COMS 3101
Programming Languages: MATLAB

Lecture 4

Fall 2013

Instructor: Ilia Vovsha

http://www.cs.columbia.edu/~vovsha/coms3101/matlab

Lecture Outline

- Review: EHW#1,2
- Plotting (figures)
- Data structures: cell arrays, structs, strings, handles
- Practical math: formulating and solving problems
- Next Lecture: optimization
- Next Lecture: Final project overview and options
- Next Lecture: advanced functionality

Useful Remarks

- Avoid making repeated calls to built-in functions
- Instead of accessing an element many times, just define and use a variable: t = M(i,j)
- Some programming languages allow you to increment variables using the operators {++,--}. This is not an option in MATLAB
- Functions that return a Boolean variable (0/1) could be used as the condition: "if isprime(N)" instead of "if isprime(N) == 1"
- Can use vectors and matrices in conditions too

Useful Remarks

- Solutions are posted (average: 80+)
- Multiply matrix M by a scalar C:
  - Answer: M*C % M.*C is redundant
- No need to increment the loop variable inside the for-loop
- On the other hand, incrementing is often essential in while-loops
- If you defined a variable for one purpose, do not redefine it for another below. This creates unnecessary confusion

Switch Statement – Usage

- When?
  - More than 2-3 choices
  - Choice is based on a common expression
- How?

```
switch switch_expr                    % Expression is a scalar or a string
    case case_expr                    % Match 'case_expr'
        statement(s)
    case {case_expr1,case_expr2}      % Match any case in the array
        statement(s)
    otherwise
        statement(s)
end
```

Switch Statement – Usage

- Brackets {} are not required unless you wish to execute the same code for more than one case
- ‘break’ statements are redundant. Unlike C++ for example, there is no ‘falling through’
- ‘otherwise’ is optional

A Silly Game

- Consider the following ‘game’:
  1. Choose any positive integer (1,2,…), call it X
  2. If X is an even integer you divide X by 2
  3. If X is an odd integer you multiply X by 3 and add 1
  4. Continue this procedure until at some point you get 1 as your next integer. At that point stop
- Claim: the game ends for every positive integer
- Can you prove the claim? Try it!

Collatz Conjecture

- Deceptively simple
- Proposed by Collatz in 1937, still unsolved (!)
- “Mathematics is not yet ripe for such problems” – Paul Erdos
- This conjecture/problem has many versions and names
- We can use MATLAB to generate some plots and perhaps obtain some insight

Collatz Conjecture – Example

- Problem: write a function that generates the sequence from N (input parameter) to 1, according to the game, and then plot each pair (sequence #, value) on a figure
- For example, if X = 3, the sequence is {3,10,5,16,8,4,2,1}. So you should plot the points (1,3), (2,10), (3,5)…(8,1)

Collatz Conjecture – Example

- Solution:

```
1.  Y = N;                          % Initialize
2.  while N > 1                     % or N ~= 1
3.          if mod(N,2) == 0        % Compute sequence
4.                  N = N / 2;
5.          else
6.                  N = (N*3) + 1;
7.          end
8.          Y(end+1) = N;
9.  end
10. LY = length(Y);
11. X = 1:LY;
12. plot(X,Y,'r.', X,Y,'b:');       % Plot red points and dotted blue line
```

Basic Plotting – Figures

- Problem:
  - plot(x,y); plot(x,z); replaces 1st plot with 2nd
- Solution: 'figure' command
  - plot(x,y); figure; plot(x,z);
  - figure(1); figure with handle #1
- Close figures:
  - Specific figure: close 1
  - All figures: close all
- MATLAB stores a handle to each figure

Basic Plotting – Figures

- Multiple plots:
  - plot(x,y); hold on; plot(x,z); hold off;
- Multiple plots, same figure:
  - subplot() command
  - figure(1); subplot(2,2,1);

Basic Plotting – Appearance

- Many options, can modify plots using the GUI
- Commands: title(), xlabel(), ylabel(), axis(), legend()
- Example:

```
1. figure(1);
2. title('test');
3. xlabel('quantity'); ylabel('price');
4. axis([1 5 1 10]);      % AXIS([XMIN XMAX YMIN YMAX])
5. grid on;               % Show grid lines
6. xlim([1 3]);           % Change x-axis limits
```

Basic Plotting – Figures

- You have created multiple figures. Which one is the 'current figure'?
- Answer: last figure you clicked on
- Better answer: use gcf() to get the handle
- Use figure(#) to make '#' the current handle
- Commands: gcf(), gca(), clf()
- Set 'object' properties using the set() command:
  - set(gca, 'XTick',[1 2 3]) % To set 'ticks' on the x-axis
- Get 'object' properties using the get() command

Saving & Loading Figures

- MATLAB has a special figure format: '.fig'
- Use openfig('name.fig') to open a saved figure
- Saving figures: use print command
  - **General Form**
  - **print** -d*format* filename
  - **Example**
  - **print –depsc 'figure.eps'**
- 'eps' is a format that stores your image in a vectorized way, which avoids quality loss after rescaling. It's particularly useful when used within LaTeX

Exercise (In Class)

- Simple plot
- Write a function that has one input parameter M, a matrix with 3 columns. Columns {1,2} of M are the {x,y} coordinates of points in the plane. Column 3 is the class to which the point belongs. The values of column 3 are one of two unknown integers
- Your function should plot all points, points in one class should be dots in red, points in the 2nd class should be squares in green. Adjust your axes properly so that the furthest points are not on the edge

Data Structures – Structs

- When a composite data structure is required, use a 'struct' (structure array)
  - Multiple fields, different data structures for each field
  - Similar in form to C++ classes
  - Use 'dot' to access fields
- Initialization:
  - S = struct([]) % Empty struct, no fields
  - S = struct ('f1', v1, 'f2', v2) % Struct with two fields
  - S.field1 = 2.5; % Create directly

Data Structures – Structs

- Functionality:
  - isstruct(S); % Check if S is a struct
  - S = setfield(S, 'field1', 5); % Set a field to a value
  - isfield(S, 'field1'); % Check if field exists
- Example:

1. S = struct('vec', [1,2,3], 'mat', rand(3));
2. my_field = 'mat';
3. isfield(S, my_field);

Structs – Note

- When multiple variables are saved in a .mat file, and then loaded into a single variable, they are saved as fields of a struct
- Example:

```
1. save file1.mat X Y Z;
2. S = load('file1');
3. isstruct(S);          % Struct with 3 fields: S.X, S.Y, S.Z
```

Data Structures – Cell Arrays

- Cell array is an 'array of matrices'.
  - Each element of a cell array can be a scalar/vector/matrix.
  - Why is it useful? Recall the 1st exercise (loading a sequence of files)
- Initialization:
  - C = cell(N) % N by N array of empty matrices
  - C = cell(N,M) % N by M array of empty matrices
  - Use brackets {} to access elements
  - Rules for regular arrays apply
  - C = {[1] [1,2,3]; rand(3), rand(3)}; % Create directly

Cell to Struct Conversion

- Commands: cell2struct(), struct2cell()
  - S = cell2struct(C, my_fields, my_dim)
  - 'my_fields' is a cell array of strings
- Example:

```
1. my_fd = {'num', 'name', 'nation'};
2. my_arr = {8, 'Iniesta', 'Spain'};
3. my_st = cell2struct(my_arr, my_fd, 2);
% You are 'folding' the dimension, size must match
```

Strings – Functions

- MATLAB is not recommended as a tool for manipulating strings. However, the functionality is available
- Check string:
  - isletter(str), isspace(str)
- Convert string:
  - lower(str), upper(str)
  - str2num(str), num2str(num)

Strings – Functions

- Operations on strings:
  - strtok(str, delim), strcmp(str1,str2), strfind(str1,pattern)
- Regular expressions (help regexp): concise and flexible means for matching strings
- Example:

1. str1 = ‘one!, no two’;
2. [bef_delim, aft_delim] = strtok(str1, ‘!’);
3. idx = strfind(str1, ‘n’);
4. res1 = strcmp(‘hi’, ‘HI’);
5. res2 = strcmp(‘hi’, lower(‘HI’);

Building a Winning Team

- Consider the following scenario: you are a manager that must assemble a team of players. You have a scouting report describing the set of skills of each player, and the salary demands of each
- Your goal is to choose your team from a large pool of players, ensuring that your players complement each other
- Since your owner is stingy, you must also minimize your total player salary, while maintaining a competitive team

BWT – Problem Formulation

- More specifically:
  - Your team should consist of P players
  - You have a pool of N players ($N \geq P$) to choose from
  - Each player 'p', demands a salary of C(p)
  - The scouting report consists of K marks for each player. Each mark is a real value in the range [0.0,1.0] indicating the player's quality with respect to a particular skill (i.e. 0.0 is 'noob', 1.0 is 'world class')
  - To ensure a competitive team, you require that the total quality of all team players for every skill 'k' is at least some value B(k)
- Given all the relevant info (player pool, salary demands, scouting report), how do you choose a set of P players, make your owner happy, and still have a competitive team?

BWT – Problem Formulation

- Assumptions:
  - Team size = P, Pool size = N, (N ≥ P).
  - C (salary) is an N-by-1 (column) vector of positive real values
  - R is an N-by-K matrix of real values in the range [0.0,1.0]. Each row represents a player in the pool. Each column represents a skill
  - B (total quality) is a 1-by-K (row) vector of positive real values
- All the relevant info is given. That is, {P,N,C,R,B} must be supplied to us
- Note: no guarantee that the parameters are set correctly
- Note: cannot have 'half' a player on a team. The player is either signed or not

BWT – Problem Formulation

- Approach:
  1. Collect all parameters {P,N,C,R,B} % Input/load data
  2. Verify that parameters are set correctly % Error checking
  3. State the problem in mathematical notation:
     - We are clearly solving a constrained optimization problem
     - We are trying to minimize a linear objective (minimize the total salary), subject to:
     - One equality constraint, a team should have exactly P players
     - K linear inequality constraints (total quality for some skill is one constraint, we have K skills)
     - Our variables must be binary {0,1}, cannot sign 'half' a player
  4. Output/verify the solution % Output/save solution

Optimization

- "Do things best under the given circumstances"
- Applications in almost every field imaginable: planning, scheduling, resource allocation, management, traffic control
- Optimization problem:
  - Make a decision
  - Express/control the quality of the decision by the objective function
  - Typically a minimization/maximization task
  - Express "circumstances" that affect the decision as constraints
  - The type of opt. prob. is determined by the nature of the objective function and the constraints

Optimization – General Form

General From:

$$\text{minimize} \quad F(x)$$

$$\text{subject to:} \quad g_i(x) \leq b_i \qquad i = 1,\ldots,m$$

- The problem is characterized by the objective function F(x), and the constraints $g_i(x)$
- The variable or vector x, belongs to some domain/set S specified by the constraints
- Linear and Quadratic programs are the most frequent problems you are likely to encounter

BWT – Problem Formulation

- Mathematical notation:
  1. Let X be the variable/solution (column) vector, $X \in \{0,1\}^N$
  2. We wish to minimize the objective function $C^T X$
  3. One equality constraint: $\sum X_i = P$
  4. K linear inequality constraints: $R^T X \geq B^T$
  5. Complete form:

$$\begin{aligned} \text{minimize} \quad & C^T X \\ \text{subject to:} \quad & R^T X \geq B^T \\ & 1^T X = P \\ & \forall p,\ X_p \in \{0,1\} \end{aligned}$$

MATLAB Optimization Toolbox

- Extensive package. Many routines, options. Plenty of documentation ('help' is not sufficient)
  - First step: define your problem clearly, write down your equations
  - Second step: find the appropriate solver (what type of problem are you solving?)
  - Third step: convert your problem to solver form. This might require combining equations, switching sign of equations & objective, adding equations
  - Fourth step: set options, call solver, examine the solution

MATLAB Optimization Toolbox

- Find appropriate solver:
  - http://www.mathworks.com/help/toolbox/optim/ug/bqnk0r0.html
  - Frequent solvers: linprog(), quadprog(), fmincon()

- Solver form example:
  - Linear program

$$\min_x f^T x \text{ such that } \begin{cases} A \cdot x \leq & b, \\ Aeq \cdot x = & beq, \\ lb \leq x \leq ub & \end{cases}$$

BWT – Problem Formulation

- MATLAB Toolbox notation:
  - Appropriate solver: bintprog()
  - Why? Solution vector is a binary integer vector, objective is linear and the constraints are linear
  - Convert to solver form:

$$\min_x f^T x \text{ such that } \begin{cases} A \cdot x \leq b, \\ Aeq \cdot x = beq, \\ x \text{ is binary} \end{cases}$$

1. f = C;
2. A = -R’; % Change sign, transpose
3. b = -B’; % Change sign, column vector
4. Aeq = ones(1,length(C)); % $1^T X = P$
5. beq = P;

bintprog()

- Converted to solver form, variables {f, A, b, Aeq, beq}
- Function call options (syntax):

```
x = bintprog(f)
x = bintprog(f,A,b)
x = bintprog(f,A,b,Aeq,beq)
x = bintprog(f,A,b,Aeq,beq,x0)
x = bintprog(f,A,b,Aeq,Beq,x0,options)
x = bintprog(problem)
[x,fval] = bintprog(...)
[x,fval,exitflag] = bintprog(...)
[x,fval,exitflag,output] = bintprog(...)
```

bintprog()

- Syntax rules (for all solvers):
  - Parameter not passed, assume it is empty
  - Parameter order is important
  - To include a subsequent parameter, but omit a preceding one, pass an empty array [ ]
  - ‘options’ is a struct specifying optimization method details. Ignore it, unless you know a thing or two about the field
  - Instead of passing many parameters, can pass a single struct ‘problem’ with appropriate fields
  - Output parameters include solution (x), value of objective function at the solution (fval), flag indicating outcome of call (exitflag), and details about the execution (output)

bintprog() – Examples

- Parameter not passed, assume it is empty:
  - x = bintprog(f) % Minimize objective without constraints
- Include subsequent parameter, omit preceding one:
  - x = bintprog(f, [], [],Aeq,beq) % No inequality constraints
- Pass a single struct:
  1. problem.f = C;
  2. problem.Aineq = -R';
  3. problem.solver = 'bintprog';
  4. x = bintprog(problem);
  - Notice that field names are slightly different
  - Must set all fields (set to empty if doesn't exist)

bintprog() – Examples

- Output parameters:
  - ‘x’: the solution vector
  - ‘exitflag’: if returns 1, problem solved successfully
  - ‘output’: a structure with solution details. For example, output.time is execution time
  - Can name parameters in any way you wish
  - [soln, fval, the_flag, soln_details] = binprog(problem)
  - If solution vector is not what it should be, you must check all output parameters to discover the problem. You should start with the ‘exitflag’, though there is no prescribed approach to detect a problem