

University of Cincinnati
Department of Electrical & Computer Engineering and Computer Science

20 ENFD 112 – Fundamentals of Programming

LABORATORY 10: DECISION ANALYSIS - NON-LINEAR SYSTEMS

Spring 2008

1. Objective

The objectives of this assignment are to formulate a common optimization problem as a non-linear system of equations and to use MATLAB to solve the system.

2. Problem

A one-day career fair is being organized for Wyoming High School. The objective is to bring speakers in from various organizations to speak about the nuances of particular careers such as Mechanical Engineering, Politics, Acting, and so on. A significant portion of the high school population has indicated interest in attending the fair. Each interested student has been asked to list p preferred speakers (p is a parameter - the actual number of preferences may vary from year to year). The goal is to create schedules for the speakers and students so that the total number of missed preferences is minimized. However, there are some realistic restrictions we have to worry about. These are:

1. There are t consecutive speaking periods (t is a parameter - the actual number of preferences may vary from year to year).
2. Every speech lasts one period.
3. There are parallel sessions. In other words, for each period there are several rooms in which speakers may speak.
4. Each speaker speaks on exactly one topic.
5. All speakers are available the entire day for speaking.
6. No two speakers speak in the same room at the same time.
7. Every speaker gives at most one speech in a period.
8. Each speaker has a set number of speeches he or she will deliver during the day. This number is fixed for each speaker and all numbers are known before scheduling.
9. Every student attends at most one speech in a period.
10. Every student must be assigned to a room in each of the t periods. Observe there may not be a speaker assigned to a room holding at least one student. In that case, attending students just sit out that period.
11. All rooms have the same capacity R (another parameter).

2.1 Part I

Schedule the speakers so as to minimize the maximum number of rooms used in a period. Assume all data is input via file with the following format (the names of items follows those given in Section 3):

```
 $n$                 % The number of speakers
 $S_1 S_2 S_3 \dots S_n$     % The number of speeches required of each speaker
 $t$                 % The number of speaking periods
```

Speakers will be identified by the order in which their required number of speeches is given on the 2nd line of the file. Comments are shown only to help clarify the meaning of lines. It will not be necessary to worry about file comments - just assume there are none. The following is a sample input file showing 6 speakers for 5 periods with number of speeches set at 2,3,4,3,2,3 for speakers 1-6, respectively.

```
6
2 3 4 3 2 3
5
```

Sample output should look like this for 4 rooms:

```
Printing speaker schedule
  3   5   1   0
  3   1   4   2
  3   4   2   6
  2   3   0   6
  5   4   0   6
```

where columns represent rooms and rows represent periods. A 0 entry at row i and column j means there is no speaker in room j during period i . According to the table, speaker 1 speaks in rooms 3 and 2 for the first two periods (total of 2 speeches), speaker 2 speaks in rooms 4,3,1 in periods 2-3 (total of 3 speeches), speaker 3 speaks in room 1 for the first 3 periods and in room 2 for the 4th period (total of 4 speeches), speaker 4 speaks in room 3 for the 2nd period and in room 2 for periods 3 and 5 (total of 3 speeches), speaker 5 speaks in rooms 2 and 1 for periods 1 and 5 (total of 2 speeches), and speaker 6 speaks in room 4 for the last three periods (total of 3 speeches). All restrictions are satisfied - this is a reasonable solution. No solution uses fewer rooms because the number of 0's is less than the number of rows.

Complete this part before continuing to part II.

2.2 Part II

Now, find the minimum room capacity which supports a schedule using the minimum number of rooms in which all student preferences are satisfied. Expect input from a file with the following format:

```

n                               % The number of speakers
S1 S2 S3 ... Sn         % The number of speeches required of each speaker
t                               % The number of speaking periods
m p                             % The number of students, the number of preferences
P1,1 P1,2 ... P1,p       % The remaining lines give student preferences, one
P2,1 P2,2 ... P2,p       % line per student, where Pi,j is the number of the
...                             % speaker that is the jth preference of student i.
Pm,1 Pm,2 ... Pm,p       % End of file at mth student

```

Speakers will be identified as in Part I. Students will be identified by the order in which their preference lines appear after the 4th line of the file. As in Part I, just assume there are no comments in the file.

The following is a sample input file showing 6 speakers for 5 periods with number of speeches set at 2,3,4,3,2,3 for speakers 1-6, respectively, and 15 students with 3 preferences each. For example, student 1 prefers to listen to speakers 1, 2 and 4 and student 15 prefers to listen to speakers 2, 3, and 5.

```

6
2 3 4 3 2 3
5
15 3
1 2 4
2 3 5
3 5 6
1 3 4
2 5 6
3 5 6
2 4 5
1 4 5
1 2 5
1 3 5
1 4 5
2 5 6
1 3 5
2 5 6
2 3 5

```

Sample output appears below where the user has selected 4 rooms of capacity 7 to hold the conference (a schedule with fewer rooms is not possible). Two schedules are shown: one for speakers, which looks like and has a similar meaning to the schedule of Part I, and one for students. In both schedules columns represent rooms and rows represent periods. In the student schedule each room/period entry contains a list of students. Thus, students 5,9,12,14 are scheduled for the lecture given by speaker 2 in room 1 during period 5. Observe that some speakers have no students during some periods.

Printing speaker schedule

5	2	0	3
5	2	6	3
1	0	4	6
1	0	4	3
2	6	4	3

Printing student schedule

[3,5,6,8,12,14,15]	[1,2,7,9,11]	[]	[4,10,13]
[2,7,8,9,10,11,13]	[1,5,12,14,15]	[]	[3,4,6]
[1,4,8,9,10,11,13]	[]	[2,7,15]	[3,5,6,12,14]
[1,4,8,9,10,11,13]	[]	[5,7,12,14]	[2,3,6,15]
[5,9,12,14]	[]	[1,4,7,8,11]	[2,3,6,10,13,15]

3. Analysis

Consider part I first. Let's define some important variables. Suppose there are n speakers. The identities of the speakers will simply be given by the set of numbers $\{1, 2, 3, \dots, n\}$. Let S_i denote the number of periods speaker i is supposed to speak in during the day. We will use t to represent the number of periods encompassing the career fair during the day and r to denote the maximum number of rooms that will be used in any one period. For convenience the rooms will be numbered $\{1, 2, 3, \dots, r\}$ and periods will be numbered $\{1, 2, 3, \dots, t\}$. This way we can reference them very easily in any program we write.

Now let's write mathematical expressions representing the restrictions. Consider the restriction (6.) that there must be *at most one speaker for each room in a single period*. Suppose, each time we schedule a speaker to a room and period, we hand that speaker a red marker stating room and period. Then, to see if the restriction is satisfied, we just need to count the red markers for each room and period. If the number is 2 or greater for one room/period combination, the restriction is not satisfied. Otherwise it is.

How do we hand out red markers in a computer program? By inventing a variable that takes the value 1 if a particular speaker i is scheduled for a particular room in a particular period and 0 otherwise (corresponding to no marker being handed out to speaker i for that room and period). Here is a description of the set of variables we create:

$$S_{i,j,k} = \begin{cases} 1 & \text{if speaker } i \text{ is scheduled for room } j \text{ in period } k \\ 0 & \text{otherwise} \end{cases}$$

Observe there are $n \cdot r \cdot t$ of these variables.

So, how do the new variables get used? Well, first we have to establish mathematical constraints (equations or inequalities) that force each of these variables to have value 0 or 1. This can be done in several different ways. For example, for all i, j, k , create the following equations:

$$S_{i,j,k} \cdot (1 - S_{i,j,k}) = 0. \tag{1}$$

Each of these equations is satisfied if and only if $S_{i,j,k}$ is 0 or 1. No other values for $S_{i,j,k}$ satisfy it.

Now we can satisfy restriction 6. with the following set of inequalities:

$$S_{1,j,k} + S_{2,j,k} + S_{3,j,k} + \dots + S_{n,j,k} \leq 1. \tag{2}$$

There are $r \cdot t$ such constraints in total. The only way any one of these constraints is satisfied is if at most one of $S_{1,j,k} \dots S_{n,j,k}$ is 1 which means at most one speaker is in room j during period k .

Equations (1) and inequalities (2) are called a quadratic system. This system has quite a number of unknowns, namely $S_{i,j,k}$ for each speaker, room, period combination. It is easy to see this system is satisfied (all equations and inequalities are satisfied) if $S_{i,j,k} = 0$ for all i, j, k which means do not schedule any speakers! We will add more constraints to the system. When all the constraints corresponding to all relevant restrictions are added in, we will get a meaningful solution.

Let's concentrate now on restriction 7. (*every speaker gives at most one speech in a period*). Similar to inequalities (2), we add constraints:

$$S_{i,1,k} + S_{i,2,k} + S_{i,3,k} + \dots + S_{i,r,k} \leq 1 \quad (3)$$

which force at most one of $S_{i,1,k} \dots S_{i,r,k}$ to 1. There are $n \cdot t$ of these constraints. If speaker i is assigned to two rooms, say 3 and 6, in the same period k , then $S_{i,3,k} = 1$ and $S_{i,6,k} = 1$ and (3) is violated because the sum that includes these two terms must be no greater than 1. Therefore, a solution to the quadratic system (1), (2), (3) satisfies restrictions 6. and 7.

Now consider restriction 8. (*a set number of speeches for each speaker*). This restriction can be satisfied by adding the following constraints to the system:

$$S_{i,1,1} + \dots + S_{i,1,t} + S_{i,2,1} + \dots + S_{i,2,t} + \dots + S_{i,r,1} + \dots + S_{i,r,t} = S_i \quad (4)$$

(alternatively, $\sum_{j=1}^r \sum_{k=1}^t S_{i,j,k} = S_i$) which force any speaker i to speak in exactly S_i periods since, for each i, j, k such that $S_{i,j,k} = 1$, speaker i speaks in some room (j in this case) at some time (period k in this case).

At this point we have all the constraints needed to solve part I. The quadratic system (1)-(4) can be solved directly in MATLAB using `quadprog`. The only question is how to represent the system. This is treated in the **Coding** section below.

To solve part II we need to define more variables and add constraints involving them. Let the number of students be m . Let R denote the capacity of each room. Identify students by number $\{1, 2, 3, \dots, m\}$ as we did for the speakers. The following variables can be created for students, just like they were created for speakers:

$$P_{i,j,k} = \begin{cases} 1 & \text{if student } i \text{ is scheduled for room } j \text{ in period } k \\ 0 & \text{otherwise} \end{cases}$$

Restriction 9. may be represented in a manner similar to inequality (3). That is,

$$P_{i,1,k} + P_{i,2,k} + P_{i,3,k} + \dots + P_{i,r,k} \leq 1. \quad (5)$$

Restriction 10 requires each student to attend a speech in every period. This can be represented in a manner similar to inequality (4) except that t can be used instead of S_i . That is,

$$P_{i,1,1} + \dots + P_{i,1,t} + P_{i,2,1} + \dots + P_{i,2,t} + \dots + P_{i,r,1} + \dots + P_{i,r,t} = t. \quad (6)$$

Restriction 11 requires all rooms to have the same capacity R . Thus, $\sum_{x=1}^m P_{x,j,k} \leq R$ to insure that room j is not overbooked in period k (we do not count the speaker).

Finally, consider satisfying all student preferences. Represent preferences as a set of student/speaker pairs

$$P = \{\langle 1, P_{1,1} \rangle, \dots, \langle 1, P_{1,4} \rangle, \langle 2, P_{2,1} \rangle, \dots, \langle 2, P_{2,4} \rangle, \dots, \langle m, P_{m,1} \rangle, \dots, \langle m, P_{m,4} \rangle\}$$

where $P_{i,j}$ are the preference numbers taken from the input file whose format is given in Section 2.2. We want to count the number of preferences satisfied. Since the product $P_{x,j,k} \cdot S_{y,j,k}$ is 1 if

and only if student x listens to speaker y in room j during period k , $\sum_{j=1}^r \sum_{k=1}^t P_{x,j,k} \cdot S_{y,j,k}$ is at least 1 if and only if student x listens to speaker y in room j during period k . Therefore, we require

$$\sum_{\langle x,y \rangle \in P} \left(\min \left\{ \sum_{j=1}^r \sum_{k=1}^t P_{x,j,k} \cdot S_{y,j,k}, 1 \right\} \right) = m \cdot p \quad (7)$$

4. Coding

This section is divided into the following parts: reading from file, choosing an optimization tool for solving Part I, dealing with linear constraints, dealing with non-linear constraints, choosing an optimization tool for solving Part II.

4.1 Reading from file

File reading is briefly mentioned since this has been done in previous labs. The important points are: use an `input` line to get a file name from a user. Get a file identifier using `fopen`. Read single numbers using `fscanf(fid, '%d', 1);`. Read a vector of, say, 4 numbers using `fscanf(fid, '%d', 4);`.

4.2 Using arrays to represent constraints

Arrays are a convenient tool for constraint representation and that is what MATLAB expects to see. By convention, array columns represent variables and array rows represent constants of constraints. A system of linear constraints can be represented as a simple inequality involving an array product. For example, the inequality of Figure 1 represents the set of constraints

$$x_1 + x_2 \leq 1 \quad (8)$$

$$x_2 + x_3 \leq 1 \quad (9)$$

$$x_3 + x_4 \leq 1 \quad (10)$$

$$x_1 + x_4 \leq 1 \quad (11)$$

$$x_2 + x_3 \leq 1 \quad (12)$$

A system of equations, instead of inequalities, can be represented similarly, as shown in Figure 2 and as follows:

$$x_1 + x_3 = 1 \quad (13)$$

$$x_2 + x_4 = 1 \quad (14)$$

$$x_2 + x_3 = 2 \quad (15)$$

Quadratic constraints can be represented similarly through an additional array multiplication. For example, the equation of Figure 3 represents the set of constraints

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Figure 1: Linear constraints as an inequality involving array multiplication. The array with two ones per row is referred to as **A** in the code below. The one dimensional array of all 1s is referred to as **b**. The actual constraints represented are shown in the text as inequalities (8)-(12).

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

Figure 2: Linear constraints as an equation involving array multiplication. The array with two ones per row is referred to as **Aeq** in the code below. The one dimensional array of all 1s is referred to as **beq**. The actual constraints represented are shown in the text as equations (13)-(15).

$$(x_1 \ x_2 \ x_3 \ x_4) \cdot \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + (1 \ 1 \ 1 \ 1) \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Figure 3: Quadratic constraints as an equation of array multiplications and addition. The square array with -1 on the diagonal is referred to as **H** in the code below. The (horizontal) array of 1s is referred to as **f** in the code. The actual constraints represented are shown in the text as equations (16)-(19).

$$-x_1^2 + x_1 = 0 \tag{16}$$

$$-x_2^2 + x_2 = 0 \tag{17}$$

$$-x_3^2 + x_3 = 0 \tag{18}$$

$$-x_4^2 + x_3 = 0 \tag{19}$$

all of which have the same form as equations (1).

MATLAB has builtin functions which take as input the constant array data of constraints and return values to variables which satisfy the constraints. For example, the system of inequalities and equations shown in Figures 1-3 can be solved using the MATLAB code shown in Figure 4. The solution returned is the transpose of $[0 \ 1 \ 1 \ 0]$. Observe, the **x** values are either 0 or 1 *only* because the constraints of Figures 1-2 are satisfied: the function **quadprog** does not actually solve the equation of Figure 2, instead it finds values for **x** that *minimize* $\mathbf{x}' * \mathbf{H} * \mathbf{x} + \mathbf{f}' * \mathbf{x}$, and the minimum occurs at 0 if the constraints of Figures 1-2 are satisfied. Hence, in finding the minimum, **quadprog** winds up solving the equation of Figure 3.

```

A = [1 1 0 0 ; 0 1 1 0 ; 0 0 1 1 ; 1 0 0 1 ; 0 1 1 0]; % Figure 1
b = [1 ; 1 ; 1 ; 1 ; 1]; % Figure 1
Aeq = [1 0 1 0 ; 0 1 0 1 ; 0 1 1 0]; % Figure 2
beq = [1 ; 1 ; 2]; % Figure 2
H = [-1 0 0 0 ; 0 -1 0 0 ; 0 0 -1 0 ; 0 0 0 -1]; % Figure 3
f = [1 1 1 1]; % Figure 3
lb = [0 ; 0 ; 0 ; 0]; % lower bounds
ub = [1 ; 1 ; 1 ; 1]; % upper bounds
x = quadprog(H,f,A,b,Aeq,beq,lb,ub); % solver

```

Figure 4: MATLAB code for solving the system shown in Figures 1-3.

4.3 Translation of variable indices to array indices

We have represented the variables of the problems with three subscripts: i, j, k for speaker, room, and period. But we said MATLAB likes to see the variables in array columns. So we need a way to translate i, j, k to a column index for both the speakers and the students. Let's start with an example. Suppose there are 2 speakers, 3 students, 4 periods, and 2 rooms. One natural translation is shown in the following table (**Idx** means column index and **Var** means variable to associate with that column):

Idx	Var	Idx	Var	Idx	Var	Idx	Var
1	$S_{1,1,1}$	11	$S_{2,1,3}$	21	$P_{1,2,1}$	31	$P_{2,2,3}$
2	$S_{1,1,2}$	12	$S_{2,1,4}$	22	$P_{1,2,2}$	32	$P_{2,2,4}$
3	$S_{1,1,3}$	13	$S_{2,2,1}$	23	$P_{1,2,3}$	33	$P_{3,1,1}$
4	$S_{1,1,4}$	14	$S_{2,2,2}$	24	$P_{1,2,4}$	34	$P_{3,1,2}$
5	$S_{1,2,1}$	15	$S_{2,2,3}$	25	$P_{2,1,1}$	35	$P_{3,1,3}$
6	$S_{1,2,2}$	16	$S_{2,2,4}$	26	$P_{2,1,2}$	36	$P_{3,1,4}$
7	$S_{1,2,3}$	17	$P_{1,1,1}$	27	$P_{2,1,3}$	37	$P_{3,2,1}$
8	$S_{1,2,4}$	18	$P_{1,1,2}$	28	$P_{2,1,4}$	38	$P_{3,2,2}$
9	$S_{2,1,1}$	19	$P_{1,1,3}$	29	$P_{2,2,1}$	39	$P_{3,2,3}$
10	$S_{2,1,2}$	20	$P_{1,1,4}$	30	$P_{2,2,2}$	40	$P_{3,2,4}$

You can verify that if the number of speakers is `nspks`, the number of students is `nstud`, the number of rooms is `nrms` and the number of periods is `nprds`, then the column indices are determined as follows:

$$\begin{aligned}
S_{i,j,k} &\rightarrow k + (j-1)*nprds + (i-1)*nprds*nrms \\
P_{i,j,k} &\rightarrow k + (j-1)*nprds + (i-1)*nprds*nrms + nspks*nprds*nrms
\end{aligned}$$

The reverse translation (from column index to i, j, k) for speakers and students is

$S_{i,j,k}$	i	$\text{floor}((\text{idx}-1)/(\text{nprds}*\text{nrms}))+1$
	j	$\text{floor}(\text{mod}((\text{idx}-1), \text{nprds}*\text{nrms})/\text{nprds}))+1$
	k	$\text{mod}((\text{idx}-1), \text{nprds}))+1$
$P_{i,j,k}$	i	$\text{floor}((\text{idx}-\text{nspks}*\text{nrms}*\text{nprds}-1)/(\text{nprds}*\text{nrms}))+1$
	j	$\text{floor}(\text{mod}((\text{idx}-\text{nspks}*\text{nrms}*\text{nprds}-1), \text{nprds}*\text{nrms})/\text{nprds}))+1$
	k	$\text{mod}((\text{idx}-\text{nspks}*\text{nrms}*\text{nprds}-1), \text{nprds}))+1$

4.4 Memoizing parameters

Life would be made a lot simpler if we could build a function, say `S`, to do the translation from i, j, k to column index and use it, for example, like this: `A(row,S(i,j,k))=1;` to place a 1 in the correct row and column according to the speaker, room, and period. But if `S` is to be a function, then some mechanism must be used to tell it the values of `nspks`, `nstud`, `nrms`, and `nprds`. Normally, these would have to be passed as arguments to the function. But we do not want to do this because it is ugly. So we make use of the following lines in the calling function:

```
setappdata(0,'nprds',nprds);
setappdata(0,'nrms',nrms);
setappdata(0,'nspks',nspks);
setappdata(0,'nstud',nstud);
```

which are invoked *after* values for the named variables have been set. These lines copy the values to a global workspace that is visible to all functions. To get the values from the global workspace the called function uses the following lines:

```
nstud = getappdata(0,'nstud');
nprds = getappdata(0,'nprds');
nspks = getappdata(0,'nspks');
nrms = getappdata(0,'nrms');
```

4.5. Example coding of a restriction

Consider restriction 6. (at most one speaker can be assigned to a room in a period) which is implemented as the following set of `nrms*nprds` inequalities:

$$S_{1,j,k} + S_{2,j,k} + S_{3,j,k} + \dots + S_{nspks,j,k} \leq 1$$

Assume the inequality constraint array is called `A` and that `A` has been initialized to all 0's using `zeros`. Each constraint is a row of `A` and constraints will be added to `A` one at a time in increasing order of row. Use a variable `con_num` to hold the number of the current row whose constraint is being built. Then the following code handles the implementation of restriction 6.:

```
con_num = 1;
for j=1:nrms
    for k=1:nprds
        for i=1:nspks; A(con_num, S(i,j,k)) = 1; end
        con_num = con_num+1;
    end
end
```

Code for another set of constraints will follow this *but without resetting con_num to 1!*

4.6. Coding of a translation function

The following shows code that implements S as used in implementing the above constraints.

```
function index = S(i,j,k);
    nprds = getappdata(0,'nprds');
    nrms = getappdata(0,'nrms');
    index = k + (j-1)*nprds + (i-1)*nprds*nrms;
end
```

Code for implementing a corresponding P (as in $P_{i,j,k}$) is similar. You will probably need to implement the inverse translation of S and P . It should be apparent how to do that from the function above and the tables in Section 4.3.

4.7. Miscellaneous

1. You may get an error from `quadprog` saying you ran out of some resource and you need to change some `options` variable to get more. Then do something similar to the following (example for `MaxIter` which has a very low default of perhaps 100 and is now going to be set to 100000).

```
options = optimset('MaxIter',100000);
x = zeros(1,n);
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x,options);
```

To use the `options` field, we are forced to provide a starting value for `x`. This is the reason for the line `x = zeros(1,n);`.

2. There are a number of other optimization functions that you might consider using instead of or in addition to `quadprog`. Some notable examples are `bintprog` which solves a linear system in which variable values must be either 0 or 1 (therefore eliminating the need for $\mathbf{x}'\mathbf{H}\mathbf{x} + \mathbf{f}'\mathbf{x}$); and `fmincon` which allows implementation of a nonlinear objective function, nonlinear inequality constraints, nonlinear equality constraints, linear inequality constraints and linear equality constraints. To find more, take a tour of the `optimization toolbox`.

5. Submission

Submit `m` files solving the two problems of Section 2 on or before June 8 using blackboard. See the course webpage at

<http://gauss.ececs.uc.edu/Courses/HTML/E112.html>

for instructions.