Massachusetts Institute of Technology
Department of Mechanical Engineering

2.003J/1.053J Dynamics & Control I

Fall 2007

Homework 1 Solution

Problem 1.1 : Matrix generation

i) You can make one of following methods listed below:

- `A = [ [1 2 3]' [4 5 6]' [7 8 9]'];`
- `A(1,:) = [1 4 7]; A(2,:) = [2 5 8]; A(3,:)=[3 6 9];`
- `A(:,1) = [1 2 3]; A(:,2) = [4 5 6]; A(:,3)=[7 8 9];`
- `A(1,1) = 1; A(2,1) = 2; A(3,1) = 3; A(1,2) = 4; A(2,2) = 5; A(3,2) = 6; A(1,3) = 7; A(2,3) = 8; A(3,3) = 9;`

and so on…

`B = [1 2 3; 0 3 1; 2 1 1];` or equivalent ways shown in creating matrix A.

```
>> A = [1 4 7; 2 5 8; 3 6 9]
A =
    1    4    7
    2    5    8
    3    6    9
>> B = [1 2 3; 0 3 1; 2 1 1]
B =
    1    2    3
    0    3    1
    2    1    1
```

ii) The most convenient method is to use `eye()` function (Please, see `eye()` in help menu.)

```
>> C = eye(3)       % eye(N) generate identity matrix with N x N
C =
```

```
     1     0     0
     0     1     0
     0     0     1
```

Problem 1.2 : Basic matrix operations

i) `A*B` gives matrix multiplication operation of matrices A and B (inner product between corresponding row of matrix A and corresponding column of matrix B), whereas `A.*B` provides element by element multiplication only between corresponding elements of matrices A and B.

```
>> A*B         % Matrix multiplication
ans =
    15    21    14
    18    27    19
    21    33    24
>> A.*B       % Element by element multiplication
ans =
     1     8    21
     0    15     8
     6     6     9
```

ii) In matrix multiplication, commutativity does not generally hold.

```
>> A*B
ans =
    15    21    14
    18    27    19
    21    33    24
>> B*A         % Result is not same as A*B
ans =
    14    32    50
     9    21    33
     7    19    31
```

iii) Following two operations are same since `.*` holds commutativity.

```
>> A.*B
ans =
```

```
     1     8    21
     0    15     8
     6     6     9
>> B.*A     % Result is same as A.*B
ans =
     1     8    21
     0    15     8
     6     6     9
```

iv) `B*D` is a 3×3 identity matrix, since $B*D = B*B^{-1} = I$.

```
>> D=B^-1       % or D=inv(B)
D =
   -0.1667   -0.0833    0.5833
   -0.1667    0.4167    0.0833
    0.5000   -0.2500   -0.2500
>> B*D          % Generates 3x3 identity matrix
ans =
     1     0     0
     1     1     0
     1     0     1
```

v) `A/B` is roughly same to `A*inv(B)` (solution to equation $XB = A$) if B is a square matrix, and `A\B` is roughly same to `inv(A)*B` (solution to equation $AX = B$) if A is a square matrix. (For more information see `mldivide()` and `mrdivide()` in help menu.) Since matrix A is singular, inverse of A does not exists. (determinant of A is equal to 0. Check it with `det(A)`.) Therefore, MATLAB gives the below warning to us for `A\B`.

```
>> A/B      % equal to A*inv(B)
ans =
   2.6667   -0.1667   -0.8333
   2.8333   -0.0833   -0.4167
   3.0000         0         0
>> A\B      % equal to inv(A)*B
Warning: Matrix is singular to working precision.
```

```
ans =
   NaN   NaN   NaN
   Inf  -Inf   Inf
  -Inf   Inf  -Inf
```

vi) $a_{ij} = 5\delta_{ij}\ (i, j = 1,2,3)$ where a_{ij} is i-th row and j-th column element of matrix. (diagonal matrix with all 5's)

```
>> E=5*eye(3)        % or E=diag([5 5 5]), and so on
E =
    5    0    0
    0    5    0
    0    0    5
```

vii) Understanding data types

i) By default, MATLAB assigns variables to `'double'` data type variables. Therefore, A, B, and C are matrices whose elements are all `'double'` data types. You can check it with `'whos'` command in MATLAB.

```
>> whos
  Name      Size            Bytes  Class    Attributes
  A         3x3                72  double
  B         3x3                72  double
  C         3x3                72  double
```

ii) `'double'` data type variable requires 8 bytes to store a variable value, and each element in matrix `A` is 8 byte long. Since matrix `A` has 3×3=9 elements (or variables), matrix `A` needs 72 bytes. Therefore, **576 bits** are required to store the information in matrix `A`, since 1 byte consists of 8 bits.

iii) New matrices `F` and `G` are displayed same as matrices `A` and `B` respectively, but they are stored in the computer with different data format.

```
>> F=int8(A)
F =
   1   4   7
   2   5   8
```

```
    3    6    9
>> G=int8(B)
G =
    1    2    3
    0    3    1
    2    1    1
```

iv) Addition between only same integer data types is supported in MATLAB, but matrix multiplication between any integer data types is not. Therefore, `F+G` produces matrix addition and `F*G` generates error as below. (See `'Arithmatic operators'` in help menu.)

```
>> F+G     % Only allowed if both are same integer types
ans =
   2    6   10
   2    8    9
   5    7   10
>> F*G     % Never allowed if at least one is integer data type
??? Error using ==> mtimes
Integer data types are not fully supported for this operation.
At least one operand must be a scalar.
```

viii) Submatrix extraction and plotting

i) First, saved data should be retrieved with `load()`. `':'` can be used to select either whole row or whole column in the matrix.

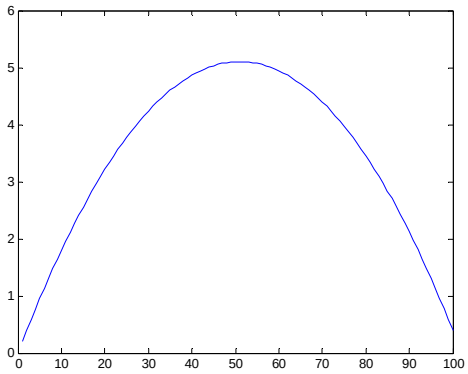
```
>> load ball.mat  % load data from ball.mat
>> t=A(:,1);     %  Extract 1st column data, and assign to 't'
```

ii)

```
>> x=A(:,2);     % Extract 2nd column data, and assign to 'x'
```

iii)

```
>> plot(t,x);         % Plot t vs. x
```



iv) `1:50` means selection from first element to 50th elements

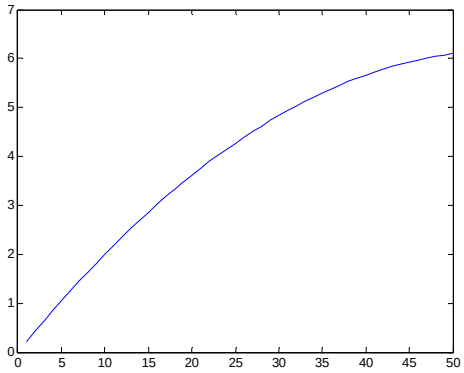
```
>> t2=A(1:50,3); %Extract first 50 elements from 3rd column of A
```

v)

```
>> x2=A(1:50,4); %Extract first 50 elements from 4th column of A
```

vi)

```
>> plot(t2,x2);        % Plot t2 vs. x2
```



vii) You can enumerate appropriate arguments for each graph (x data, y data, and linespec) sequentially in the `plot()` function. `'r'` means red color and `'b'` indicates blue color. (See `'linespec'` in help menu) Alternatively, you can also overlap several plots with `'hold on'` and `'hold off'` (See `'hold'` in help menu)

```
>> plot(t,x,'r',t2,x2,'b');
                    or
>>plot(t,x,'r'); hold on; plot(t2,x2,'b'); hold off;
```