, $i = 1, 2, j = 1, 2, t = 1, \ldots, 6$,

$S_{i,t}$ = spillage from reservoir i during hour t, $i = 1, 2, t = 1, \ldots, 6$,

$M_{i,t}$ = contents of reservoir i at the end of hour t, $i = 1, 2, t = 1, \ldots, 6$.

Objective function

maximise $\quad \lambda_f((\mu_{1,1} + \mu_{2,1})M_{1,6} + \mu_{2,1}M_{2,6})$.

Constraints

$$M_{1,t} - M_{1,t-1} + Q_{1,1,t} + Q_{1,2,t} + S_{1,t} = V_1, \qquad t = 1, \ldots 6,$$

$$M_{2,t} - M_{2,t-1} + Q_{2,1,t} + Q_{2,2,t} + S_{2,t} - Q_{1,1,t} - Q_{1,2,t} - S_{1,t} = V_2, \qquad t = 1, \ldots 6,$$

$$\sum_{i=1}^{2}\sum_{j=1}^{2} \mu_{i,j} Q_{i,j,t} = D_t, \qquad t = 1, \ldots 6.$$

Variable limits

$$0 \le Q_{i,j,t} \le \overline{Q}_{i,j}, \qquad i = 1, 2, j = 1, 2, t = 1, \ldots, 6,$$

$$0 \le S_{i,t}, \qquad i = 1, 2, t = 1, \ldots, 6,$$

$$0 \le M_{i,t} \le \overline{M}_i, \qquad i = 1, 2, t = 1, \ldots, 6.$$

B.1 GAMS

GAMS is a programming language especially dedicated to formulation of optimisation problems. The idea is that problems and inputs are formulated in a standardised way—regardless of whether the problem at hand is linear or non-linear—and then the problem is passed on to special solvers. A GAMS program is written in an ordinary text file (it is most straightforward to use the built-in editor). When executing the program (choose Run in the menu File) GAMS will process the file and check that the problem syntax is correct. If that is the case then the problem is passed to a suitable solver. The solution (or an error message if no solution was found) is then returned to GAMS, where it is possible to choose how to present the results. Finally, the result is written to a text file having the same name as the program file, but with the suffix .lst. This listing file can be viewed in the built-in editor. Any errors encountered during the run results in error messages in the listing file.

We start by defining the indices to be used in the program. GAMS requires that all possible index values are states, in order to allow the program to check that other statements only use allowed values. The syntax for defining index values is

```
set symbol "explanatory text" /element1, element2/;
```

The keyword `set` (we may also use the plural form `sets`) states that an index is going to be defined, and is followed by the symbol used to denote the index. Then an explanatory text follows, which GAMS will echo when listing the results. The explanatory text may not exceed 80 characters. If no slashes, commas or semi-colons are used in the text, there is no need for the quotation marks. Finally, the possible index values are listed within the slashes. It is allowed to define more than one index in the same `set` statement; the most straightforward is then to define each index in a row of its own.

Indices of power plants

Degerforsen 1, Edensforsen 2.

```
Sets
  i power plant
    /Degerforsen, Edensforsen/
  j segment /segment1*segment2/
  t hour /hour1*hour6/
;
```

Notice that we may define a sequence of numbered index values using `*`; the allowed values of the index `t` are thus `"hour1"`, `"hour2"`, `"hour3"`, `"hour4"`, `"hour5"` and `"hour6"`.

The next step is to declare the parameters of the problem. The syntax for declaring parameters is similar to the one for index values:

Parameters

The following parameters are given:

$\overline{H}_i$ = installed capacity in power plant i =

$$= \begin{cases} 62 & i = 1, \\ 63 & i = 2, \end{cases}$$

$\overline{Q}_i$ = maximal discharge in power plant i =

$$= \begin{cases} 300 & i = 1, \\ 270 & i = 2, \end{cases}$$

```
Parameters
Hmax(i)    installed capacity in
           power plant i
           /Degerforsen 62,
            Edensforsen 63/
Qtotmax(i)maximal discharge in
           power plant i
           /Degerforsen 300,
            Edensforsen 270/
```

$\overline{M}_i$ = maximal contents of reservoir i =

$$= \begin{cases} 5\,000\,000/3\,600 & i = 1, \\ 4\,000\,000/3\,600 & i = 2, \end{cases}$$

λ_f = expected future electricity price =

$= 185,$

w_i = mean annual flow =

$$= \begin{cases} 163 & i = 1, \\ 164 & i = 2, \end{cases}$$

D_t = contracted load for hour t =

$$= \begin{cases} 90 & t = 1, \\ 98 & t = 2, \\ 104 & t = 3, \\ 112 & t = 4, \\ 100 & t = 5, \\ 80 & t = 6. \end{cases}$$

```
Mmax(i)    maximal contents of
           reservoir i
           /Degerforsen 5e6,
            Edensforsen 4e6/
lambdaf    expected future
           electricity price /185/
w(i)       mean annual flow at
           power plant i
           /Degerforsen 163,
            Edensforsen 164/
D(t)       contracted load hour t
           /hour1 90, hour2 98,
            hour3 104, hour4 112,
            hour5 100, hour6 80/
Mstart(i)  start contents of
           reservoir i
V(i)       local inflow to
           reservoir i
Qmax(i,j)  "maximal discharge in
           power plant i,
           segment j"
mu(i,j)    "marginal production
           equivalent for power
           plant i, segment j"
;
```

When a parameter has more than one index, a dot is used to separate different index values. The maximal discharge in each power plant and segment could thus be stated in the following manner:[1]

```
Parameters
Qmax(i,j) "maximal discharge in power plant i, segment j"
          /Degerforsen.segment1 225,
           Edensforsen.segment1 202.5,
           Degerforsen.segment2 75,
           Edensforsen.segment2 67.5/
;
```

However, in order to avoid rounding errors it might be better to let GAMS perform the calculations. In that case, it is sufficient to declare that the parameter exists and then assign values later:

The following parameters are calculated based on the given parameters:

$M_{i,0}$ = start contents of reservoir $i = 0.5\overline{M}_i$, $i = 1, 2,$

V_i = local inflow to reservoir i =

$$= \begin{cases} w_1 & i = 1, \\ w_2 - w_1 & i = 2, \end{cases}$$

```
Mmax(i) = Mmax(i)/3600;
Mstart(i) = 0.5*Mmax(i);
V(i) = w(i) - w(i-1);
Qmax(i,"segment1") =
  0.75*Qtotmax(i);
Qmax(i,"segment2") = Qtotmax(i) -
  Qmax(i,"segment1");
mu(i,"segment1") =
  Hmax(i)/(Qmax(i,"segment1") +
  0.95*Qmax(i,"segment2"));
mu(i,"segment2") =
  0.95*mu(i,"segment1");
```

1. An alternative approach would be to use the key word `Table`. See *GAMS Users Guide* (which can be accessed from the help menu in the GAMS window) for further details.

$\overline{Q}_{i,1}$ = maximal discharge in power plant i,
segment 1 = $0.75\overline{Q}_i$, $i = 1, 2$,

$\overline{Q}_{i,2}$ = maximal discharge in power plant i,
segment 2 = $\overline{Q}_i - \overline{Q}_{i,1}$, $i = 1, 2$,

$\mu_{i,1}$ = marginal production equivalent of
power plant i, segment 1 =

$$= \frac{\overline{H}_i}{\overline{Q}_{i,1} + 0.95\overline{Q}_{i,2}},\ i = 1, 2,$$

$\mu_{i,2}$ = marginal production equivalent of
power plant i, segment 2 = $0.95\mu_{i,1}$.

Notice that GAMS automatically performs assignments over all possible values of the indices involved in the expression. Thus, the first row in the code above will be interpreted by GAMS as

```
Mmax("Degerforsen") = Mmax("Degerforsen")/3600;
```

and

```
Mmax("Edensforsen") = Mmax("Edensforsen")/3600;
```

Another feature of GAMS is that expressions which contain non-existing index values will automatically be removed from the calculations. When the index `i` is equal to `"Edensforsen"` the row

```
V(i) = w(i) - w(i-1);
```

is read as

```
V("Edensforsen") = w("Edensforsen") - w("Degerforsen");
```

i.e., `i - 1` is automatically replaced by the previous index value. But when `i` is equal to `"Degerforsen"` there is no previous index value (as `"Degerforsen"` is the first power plant in the definition of the set); hence this part is removed and then the row reads

```
V("Degerforsen") = w("Degerforsen");
```

We must also declare which optimisation variables we have in the problem. Notice that we already now can state some variable limits by using the keywords `positive`, `negative`, `binary`, `integer` and `free`. GAMS requires that there is at least one free variable in each problem, as the variable to be maximised or minimised must be able to assume any value. The declarations are similar to the index values and parameters, i.e., a symbol is stated followed by an explanatory text (however, no values are stated):

Optimisation variables

$Q_{i,j,t}$ = discharge in power plant i,
segment j, during hour t, $i = 1, 2$,
$j = 1, 2$, $t = 1, \ldots, 6$,

$S_{i,t}$ = spillage from reservoir i
during hour t, $i = 1, 2$, $t = 1, \ldots, 6$,

$M_{i,t}$ = contents of reservoir i at the end
of hour t, $i = 1, 2$, $t = 1, \ldots, 6$.

```
Positive variables
  Q(i,j,t)   "discharge in power
              plant i, segment j,
              during hour t"
  S(i,t)     spillage from
              reservoir i
              during hour t
  M(i,t)     contents of reservoir
              i at the end of
              hour t
```

Table B.1 GAMS equation types.

Type of equations	Mathematical symbol	Sense
=e=	=	Left hand side must equal right hand side
=l=	≤	Left hand side must be less than or equal to the right hand side
=g=	≥	Left hand side must be greater than or equal to right hand side

```
;

Free variable
  z         value of stored water
;
```

The relation between the optimisation variables are described by equations in GAMS. An equation must first be declared; this is done in a similar manner as for variables. Then, the actual equation is defined according to the following syntax:

```
symbol .. LHS type RHS;
```

Here, `symbol` is the name the equation was given when declared. The name of the equation is separated from the equation definition by two dots. Both the left hand side (`LHS`) and right hand side (`RHS`) are mathematical expressions. The relation between the left and right hand sides are depending on the `type` of the equation (see table B.1). There is no difference in syntax between the objective function and the constraints; the objective function is simply the equation stating the value of the free variable to be minimised or maximised:

Objective function

maximise $\lambda_f((\mu_{1,1} + \mu_{2,1})M_{1,6} + \mu_{2,1}M_{2,6})$.

```
Equation
  objfnc  objective function;

objfnc..  z =e= lambdaf*
            ((mu("Degerforsen",
                 "segment1") +
              mu("Edensforsen",
                 "segment1"))
            *M("Degerforsen",
               "hour6")
          + mu("Edensforsen",
               "segment1")
            *M("Edensforsen",
               "hour6"));
```

The constraints are also stated as equations. Once again we can utilise that GAMS removes expressions if the index value does not exist. However, in the hydrological constraints there is a complication, which cannot be managed by GAMS itself. In the first hour there is no optimisation variable $M_{i,t-1}$, but there is a parameter $M_{i,0}$, which in the program has been given the symbol `Mstart(i)`. This parameter should only be included in those hydrological constraints for which `t` is equal to `"hour1"`. This is implemented by using a so-called dollar condition, which means that after an expression we may type a `$`-sign, followed by a condition. If this condition is not fulfilled the expression will be removed, in the same way as non-existing index values are removed. The function `ord` can be used to obtain the position of an index value in a set. As `"hour1"` is the first value in the set `t`, the condition `ord(t) = 1` will only be fulfilled for the hydrological balances during the first hour.

Table B.2 GAMS default variable limits.

Variable type	Lower limit (.LO)	Upper limit (.UP)	Other limitations
free	$-\infty$	$+\infty$	
positive	0	$+\infty$	
negative	$-\infty$	0	
binary	0	1	Integer variable
integer	0	100	Integer variable

Constraints

$$M_{1,t} - M_{1,t-1} + Q_{1,1,t} + Q_{1,2,t} + S_{1,t} = V_1, \qquad t = 1, \ldots 6,$$

$$M_{2,t} - M_{2,t-1} + Q_{2,1,t} + Q_{2,2,t} + S_{2,t} - Q_{1,1,t} - Q_{1,2,t} - S_{1,t} = V_2, \qquad t = 1, \ldots 6,$$

$$\sum_{i=1}^{2}\sum_{j=1}^{2} \mu_{i,j} Q_{i,j,t} = D_t, \qquad t = 1, \ldots 6.$$

```
Equations
  hydbal(i,t) "hydrological
              balance in power
              plant i, hour t"
  loadbal(t)  load balance of
              hour t
;

hydbal(i,t).. M(i,t) =e= M(i,t-1)
  + Mstart(i)$(ord(t)
                          = 1)
  - sum(j,Q(i,j,t))
  - S(i,t)
  + sum(j,Q(i-1,j,t))
  + S(i-1,t)
  + V(i);
loadbal(t)..  sum((i,j),mu(i,j)*
                    Q(i,j,t))
              =e= D(t);
```

All optimisation variables in GAMS are given variable limits according to their type (see table B.2). It is simple to change the default values in those cases when a variable has another limit. The upper limit of a variable is accessed by the symbol of the variable followed by the suffix `.UP` and similarly, the lower limit has the suffix `.LO`. In this problem all variables have the lower limit zero and therefore do not need to be changed. The spillage has no upper limit; hence, the spillage uses the default upper value too. The only thing that needs to be done is to set the upper limits of discharge and reservoir contents:

Variable limits

$$0 \le Q_{i,j,t} \le \overline{Q}_{i,j}, \quad i = 1, 2, j = 1, 2, t = 1, \ldots, 6,$$

$$0 \le S_{i,t}, \qquad i = 1, 2, t = 1, \ldots, 6,$$

$$0 \le M_{i,t} \le \overline{M}_i, \qquad i = 1, 2, t = 1, \ldots, 6.$$

```
loop(t,Q.up(i,j,t) = Qmax(i,j);
       M.up(i,t) = Mmax(i));
```

We have already seen that assignments automatically are performed for all possible values of an index. However, in this case we have more indices in the parameter that is to be assigned than in the parameter from which the values are taken. Therefore, we use a `loop`-statement to perform the same assignment for every hour. Thus, as a result of the loop, GAMS is performing the assignments

```
Q.UP(i,j,"hour1") = Qmax(i,j);
```

```
M.UP(i,"hour1") = Mmax(i);
Q.UP(i,j,"hour2") = Qmax(i,j);
M.UP(i,"hour2") = Mmax(i);
```

etc.

We can now solve the problem. First we give a name to the model, while stating which equations are included in the model (in this case it is all equations, which is stated by `/all/`). Then we ask GAMS to solve the optimisation problem. First, we must specify which type of solver that is to be used. In this case we have an LP problem; therefore, we state that the problem should be solved "`using lp`". It we would like to solve a MILP problem, we would have stated "`using mip`", and if we would like to solve a non-linear problem, we would state "`using nlp`". GAMS will automatically select the solver which is most appropriate for problems of the specified type. Finally, we must state which variable is to be minimised or maximised. In this case it is `z` that shall be maximised. Remember that the variable to be maximised or minimised must be declared as a free variable.

```
model hydroplanning /all/;

solve hydroplanning using lp
  maximizing z;
```

Finally we instruct GAMS to write the most important results at the end of the listing file. At the same time, we can also calculate the total discharge and electricity generation in each power plant and hour (cf. the discharge plan in table 5.5). To calculate the total discharge and electricity generation we need the optimal values of the discharge $V_{i,j,t}$. When GAMS is solving the optimisation problem the optimal solution is stored in parameters having the same name as the optimisation variable followed by the suffix `.L`. Hence, the optimal discharge is accessed through `Q.L`. We can also obtain the dual variables of both constraints and variable limits. The dual variables are stored in parameters which have the same name as the constraint or the optimisation variable followed by the suffix `.M`.

```
Parameters
  Qtot(i,t) "total discharge in
            power plant i,
            hour t"
  H(i,t)    "generation in power
            plant i, hour t"
;

loop((i,t),Qtot(i,t) =
            sum(j,Q.L(i,j,t));
           H(i,t) = sum(j,
            mu(i,j)*Q.L(i,j,t)));

display M.L, Q.L, Qtot, H, S.L;
display hydbal.M, loadbal.M;
```

B.2 MATLAB

Matlab is a general programming language for technical computing. There is a large range of ready-made functions for many areas of computing. With the exception of the most basic mathematical functions—which are included in the software—all functions are written in ordinary text files (it is most straightforward to use the built-in editor).

Functions to solve optimisation problems are found in the so-called Matlab Optimization Toolbox, but there are also freeware functions to be found on the Internet. Below it is demonstrated how our problem can be solved using `linprog` (which is included in Optimization Toolbox). As most Matlab functions for solving LP problems, `linprog` requires that the problem is formulated using matrices. In this case the problem should be on the form

$$\text{minimise} \quad f^T x \tag{B.1}$$

$$\text{subject to} \quad Ax \le b, \tag{B.1a}$$

$$A_{eq} x = b_{eq}, \tag{B.1b}$$

$$l_b \le x \le u_b. \tag{B.1c}$$

The function `linprog` can be called with different inputs and outputs, depending on how we wish to use the function.[2] Below follows a description of what is needed to solve the example in this appendix.

Assume that a problem has N_{ineq} inequality constraints, N_{eq} equality constraints and M variables. In that case `linprog` uses the following inputs:

- `f` coefficients of the objective function (column vector with M elements),
- `A` coefficients of the inequality constraints (matrix with N_{ineq} rows and M columns),
- `b` right hand side of the inequality constraints (column vector with N_{ineq} elements),
- `Aeq` coefficients of the equality constraints (matrix with N_{eq} rows and M columns),
- `beq` right hand side of the equality constraints (column vector with N_{eq} elements),
- `lb` lower bound of the optimisation variables (column vector with M elements),
- `ub` upper bound of the optimisation variables (column vector with M elements).

The result of a call is

- `x` optimal solution (column vector with M elements).

The difficulty when solving the planning problem is to correctly define the inputs necessary to call `linprog`. To make sure that no mistakes are made in this process, it is appropriate to write a Matlab program which follows the original problem formulation as closely as possible. Hence, we start by clarifying the index values we use for the power plants:

Indices of power plants

Degerforsen 1, Edensforsen 2.

```
% Indices of the power plants:
% Degerforsen 1, Edensforsen 2
```

In Matlab everything on a row after a `%`-sign is considered a comment. Hence, no calculations are performed by the two rows above—they are just intended to make the program easier to understand.

The next step is to define all parameter values:

2. Please refer to the built-in help function of Matlab for a more detailed description.

Parameters

The following parameters are given:

$\overline{H}_i$ = installed capacity in power plant i =

$$= \begin{cases} 62 & i = 1, \\ 63 & i = 2, \end{cases}$$

$\overline{Q}_i$ = maximal discharge in power plant i =

$$= \begin{cases} 300 & i = 1, \\ 270 & i = 2, \end{cases}$$

$\overline{M}_i$ = maximal contents of reservoir i =

$$= \begin{cases} 5\,000\,000/3\,600 & i = 1, \\ 4\,000\,000/3\,600 & i = 2, \end{cases}$$

λ_f = expected future electricity price =

$= 185,$

w_i = mean annual flow =

$$= \begin{cases} 163 & i = 1, \\ 164 & i = 2, \end{cases}$$

D_t = contracted load for hour t =

$$= \begin{cases} 90 & t = 1, \\ 98 & t = 2, \\ 104 & t = 3, \\ 112 & t = 4, \\ 100 & t = 5, \\ 80 & t = 6. \end{cases}$$

```
Given parameters
npowerplants = 2;
nhours = 6;
Hmax = [62; 63]; % Hmax(i)
Qtotmax = [300; 270];
  % Qtotmax(i)
Mmax = [5e6; 4e6]/3600; % Mmax(i)
lambdaf = 185;
w = [163; 164]; % w(i)
D = [90 98 104 112 100 80];
    % D(t)
```

The following parameters are calculated based on the given parameters:

$M_{i,0}$ = start contents of reservoir $i = 0.5\overline{M}_i$, $i = 1, 2,$

V_i = local inflow to reservoir i =

$$= \begin{cases} w_1 & i = 1, \\ w_2 - w_1 & i = 2, \end{cases}$$

$\overline{Q}_{i,1}$ = maximal discharge in power plant i, segment $1 = 0.75\overline{Q}_i$, $i = 1, 2,$

```
% Calculated parameters
Mstart = 0.5*Mmax; % Mstart(i)
V = [w(1); w(2)-w(1)]; % V(i)
% Qmax(i,j)
Qmax(:,2) = Qtotmax - Qmax(:,1);
Qmax(:,1) = .75*Qtotmax;
% mu(i,j)
mu(:,1) = Hmax./(Qmax(:,1) +
  0.95*Qmax(:,2));
mu(:,2) = 0.95*mu(:,1);
```

$\overline{Q}_{i,2}$ = maximal discharge in power plant i, segment 2 = $\overline{Q}_i - \overline{Q}_{i,1}$, $i = 1, 2$,

$\mu_{i,1}$ = marginal production equivalent of power plant i, segment 1 = $= \dfrac{\overline{H}_i}{\overline{Q}_{i,1} + 0.95\overline{Q}_{i,2}}$, $i = 1, 2$,

$\mu_{i,2}$ = marginal production equivalent of power plant i, segment 2 = $0.95\mu_{i,1}$.

Notice that we try to choose the same parameter name in Matlab as when we originally formulated the problem. It is also recommended to insert a comment explaining how the Matlab matrices are indexed. For example, we have above chosen to put all marginal production equivalents in a matrix `mu`, where the element on row `i`, column `j`, correspond to the marginal production equivalent of power plant i, segment j, i.e., $\mu_{i,j}$.

When an optimisation problem is formulated in matrix form, all optimisation variables are collected in a column vector x, i.e., each element in x should correspond to a unique optimisation variable in the original problem formulation, as in the following example:

$$x = [Q_{1,1,1}\, Q_{1,2,1}\, S_{1,1}\, M_{1,1}\, Q_{2,1,1}\, Q_{2,2,1}\, S_{2,1}\, M_{2,1} \ldots M_{3,6}]^T.$$

It is arbitrary in which order the variables area sorted into the vector x, but it is of course necessary to be consequent throughout the Matlab code. It is simplifying to use a "dictionary", where it is easy to identify which element in x corresponds to a certain variable. To implement this, we use the following code:

Optimisation variables

$Q_{i,j,t}$ = discharge in power plant i, segment j, during hour t, $i = 1, 2$, $j = 1, 2$, $t = 1, \ldots, 6$,

$S_{i,t}$ = spillage from reservoir i during hour t, $i = 1, 2$, $t = 1, \ldots, 6$,

$M_{i,t}$ = contents of reservoir i at the end of hour t, $i = 1, 2$, $t = 1, \ldots, 6$.

```
% Variables
pos = 0;
for t = 1:nhours
  for i = 1:npowerplants
    pos = pos + 1;
    Qpos(i,1,t) = pos;
    pos = pos + 1;
    Qpos(i,2,t) = pos;
    pos = pos + 1;
    Spos(i,t) = pos;
    pos = pos + 1;
    Mpos(i,t) = pos;
  end
end
nvariables = pos;
```

In the code above the variable `pos` is just a counter, which tracks the last used position in i x. When the loops of the program has been performed, each variable in the problem will have been assigned a position and these positions have been stored in the three matrices `Vpos`, `Spos` and `Mpos`. These matrices are indexed in the same manner as the corresponding optimisation variable. Hence, if we want to know where in the vector x we will find $M_{2,3,4}$we should check the value of `Mpos(2,3,4)`.

Now we can define the objective function. The objective function in (B.1) is read

$$f^T x = f_1 x_1 + f_2 x_2 + \ldots + f_M x_M. \tag{B.2}$$

When calling `linprog` we just need to state the coefficients, *f*. If we look at the problem formulation we find that only two optimisation variables are included in the objective function, which means that most of the coefficients in *f* are equal to zero. To shorten the Matlab code we can start by a vector of the right size, which just hold zeros (such a vector is generated by the function `zeros`). Then we change just those coefficients that are non-zero. For example, the coefficient before $M_{2,6}$ is equal to $\mu_{2,1}$ and if $M_{2,6}$ is the *n*:th element of the vector *x* then the *n*:th element of *f* should equal $\mu_{2,1}$. To know the position of $M_{2,6}$ we use the earlier defined "dictionary" `Mpos`:

Objective function

maximise $\lambda_f((\mu_{1,1} + \mu_{2,1})M_{1,6} + \mu_{2,1}M_{2,6})$.

```
% Objective function:
f = zeros(nvariables,1);
f(Mpos(1,nhours)) = ...
  lambdaf*(mu(1,1) + mu(2,1));
f(Mpos(2,nhours)) = ...
  lambdaf*mu(2,1);
```

The constraints are built in a similar manner as the objective function. Each constraint corresponds to a certain row in the matrices *A* and A_{eq} respectively, and a certain row in the column vectors *b* and b_{eq} respectively. We use a counter `cnstr` to keep track of which constraint we currently define; each time we start stating the coefficients of a new constraint we first increase the value of `cnstr` by one.

Notice that in the hydrological constraints we are forced to use an `if`-clause to manage different cases. The first hour (i.e., then `t = 1`) the reservoir contents at the end of the previous hour is a known parameter, which should be added to the right hand side of the constraint, whereas $M_{i,t-1}$ is an optimisation variable for the other hours. Moreover, in the hydrological balance of Degerforsen (which we defined as power plant 1) there is no incoming water from discharge and spillage in the power plant upstream; hence, these coefficients should only be included if `i > 1`.

Constraints

$$M_{1,t} - M_{1,t-1} + Q_{1,1,t} + Q_{1,2,t} + S_{1,t} = V_1, \qquad t = 1, \ldots 6,$$

$$M_{2,t} - M_{2,t-1} + Q_{2,1,t} + Q_{2,2,t} + S_{2,t} - Q_{1,1,t} - Q_{1,2,t} - S_{1,t} = V_2, \qquad t = 1, \ldots 6,$$

$$\sum_{i=1}^{2}\sum_{j=1}^{2} \mu_{i,j} Q_{i,j,t} = D_t, \qquad t = 1, \ldots 6.$$

```
% Constraints
nconstraints = 3*nhours;
Aeq = zeros(nconstraints,...
            nvariables);
beq = zeros(nconstraints,1);
cnstr = 0;
for t = 1:nhours
  for i = 1:npowerplants
    contrnr = cnstr + 1;
    Aeq(cnstr,Mpos(i,t)) = 1;
    if k == 1
      beq(cnstr) = beq(cnstr) ...
                   + Mstart(i);
    else
      Aeq(cnstr,Mpos(i,t-1)) ...
        = -1;
    end
    Aeq(cnstr,Qpos(i,1,t)) = 1;
    Aeq(cnstr,Qpos(i,2,t)) = 1;
    Aeq(cnstr,Spos(i,t)) = 1;
    if i > 1
      Aeq(cnstr,Qpos(i-1,1,t))...
        = -1;
```

```
      Aeq(cnstr,Qpos(i-1,2,t))...
        = -1;
      Aeq(cnstr,Spos(i-1,t))...
        = -1;
    end
    beq(cnstr) = beq(cnstr) + ...
                 V(i);
  end
  cnstr = cnstr + 1;
  for i = 1:npowerplants
    for j = 1:2
      Aeq(cnstr,Qpos(i,j,t)) =
        ...
        mu(i,j);
    end
  end
  beq(cnstr) = D(t);
end
```

Finally we have to define the upper and lower limits on all optimisation variables. The lower limits are all zeros; therefore, the vector `lb` is easily defined directly by the function `zeros`. The upper limits are different for different variables. The most common limit is $+\infty$ and we may therefore start from a vector of the correct size, where all elements are equal to $+\infty$ (such a vector can be generated by the function `inf`) and then we change the upper limits that are not infinite:

Variable limits

$$0 \le Q_{i,j,t} \le \overline{Q}_{i,j}, \quad i = 1, 2, j = 1, 2, t = 1, \ldots, 6,$$

$$0 \le S_{i,t}, \qquad i = 1, 2, t = 1, \ldots, 6,$$

$$0 \le M_{i,t} \le \overline{M}_i, \qquad i = 1, 2, t = 1, \ldots, 6.$$

```
% Variable limits:
lb = zeros(nvariables,1);
ub = inf(nvariables,1);
for t = 1:nhours
  for i = 1:npowerplants
    for j = 1:2
      ub(Qpos(i,j,t)) = ...
         Vmax(i,j);
    end
    ub(Mpos(i,t)) = Mmax(i);
  end
end
```

Now that all inputs have been defined (we have no inequality constraints, so there is no need to define `A` and `b`; they are replaced by empty matrices in the call to `linprog`) and are ready to solve the optimisation problem. As `linprog` solves a minimisation problem and we have formulated a maximisation problem we must remember to negate the objective function! The function `linprog` can in addition to the optimal solution also return additional information about the problem. For example is the value of the dual variables of both constraints and variable limits calculated. The dual variables are the fifth value returned by `linprog`. Hence, we are only interested of the first and fifth value, and the values in between are stored in the waste parameters `waste1`, `waste2` and `waste3`:

```
% Solve the optimisation problem:
[x,waste1,waste2,waste3,y] =
  linprog(-f,[],[],...
          Aeq,beq,lb,ub);
```

The function returns the optimal solution stored in the vector `x`. To get a better overview of the optimal solution we can extract the individual optimisation variables and put them in separate

matrices for each kind of variable (discharge, spillage and reservoir contents). At the same time, we can also calculate the total discharge and electricity generation in each power plant and hour (cf. the discharge plan in table 5.5). The dual variables are returned in a Matlab structure. A dot followed by the field name in question is used to access a field in the structure., The structure storing the dual variables has the fields `.ineqlin` (dual variables of the inequality constraints), `.eqlin` (dual variables of the equality constraints), `.upper` (dual variables of the upper variable limits) and `.lower` (dual variables of the lower variable limits).

```
% Sort the results:
Qres = [];
Sres = [];
Mres = [];
for t = 1:nhours
  for i = 1:npowerplants
    for j = 1:2
      Qres(i,j,t) = ...
            x(Vpos(i,j,t));
    end
    Sres(i,t) = x(Spos(i,t));
    Mres(i,t) = x(Mpos(i,t));
    Qtot(i,t) = sum(Qres(i,:,t));
    H(i,t) = ...
      mu(i,1)*Qres(i,1,t) + ...
      mu(i,2)*Qres(i,2,t);
  end
end
optvalue = round(f'*x)
Mres, Qres, Qtot, H, Sres
dualvariables = y.eqlin
```