Matlab Basics

Paul Schrimpf

January 14, 2009
Overview

Goals
- Matlab features
- Program design
- Numerical methods
Topics to be covered

- Essentials of Matlab
- Using Matlab’s features to design good programs
  - Example: dynamic programming
- Optimization and integration
  - Example: maximum likelihood
- Object-oriented programming
  - Example: automatic differentiation
Matlab References

- `help function` or more detailed, `doc function`
- Matlab Primer
- MATLAB on Athena
- 10.34 Matlab tutorial
- Numerical Computing with Matlab
- Art of Matlab
Operators

Matrix Operators

\begin{verbatim}
1   a+b;
2   a-b;
3   a*b;
4   a^n;
5   A';
6   A \ b; \ % returns x s.t. A*x=b
7   A / b; \ % returns x s.s. x*A=b
\end{verbatim}

Array Operators

\begin{verbatim}
1   a.*b;
2   a.^n;
3   a./b; \ % these are
4   a./b; \ % equivalent
5   a & b; \ % don't confuse with &&
6   a | b; \ % don't confuse with ||
7   ~a;
8   arrayfun(fn,a); \ % evaluate fn at each element of a (usually in
\end{verbatim}
Avoid Matrix Inversion

\[
K = 2000; \quad N = K+1; \quad b = \text{ones}(K,1);
\]
\[
x = \text{randn}(N,K); \quad y = x*b + \text{randn}(N,1);
\]
\[
xx = x'\cdot x;
xy = x'\cdot y;
\]

% from slow to fast ...
\[
tic; \quad \text{bhat1} = (xx)^{-1}\cdot xy; \quad toc;
\]
\[
tic; \quad \text{bhat2} = \text{inv}(xx)\cdot xy; \quad toc;
\]
\[
tic; \quad \text{bhat3} = xx \backslash xy; \quad toc;
\]

- \backslash is also more accurate, see purpose of inv
- Example: funWithInv.m
Array Functions

Arrays of Constants

1. eye(10); % 10 by 10 identity
2. zeros(3); % 3 by 3 of zeros
3. zeros(2,3); % 2 by 3 of zeros
4. ones(31,35,69);
5. 1:5; % [1 2 3 4 5]

Vector Functions

1. sum(a,2); % sum along 2nd dimension of x
2. max(a); % max along 1st dimension of a
3. any(a,2); % a(:,1) | a(:,2) | ...
4. all(b); % b(1,:) & b(2,:) & ...
5. cumprod(a); % cumulative product
More Functions

- All standard mathematical functions – linear algebra, special functions, polynomials, etc
- Manipulating arrays – sort, permute, find, set operations
- Strings – regexp, findstr, etc
- Use the Matlab Function Reference
Flow Control

```matlab
if (j==3)
  % ... some commands ...
elseif (j>4)
  % ... some other commands ...
else
  % ... some other commands ...
end

for j=lo:hi
  x(j) = sqrt(j);
end

epsilon = 1;
while (1-epsilon ~= 1)
  epsilon = epsilon*0.99
end
```
Warning – Arrays and Flow Control

A = [1 2 3]; B = A; C = [1 2 2];
if A==B
  fprintf('A==B\n');
end

if A==C % what message will be printed?
  fprintf('A==C\n');
elseif A~=C
  fprintf('A~==C\n');
else
  fprintf('~(A==C) && ~(A~==C) !?\n');
end

Output
1. \( A = \text{magic}(4); \) % 4 by 4 magic matrix
2. \( A(2,3); \) % by subscript
3. \( A(5); \) % by linear index — \( A(5) = A(1,2) \)
4. \( \text{ind2sub}(\text{size}(A),5); \) % convert linear index to subscripts
5. \( \text{bigA} = A>10; \) % logical 4 by 4 matrix
6. \( A(\text{bigA}); \) % vector of elements of \( A > 4 \), in order of linear index
Array Subscripting

1. A = eye(2);
2. B = rand(3,2,2);
3. A(1,:)  \% [1 0]
4. A(:,2)  \% [0; 1]
5. try
6.   B(1,:,:)+A;  \% not allowed
7. catch
8.   squeeze(B(1,:,:))+A;
9. end
10. B(1);  \% = B(1,1,1)
11. A(3);  \% = A(1,2) – matrices stored columnwise
12. B(2,A==1);  \% [B(2,1,1) B(2,2,2)]
Structures

- Way of organizing related data
- Create a structure, `s`, with fields, `x`, `y`, and `name`

```matlab
s.y = 1;
s.x = [1 1];
s.name = 'foo';
% or equivalently
s2 = struct('y',1,'x',[1 1],'name','foo');
```

- Use the fields like normal variables
- Can create arrays of structures

```matlab
for i=10:(-1):1
    s(i).y = rand();
    s(i).x = [i:i+2];
    s(i).name = sprintf('name%d',i);
end
```
Structures

- Structure array → normal array

```
1  % slow, explicit way
2  for i=1:length(s)
3    X(:,i) = s(i).x;
4  end
5  % equivalent fast way
6  X = [s.x]; % rationale: s.x is a comma separated list
```

- Test for equality

```
1  isequal(s1,s2); % works for any s1, s2
```
Structures

- Get a list of fields

```matlab
f = fieldnames(s);  % creates cell array containing names of s
```

- Dynamic field reference:

```matlab
s.x       % a static reference to s.x
s.('x')   % dynamic reference to s.x
```
Structures

- Loop over fields

```matlab
f = fields(s); % fields() equivalent to fieldnames()
for i=1:length(f)
    doSomething(s.{f{i}}); % do something to each field
end
% equivalently,
for f=fields(s)'
    doSomething(s.(char(f)));
end
% most compact
structfun(@doSomething,s);
```
Cell Arrays

- Cell arrays can have entries of arbitrary datatype

```matlab
1. a = cell(3,2); % create 3 by 2 cell array
2. a{1,1} = 1;
3. a{3,1} = 'hello';
4. a{2,2} = randn(100,100);
```

- Useful for strings and avoiding `squeeze()`
- Using cell arrays with other data types can be tricky
  - Indexing with () gives elements of cell arrays, which are themselves cells
  - Indexing with {} converts elements of cell arrays to their underlying type, returns comma separated list if not singleton

```matlab
1. a = {[1 2], 3}; % create 2 by 1 cell array
2. y = a{1}; % y is 1 by 2 numeric array
3. ycell = a(1); % is 1 by 1 cell array
4. x = y+1; % allowed
5. xcell = ycell+1; % not allowed
6. onetwothree = [a{1:2}]; % = [1 2 3]
```
Commenting

- Comments are anything after a `%` or a ...
- Special comments:
  - First contiguous block of comments in an m-file are that file’s help
    - `% See also FUNCTION` creates clickable link to help for function.m
    - Always include: a description of what the function does, what inputs are expected, and what kind of output will be produced
  - Code “cells” are delimited by `%% Cell title`
    - Matlab editor has special abilities for working with cells
    - `publish('file.m')` runs file.m and makes nice output

```matlab
1   % publish all m-files in current directory
2   files = dir('*.m');
3   cellfun(@(x) publish(x,struct('evalCode',false)), ...
4       {files.name},'UniformOutput',false);
```
Nobody writes a program correctly the first time

A debugger lets you pause your program at an arbitrary point and examine its state

Debugging lingo:
- breakpoint = a place where the debugger stops
- stack = sequence of functions that lead to the current point; up the stack = to caller; down to the stack = to callee
- step = execute one line of code; step in = execute next line of code, move down the stack if a new frame is added; step out = execute until current frame exits
- continue = execute until the next breakpoint
Matlab Debugging

- Buttons at top of editor – set/clear break points, step, continue
- More under Debug menu or from the command line:
  - Set breakpoints

```matlab
1  dbstop in mfile at 33 % set break point at line 33 of mfile
2  dbstop in mfile at func % stop in func() in mfile
3  dbstop if error % enter debugger if error encountered
4  dbstop if warning
5  dbstop if naninf
```

- `dbstack` prints the stack
- `dbup` and `dbdown` move up and down the stack
- `mlint file` analyzes file.m for potential errors and inefficiencies, messages also shown on right edge of editor

```matlab
1  for i=1:10
2    x(i) = i; %#ok (tells mlint to ignore this line)
3  end
```
Profiling

- Display how much time each part of a program takes
- Use to identify bottlenecks
  - Try to eliminate them
- Could also be useful for debugging – shows exactly what lines were executed and how often
Matlab Profiler

- `profile on` makes the profiler start collecting information
- `profile viewer` shows the results
- Very nice and easy to use
Example: Diffs in Diffs Simulation

- From 382: recreate and extend simulations from Betrand, Duflo, and Mullanaithan (2004)
- Illustrates:
  - Importing data
  - Lots of subscripting
  - Use of structures
  - Random numbers
  - Comments and publishing

Code
Exercises

1. Take a simple program that you have written in another language and rewrite it in Matlab.

2. Taken from the art of Matlab blog: “Q: Suppose there is a multiple-choice quiz, and for each question, one of the responses scores 0 points, one scores 3 points, one scores 5 points, one scores 8 points, and one scores 10 points. If the quiz has 4 questions, and assuming that each taker answers all of the questions, then which totals per taker are not possible? For example, it would not be possible to finish the quiz with a total score of 2. If the quiz had 7 questions? Can you generalize the code so that the number of questions can be varied by varying a single assignment?”

3. Write a collection of Matlab functions for linear regression. You could include OLS, GLS, SUR, IV, 3SLS, etc.