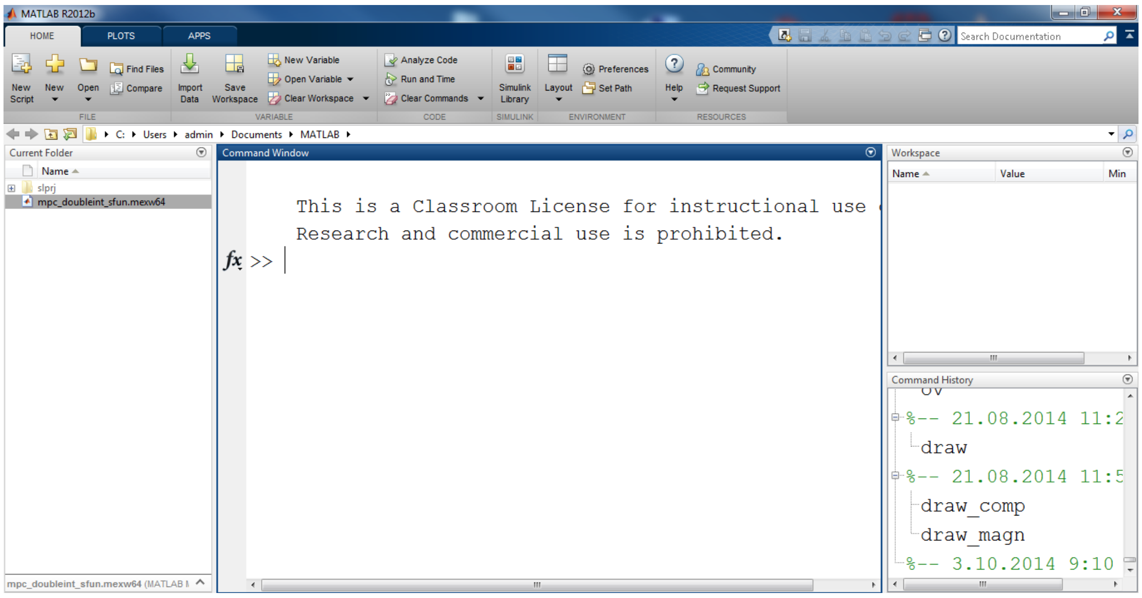
ISS0030 Modeling and Optimization

Practice in MATLAB

General overview



```
This is a Classroom License for instructional use
Research and commercial use is prohibited.
fx >>
```

```
%-- 21.08.2014 11:2
draw
%-- 21.08.2014 11:5
draw_comp
draw_magn
%-- 3.10.2014 9:10
```

Remark 1. *If any part of the main window is missing, it is possible to find and recover from the toolbar menu* ***Layout****: Command Window, Command History, Current Folder, Workspace, etc.*

Remark 2. *If you forget the name of a team, you can use either* ***Product Help*** *in the toolbar menu or type* ***help*** `command name` *in command window.*

Remark 3. *If you want to change style (font, font size, etc.), it is achievable via Preferences.*

Basic operations in MATLAB

```
>> a = 1 % some comment
a =
    1
>> b = -2; % the semicolon symbol (;)
>> b^2 % ^ - power symbol; ENG: Shift + 6; EST: AltGr + \"{A}
ans =
    4
>> ab = a + b % sum of a and b
ab =
    -1
>> 1 + 2*sqrt(ab) % square root
ans =
    1.0000 + 2.0000i
```

```
>> log(0) % Inf = infinity
ans =
    -Inf
>> var1 = 3.1415e+3 % 3.1415e+3 is the same as 3.1415*10^3
var1 =
    3.1415e+003
clear all % removes all variables from your base workspace
```

Vectors and matrices

```
>> a = [1 2 3 4 3 2 1] % row vector
a =
    1 2 3 4 3 2 1
>> b = a + 3 % element wise addition
b =
    4 5 6 7 6 5 4
>> A = [9 2 3; -1 3 4; 0 2 1] % symbol ; separates rows of the matrix
A =
     9    2    3
    -1    3    4
     0    2    1
>> At = A' % symbol ' denotes the transpose of the matrix
At =
     9   -1    0
     2    3    2
     3    4    1
>> B = A*At % symbol (*) denotes multiplication of matrices
B =
    94    9    7
     9   26   10
     7   10    5
>> A % displays the content of the variable
A =
     9    2    3
    -1    3    4
     0    2    1
>> A(1,1) + A(2,3) % A(i,j) - ith row and jth column
ans =
    13
>> A(3,3) = 4 % replacement
A =
     9    2    3
    -1    3    4
     0    2    4
>> row1 = 2:5 % create array with step equals to 1
row1 =
    2 3 4 5
>> row2 = 6:-0.5:3.5 % array with step equals to 0.5
row2 =
    6.0000 5.5000 5.0000 4.5000 4.0000 3.5000
```

```
>> A(2:3,1:2) % from 2nd to 3rd row; from 1st to 2nd column
ans =
    -1 3
    0 2
>> inv(A) % returns the inverse of the square matrix A
ans =
     0.1053   -0.0526   -0.0263
     0.1053    0.9474   -1.0263
    -0.0526   -0.4737    0.7632
>> diag(A) % returns the main diagonal of A
ans =
    9
    3
    4
>> eig(A) % returns a vector of the eigenvalues of matrix A
ans =
    0.6426
    7.6787 + 0.4106i
    7.6787 - 0.4106i
>> p1 = poly(A) % returns characteristic polynomial of matrix A
p1 =
    1.0000 -16.0000 69.0000 -38.0000
>> roots(p1) % returns roots of the polynomial p1
ans =
    7.6787 + 0.4106i
    7.6787 - 0.4106i
    0.6426
>> p2 = [1 -2 0 5]; % row vector contains the coefficients
% of a polynomial, ordered in descending powers, i.e.
% p2 = x^3 - 2x^2 + 5
>> conv(p1,p2) % polynomial multiplication
ans =
    Columns 1 through 7
        1.0000 -20.0000 126.0000 -192.0000 -491.0000 956.0000 -380.0000
    Column 8
        0
```

Optimization toolbox: function `linprog`

Additional information: Start → Toolboxes → Optimization → Help

The function

```
x = linprog(c, A, b, Aeq, beq, lb, ub)
```

solves the linear programming problem specified by

$$z = c^T x \to \min$$

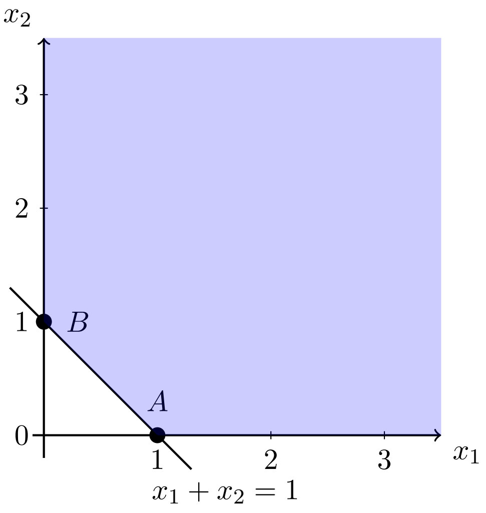
subject to constraints

$$A \cdot x \leq b$$
$$Aeq \cdot x = beq$$
$$lb \leq x \leq ub$$

where f, x, b, beq, lb, and ub are vectors, and A, Aeq are matrices.

Example 1: Recall example from Lecture 2. Consider the following linear programming problem

$$z = x_1 + 2x_2 \to \min$$
$$x_1 + x_2 \geq 1$$
$$x_1 \geq 0, x_2 \geq 0.$$



Minimum: $z_{\min}(A) = z_{\min}(1, 0) = 1$.

From description of the problem presented above we can create the following variables

```
>> c = [1; 2];
>> A = [-1 -1];
>> b = -1;
>> lb = zeros(2,1);
```

Now, we can use the function `linprog` to solve the linear programming problem with the following input arguments[1]

```
>> [x, fval] = linprog(c, A, b, [], [], lb);
Optimization terminated.

x =
    1.0000
    0.0000
```

[1]If some of the arguments is missing or unknown, write `[]` instead.

```
fval =
    1.0000
```

Hence, the optimal solution is $z_{\min}(1, 0) = 1$ and coincides with that obtained before.

Example 2: Recall example from Lecture 4. Consider the following linear programming problem

$$\begin{aligned} z = 3x_1 + 4x_2 \to \max \\ 2x_1 + 4x_2 \le 120 \\ 2x_1 + 2x_2 \le 80 \\ x_1 \ge 0, x_2 \ge 0. \end{aligned}$$

This is a problem of maximization. In order to use MATLAB functionality, we need to convert the objective function as

$$z' = -3x_1 - 4x_2 \to \min,$$

where $z' = -z$. Next, we create the following variables

```
>> c = [-3 -4];
>> A = [2 4; 2 2];
>> b = [120; 80];
>> lb = zeros(2,1);
```

and use the function `linprog` to solve the linear programming problem

```
>> [x, fval] = linprog(c, A, b, [], [], lb);
Optimization terminated.

x =
   20.0000
   20.0000

fval =
 -140.0000
```

The optimal solution is found. Hence, the solution of the original problem is $z_{\max}(20, 20) = 140$.

Example 3: Consider transportation problem presented by the following table

Warehouse	Destinations					Reserve
	D_1	D_2	D_3	D_4	D_5	
W_1	5	6	2	6	7	10
W_2	3	1	6	1	3	50
W_3	8	5	4	7	9	60
Requirement	10	30	20	10	50	

Transportation problem can be written as the linear programming problem

$$z = \sum_{i=1}^{3} \sum_{j=1}^{5} c_{ij} x_{ij} \to \min$$

subject to the constraints

$$\begin{aligned}
x_{11} + x_{12} + x_{13} + x_{14} + x_{15} &= 10 \\
x_{21} + x_{22} + x_{23} + x_{24} + x_{25} &= 50 \\
x_{31} + x_{32} + x_{33} + x_{34} + x_{35} &= 60 \\
x_{11} + x_{21} + x_{31} &= 10 \\
x_{12} + x_{22} + x_{32} &= 30 \\
x_{13} + x_{23} + x_{33} &= 20 \\
x_{14} + x_{24} + x_{34} &= 10 \\
x_{15} + x_{25} + x_{35} &= 50 \\
x_{ij} &\geq 0 \qquad\qquad 1 \leq i \leq 3, 1 \leq j \leq 5.
\end{aligned}$$

One can easily check that the transportation problem is in the balanced form, i.e., $\sum_{i=1}^{3} a_i = 120 = \sum_{j=1}^{5} b_j$. Moreover, from the constraints presented above we can construct the corresponding 8×15- and 8×1-dimensional matrices

$$Aeq = \left[\begin{array}{ccccc|ccccc|ccccc}
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

and

$$beq = \begin{bmatrix} 10 & 50 & 60 & 10 & 30 & 20 & 10 & 50 \end{bmatrix}^{\mathrm{T}},$$

respectively. Next, the function `linprog` can be used as follows

```
>> x = linprog(c, [], [], Aeq, beq, zeros(15,1));
Optimization terminated.
```

Then, the minimal transportation cost is then

$$\begin{aligned}
z = c_{11}x_{11} + c_{24}x_{24} + c_{25}x_{25} + c_{32}x_{32} + c_{33}x_{33} + c_{34}x_{34} + c_{35}x_{35} = \\
= 50 + 6.3448 + 130.9656 + 150 + 80 + 25.5864 + 57.1032 = 500.
\end{aligned}$$

Control System toolbox: basic functions

Additional information: Start → Toolboxes → Control System → Help

In order to generate random continuous-time model of the nth order with p outputs and m inputs, the following command can be used

```
>> sys = rss(n, p, m) % random test model
```

In order to create a SS object representing the continuous-time state-space model from the known matrices A, B, C and D, the following command can be used

```
>> sys_ss = ss(A, B, C, D) % creates a SS object
```

Controllability and observability properties can be checked as

```
>> Qc = ctrb(sys) % compute controllability matrix for the model
>> rank(Qc) % rank of the controllability matrix
```

and

```
>> Qo = obsv(sys) % compute observability matrix for the model
>> rank(Qo) % rank of the observability matrix
```

Eigenvalues of the system can be calculated by means of the following function

```
>> eig(A) % eigenvalues
```

In order to find transfer function description of the system, the following command can be used

```
>> tf(sys) % convert the model to transfer function form
```

LTI Viewer: allows to perform a brief analysis of the system

```
>> ltiview(sys) % open an LTI Viewer containing the step response
```

Pole placement

Full state feedback or pole placement, is a method employed in feedback control system theory to place the closed-loop poles of a plant in pre-determined locations in the s-plane. Placing poles is desirable because the location of the poles corresponds directly to the eigenvalues of the system, which control the characteristics of the response of the system. The system must be considered controllable in order to implement this method.

Let a system is given in the state-space form

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{1}$$

The poles of the system are the roots of the characteristic equation given by

$$|sI - A| = 0.$$

We shall choose the control signal to be

$$u = -Kx. \tag{2}$$

This means that the control signal u is determined by an instantaneous state. Such a scheme is called state feedback. The $1 \times n$ matrix K is called the state feedback gain matrix. We assume that all state variables are available for feedback. The latter comes from the complete state controllability property of the system.

Substituting equation (2) into equation (1) gives

$$\begin{aligned}\dot{x} &= (A - BK)x \\ y &= (C - DK)x\end{aligned}$$

It obvious that the stability and transient response characteristics are determined by the eigenvalues of matrix $A - BK$. If matrix K is chosen properly, the matrix $A - BK$ can be made an asymptotically stable matrix. The eigenvalues of matrix $A - BK$ are called the regulator poles, which have to be placed in the left-half s plane. The latter can be done by calculating roots of the characteristic equation, $\det[sI - (A - BK)]$ and comparing the terms of this equation with those of the desired characteristic equation.

Example 4: Given state-space model of the inverted pendulum on a cart

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx + Du\end{aligned}$$

in which $x = \begin{bmatrix}\theta & \dot{\theta} & d & \dot{d}\end{bmatrix}^T$, θ [rad] is angle of pendulum, d [m] is position of a cart with respect to the origin, u [N] is force of motion (model input),

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \dfrac{M+m}{M \cdot l} g & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\dfrac{m}{M} g & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ -\dfrac{1}{M \cdot l} \\ 0 \\ \dfrac{1}{M} \end{bmatrix}, \quad C = I_{4\times4}, \quad D = 0_{4\times4}$$

and parameters are defined as: mass of cart $M = 5$ [kg], mass of pendulum $m = 0.5$ [kg], distance to pendulum center of mass $l = 0.5$ [m], acceleration of gravity $g = 9.81$ [m/s^2].

The matrix K can be calculated using MATLAB function `place` as follows

```
% initialization of parameters
>> g = 9.8;
>> M = 2;
>> m = 0.2;
>> l = 0.6;

% matrices of the model
>> A = [0 1 0 0; g*(M + m)/(M*l) 0 0 0; 0 0 0 1; -m/M*g 0 0 0]
>> B = [0; -1/(M*l); 0; 1/M]
>> C = eye(4)    % unit matrix
>> D = zeros(4,1)    % zero matrix

>> sys = ss(A, B, C, D)    % create state-space model

% controllability
```

```
>> Qc = ctrb(sys)
>> rank(Qc)

>> eig(sys)    % eigenvalues of the system

% Synthesis of the closed-loop system
>> s1 = -1; s2 = -2; s3 = -3; s4 = -4;    % desired poles
>> S = [s1 s2 s3 s4]

>> K = place(sys.a, sys.b, S)    % calculate values for the matrix K
```

Solving quadratic optimal regulator problems with MATLAB

The material presented in this section is based on Ogata's book (Section 10-8). An advantage of the quadratic optimal control method over the pole-placement method is that the former provides a systematic way of computing the state feedback control gain matrix.

In MATLAB, the command

```
lqr(A,B,Q,R)
```

solves the continuous-time, linear, quadratic regulator problem and the associated Riccati equation. This command calculates the optimal feedback gain matrix K such that the feedback control law

$$u = -Kx$$

minimizes the performance index

$$J = \int_0^\infty (x^* Q x + u^* R u) \mathrm{d}t$$

subject to the constraint equation

$$\dot{x} = Ax + Bu.$$

Another command

```
[K,P,E]=lqr(A,B,Q,R)
```

returns the gain matrix K, eigenvalue vector E, and matrix P, the unique positive-definite solution to the associated matrix Riccati equation

$$PA + A^* P - PBR^{-1} B^* P + Q = 0.$$

Note that if matrix $A - BK$ is a stable matrix, such a positive-definite solution P always exists. The eigenvalue vector E gives the closed-loop poles of $A - BK$. It is important to note that for certain systems matrix $A - BK$ cannot be made a stable matrix, whatever K is chosen. In such a case, there does not exist a positive-definite matrix P for the matrix Riccati equation. For such a case, the commands

```
K=lqr(A,B,Q,R)
[K,P,E]=lqr(A,B,Q,R)
```

do not give the solution.

Example 5: Consider the system described by

$$\dot{x} = Ax + Bu,$$

where

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The performance index J is given by

$$J = \int_0^\infty (x'Qx + u'Ru)\mathrm{d}t,$$

where

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} 1 \end{bmatrix}.$$

Assume that the following control u is used

$$u = -Kx.$$

Determine the optimal feedback gain matrix K. The optimal feedback gain matrix K can be obtained by solving the following Riccati equation for a positive-definite matrix P

$$A'P + PA - PBR^{-1}B'P + Q = 0.$$

The result is

$$P = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}.$$

Substituting this P matrix into the following equation gives the optimal K matrix

$$K = R^{-1}B'P = \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix}.$$

Thus, the optimal control signal is given by

$$u = -Kx = -x_1 - x_2.$$

The same problem can be solved by MATLAB

```
% ----- Design of quadratic optimal regulator system -----
A = [0 1;0 -1];
B = [0;1];
Q = [1 0; 0 1];
R = [1];
K = lqr(A,B,Q,R)
K =
1.0000 1.0000
```