COMS 3101
Programming Languages: MATLAB

Lecture 5

Fall 2013

Instructor: Ilia Vovsha

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

Lecture Outline

- Review: HW#3
- Practical math / optimization (continued)
- Advanced functionality:
  - Time, error, sets
  - Function handles
  - Variable number of function arguments
  - 3D & area plots

Useful Remarks

- Suppose the objective is to access all rows of a matrix (M) which satisfy some condition. Instead of using 'find', can directly test for the condition, and then use the 0-1 'logical' vector to access relevant rows
- Example:

```
1. pts = M(:,3) >= 5                % pts is a 0/1 'logical' vector
2. plot( M(pts,1), M(pts,2) )       % plotting row k, only if pts(k) == 1
```

HW #3

- For HW3b, assumptions were: vector C of length two with distinct values:

```
1. if length(C) ~= 2
2.    return;
3. end
4. if C(1) == C(2)
5.    return;
6. end
```

- Why do we check "length(C) ~= 2" first?
  - "if (length(C) ~= 2 || C(1) == C(2))" is also OK

Loading Data

- Three step process:
  1. Identify the type of data you have
  2. Find the relevant function(s)
  3. Choose the appropriate format
- Example: “Each file has strictly numeric data separated by delimiter…range of data is specified”
  - Step 2: dlmread(), dlmwrite()
  - Step 3: “RESULT = DLMREAD(FILENAME,DELIMITER,RANGE)”
  - Reads the range specified by RANGE = [R1 C1 R2 C2]

num2cell()

- Problem 1: create a vector ‘V’ of length N, where each element, V(k) = 2^k
- Problem 2: create a cell array ‘C’ of length N, where each element, C{k} = 2^k
- Solution (1):

```
1. idx = 1:N;
2. V = 2.^idx;          % The dot is required here
```

- Solution (2):
  - Cell array, solution (1) doesn’t work!
  - Instead of using a for-loop could convert numeric array to cell

num2cell()

- Problem 2: create a cell array ‘C’ of length N, where each element, C{k} = 2^k
- Solution (2):

```
1.  idx = 1:N;
2.  V = 2.^idx;              % V is a numeric array
3.  C = num2cell(V);         % Converted V to a cell array C
```

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, ($N \geq P$)
  - 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

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

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

1st Step – Approach

- 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

1^{st} Step – Notation

- 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^TX
  3. One equality constraint: $\sum X_i = P$
  4. K linear inequality constraints: $R^TX \geq B^T$
  5. Complete form:

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

2nd Step – Choose Solver

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

- Solver form example:
  - Linear program

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

3rd Step – Solver Form

- 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;

4th Step – Call Solver

- 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(...)
```

General Solver Rules

- 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 = 'binprog';
  4. x = bintprog(problem);
  - Notice that field names are slightly different
  - Must set all fields except x0 (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

Time

- Stopwatch to record elapsed time (in seconds):
  - 'tic' to start the stopwatch
  - 'toc' to stop it
- Examples:
  1. tic; % code toc; % Displays elapsed time
  2. t1 = tic; % code t2 = toc(t1) % Stores elapsed time in 't2'
  3. tic; % code toc;

     % code toc; % 2nd toc displays elapsed time since tic!
- Can also obtain date, CPU-time (clock, cputime, datestr)

Error

- To handle errors and abort execution:
  - error (‘*some message*’) % Aborts execution and prints the message
- More ‘sophisticated’ error messages:
  - error ('*msgString*', *v1*, *v2*)
  - Assumes message string contains format specifiers (%s, %d)
  - Replaces each specifier with the appropriate parameter
- “Catch” errors:
  - “Try” some code, if you “catch” an error, override default code

```
1. try
2.     % test code
3. catch exception
4.     % error code;
5. end
```

Sets

- Relevant functions: unique(), intersect(), union(), setdiff(), ismember()
- General form:
  - <function>(A, B) % Applies function to vectors {A,B}
  - <function>(A, B, 'rows') % Applies function to matrices {A,B}
  - [C, IA, IB] = <function>(…) % Returns combination & indices
- Rules:
  - If {A,B} are matrices, must supply the 'rows' parameter
  - {A,B} must have the same number of columns
- ismember(A, B): returns a 0/1 array with the size of A. Where element is set to 1 if element is in B

Sets – Examples

- Define: V1 = [1,2,3,4]; V2 = [0,1,2];
- Define: A1 = [1,0;1,1;1,2]; A2 = [1,1;1,1;1,2];
- Examples:
  1. ans = intersect (V1, V2)
  2. ans = union (V1, V2)
  3. ans = union (A1, A2)
  4. ans = setdiff(V1, V2)
  5. ans = ismember (A1, 1)
  6. ans = ismember (V1, [-1,2,4])

textscan()

- Read formatted data from a text file
- textread() is another (deprecated) function with similar functionality
- How does it work?
  - *C* = textscan (*FID*, '*format*')
  - Recall that to open a text file, we need to create an identifier:

    FID = fopen ('myfile.text', 'r');
  - *'format'* is a string specifying how each line should be read
  - 'C' is a cell array. The number of specifiers (*'format'*) determines the number of cells

Function Handles @

- Function handle: a variable that stores an identifier for a function
  - h1 = @min; % ‘h1’ can now be used instead of ‘min’
  - val = h1(rand(3)) % Same as val = min(rand(3))
- Can use handles in cell arrays or structs, but not in regular arrays:
  - C = {@min, @max, @mean};
  - S.a = @min; S.b = @max; S.c = @mean;
  - ~~A = [@min,@max];~~ WRONG!

Function Handles @

- Purpose of function handles:
  - Suppose the user should be able to choose which sub-routine to use. Thus, a mechanism to pass a parameter which specifies the sub-routine is required
  - Can't pass a function, so pass a handle instead
  - Also possible to define functions on the fly (anonymous functions):
    1. sqr = @(x) x.^2;
    2. a = sqr(5);

Exercise (In Class)

- Sets & handles
- Write a function with 3 input parameters {A, B, fh}, which applies the function stored in the handle 'fh' to the vectors/matrices stored in {A,B}. You can assume that the function in 'fh' can only be one of the "set" methods (union, intersect, setdiff, ismember)

Variable Input Arguments

- Function which can have any number of input args
- Rules:
  - Keyword: varargin
  - 'varargin' is a cell array containing the optional arguments ONLY
  - Must be declared as the last input argument
  - Must be lowercase
  - Example: "function my_fun(x,y,z,varargin)"
  - Similarly, use varargout to create a function with any number of output args. The same rules apply (last, lowercase)

Exercise (In Class)

- Variable input arguments
- Write a function with *at least* 3 input parameters {A, B, fh1, fh2, fh3…}, which applies the functions stored in the handles 'fh1' (fh2, fh3…if present) to the vectors/matrices stored in {A,B}. You can assume that the function in 'fh#' can only be one of the "set" methods (union, intersect, setdiff, ismember)

More Plotting – 3D

- Very similar to 2D plotting
- Keyword: plot3()
- Example:

  1. X = 0:0.1:1; Y = X; Z = X.^2+Y.^2;
  2. plot3(X,Y,Z);
  3. xlabel('locX'); ylabel('locY'); zlabel('power');
  4. xlim([0 1]); zlim([0 40]);

More Plotting – Surface

- 'plot' functions produce line plots. Suppose we wish to display a 3D surface
- Keywords: meshgrid(), mesh(), surf()
- The command meshgrid(X,Y) creates a grid (domain) with all combinations of {X,Y} elements. mesh(X,Y,Z) / surf() are then used to plot the surface Z = f(X,Y) over the grid

3D Surface – Example

- Example:

```
1. X = 1:4;  Y = 1:3;
2. [Xg, Yg] = meshgrid(X,Y);        % produce a grid
3. Z = X.^2 + Y.^2;                 % Z = f(X,Y)
4. mesh(X,Y,Z);                     % plot
5. surf(X,Y,Z);                     % plot
```

- 'mesh' plots wireframe. 'surf' plots with shading
- Can pass a 4th parameter specifying color e.g. "surf(X,Y,Z,C)". Otherwise, color is proportional to mesh height

3D Surface – Example

- surf vs. mesh