FAME: Applied Econometrics
Lecture Note #1

Michael Rockinger

August 26, 2004

1 Introduction

Computer programs: -Matlab/Gauss
-Eviews/Rats
-S+/R
-Stata/Sas
-Ad hoc modules - extreme value theory
-Lesage Econometric toolbox
http://www.spatial-econometrics.com/
- C/C++ (for instance to write DLLs, Bayesian analysis)

Historical background (Eispack/Linpack - now Lapack)

2 Matlab

2.1 Command window

```
format long
format short
format bank        % two decimals

get(0,'Format')    % object oriented programming
clc
clear
```

clear name

help

ver

whos

diary

2.2 Editor

2.2.1 basic matrix instructions

learning1.m

```
>>
echo on %also write intstructions on screen
A=[ 1 2 3; 4 5 6] % creates a Matrix
A =
 1.00 2.00 3.00
 4.00 5.00 6.00
b=[88; 99]; % notice the difference due to ;
disp('a column vector');
a column vector
b
b =
 88.00
 99.00
fprintf('another way to print b but using formatted output.  b=%12.4f\n',b);
another way to print b but using formatted output.  b= 88.0000
another way to print b but using formatted output.  b= 99.0000
disp('what is the precision of Matlab');
what is the precision of Matlab
fprintf('pi=%25.20f',pi);
pi= 3.14159265358979310000
A(:,1) % the first column of A
ans =
```

```
 1.00
 4.00
A(2,:)  % the second row of A
ans =
 4.00 5.00 6.00
A(1,3) % an element out of A
ans =
 3.00
A(2,2:3)=[-2 -3]; % replace blocks in A
A
A =
 1.00 2.00 3.00
 4.00 -2.00 -3.00
B=[1 7 3 ; 5 6 7] % another matrix
B =
 1.00 7.00 3.00
 5.00 6.00 7.00
B' %transpose of B
ans =
 1.00 5.00
 7.00 6.00
 3.00 7.00
C=B'*A %yet another matrix
C =
 21.00 -8.00 -12.00
 31.00 2.00 3.00
 31.00 -8.00 -12.00
5*C % multiply a matrix by a scalar
ans =
 105.00 -40.00 -60.00
 155.00 10.00 15.00
 155.00 -40.00 -60.00
D=[A;B] % vertical concatenation (combine vertically A and B)
```

```
D =
 1.00 2.00 3.00
 4.00 -2.00 -3.00
 1.00 7.00 3.00
 5.00 6.00 7.00
D=[A B] % horizontal concatenation
D =
 Columns 1 through 4
 1.00 2.00 3.00 1.00
 4.00 -2.00 -3.00 5.00
 Columns 5 through 6
 7.00 3.00
 6.00 7.00
A.*B % Haddamar product (element by element product)
ans =
 1.00 14.00 9.00
 20.00 -12.00 -21.00
E=[] % an empty matrix
E =
 []
E=[E;A;A] %concatenate matrices (useful to store results in E)
E =
 1.00 2.00 3.00
 4.00 -2.00 -3.00
 1.00 2.00 3.00
 4.00 -2.00 -3.00
A+B %sum of matrices
ans =
 2.00 9.00 6.00
 9.00 4.00 4.00
A-B % difference of matrices
ans =
 0 -5.00 0
```

```
 -1.00 -8.00 -10.00
a=A(1,:)
a =
 1.00 2.00 3.00
b=B(:,2)
b =
 7.00
 6.00
[r,c]=size(A) %rows and columns of A
r =
 2.00
c =
 3.00
size(A,1) % row dimension of A
ans =
 2.00
size(A,2) % column dimension of A
ans =
 3.00
% notice:  a.*b would return a mistake
kron(a,b)
ans =
 7.00 14.00 21.00
 6.00 12.00 18.00
clear a
%just typing a returns an error
A=1:3 % creates a sequence for A. Notice A is a row!!!
A =
 1.00 2.00 3.00
A(end)
ans =
 3.00
A(end-1)
```

```
ans =
 2.00
B='I am a string of characters'
B =
I am a string of characters
C='; lots of them'
C =
; lots of them
D=[B C] % horizontal concatenation
D =
I am a string of characters; lots of them
S=char(128) % the Euro
S =
€
D=[D ' ' S]
D =
I am a string of characters; lots of them €
E=strvcat(B,'Yes') % vertical concatenation.  Notice padding
E =
I am a string of characters
Yes
disp(E)
I am a string of characters
Yes
E=strcat(B,' No') % equivalent to [B ' No'].  Emphasis on string operation
E =
I am a string of characters No
double('A') % to go from char to integer
ans =
 65.00
double('a')
ans =
 97.00
```

```
x=randn(3,1) % generates normal random numbers
x =
 0.53
 0.22
 -0.92
str1 = num2str(min(x)) % a first string
str1 =
-0.9219
str2 = num2str(max(x)) % another one
str2 =
0.52874
disp(str1); %only strings can get displayed with disp
-0.9219
['from smallest ' str1 ' to largest ' str2]
ans =
from smallest -0.9219 to largest 0.52874
s=num2str(min(x),12) % more decimals
s =
-0.921901624356
['now display 12 decimals ',s]
ans =
now display 12 decimals -0.921901624356
str = '12.3389e-1';
val = str2num(str) % goes from string to real
val =
 1.23
val+12
ans =
 13.23
% do not confuse with cells.  These are structures.
% cells can be useful for collecting output from a function
A={[1; 2] 'I scream for ice cream' }
A =
```

```
 [2x1 double] [1x22 char]
A{1,1}
ans =
 1.00
 2.00
A{1,2}
ans =
I scream for ice cream
```

>>

Comments: %

Lines that are too: long use three points and continue on the next line ...

2.2.2 some math functions

learning2.m

```
A=[1 2 3; 1 2 3]
A =
 1 2 3
 1 2 3
rank(A) %rank of A
ans =
 1
B=[1 2; 3 4]
B =
 1 2
 3 4
rank(B)
ans =
 2
det(B) %determinant
ans =
 -2
trace(B) %trace
ans =
```

```
 5
[V,D] = eig(B) %eigen vectors and eigen values
V =
 -0.82456484013239 -0.41597355791928
 0.56576746496899 -0.90937670913212
D =
 -0.37228132326901 0
 0 5.37228132326901
[B*V(:,1) D(1,1)*V(:,1)] %verification that B v_1= lda_1 v_1
ans =
 0.30697008980559 0.30697008980559
 -0.21062466052121 -0.21062466052121
inv(B)
ans =
 -2.00000000000000 1.00000000000000
 1.50000000000000 -0.50000000000000
c=[3; 4]
c =
 3
 4
inv(B)*c % one way to compute
ans =
 -2.00000000000000
 2.50000000000000
% a much better way to go (for numerical precision)
B\c
ans =
 -2.00000000000000
 2.50000000000000
E=eye(2)
E =
 1 0
 0 1
```

```
% if one needs to invert B use (faster + more precise)
B\E
ans =
 -2.00000000000000 1.00000000000000
 1.50000000000000 -0.50000000000000
% condition number (ratio of largest to smallest eigenvalue)
% if larger than 30'000 be careful when invering matrix
cond(B)
ans =
 14.93303437365927
C=[1 2; 0.9999 2.0001]
C =
 1.00000000000000 2.00000000000000
 0.99990000000000 2.00010000000000
cond(C)
ans =
 3.333400003665147e+004
%Choleski decomposition matrix must be positive definite
B=[1 0.1; 0.1 3]
B =
 1.00000000000000 0.10000000000000
 0.10000000000000 3.00000000000000
X=chol(B)
X =
 1.00000000000000 0.10000000000000
 0 1.72916164657906
X'*X %verification
ans =
 1.00000000000000 0.10000000000000
 0.10000000000000 3.00000000000000
```

The usual set of trigonometric functions works

sin, cos, tan, cot, sinh, cosh, tanh, coth, asin, acos, atan, acot, and arc-hyperbolic functions.

Further functions

name	instruction	name	instruction
Exponential	exp	Round towards plus infinity	ceil
Natural logarithm	log	Round towards nearest integer	round
Base 2 logarithm	log2	Modulus	mod
(base 10) logarithm	log10	Signum	sign
Square root	sqrt	Prime factors	factor
Absolute value	abs	Factorial function	factorial
Complex conjugate	conj	Greatest common divisor	gcd
Complex number	complex	True for prime numbers	isprime
Phase angle	angle	Least common multiple	lcm
Imaginary unit	i ou j	Choose k among N	nchoosek
Complex imaginary part	imag	All possible permutations	perms
True for real array	isreal	Generate list of prime numbers	primes
Complex real part	real		
Round towards zero	fix		
Round towards minus infinity	floor		

Some generally useful functions (also for elementary statistics)

name	instruction	name	instruction
Cumulative product	cumprod	Product of array elts.	prod
Cumulative sum	cumsum	Sort elements in ascd. order	sort
Cum. trapezoidal num. integration	cumtrapz	Sort rows in ascd. order	sortrows
Maximum elt. of array (also index)	max	Standard deviation	std
Average = mean of arrays	mean	Sum of array elements	sum
Median value of arrays	median	Trapezoidal num. integ.	trapz
Minimum elt. of array (also index)	min	Variance	var
Differences	diff	Numerical gradient	gradient
Correlation coefficients	Correlation	Covariance matrix	cov

2.2.3 Mathematical operators

```
+ - * /     %addition, subtraction, multiplication, division
  ' % ! ^   %transpose, moduls division, factional, exponentiation
```

2.2.4 Relational operators

==	~=	>	<	>=	<=
equal	not equal	greater than	less than	greater or equal	less or equal

A=5 is not the same as A==5;

2.2.5 Logical operators

& AND

| OR

~ NON

xor

Sometimes a true condition corresponds to 1. Matlab does not always adhere to this because of rounding error.

2.3 Programming

Edit a filename: Ex. gmm1.m

good habit→	*.m	program file
	*.txt	data file
	*.dat	data file
	*.asc	data file
	*.src	source codes

further good habits→ i,j,k,l → index variables

small →letters scalar

capital → letters matrices

notice that a and A are not the same thing for Matlab

_A underscore as a first letter is not allowed

3A not allowed. A3=2. This is allowed

2.3.1 A first 'official' program to illustrate input/output

Create a database in Excel. Save it as an ascii file my2.txt. Use tabulation as operator.

1 6.7 2.33

2	8.9	1.25
3	3.3	8.7
4	8.7	9.3
5	2.3	10.33
6	2.99	15

Matlab is as good (or as bad) as other programs at reading files. If you have an Excel file, then the easiest is to use something like

`A = xlsread('filename').`

The following program performs some basic input/output.

```
function learning3()
% learning3.m
% a first matlab function
% loads some ascii data,
% creates with that data a Matlab dataset.  Loads data and displays it
%
clc;
fid = fopen('d:\\rocky\\my2.txt','r')
[A,T] = fscanf(fid,'%f %f %f'); % notice:  no missing values allowed
fclose(fid);

format short;
A(1:3)
A(end-2:end)
disp('T');
T
x=reshape(A,3,T/3)' %create 3 rows then transpose Each unit has T/3 elements
x

% now writes data into a new file
disp('Save data into file');
save 'd:\finbox\my2' x;
whos
```

```
clear;
disp('whos after clear')
whos
load 'd:\finbox\my2' x
x

% save as an ascii file
fid = fopen('d:\\finbox\\my2.txt','w');
fprintf(fid,'%5.0f %8.3f %8.3f \n',x); %don't forget the \n
% notice \n is readable under word but not under wordpad...
fclose(fid);
```

Running this file produces

```
fid =
 3
ans =
 1.0000
 6.7000
 2.3300
ans =
 6.0000
 2.9900
 15.0000
T
T =
 18
x =
 1.0000 6.7000 2.3300
 2.0000 8.9000 1.2500
 3.0000 3.3000 8.7000
 4.0000 8.7000 9.3000
 5.0000 2.3000 10.3000
 6.0000 2.9900 15.0000
Save data into file
```

```
 Name Size Bytes Class
 A 18x1 144 double array
 T 1x1 8 double array
 ans 3x1 24 double array
 fid 1x1 8 double array
 x 6x3 144 double array
Grand total is 41 elements using 328 bytes
```

```
whos after clear
```

```
x =
 1.0000 6.7000 2.3300
 2.0000 8.9000 1.2500
 3.0000 3.3000 8.7000
 4.0000 8.7000 9.3000
 5.0000 2.3000 10.3000
 6.0000 2.9900 15.0000
```

2.3.2 Conditional branching

```
for m = 1:k
    for n = 1:k
        if m == n
            a(m,n) = 2;
        elseif abs(m-n) == 2
            a(m,n) = 1;
        else
            a(m,n) = 0;
        end
    end
end
```

For k=5 you get the matrix

```
a =
 2 0 1 0 0
 0 2 0 1 0
```

```
1 0 2 0 1
0 1 0 2 0
0 0 1 0 2
```

2.3.3 A really small test

If condition

do something;

end

notice the (non)-use of ;

Tests may be nested.

2.3.4 While/Until loops

while condition

do something;

end

Illustration

WhileEx.m

```
i=1;
while i<10
 [i i^2]
 i=i+1;
end
Res=[];
A=zeros(10,1);
for x=1:0.1:2
 Res=[Res; [x x^2]]; % only ok for short series else pre-declare size
 A(floor(x*10),:)=x
end
Res
disp('A');
A
```

Remark: Until (of some other languages) can always be recoded as while⇔ Matlab only implements while.

The loop is tested before it is executed → loop may not be executed at all.

if condition depends on something affected within loop, the programmer must ensure that he does not forget "something". This is a typical source of error getting your program to run for ever.

2.3.5 For loops

loops.m

```
k=3; Res=[];
for m = 1:k
    Res=[Res;[m m^2]];
end
```

produces

```
Res =
 1 1
 2 4
 3 9
```

→It's much faster than while

→Requires that number of steps are known

The instruction break may be used to get out of a for or a while loop.

2.3.6 Function evaluations

involvs **fcall.m** and **do_more_Comput.m**

```
function fcall()
 % illustrates how to nest functions and local/globalness of variables
 global c
 a=2
 b=3
 c=10

 [x,y]=do_Comput(a,b) %here involves two elements that are returned

 function [f,g]=do_Comput(a,b);
 global c
```

```
 f=a^2+c;
 g=a*b*c;
```

this produces

```
a =
 2
b =
 3
c =
 10
x =
 14
y =
 60
```

Sometimes you wish to develop a general program such as an optimizer (see a bit later) or a module that is very general and that you wish to run on many functions (think about a Max Lik program). In that case, you have to provide the name of a function. This can be achieved via `@function_name` at the level of the calling program and via `feval(function_name)` at the function level.

showeval.m

```
function showeval()
% illustrates how 'feval' operates
% here compute gradients for two trivial functions at trivial points
x=1.0;
grad(@f,x) % evaluates grandient of f at x
grad(@g,x)
function y=f(x);
y=x^2;
function y=g(x)
y=x^3;
function gr=grad(fnam,x)
h=0.0000001;
f1=feval(fnam,x);
f2=feval(fnam,x+h);
```

```
gr=(f2-f1)/h
```

2.4 Using the optimizer

Optim_Ilustrate.m

```
function Optim_Ilustrate()
% illustrates how to use the optimization package
clc;
options = optimset('MaxFunEvals', 1000, ...
'TolFun', 1e-5,...
 'TolX', 1e-5,...
'LargeScale','off',...
'Diagnostics','on',...
'Display','iter');
beta0=[1 2]; % starting values
A=3;
[beta,Qmin,exitflag] = fminunc(@obj_func,beta0,options,A);
disp('the optimum is');
beta
disp('whereas it should be ');
[A;-1]
function z=obj_func(beta,A);
x=beta(1);
y=beta(2);
z=(x-A)^2+(y+1)^2;
```

running this program produces:

```
> In C:\MATLAB6p1\toolbox\optim\private\diagnose.m at line 20
 In C:\MATLAB6p1\toolbox\optim\fminunc.m at line 221
 In D:\finbox\Learning4.m at line 12
Number of variables:  2
Functions
 Objective:  obj_func
 Gradient:  finite-differencing
```

```
 Hessian:  finite-differencing (or Quasi-Newton)

Algorithm selected
 medium-scale:  Quasi-Newton line search
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 End diagnostic information
 Directional
 Iteration Func-count f(x) Step-size derivative
 1 2 13 0.5 -52
 2 8 3.22214e-017 0.5 2.62e-009

Optimization terminated successfully:
 Search direction less than 2*options.TolX
the optimum is
beta =
 3.0000 -1.0000
whereas it should be
ans =
 3
 -1
```

Notice: a function can only be nested within a function

Compilation mcc -x filename

2.4.1 Efficient programming

Matlab procedures which are rationalized exist for an incredible number of situations.

- these procedures are pre-tested. Keeps your programming fast
- procedures are vectorized. They are much faster than loops for instance.
- Time your loops using tic toc
- Always put the largest loop as most innermost loop.
- Declare repeated operations outside loop (for instance if you need to compute a loop involving i*gamma(3) where i is some increment, then it would be too costly to

evaluate in each loop the gamma function. Compute out of the loop g=gamma(3) and use in the loop i*g.

Declare place holders rather than using concatenation. For instance use

```
nh=n/2;
np1h=(n+1)/2;
y=zeros(T,1);
j=1;
while j<=T % notice, here there is not need for a ;
    y(j)=gamma(nh^j)/gamma(np1h^j)*x(j); % x defined elsewhere
    j=j+1;
end %again no need for a ;
```

which is much faster than

```
j=1;
y=[];
while j<=T
    y=[y; (gamma(n/2)^j)/gamma(((n+1)/2)^j)*x(j)];
    j=j+1;
end
```

Other example

speed.m

3 The graphics module

3.1 Basic graphs (and more general programming)

The following program is contained in **ElemStats.m**

```
>>
echo on
u=randn(5,2) % 5 uniform random numbers in two columns
u =
 -0.0132 1.7701
 -0.5803 0.3255
 2.1363 -1.1190
```

```
 -0.2576 0.6204
 -1.4095 1.2698
u(:,2)=u(:,1)+0.3*u(:,2)
u =
 -0.0132 0.5179
 -0.5803 -0.4826
 2.1363 1.8006
 -0.2576 -0.0715
 -1.4095 -1.0286
plot(u(:,1),u(:,2),'s',...
 'MarkerEdgeColor','k',...
 'MarkerFaceColor','g',...
 'MarkerSize',10)
title('scatterplot of u(:,1) against u(:,2)');
% in the plot instruction 's' means symbols="Marker"
[ma,ma_idx]=max(u)
ma =
 2.1363 1.8006
ma_idx =
 3 3
max(u) % notice that Matlab functions are overloaded
ans =
 2.1363 1.8006
[mi,mi_idx]=min(u)
mi =
 -1.4095 -1.0286
mi_idx =
 5 5
mu=mean(u)
mu =
 -0.0249 0.1471
sum(u)/size(u,1)
ans =
```

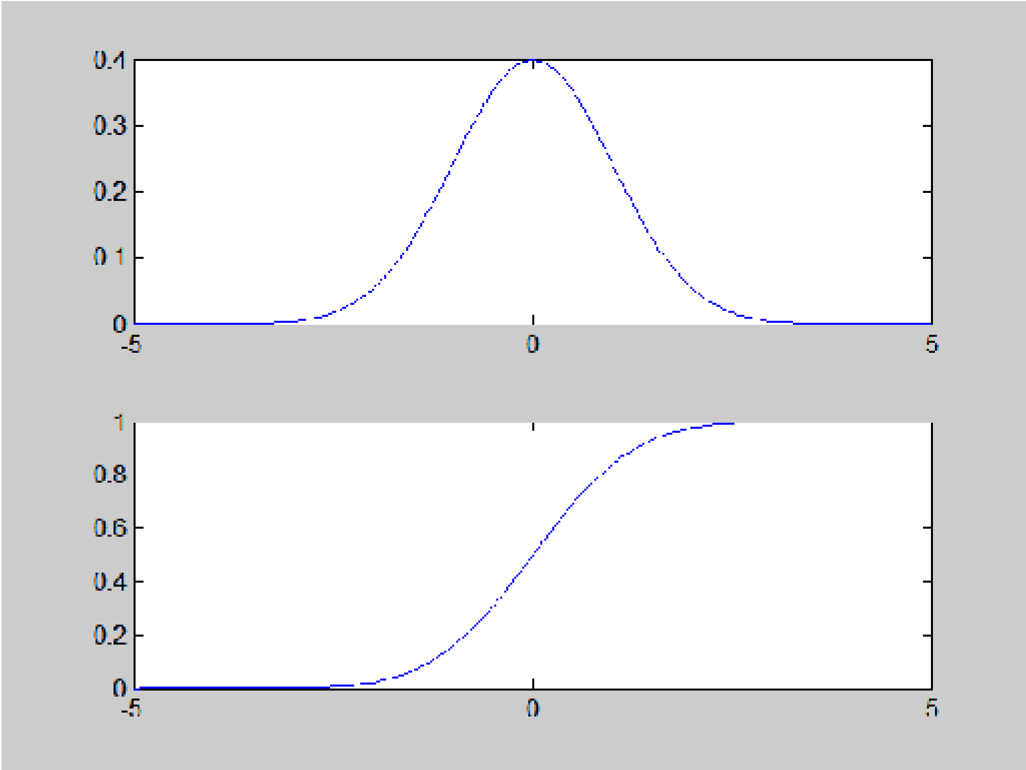
```
 -0.0249 0.1471
median(u)
ans =
 -0.2576 -0.0715
sort(u)
ans =
 -1.4095 -1.0286
 -0.5803 -0.4826
 -0.2576 -0.0715
 -0.0132 0.5179
 2.1363 1.8006
sortrows(u)
ans =
 -1.4095 -1.0286
 -0.5803 -0.4826
 -0.2576 -0.0715
 -0.0132 0.5179
 2.1363 1.8006
uu = u - kron( mu, ones(size(u,1),1)); %this is truly stupid
mean( uu.^2 )
ans =
 1.3901 0.9396
T=size(u,1);
mean(uu.^2)*(T-1)/T
ans =
 1.1121 0.7517
v=var(u)
v =
 1.7377 1.1745
sqrt(v)
ans =
 1.3182 1.0838
std(u)
```

```
ans =
 1.3182 1.0838
mu=0.1;
sig=0.2;
S0=100;
x=mu+sig*u;
x1=exp(x);
x1=[ones(1,2); x1]; % first observation
St1=100*cumprod(x1)
St1 =
 100.0000 100.0000
 110.2262 122.5772
 108.4709 123.0045
 183.7808 194.8715
 192.9093 212.3079
 160.8246 191.0087
x=[zeros(1,2); x]; % again for first observation
St2=100*exp(cumsum(x))
St2 =
 100.0000 100.0000
 110.2262 122.5772
 108.4709 123.0045
 183.7808 194.8715
 192.9093 212.3079
 160.8246 191.0087
A=[1 2 3; 2 2 4; 2 1 2]
A =
 1 2 3
 2 2 4
 2 1 2
prod(A)
ans =
 4 4 24
```

```
cumprod(A)
ans =
 1 2 3
 2 4 12
 4 4 24
dx=0.1
dx =
 0.1000
x=-5:dx:5; % some fine grid
y=1/sqrt(2*pi)*exp(-0.5*x.^2);
figure(1)
subplot(2,1,1)
plot(x,y);
cdf=cumtrapz(y)*dx;
subplot(2,1,2)
plot(x,cdf);
trapz(y)*dx
ans =
 1.0000
```

3.2 Another graphical example

```
%Graphcapa.m
% illustrates graphical capabilities
x=randn(10,1);
plot(x,'--+'); % plots x against time
axis([0 13 -3 3]);
title('a nice simulation')
xlabel('time goes by')
ylabel('the realizations')
legend('name of curve')
text(8,-2,'Right lower corner');
```



3.3 More advanced issues

clc clears command window

clf clears current graphics window

hold on / hold off to superpose plots

subplot(m,n,p) splits graphics window into m rows and n celles. p corresponds to the number of subplot.

how to glue a figure into word-processor

Object oriented stuff: get/set

get try `get(0)` in command window

```
% plot2.m again a plot
%
x=randn(10,1);
y=1+0.2*randn(10,1);
plot(x,y)
get(gcf,'Position') %gcf is generic graphics window handle
```