MATH3290 Mathematical Modeling

MATLAB Review

12th September 2017

Outline

Basic Operations

a. Addition (Subtraction)

1. Between scalars:
   e.g.

```
2 + 4 = ?  3.14 + 1.57 = ?  2/7 + 0.16 = ?
```

2. Between scalar and vector (or matrix):
   e.g.

```
1+ [1;1;1;1] = ?
1+ [1,0;0,1] = ?
```

3. Between vectors / matrice:
   e.g.

```
[1;2;3;4]+[5;6;7;8] = ?
[1;2;3;4] + [5;6;7]
```

(Error)

Basic Operations (cont.)

Remark:

1. Each row of matrix end with the sign `";"`.
2. Sometimes it is useful to grasp part of the matrix using the sign `":"` and the keyword `"end"`.

```
» A = magic(5);
 A(2:3,4:5)  = ?
  A(1:3,:)   = ?
   A(:,1:3)  = ?
 A(:,2:end)  = ?
```

3. For subtraction, one can just replaced `"+"` by the sign `"−"` with exactly the same syntax.

Basic Operations (cont.)

b. Multiplication

1. Between scalars:
   e.g.

```
2 * 4 = ?  3.14 * 1.57 = ?  2.7 * 3/2 = ?
```

2. Between scalar and vector (or matrix):
   e.g.

```
3.5 * [1;1;1;1] = ?
pi * [1,2;3,4] = ?
```

3. Between matrice (Matrix-Multiplication) e.g.

```
[1 2 3] * [4;5;6] = ?
[1 2 3; 4 5 6] * [7 8; 9 10; 11 12] = ?
```

Basic Operations (cont.)

4. Between matrix (entry-wise)

```
A = [1 2 3; 4 5 6];

B = [7 8 9; 10 11 12];

A .* B = ?

a = [1;2;3], b = [4;5;6];

sum(a.*b) = ?
```
(inner product)

```
a' * b = ?
```
(inner product)

Remark: The build-in function "`sum`" is used to obtain the sum of all entries of a vector.

Basic Operations (cont.)

c. Division (Inversion)
e.g.

```
2/7 = ?

2\7 = ?

A = [1 2; 4 5], B = [7; 10]; x =A\B; x = ?

A ^ (-1) = ? inv(A) = ?
```

Remark: We prefer to use `x = A\b` more than `x = inv(A) * b`.

Basic Operations (cont.)

d. Power
e.g.

```
2^5 = ?  power(2,5) = ?
A = [1 2; 4 5], B = [7 8; 10 11];
A .^ B = ?
```

e. Square root or *n*-th root
e.g.

```
27^(1/3) = ?  nthroot(27,3) = ?
16^(1/2) = ?  sqrt(16) = ?
```

Summary

- Basic operations including: addition, subtraction, multiplication, division and power.
- Display format in MATLAB: `short`, `long`, `rat`, `bank` etc.
- For some operations (e.g. multiplication or power), there are entry-wise operations `.*` or `.^` .
- One can solve the linear system (assume `inv(A)` exists)

$$Ax = b$$

by using the "\" in MATLAB:

```
x = A\b.
```

Build-in Functions

There are many useful build-in functions in MATLAB. Here, we just name very few of them, which are frequently used in this course.

`zeros`	`mean`	`save/load`
`ones`	`norm`	`diag`
`repmat`	`sum`	`eye`
`kron`	`clear/clc`	`floor/round/ceil`
`linspace`	`abs`	`whos`
`disp`	`max/min`	`rand`
`mod`	`det/rank`	`eig`

Define a function

Sometimes the user want to define a function by oneself and there are two ways to do so.

1. Define a function in a separate `m.file`:

   ```
   function [outputs] = function name(inputs)
    program statements ...
   end
   ```

   e.g.

   ```
   function[x1, x2] = quadsolver(a,b,c)
    d = sqrt(b^2-4*a*c);
    x1 = (-b + d) / (2*a);
    x2 = (-b - d) / (2*a);
   end
   ```

2. Function handle:

```
 function name = @ (list of arguments)
mathematical expressions
```

e.g.

```
 m = 500, n = 1000;
 A = rand(n,m); b = sign(rand(m,1)-0.5);
 f = @(w,c) ...
 e'*log(1+exp(-b.*(A'*w + c)))/m;
```

e.g.

```
 quadsolver = @(a,b,c) ...
 [(-b+sqrt(b^-4*a*c))/(2*a)
(-b-sqrt(b^-4*a*c))/(2*a)];
```

Syntax

a. If-else

The syntax of if-else statement:

```
if( logical expression )
 program statements ...
end
```

or

```
if( logical expression )
 program statements ...
else
 another program statements ...
end
```

Remark: Usually, the logical expressions can be some variables, inequality or equality

Logical symbols

These symbols are used to formulate the logical expressions.

`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to
`==`	Equal to
`~=`	Not equal to

e.g.

```
if( b^2-4*a*c >= 0 )
 disp('This quad-eqn has two real roots.')
else
 disp('This quad-eqn has two imaginary
roots.')
end
```

For-loop Statement

To execute some statements specific number of times (e.g. solving ODE with Euler's scheme), we introduce `for-loop` in MATLAB.

```
for idx = 'some vector'
 program statements ...
 ...
end
```

Remark: Usually, `'vector'` used for the `idx` is `1:1:n`, where `n` is a large integer.

For-loop Statement (cont.)

Example I: Forming Hilbert matrix

$$H = (h_{ij})_{k\times k} = \Big(1/(i+j-1)\Big)_{k\times k}.$$

```
k = 5;
H = zeros(k,k);
for m = 1:k
 for n = 1:k
   H(m,n) = 1/(m+n-1);
 end
end
```

For-loop Statement (cont.)

Examle II:

```
for s = 1.0:  -0.1 :  0.0
   disp(s);
end

for s = [1 5 8 17]
   disp(s);
end
```

While-loop Statement

Repeatedly execute statements when the condition is true. The syntax of `while-loop` in MATLAB is as follows:

```
while expression
 ...  program statements ...
 ...
end
```

Remark: First, if the expression is true, then the program statements will be executed. After finishing all the statements, the expression will be determined again. The program will keep running if the result of expression is true, otherwise terminate.

Remark: Try to avoid `infinite-loop` when using `while-loop`.

Example: (Classical) gradient method
Consider the following minimization problem: given a differentiable convex function f

$$\min_{x\in\mathbb{R}^n} f(x).$$

The algorithm: choose x^0 and repeat

$$x^{k+1} = x^k - t_k \nabla f(x^k), \quad k = 0, 1, 2, \cdots .$$

Note that $p_k := -\nabla f(x^k)$ is a descent direction. How to determine step size t_k?

Step size rules: backtracking line search
Initialize t_k at some fixed $\hat{t} > 0$ (e.g. $\hat{t} = 1$), repeat $t_k \leftarrow \beta t_k$ until the following condition holds:

$$f(x - t_k \nabla f(x)) < f(x) - \alpha t_k \|\nabla f(x)\|_2^2.$$

Remark: This is called `Amijo condition` and the parameters satisfy $0 < \beta < 1$ and $0 < \alpha \leq 0.5$.

Question: how to code the algorithm and this line search up? Currently, let f be the function defined as follows:

$$f(x) = f(x_1, x_2) = e^{-0.1}\left(e^{x_1+3x_2} + e^{x_1-3x_2} + e^{-x_1}\right).$$

While-loop

Example: Gradient line search in optimization.

```
clear
x = [0;0];
alpha = 0.1;   beta = 0.7;
frecord = zeros(maxiter,1);
grecord = zeros(maxiter,1);
f = @(x) (exp([1 3]*x) + exp([1 -3]*x) + ...
  exp([-1 0]*x))*exp(-0.1);
gradf = @(x) (exp([1 3]*x)*[1;3] + ...
  exp([1 -3]*x)*[1;-3] + ...
  exp([-1 0]*x)*[-1;0])*exp(-0.1);
gradnorm = norm(gradf(x));
iter = 1; tol = 1e-4;
```

While-loop

```
while gradnorm >= tol && iter <= maxiter
  grad = gradf(x);
  grecord(iter) = norm(grad);
  fvalx = f(x);
  frecord(iter) = fvalx;
  t=1; y = x-t*grad;
  fvaly = f(y);
  while fvaly >= fvalx - alpha*t*norm(grad)^2
     t = beta*t;
    y = x-t*grad;
  end
  x = y;
  gradnorm = norm(gradf(x));
  iter = iter + 1;
end
figure(2);
semilogy((1:size(grecord,1)),grecord);
```

A Typical Example

Plotting graph is important. A graph is better than a thousand words. You are highly recommended to sketch some figures by MATLAB or other software in your assignments if necessary.

First, we will look at a typical example in the assignment.
Let M_n be the population of mouse after n years and O_n be the owl population after n years. The ecologist has suggested the following model:

$$\begin{aligned} M_{n+1} &= 1.2M_n - 0.001 O_n M_n, \\ O_{n+1} &= 0.7O_n + 0.002 O_n M_n. \end{aligned}$$

The ecologist wants to know if the two species can coexist in the habitat and if the outcome is sensitive to the starting populations.

Learning by doing

If the initial populations are given: $O_0 = 150$ and $M_0 = 200$, how do the populations of two species change after 40 years? For this problem, we can write a program to testify. Here is the code:

```
M = 200; O = 150;
n = 40;
record = zeros(2,n+1);record(:,1) = [M;O];
for i = 2:n+1
  Mnew = 1.2*M - 0.001*O*M;
  Onew = 0.7*O + 0.002*O*M;
  record(:,i) = [Mnew;Onew];
  M = Mnew;
  O = Onew;
end
```

Question: How can we show our result?

Usually, we use the build-in function `plot` to draw our figures. We can use the keyword `figure` to open the window of figure in MATLAB. For example:

```
fh = figure;
h = plot(X1,Y1, ...  ,Xn,Yn);
```

In our case, `X1 = X2 = 1:(n+1)` and `Y1 = record(1,:)`, `Y2 = record(2,:)`. Hence, we can draw a figure by the following command:

```
fh = figure;
h = plot(X1,Y1,X2,Y2);
```

Further, we can set the properties of the figure such as the `title`, `label` and `legend`. Finally, don't forget to save your figure as a `fig.file`.

When we are coding, there are always bugs (`error`) in the program. Sometime, the bugs come from the error of syntax. This kind of bugs are easy to find out.

Another type of bugs come from the logical wrong program with correct syntax. This is usually hard to find out and obtain unexpected result.

Let us take a look at an example: `quadsolver.m`.

```
function[x1, x2] = quadsolver(a,b,c)
 d = sqrt(b^2-4*a*c);
 x1 = (-b + d) / (2*a);
 x2 = (-b - d) / (2*a);
end
```

We want to determine if the root is real or complex. The key point is to make use of the quantity $b^2 - 4ac$.

```
function[x1, x2] = quadsolver(a,b,c)
 d = sqrt(b^2-4*a*c);
 if (d >=0)
   disp('This quad-eqn has two real roots.');
 else
   disp('This quad-eqn has two imaginary
roots.');
 end
 x1 = (-b + d) / (2*a);
 x2 = (-b - d) / (2*a);
end
```

We test two cases: $x^2 - 2x + 1 = 0$ and $x^2 + 1 = 0$.

```
» [x1,x2] = quadsolver(1,-2,1);
This quad-eqn has two real roots.
```

Good to know!

```
» [x1,x2] = quadsolver(1,0,1);
This quad-eqn has two real roots.
```

Why? What happen?

Let us go back to the code and see what is wrong.

Tips: we can set `Breakpoint` in the program so that we can see how the program runs step by step.

After scanning how the program runs, we change the code a little bit and see what happen then.

```
function[x1, x2] = quadsolver(a,b,c)
 d = (b^2-4*a*c);
 if (d >=0)
   disp('This quad-eqn has two real roots.');
 else
   disp('This quad-eqn has two imaginary
roots.');
 end
 x1 = (-b + d) / (2*a);
 x2 = (-b - d) / (2*a);
end
```

```
» [x1,x2] = quadsolver(1,0,1);
This quad-eqn has two imaginary roots.
```

Problem fixed.

Remark: The logical sign `'>='` or `'<='` only makes use of the real part of a complex number and we should avoid to compare a complex number to another quantity, no matter this quantity is complex or not.

Help and Doc

The most efficient way to find a solution for a specific MATLAB program is to use the keywords `'help'` and `'doc'` in MATLAB. For example, we know that one can use function `'plot'` to draw some figures but we do not know how to edit the properties of a figure after we drew it. Then, we can type the following command in the command window:

```
» doc plot
```

Ask smartly

Another way is to search the answer from internet, especially from **Google**. Try to use the accurate key-words to represent your problem.

We can ask question on the forum, e.g. `Stackexchange`. Before asking the question, we should make our problem as clear as possible and let people find it easier to provide the answer (very IMPORTANT).

Practice problems

1. Consider the ordinary differential equation with initial condition

$$\frac{dy}{dt} = f(t, y) \quad y(t_0) = y_0, \quad t \in [t_0, T]$$

where y_0 is given. One can use Euler's method to numerically solve this ODE. The scheme is as follows: for $n = 0, 1, \cdots,$

$$y_{n+1} = y_n + f(t_n, y_n)\Delta t,$$

Δt is a given step size of time, $t_n = t_0 + n\Delta t$ and $T = t_0 + N\Delta t$.

In this problem, assume the function $f(t, y)$ and initial condition are given as follow:

$$f(t, y) = 1 + \frac{y}{t} \quad t \in [1, 10] \quad \text{and} \quad y(1) = 2.$$

Take $\Delta t = 0.1, 0.2$ and 0.5. Write a program to solve the ODE. Record all the y_n and plot y_n against t_n. Also, compare the numerical solution to the exact solution of the ODE

$$y(t) = t \log(t) + 2t.$$

Hint: To compare, one can define a function handle

```
g(t):=t*log(t)+2*t
```

and the vector variable `error := abs(g(tn) - yn)`, where `yn` is your numerical approximation.

Practice problems

2. Assume that you are given a data set $\{x_i, y_i\}_{i=1}^N$. We want to find a parabola $f(x; p_0, p_1, p_2) := p_0 + p_1 x + p_2 x^2$ such that the mean square error is minimized:

$$S(p_0, p_1, p_2) := \sum_{i=1}^{N} |y_i - f(x; p_0, p_1, p_2)|^2.$$

Assume that $N = 100$ and the data are included in the file `LSdata.mat`. Write a program to `load` the data and find out the parameters p_0, p_1 and p_2. Plot out the parabola $f(x; p_0, p_1, p_2)$ over the interval $x \in [-3, 3]$.

Hint: Use the method of **normal equation** (Will be discussed in the next tutorial section `^.^`).

Hint: We can plot the parabola against

```
x = linspace(-3,3,L)
```

in MATLAB, where `L` is a large constant.

The End.