**Stsm properties**

%=========================================================================

%

% Program to identify properties of US macro variables

%

%=========================================================================

%

%--------------------------- Functions ----------------------------------

%

%--------------------------------------------------------------------------

% ACF function based on running a sequence of OLS regressions

%--------------------------------------------------------------------------

load('sims\_data.mat')

%% As the function does not have the same name as the file itself, %

%it will not run. It will work as a property for the simulation code. You

%can call the function from simulation but the function itself will not

%execute.

function acorr = acf(y,lags)

acorr = zeros(lags,1);

t = length(y);

for i = 1:lags

y0 = trimr(y,i,0);

x = [ ones(t-i,1) trimr(y,0,i)];

b = inv(x'\*x)\*x'\*y0;

acorr(i) = b(2);

end

end

%--------------------------------------------------------------------------

% PACF function based on running a sequence of OLS regressions

%--------------------------------------------------------------------------

function pcorr = pacf(y,lags)

pcorr = zeros(lags,1);

t = length(y);

x = ones(t,1);

for i = 1:lags

y0 = trimr(y,i,0);

x = [trimr(x,1,0) trimr(y,0,i)];

b = inv(x'\*x)\*x'\*y0;

pcorr(i) = b(i+1);

end

end

stsm simulate

%=========================================================================

%

% Simulate an ARMA(2,2) model and compute the ACF and the PACF

%

%==========================================================================

function stsm\_simulate( )

clear all

clc

RandStream.setGlobalStream( RandStream('mt19937ar','seed',12) );

t = 200; % Define the sample size

lags = 10; % Number of lags in the ACF and the PACF

% Generate error process

% Step #1: we are taking an additional 100 observation. We are making a column of

% random variable of 301 rows and 1 column (200+101). It is advised in the

% exercise that we should simulate t+100 observations and then exclude the

% first 100 observations to avoid initialisation problem. so, (n+1)+100=T

% t+101.

ut = sqrt(0.1)\*randn(t+101,1); %An additional 100 observations

% ARMA(0,0)

mue = 0.0;

phi1 = 0.0;

phi2 = 0.0;

si1 = 0.0;

si2 = 0.0;

% Recserar: an autoregressive recursive series. autocorrelation function. The AR parameters are determined by the first

% p+1 elements of the autocorrelation function. The full autocorrelation

% function can then be derived by recursively calculating.The idea of

% recursive sequences in which later terms are deduced from earlier ones.

% Fibonacci retracement can be an example.For a sequence a1, a2, a3, . . . , an, . . . a recursive formula is a formula that requires the computation of all previous terms in order to find the value of an .

%Note: Recursion is an example of an iterative procedure.

% So to run autoregression we are using recserar. For computations of

% Fibonacci or similar autoregressive data calculation we should always use

% recserar. In the first stage we are trimming the first two rows of ut.

% Which gives it a 299 rows (number of additional p-1 th row, condition for

% recursive estimation, from the earlier 100+1+1 row. For the sake of

% iteration the next part of equation has first and the last row of the

% column omitted. Than si2 has last two rows omitted. Now look at the

% equation itself. It follows y = recserar(x,y0,a) structure where, % Inputs:

% x = a matrix of dimensions (n,k) (exogenous variables)

% y0 = a matrix of dimensions (p,k) (starting values)

% a = a matrix of dimensions (p,k) (lag parameters)

%

% Mimics the Gauss routine. So the first matrix gives you exogenous

% variables with autoregreeive recursion or iteration. Starting values

% are 0. Phi are the lag parameters as conditioned in basic equation. the omission follows 0 to 2 bevause we are calculating an ARMA(2,2)

% model...nothing else...if it was ARMA (3,3), the trimming would have

% followed 0 to 3 and we would have added upto si3.

yt = recserar( mue + trimr(ut,2,0) + si1\*trimr(ut,1,1) ...

+ si2\*trimr(ut,0,2) , [ 0.0; 0.0 ] , [ phi1;phi2 ] );

% Step 2: (Avoiding initialisation problem)- the following y0t excludes the

%first 100 rows and give you the latest values or the new values only in

% a new (n-1)=t=199 row matrix.

%%This is the cycle with new data.And so acf and pacf is applied on new 199 rows, not the earlier ones.

y0t = trimr(yt,100,0);

acf0 = acf( y0t,lags );

pacf0 = pacf( y0t,lags );

% ARMA(1,0)

mue = 0.0;

phi1 = 0.7;

phi2 = 0.0;

si1 = 0.0;

si2 = 0.0;

% this one is also same as earlier. we just have a new phi1 that changes

% the entire result.

yt = recserar( mue + trimr(ut,2,0) + si1\*trimr(ut,1,1) ...

+ si2\*trimr(ut,0,2) , [ 0.0; 0.0 ] , [ phi1;phi2 ] );

y1t = trimr(yt,100,0); % Trim the data

acf1 = acf( y1t,lags ); % Compute the ACF

pacf1 = pacf( y1t,lags ); % Compute the PACF

% ARMA(2,0)

mue = 0.0;

phi1 = 0.7;

phi2 = -0.5;

si1 = 0.0;

si2 = 0.0;

% this one is also same as earlier. we just have two new phis that changes

% the entire result.

yt = recserar( mue + trimr(ut,2,0) + si1\*trimr(ut,1,1) ...

+ si2\*trimr(ut,0,2) , [ 0.0; 0.0 ] , [ phi1;phi2 ] );

y2t = trimr(yt,100,0); % Trim the data

acf2 = acf( y2t,lags ); % Compute the ACF

pacf2 = pacf( y2t,lags ); % Compute the PACF

% this one is also same as earlier. we just have a new si1 that changes

% the entire result.

% ARMA(0,1)

mue = 0.0;

phi1 = 0.0;

phi2 = 0.0;

si1 = 0.9;

si2 = 0.0;

yt = recserar( mue + trimr(ut,2,0) + si1\*trimr(ut,1,1) ...

+ si2\*trimr(ut,0,2) , [ 0.0; 0.0 ] , [ phi1;phi2 ] );

y3t = trimr(yt,100,0);

acf3 = acf(y3t,lags); % Compute the ACF

pacf3 = pacf(y3t,lags); % Compute the PACF

% this one is also same as earlier. we just have two new si's that changes

% the entire result.

% ARMA(0,2)

mue = 0.0;

phi1 = 0.0;

phi2 = 0.0;

si1 = -0.2;

si2 = 0.7;

yt = recserar( mue + trimr(ut,2,0) + si1\*trimr(ut,1,1) ...

+ si2\*trimr(ut,0,2) , [ 0.0; 0.0 ] , [ phi1;phi2 ] );

y4t = trimr(yt,100,0);

acf4 = acf(y4t,lags); % Compute the ACF

pacf4 = pacf(y4t,lags); % Compute the PACF

% ARMA(1,1)

mue = 0.0;

phi1 = 0.8;

phi2 = 0.0;

si1 = 0.7;

si2 = 0.0;

yt = recserar( mue + trimr(ut,2,0) + si1\*trimr(ut,1,1) ...

+ si2\*trimr(ut,0,2) , [ 0.0; 0.0 ] , [ phi1;phi2 ] );

y5t = trimr(yt,100,0);

acf5 = acf(y5t,lags); % Compute the ACF

pacf5 = pacf(y5t,lags); % Compute the PACF

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*\*

%\*\*\* Plot the series

%\*\*\*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Switch off TeX interpreter and clear figure

set(0,'defaulttextinterpreter','none');

figure(1);

clf;

t= 0:lags;

%--------------------------------------------------------%

% Panel (a)

subplot(3,3,1);

plot(y2t,'-k');

title('(a) ARMA(2,0)');

%ylabel('$y\_t$');

%xlabel('$t$');

axis tight

box off;

%--------------------------------------------------------%

% Panel (b)

subplot(3,3,2);

bar(t,[1; acf2]);

title('(b) ACF ARMA(2,0)');

%ylabel('ACF');

%xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

% Panel (c)

% Panel (c)

%subplot(m,n,p) divides the current figure into an m-by-n grid and creates an axes for a subplot in the position specified by p. MATLAB® numbers its subplots by row, such that the first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. If the axes already exists, then the command subplot(m,n,p) makes the subplot in position p the current axes.

subplot(3,3,3);

%bar (x,y).bar(x,y) draws the bars at the locations specified by x.,

bar(t,[1; pacf2]);

title('(b) PACF ARMA(2,0)');

%ylabel('PACF');

%xlabel('lag');

%ax = gca returns the handle to the current axes for the current figure. If an axes does not exist, then gca creates an axes and returns its handle. You can use the axes handle to query and modify axes properties. For more information, see Axes Properties.

%Set the font size, tick direction, tick length, and y-axis limits for the current axes. Use gca to refer to the current axes. Starting in R2014b, you can use dot notation to set properties.

%If you are using an earlier release, use the set function instead.

%findobj returns handles of the root object and all its descendants without assigning the result to a variable.

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

%--------------------------------------------------------%

% Panel (d)

subplot(3,3,4);

plot(y4t,'-k');

title('(a) ARMA(0,2)');

%ylabel('$y\_t$');

%xlabel('$t$');

axis tight

box off;

%--------------------------------------------------------%

% Panel (e)

subplot(3,3,5);

bar(t,[1; acf4]);

title('(b) ACF ARMA(0,2)');

%ylabel('ACF');

%xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

% Panel (f)

subplot(3,3,6);

bar(t,[1; pacf4]);

title('(b) PACF ARMA(0,2)');

%ylabel('PACF');

%xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

%--------------------------------------------------------%

% Panel (g)

subplot(3,3,7);

plot(y5t,'-k');

title('(a) ARMA(1,1)');

%ylabel('$y\_t$');

xlabel('$t$');

axis tight

box off;

%--------------------------------------------------------%

% Panel (h)

subplot(3,3,8);

bar(t,[1; acf5]);

title('(b) ACF ARMA(1,1)');

%ylabel('ACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

% Panel (i)

subplot(3,3,9);

bar(t,[1; pacf5]);

title('(b) PACF ARMA(1,1)');

%ylabel('PACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

laprint(1,'stsmsimulate','options','factory');

figure(2);

clf;

%--------------------------------------------------------%

% Panel (a)

subplot(3,3,1);

plot(y0t,'-k');

title('(a) ARMA(0,0)');

%ylabel('$y\_t$');

xlabel('$t$');

axis tight

box off;

%--------------------------------------------------------%

% Panel (b)

subplot(3,3,2);

bar(t,[1; acf0]);

title('(b) ACF ARMA(0,0)');

%ylabel('ACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

% Panel (c)

subplot(3,3,3);

bar(t,[1; pacf0]);

title('(b) PACF ARMA(0,0)');

%ylabel('PACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

%--------------------------------------------------------%

% Panel (d)

subplot(3,3,4);

plot(y1t,'-k');

title('(a) ARMA(1,0)');

%ylabel('$y\_t$');

xlabel('$t$');

axis tight

box off;

%--------------------------------------------------------%

% Panel (e)

subplot(3,3,5);

bar(t,[1; acf1]);

title('(b) ACF ARMA(1,0)');

%ylabel('ACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

% Panel (f)

subplot(3,3,6);

bar(t,[1; pacf1]);

title('(b) PACF ARMA(1,0)');

%ylabel('PACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

%--------------------------------------------------------%

% Panel (g)

subplot(3,3,7);

plot(y2t,'-k');

title('(a) ARMA(2,0)');

%ylabel('$y\_t$');

xlabel('$t$');

axis tight

box off;

%--------------------------------------------------------%

% Panel (h)

subplot(3,3,8);

bar(t,[1; acf2]);

title('(b) ACF ARMA(2,0)');

%ylabel('ACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

% Panel (i)

subplot(3,3,9);

bar(t,[1; pacf2]);

title('(b) PACF ARMA(2,0)');

%ylabel('PACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%laprint(1,'simrma','options','factory');

%--------------------------------------------------------%

%--------------------------------------------------------%

figure(3)

clf;

% Panel (a)

subplot(3,3,1);

plot(y3t,'-k');

title('(a) ARMA(0,1)');

%ylabel('$y\_t$');

xlabel('$t$');

axis tight

box off;

%--------------------------------------------------------%

% Panel (b)

subplot(3,3,2);

bar(t,[1; acf3]);

title('(b) ACF ARMA(0,1)');

%ylabel('ACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

% Panel (c)

subplot(3,3,3);

bar(t,[1; pacf3]);

title('(b) PACF ARMA(0,1)');

%ylabel('PACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

%--------------------------------------------------------%

% Panel (d)

subplot(3,3,4);

plot(y4t,'-k');

title('(a) ARMA(0,2)');

%ylabel('$y\_t$');

xlabel('$t$');

axis tight

box off;

%--------------------------------------------------------%

% Panel (e)

subplot(3,3,5);

bar(t,[1; acf4]);

title('(b) ACF ARMA(0,2)');

%ylabel('ACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

% Panel (f)

subplot(3,3,6);

bar(t,[1; pacf4]);

title('(b) PACF ARMA(0,2)');

%ylabel('PACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

%--------------------------------------------------------%

% Panel (g)

subplot(3,3,7);

plot(y5t,'-k');

title('(a) ARMA(1,1)');

%ylabel('$y\_t$');

xlabel('$t$');

box off;

%--------------------------------------------------------%

% Panel (h)

subplot(3,3,8);

bar(t,[1; acf5]);

title('(b) ACF ARMA(1,1)');

%ylabel('ACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%--------------------------------------------------------%

% Panel (i)

subplot(3,3,9);

bar(t,[1; pacf5]);

title('(b) PACF ARMA(1,1)');

%ylabel('PACF');

xlabel('lag');

h = findobj(gca,'Type','patch');

set(h,'FaceColor','w','EdgeColor','k')

axis tight

box off;

%laprint(1,'simarma','options','factory');

end

%

%------------------------- Functions -------------------------------------%

%

%-------------------------------------------------------------------------%

% ACF - based on running a sequence of OLS regressions

%-------------------------------------------------------------------------

%%As the function does not have the same name as the file itself,

%%it will not run. It will work as a property for the simulation code. You

%%can call the function from simulation but the function itself will not

%%execute.

function acorr = acf(y,lags)

% y rows and number of lags column%

acorr = zeros(lags,1);

% Number of lags row and one column of zeros %

t = length(y);

for i = 1:lags

% It will start at 1 and will continue iteration upto lag input number%

y0 = trimr(y,i,0);

% trimr will take the y matrix, will omit upto i in from the beginning of row and nothing will be omitted from the end of row. So if there are 199

% This is important...We need to remember if we are going to do matrix multiplication we will multiply with ones matrix always to get continues iteration/ MCM (matrix chain multiplication)(cumulation) until we get to a 2 by 2 matrix from a very big matrix. In case we will do matrix summation we need to use zeroes matrix, not ones to get the cumulative matrix.

%For example

%%x= x1+x2+x3+x4, sum= zero matric, for i=1:4, sum=sum+x!

% It will give you, for i=1, sum=x1, for i=2, sum=x1+x2 and will continue

% until it reaches x4.

% In this step of converting the i by 1 matrix to 2 by 2 matrix (squared matrix)..So now we are converting the i by 1 matrix to i by 2 matrix%%

x = [ ones(t-i,1) trimr(y,0,i)];

xp=x';

% we just multiplied matrix x with its inverse or x'.and we have a squared

% matrix or a two by two matrix

x00 =xp\*x;

% We can only use inv on squared matrix. Remember, while (ix1)x(ix1)'

% will give you a squared matrix by rotating and multiplying rows and

% columns, inv is 1/x only. It will rotate the entire matrix by replacing its rows and columns.

x11 =inv(x00);

x12 = x11\* x';

x13 = x12 \*y0;

% x13 is the desired two by two matrix

b= x13;

% b = inv(x'\*x)\*x'\*y0; This is simply Least square solution for estimation of beta using the least square estimation method. Least square estimation is finding that particular value of parameter that minimizes sum of squared error or minimizes RSE from a linear model. This equation simply gives you , that is the parameter value minimizing RSE. Simple.Page 146 of the 2nd book.

acorr(i) = b(2);

end

end

%-------------------------------------------------------------------------%

% PACF - based on running a sequence of OLS regressions

%-------------------------------------------------------------------------%

function pcorr = pacf(y,lags)

pcorr = zeros(lags,1);

t = length(y);

x = ones(t,1);

for i = 1:lags

y0 = trimr(y,i,0);

% Wow, the following x is same as x in acf..exactly same...here

% we have produced the ones matrix earlier, and than we trim...we

% could simply followedd the acf way or we could use this in acf...

% just remeber, they did t-i in acf and here its just 1 excluded.

% I think to maintain dimension, it should have been trimr(x,i,0).

% it is just x = [ones(t-i,1) trimr(y,0,i)]

x = [trimr(x,1,0) trimr(y,0,i)];

b = inv(x'\*x)\*x'\*y0;

pcorr(i) = b(i+1);

end

end

stsm roots

roots

Polynomial roots

[expand all in page](javascript:void(0);)

The roots function solves polynomial equations of the form p1xn+...+pnx+pn+1=0. Polynomial equations contain a single variable with nonnegative exponents. To find the roots of other types of equations, use [fzero](file:///C:\Program%20Files\MATLAB\R2015a\help\matlab\ref\fzero.html).

**Syntax**

* r = roots(p)[example](file:///C:\Program%20Files\MATLAB\R2015a\help\matlab\ref\roots.html#buo5g2u-1)

**Description**

[example](file:///C:\Program%20Files\MATLAB\R2015a\help\matlab\ref\roots.html#buo5g2u-1)

r = roots([p](file:///C:\Program%20Files\MATLAB\R2015a\help\matlab\ref\roots.html?searchHighlight=roots#inputarg_p)) returns the roots of the polynomial represented by p as a column vector.

p is a vector containing n+1 polynomial coefficients, starting with the coefficient of xn. A coefficient of 0 indicates an intermediate power that is not present in the equation. For example, p = [3 2 -2] represents the polynomial 3x2+2x−2.

**Examples**

[collapse all](javascript:void(0);)

[Roots of Quadratic Polynomial](javascript:void(0);)

[Roots of Quartic Polynomial](javascript:void(0);)

Solve the equation $x^4 - 1 = 0$.

Create a vector to represent the polynomial, then find the roots.

p = [1 0 0 0 -1];

r = roots(p)

r =

-1.0000 + 0.0000i

0.0000 + 1.0000i

0.0000 - 1.0000i

1.0000 + 0.0000i

**Input Arguments**

[expand all](javascript:void(0);)

[p — Polynomial coefficients](javascript:void(0);)vector

**More About**

[collapse all](javascript:void(0);)

[Tips](javascript:void(0);)

* Use the [poly](file:///C:\Program%20Files\MATLAB\R2015a\help\matlab\ref\poly.html) function to obtain a polynomial from its roots: p = poly(r). The poly function is the inverse of the roots function.

[Algorithms](javascript:void(0);)

The roots function considers p to be a vector with n+1 elements representing the nth degree characteristic polynomial of an n-by-n matrix, A. The roots of the polynomial are calculated by computing the eigenvalues of the companion matrix, A.

A = diag(ones(n-1,1),-1);

A(1,:) = -p(2:n+1)./p(1);

r = eig(A)

The results produced are the exact eigenvalues of a matrix within roundoff error of the companion matrix, A. However, this does not mean that they are the exact roots of a polynomial whose coefficients are within roundoff error of those in p.

More on polynomial roots:

# Polynomial Roots

[DOWNLOAD Mathematica Notebook](http://mathworld.wolfram.com/notebooks/Algebra/PolynomialRoots.nb)

A root of a polynomial P(z) is a number z_i such that P(z_i)=0. The [fundamental theorem of algebra](http://mathworld.wolfram.com/FundamentalTheoremofAlgebra.html) states that a [polynomial](http://mathworld.wolfram.com/Polynomial.html) P(z) of degree n has nroots, some of which may be degenerate. For example, the roots of the polynomial

|  |  |
| --- | --- |
| x^3-2x^2-x+2=(x-2)(x-1)(x+1) | (1) |

are -1, 1, and 2. Finding roots of a polynomial is therefore equivalent to [polynomial factorization](http://mathworld.wolfram.com/PolynomialFactorization.html) into factors of degree 1.

Any [polynomial](http://mathworld.wolfram.com/Polynomial.html) can be numerically factored, although different [algorithms](http://mathworld.wolfram.com/Algorithm.html) have different strengths and weaknesses.

The roots of a polynomial equation may be found exactly in the [Wolfram Language](http://www.wolfram.com/language/) using [Roots](http://reference.wolfram.com/language/ref/Roots.html)[lhs==rhs, var], or numerically using [NRoots](http://reference.wolfram.com/language/ref/NRoots.html)[lhs==rhs,var]. In general, a given root of a polynomial P(x)=x^n+a_(n-1)x^(n-1)+...+a_0 is represented as [Root](http://reference.wolfram.com/language/ref/Root.html)[#^n+a[n-1]#^(n-1)+...+a[0]&, k], where k=1, 2, ..., n is an index identifying the particular root and the pure function polynomial is [irreducible](http://mathworld.wolfram.com/IrreduciblePolynomial.html). Note that in the [Wolfram Language](http://www.wolfram.com/language/), the ordering of roots isdifferent in each of the commands [Roots](http://reference.wolfram.com/language/ref/Roots.html), [NRoots](http://reference.wolfram.com/language/ref/NRoots.html), and Table[Root[p, k], {k, n}].

In the [Wolfram Language](http://www.wolfram.com/language/), algebraic expressions involving [Root](http://reference.wolfram.com/language/ref/Root.html) objects can be combined into a new [Root](http://reference.wolfram.com/language/ref/Root.html) object using the command [RootReduce](http://reference.wolfram.com/language/ref/RootReduce.html).

In this work, the nth root of a polynomial P(x) in the ordering of the [Wolfram Language](http://www.wolfram.com/language/)'s [Root](http://reference.wolfram.com/language/ref/Root.html) object is denoted (P(x))_n, where x is a dummy variable. In this ordering, real roots come before complex ones and [complex conjugate](http://mathworld.wolfram.com/ComplexConjugate.html) pairs of roots are adjacent. For example,

|  |  |  |  |
| --- | --- | --- | --- |
| (x^2+x+1)_1 | = | 1/2(-1-isqrt(3)) | (2) |
| (x^2+x+1)_2 | = | 1/2(-1+isqrt(3)) | (3) |

and

|  |  |  |  |
| --- | --- | --- | --- |
| (x^3+x+1)_1 | approx | -0.68232 | (4) |
| (x^3+x+1)_2 | approx | 0.34116-1.1615i | (5) |
| (x^3+x+1)_3 | approx | 0.34116+1.1615i. | (6) |

Let the [roots](http://mathworld.wolfram.com/Root.html) of the polynomial

|  |  |
| --- | --- |
| P(x)=a_nx^n+a_(n-1)x^(n-1)+...+a_1x+a_0 | (7) |

be denoted r_1, r_2, ..., r_n. Then [Vieta's formulas](http://mathworld.wolfram.com/VietasFormulas.html) give

|  |  |  |  |
| --- | --- | --- | --- |
| sum_(i=1)^(n)r_i | = | -(a_(n-1))/(a_n) | (8) |
| sum_(i,j; i<j)^(n)r_ir_j | = | (a_(n-2))/(a_n) | (9) |
| sum_(i_1,i_2,...,i_k; i_1<i_2<...<i_k)^(n)r_(i_1)r_(i_2)...r_(i_k) | = | (-1)^k(a_(n-k))/(a_n). | (10) |

These can be derived by writing

|  |  |
| --- | --- |
| P(x)=a_n(x-r_1)(x-r_2)...(x-r_n), | (11) |

expanding, and then comparing the coefficients with (◇).

Given an nth degree polynomial a_nx^n+...+a_1x+a_0, the [roots](http://mathworld.wolfram.com/Root.html) can be found by finding the [eigenvalues](http://mathworld.wolfram.com/Eigenvalue.html) lambda_i of the [matrix](http://mathworld.wolfram.com/Matrix.html)

|  |  |
| --- | --- |
| [-a_1/a_0 -a_2/a_0 -a_3/a_0 ... -a_n/a_0; 1 0 0 ... 0; 0 1 0 ... 0; | | 1 ... 0; 0 0 0 ... 0] | (12) |

and taking r_i=1/lambda_i. This method can be computationally expensive, but is fairly robust at finding close and multiple roots.

If the [coefficients](http://mathworld.wolfram.com/Coefficient.html) of the [polynomial](http://mathworld.wolfram.com/Polynomial.html)

|  |  |
| --- | --- |
| d_nx^n+d_(n-1)x^(n-1)+...+d_0=0 | (13) |

are specified to be [integers](http://mathworld.wolfram.com/Integer.html), then rational roots must have a [numerator](http://mathworld.wolfram.com/Numerator.html) which is a factor of d_0 and a [denominator](http://mathworld.wolfram.com/Denominator.html) which is a factor of d_n (with either sign possible). This is known as the [polynomial remainder theorem](http://mathworld.wolfram.com/PolynomialRemainderTheorem.html).

If there are no [negative](http://mathworld.wolfram.com/Negative.html) [roots](http://mathworld.wolfram.com/Root.html) of a [polynomial](http://mathworld.wolfram.com/Polynomial.html) (as can be determined by [Descartes' sign rule](http://mathworld.wolfram.com/DescartesSignRule.html)), then the [greatest lower bound](http://mathworld.wolfram.com/GreatestLowerBound.html) is 0. Otherwise, write out the[coefficients](http://mathworld.wolfram.com/Coefficient.html), let n=-1, and compute the next line. Now, if any [coefficients](http://mathworld.wolfram.com/Coefficient.html) are 0, set them to minus the sign of the next higher [coefficient](http://mathworld.wolfram.com/Coefficient.html), starting with the second highest order [coefficient](http://mathworld.wolfram.com/Coefficient.html). If all the signs alternate, n is the greatest lower bound. If not, then subtract 1 from n, and compute another line. For example, consider the [polynomial](http://mathworld.wolfram.com/Polynomial.html)

|  |  |
| --- | --- |
| y=2x^4+2x^3-7x^2+x-7. | (14) |

Performing the above [algorithm](http://mathworld.wolfram.com/Algorithm.html) then gives

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 2 | 2 | -7 | 1 | -7 |
| -1 | 2 | 0 | -7 | 8 | -15 |
| -- | 2 | -1 | -7 | 8 | -15 |
| -2 | 2 | -2 | -3 | 7 | -21 |
| -3 | 2 | -4 | 5 | -14 | 35 |

so the greatest lower bound is -3.

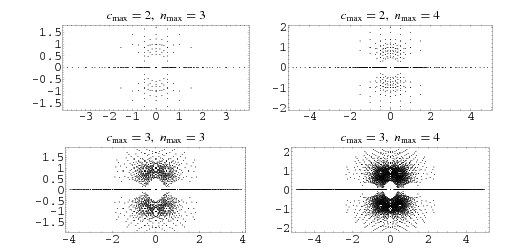
If there are no [positive](http://mathworld.wolfram.com/Positive.html) [roots](http://mathworld.wolfram.com/Root.html) of a [polynomial](http://mathworld.wolfram.com/Polynomial.html) (as can be determined by [Descartes' sign rule](http://mathworld.wolfram.com/DescartesSignRule.html)), the [least upper bound](http://mathworld.wolfram.com/LeastUpperBound.html) is 0. Otherwise, write out the[coefficients](http://mathworld.wolfram.com/Coefficient.html) of the [polynomials](http://mathworld.wolfram.com/Polynomial.html), including zeros as necessary. Let n=1. On the line below, write the highest order [coefficient](http://mathworld.wolfram.com/Coefficient.html). Starting with the second-highest [coefficient](http://mathworld.wolfram.com/Coefficient.html), add n times the number just written to the original second [coefficient](http://mathworld.wolfram.com/Coefficient.html), and write it below the second [coefficient](http://mathworld.wolfram.com/Coefficient.html). Continue through order zero. If all the [coefficients](http://mathworld.wolfram.com/Coefficient.html) are [nonnegative](http://mathworld.wolfram.com/Nonnegative.html), the least upper bound is n. If not, add one to x and repeat the process again. For example, take the[polynomial](http://mathworld.wolfram.com/Polynomial.html)

|  |  |
| --- | --- |
| y=2x^4-x^3-7x^2+x-7. | (15) |

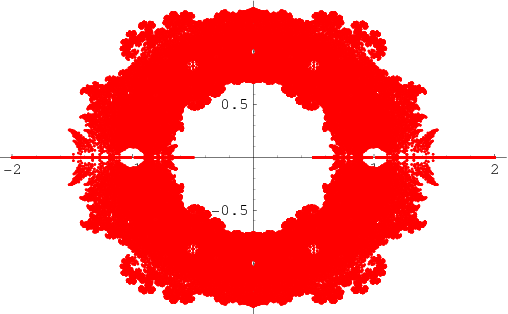
Performing the above [algorithm](http://mathworld.wolfram.com/Algorithm.html) gives

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 2 | -1 | -7 | 1 | -7 |
| 1 | 2 | 1 | -6 | -5 | -12 |
| 2 | 2 | 3 | -1 | -1 | -9 |
| 3 | 2 | 5 | 8 | 25 | 68 |

so the [least upper bound](http://mathworld.wolfram.com/LeastUpperBound.html) is 3.



Plotting the roots in the complex plane of all polynomials up to some degree with integer coefficients less than some cutoff integer in absolute value shows the beautiful structure illustrated above (Trott 2004, p. 23).



An even more stunning figure is obtained by plotting all roots of all polynomials with coefficients +/-1 up to degree n (Borwein and Jörgenson 2001; Pickover 2002; Bailey et al. 2007, p. 18).

**SEE ALSO:**

%=========================================================================

%

% Program to identify properties of US macro variables

We are looking for stationarity in each variable.

%

%=========================================================================

% look into page 15 of statistics in matlab a primar

function stsm\_roots()

clear all

clc

% Read the data: quarterly US data from Jan-1959 to Dec-1998

load simsdata.mat

% Define varaibles

r = ytdata(:,1);

lex = log( ytdata(:,2) );

lcp = log( ytdata(:,3) );

lm = log( ytdata(:,4) );

lp = log( ytdata(:,5) );

lo = log( ytdata(:,6) );

sdum = ytdata(:,7:17);

% Construct variables for use in VAR: from 1960:1 to 1998:12

% interest rate and the annual % growth rates in money, price and output

yvar = [r lm lp lo ];

%%Its a matrix Concatenating. The second part of matrix is MCM or Matrix

%%Chain Multiplication. This has been explained in granger cause problem.

% The first row is simply interest rate, excluding the first 12 months

% the next part is other variables, where the growth rate is measured by

% excluding the first and last year and taking differences...very easy.

y = [trimr(yvar(:,1),12,0) 100\*(trimr(yvar(:,2:4),12,0) - trimr(yvar(:,2:4),0,12)) ];

% Compute the roots of the AR(4) model for the interest rate. The polynomial roots

y = r;

% Recall that we are multiplying with one so that we get iterated

% projection from a1,a2,a3......an. in case of matrix multiplication we

% multiply with ones for the purpose of whole matrix concetanation. In

% mathematics, concatenation is the joining of two numbers by their numerals. That is, the concatenation of 123 and 456 is 123456. Concatenation of numbers a and b is denoted a||b.

% Here we are estimating a AR(4) model for interest rate and determining if the process is stationary or not.

theta = [ones(length(y)-4,1) trimr(y,3,1) trimr(y,2,2) trimr(y,1,3) trimr(y,0,4)] \trimr(y,4,0);

% There might be a question as why are we making a one column matrix

% starting from theta(5)...because the polynomial equation is

% 1-phi1z-phiz2z^2-phi3z^3-phi4z^4=0...so we are starting from 5 and

% ending at 2 in relation to the equation given in the problem.

c = [-theta(5); -theta(4); -theta(3); -theta(2); 1 ];

%The roots function solves polynomial equations of the form p\_1^x^n+p\_n^x+p\_n+1=0.....+p. Polynomial equations contain a single variable with nonnegative exponents. Use the poly function to obtain a polynomial from its roots: p = poly(r). The poly function is the inverse of the roots function.

rt = roots( c );

format short

disp('AR(4) model for interest rates')

disp('------------------------------')

disp(' Roots Absolute Value of Roots')

disp( [ rt abs( rt ) ] )

disp(' ')

clear c rt;

% Compute the roots of the AR(2) model for real output growth

%in real output growth we are only concenrned with percentage growth of

%output...

y = 100\*(trimr(lo,12,0) - trimr(lo,0,12));

theta = [ones(length(y)-2,1) trimr(y,1,1) trimr(y,0,2) ]\trimr(y,2,0);

% equation is 1-phi1z-phiz2z^2=0

c = [ -theta(3); -theta(2); 1 ];

rt = roots( c );

format short

disp('AR(2) model for real output growth')

disp('----------------------------------')

disp(' Roots Absolute Value of Roots')

disp( [ rt abs( rt ) ] )

disp(' ')

clear c rt;

% Compute the roots of the AR(2) model for logarithm of output given in the question

y = log( ytdata(:,6) );

theta = [ones(length(y)-2,1) trimr(y,1,1) trimr(y,0,2) ]\trimr(y,2,0);

c = [ -theta(3); -theta(2); 1 ];

rt = roots( c );

format short

disp('AR(2) model for log of real output')

disp('----------------------------------')

disp(' Roots Absolute Value of Roots')

disp( [ rt abs( rt ) ] )

disp(' ')

clear c rt;

% Compute the roots of the VAR(2) model

% Now this problem is slightly different..Because the quiestion

% suggests to comute VAR(2)..not AR(2)...containing a constat, annual

% percentage growth rate of money and inflation rate and the polynomial

% is now in absolute terms. because it is VAR(2) we are going upto

% phi2..

yvar = [ lm lp ];

y = trimr(yvar,12,0) - trimr(yvar,0,12);

X = [ones(length(y)-2,1) trimr(y,1,1) trimr(y,0,2)];

theta = X\trimr(y,2,0);

% Please note we are taking inverse valuies of thete here... %Because mue is for two variables theta we are trimming upto 4...and phii is excluding 2 from the last.giving the coefficients some dimension….

mu = trimr(theta,0,4)';

phi1 = trimr(theta,1,2)';

phi2 = trimr(theta,3,0)';

%%%%%%

[X,e] = polyeig(A0,A1,...Ap) solves the polynomial eigenvalue problem of degree p

(A0+λA1+…+λPAp)x=0

where polynomial degree p is a non-negative integer, and A0,A1,...Ap are input matrices of order n. The output consists of a matrix X of size n-by-n\*p whose columns are the eigenvectors, and a vector e of length n\*p containing the eigenvalues.

% polyeig gives you inverse of roots..

% eye is identity matrix. I = eye([n](file:///C:\Program%20Files\MATLAB\R2015a\help\matlab\ref\eye.html?searchHighlight=eye#inputarg_n)) returns an n-by-n identity matrix with ones on the main diagonal and zeros elsewhere.

Create a 4-by-4 identity matrix.

I = eye(4)

I =

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

% rt is roots in the following…

% ~ The tilde character can be used in function definitions to represent an input argument that is unused within the function. It can also be used to indicate that an output argument of function call is to ignored. In this case, it must appear within [ ] and separated by commas from any other arguments.%%%%%%%%%%

[~,rt] = polyeig(eye(2),-phi1,-phi2);

Other

%~ Logical NOT. ~A performs a logical NOT of input array A, and returns an array containing elements set to either logical 1 (TRUE) or logical 0 (FALSE).

% An element of the output array is set to 1 if A contains a zero value

% element at that same array location. Otherwise, that element is set to

% 0.

%

% B = NOT(A) is called for the syntax '~A' when A is an object.

%

% ~ can also be used to ignore input arguments in a function definition,

% and output arguments in a function call. See "help punct"

% Copyright 1984-2005 The MathWorks, Inc.

format short

disp('VAR(2) model of money and inflation')

disp('----------------------------------')

disp(' Roots Absolute Value of Roots')

disp( [ rt abs( rt ) ] )

disp(' ')

clear rt

% Compute the roots of the VAR(2) model containing

% interest rate, money growth rate, inflation and real GDP growth rate

yvar = [ r lm lp lo ];

y = [trimr(yvar(:,1),12,0) 100\*(trimr(yvar(:,2:4),12,0) - trimr(yvar(:,2:4),0,12)) ];

X = [ ones(length(y)-2,1) trimr(y,1,1) trimr(y,0,2) ];

theta = X\trimr(y,2,0);

% Interesting, theta was 17 by 4 matrix...which is turned to only 4 by 1

% matrix in mu and phi1 and phi2 both turns this to 4 by 4 matrix. because

% we are delaing with 4 variables, inm mue we simply exclude 4 X 2 = 8 and

% in phi1 we exclude the last 4, the first 1 is exlcluded to maintain

% dimension with phi 2......the rotation of theta ensures 4 by 4 matrix.

mu = trimr(theta,0,8)';

phi1 = trimr(theta,1,4)';

phi2 = trimr(theta,5,0)';

[~,rt] = polyeig(eye(4),-phi1,-phi2);

format short

disp('VAR(2) model of interest rate, money, inflation and real gdp')

disp('------------------------------------------------------------')

disp(' Roots Absolute Value of Roots')

disp( [ rt abs( rt ) ] )

disp(' ')

clear rt

% Compute the roots of the VAR(4) model containing

% interest rate, money growth rate, inflation and real GDP growth rate

yvar = [ r lm lp lo ];

y = [trimr(yvar(:,1),12,0) 100\*(trimr(yvar(:,2:4),12,0) - trimr(yvar(:,2:4),0,12)) ];

X = [ ones(length(y)-4,1) trimr(y,3,1) trimr(y,2,2) trimr(y,1,3) trimr(y,0,4)];

theta = X\trimr(y,4,0);

% Again unlike the previous VAR(2) calculations, this problem deals with

% VAR(4), so the sequuence of exclusion changes....remember the mue

% exclusion is 4^2=16; or sequence can be simply...the phi exclusion follows,0, 1,5,9,18 following n+3

% and last row exlusion is following n+4 which is 0,4,8,12,16...this

% includes mu.....

mu = trimr(theta,0,16)';

phi1 = trimr(theta,1,12)';

phi2 = trimr(theta,5,8)';

phi3 = trimr(theta,9,4)';

phi4 = trimr(theta,13,0)';

[~,rt] = polyeig(eye(4),-phi1,-phi2,-phi3,-phi4);

format short

disp('VAR(2) model of interest rate, money, inflation and real gdp')

disp('------------------------------------------------------------')

disp(' Roots Absolute Value of Roots')

disp( [ rt abs( rt ) ] )

disp(' ')

end

stsm-arma%=========================================================================

%

% Program to estimate an ARMA model

%

%=========================================================================

function stsm\_arma( )

clear all

clc

% Read the data: quarterly US data from Jan-1959 to Dec-1998

load sims\_data.mat

% Define variables

r = ytdata(:,1);

lex = log( ytdata(:,2) );

lcp = log( ytdata(:,3) );

lm = log( ytdata(:,4) );

lp = log( ytdata(:,5) );

lo = log( ytdata(:,6) );

sdum = ytdata(:,7:17);

t = length(ytdata);

% Choose y variable

y = r; % Interest rate

% Here we are only doing for interest rate….when we do the calculations for annual growth and annual inflation we just change the value of y….

%y = 100\*(trimr(lo,12,0) - trimr(lo,0,12)); % Annual output growth

%y = 100\*(trimr(lp,12,0) - trimr(lp,0,12)); % Annual inflation

% in the question it is mentioned to use BFGS algorithm....

bstart = 0.1\*ones(3,1);

%Use 'trust-region' algorithm if possible when set to the default 'on'. Use 'quasi-newton' algorithm when set to 'off'.

%The LargeScale algorithm requires you to provide the gradient (see the preceding description of fun), or else fminunc uses the 'quasi-newton' algorithm. For information on choosing the algorithm, see Choosing the Algorithm.

options = optimset('LargeScale','off','Display','iter');

%fminunc, Find minimum of unconstrained multivariable function.Finds the minimum

%of a problem specified by min f(x). f(x) is a function that returns a

%scaler, x is a matrix.[x,fval,exitflag,output,grad,hessian] = fminunc(...)

%fminunc(...) returns in hessian the value of the Hessian of the objective function fun at the solution x.

[ theta,fval,~,~,~,H ] = fminunc( @(b) neglogl(b,y),bstart,options );

lnl1 = -fval;

vc = (1/(t-1))\*inv(H);

% Wald test (AR)

r = [ 0 1 0 ];

q = 0;

w = (r\*theta - q)'\*inv(r\*vc\*r')\*(r\*theta - q);

disp(' ')

disp(['Wald statistic (AR) = ', num2str(w) ]);

disp(['p-value = ', num2str(1-chi2cdf(w,1)) ]);

disp(' ')

% Wald test (MA)

r = [ 0 0 1 ];

q = 0;

w = (r\*theta - q)'\*inv(r\*vc\*r')\*(r\*theta - q);

disp(' ')

disp(['Wald statistic (MA) = ', num2str(w) ]);

disp(['p-value = ', num2str(1-chi2cdf(w,1)) ]);

disp(' ')

% Wald test (Joint)

r = [ 0 1 0 ;

0 0 1 ];

q = [ 0 ; 0 ];

w = (r\*theta - q)'\*inv(r\*vc\*r')\*(r\*theta - q);

disp(' ')

disp(['Wald statistic (Joint) = ', num2str(w) ]);

disp(['p-value = ', num2str(1-chi2cdf(w,2)) ]);

disp(' ')

% Likelihood Ratio test

% Located in book page 128 and search page 166...LR=-2T(Lg of

% LT(theta0)-Lg of LT(theta1))...very sinmple....their are the

% estimated parameters.

lnl0 = -neglogl( [mean(y) 0 0],y );

lr = -2\*(t-1)\*(lnl0 - lnl1);

disp(' ')

disp(['LR statistic (Joint) = ', num2str(lr) ]);

disp(['p-value = ', num2str(1-chi2cdf(lr,2)) ]);

disp(' ')

end

%--------------------------- Subroutines ----------------------------------

%

%--------------------------------------------------------------------------

% Log-likelihood function for an ARMA model

%--------------------------------------------------------------------------

%example 13.18 in the book, page 522,book page 484 conditional log likelihood function

%of N dimensional VAR(p) model.

function f = neglogl( b,y )

%[m,n] = size(X) returns the size of matrix X in separate variables m and n.

[ t,n ] = size( y );

v = zeros( t,1 );

lf = zeros( t-1,1 );

% First loop over MA term

% page 522, book page 484 in the book gives residual covariance matrix V where i starts from 2….

for i = 2:t

[v(i)](neglogl.pdf) = y(i)-b(1)-b(2)\*y(i-1)-b(3)\*v(i-1);

end

v = trimr( v,1,0 );

vc = v'\*v/(t-1);

for i = 1:t-1;

%d = det(A) returns the determinant of square matrix A.

%When a matrix is used to represent linear transformations (as is commonly the case in 3D graphics), the determinant effectively represents the degree of unambiguousness inside a matrix.

%That is, if determinant is non-zero, the matrix can be inverted because the components don't "cancel out" each other (which would cause zero determinant).

%page 522 aka book page 484 equation 13.17. In the equation vc is

%covariance matrix and absolute is determined by its determinant. Starting

%value of t is 2. inv(vc)=1\V. vc is the residual covariance matrix, obtained in the final iteration and v

%is the disturbance term.It has zero mean and its covriance matrix comes

%from identity matrix.page 522 for details.t stat is used.

lf(i) = -0.5\*n\*log(2\*pi) - 0.5\*log(det(vc)) ...

- 0.5\*v(i,:)\*inv(vc)\*v(i,:)';

end

f = -mean( lf );

end

stsm gaussn

%=========================================================================

%

% Program to estimate an ARMA(1,1) using the GAUSS-NEWTON algorithm

%

%=========================================================================

clear all;

clc;

RandStream.setGlobalStream( RandStream('mt19937ar','seed',12345) );

% Alternative starting values

theta = [0.2 ; 0.2 ; 0.2 ];

%theta = [0.0 ; 0.0 ; 0.0 ];

%theta = [0.0 ; 0.3 ; 1.1 ];

%theta = [0.0 ; 1.1 ; 0.3 ];

% Alternative sample sizes

t = 200;

%t = 500;

% Simulate the data (discard first 100 simulated observations)

mu = 0.0;

phi1 = 0.8;

si1 = 0.3;

% in order to discard the first 100 simulated observations, you must add

% 100 observations first.

vt = randn(t+101,1);

% we are trimming one from first and last of vt is probably due to

% dimensional adjustment....it is because vt has (n+1) observations, so we

% are removing one rows in the equation to bring it down to n.

yt = recserar( mu + trimr(vt,1,0) + si1\*trimr(vt,0,1) , 0.0 , phi1);

yt = trimr(yt,100,0);

% Gauss Newton algorithm

crit = 0.00001; % Convergence criterion

maxit = 20; % Maximum number of iterations

i = 1;

while i <= maxit

% vt is being redefined in the [gauss-newton algorithm](../Theories/Gauss–Newton%20algorithm%20-%20Wikipedia,%20the%20free%20encyclopedia.pdf)

%theta(1) or theta(2) means only the elements (1) or (2) within theta

%matrix is being used. first chapter of statitics in matlab for expamle X=[ 2 3 1], where X(2)=3. %Look into zt...its simple,[ones(length(vt),1)trimr(yt,0,1)trimr(vt,0,1)]

% its just adjusting for dimensions. the equation v(t) follows gauss

% newton algorithm.

%

vt = recserar( trimr(yt,1,0) - theta(1) - theta(2)\*trimr(yt,0,1) , 0.0 , -theta(3));

z1t = recserar( ones(length(vt),1) , 0.0 , -theta(3) );

z2t = recserar( trimr(yt,0,1) , 0.0 , -theta(3) );

z3t = recserar( [ 0.0 ; trimr(vt,0,1) ] , 0.0 , -theta(3));

zt = [ z1t z2t z3t ];

% we will need zt to estimate covariance matrix later. vt is Vt, the

% unrestricted maximum likelihood estimator of covariance matrix V in page 164.

% Remove all starting zeros

vt = trimr(vt,2,0);

zt = trimr(zt,2,0);

% we are developing a program which will show different parameter vector

% matrix for different inputs of iteration

disp(['Iteration = ', num2str(i) ]);

disp(['Parameter vector = ', num2str(theta') ]);

disp(' ');

dtheta = zt\vt;

% Check convergence and update parameters

if dtheta'\*dtheta < crit

break

else

theta = theta + dtheta;

if i == maxit

disp(['Failed to converge after iteration ', num2str(maxit) ]);

end

end

i = i + 1;

end

% Now here, sig2 is representing Vt the unrestricted maximum likelihood

% estimator using vt (page 127, search 165). But vc is covariance matrix which has been

% noramilized using an additional parameter matrix zt.t-stats is theta.\standard errors from formula

sig2 = (vt'\*vt)/t;

vc = sig2\*inv(zt'\*zt);

se = sqrt(diag(vc));

disp(' ')

disp('Parameters Std. errors t-stats')

disp( [theta se theta./se ]);

disp(' ')

disp('Estimated asymptotic covariance matrix')

disp( vc )

% Wald test (page 166, book page 128)

w = theta(3)^2/vc(3,3);

disp(' ')

disp(['Wald statistic (si1 = 0.0) = ', num2str(w) ]);

disp(['p-value = ', num2str(1-chi2cdf(w,1)) ]);

disp(' ')

% LM test :page 169

% First regression

y = trimr(yt,1,0);

x = [ones(length(y),1) trimr(yt,0,1) ];

e = y - x\*(x\y);

% Second regression

y = trimr(e,1,0);

x = [ones(length(y),1) trimr(yt,1,1) trimr(e,0,1) ];

e = y - x\*(x\y);

%bsxfun = Apply element-by-element binary operation to two arrays with

%singleton expansion enabled( 1 matrix for example matlab will generally auto disable singleton expansion so if A is 3 by 2 by 1 by 1 matrix matlab will automatically transform it a 3 by 2 matrix. This is singleton).

%C = bsxfun(fun,A,B) applies the element-by-element binary operation specified by the function handle fun to arrays A and B, with singleton expansion enabled.

%Use bsxfun to subtract the column mean from the corresponding column elements of a matrix, A.

%%%for example, A = [1 2 10; 1 4 20;1 6 15] ;so C = bsxfun(@minus, A, mean(A))

% Here bsxfun is doing a deduction of mean(y) from y itself. The beauty of

% bsxfun is not limited here. Remember the singleton expansion? where the

% singleton expansion is generally omitted, **with bsxfun Singleton expansion allows bsxfun to expand the input vectors into a full matrix**.

%Call a custom-defined binary function with bsxfun by specifying a handle to the function.

%example--

%fun = @(A,B) A.\*sin(B); A = 1:7; B = pi\*[0 1/4 1/3 1/2 2/3 3/4 1].'; C

%=bsxfun(fun,A,B)%%%

y = bsxfun(@minus,y,mean(y));

rsq = 1 - (e'\*e)/(y'\*y);

lm = length(yt)\*rsq;

disp(' ')

disp(['LM statistic (si1 = 0.0) = ', num2str(lm) ]);

% We should always remember to use simple chi asquare with this equation.

disp(['p-value = ', num2str(1-chi2cdf(lm,1)) ]);

disp(' ')

stsm lm

%=========================================================================

%

% Program to demonstrate properties of the LM test

%

%=========================================================================

clear all;

clc;

RandStream.setGlobalStream( RandStream('mt19937ar','seed',12345) );

% Sample sizes

t = 200;

% Simulate the data (discard first 100 simulated observations)..Here the

% values of mue , phi1 and si1 values are taken from p[roblem 5 as in this

% problem it is mentioned to use the ARMA(1,1) model of exervise 5.

mu = 0.0;

phi1 = 0.8;

si1 = 0.3;

vt = randn(t+101,1);

% This is the simulation stage

yt = recserar( mu + trimr(vt,1,0) + si1\*trimr(vt,0,1) , 0.0 , phi1);

% Now discard the first 100 observations taking only the latest values

yt = trimr(yt,100,0);

% Demonstrate the equivalence result

**%Very important to remember, under all circumstances the codes for lm tests remain same for AR or for MA. The only thing that matters depending on whether you are dealing with ARMA(1,1) or ARMA(2,2) or whatever data you input in mue, phi,si1 will change the results for LM test. The changes you will need to bring, you will bring in yt equation in th earlier stage only.**

% LM test of AR(1) : Following is the basic of lm test everywhere. Just memorize it...always remember the steps. Explained earlier in stsm\_gaussn. Theories can be found in Page 132 of book and 169 in search.

% First regression

y = yt;

x = ones(length(y),1);

v = y - x\*(x\y);

% Second regression

y = trimr(v,1,0);

x = [ones(length(y),1) trimr(v,0,1) ];

e = y - x\*(x\y);

y = bsxfun(@minus,y,mean(y));

rsq = 1 - (e'\*e)/(y'\*y);

lm = (t-1)\*rsq;

disp(' ')

disp(['LM statistic (phi1 = 0.0) = ', num2str(lm) ]);

disp(['p-value = ', num2str(1-chi2cdf(lm,1)) ]);

disp(' ')

% LM test of MA(1): the only difference is in first difference instead of v

% we use e and in second regression we trim e.

% First regression

y = yt;

x = ones(length(y),1);

e = y - x\*(x\y);

% Second regression

y = trimr(e,1,0);

x = [ones(length(y),1) trimr(e,0,1) ];

e = y - x\*(x\y);

y = bsxfun(@minus,y,mean(y));

rsq = 1 - (e'\*e)/(y'\*y);

lm = (t-1)\*rsq;

disp(' ')

disp(['LM statistic (si1 = 0.0) = ', num2str(lm) ]);

disp(['p-value = ', num2str(1-chi2cdf(lm,1)) ]);

disp(' ')

% Demonstrate singularity result: the difference is we have only v and in

% second regression we just construct a matrix trimming v. No estimation of

% second regression.

% LM test (joint AR and MA). This is important as in many cases you will be dealing with ARMA(1,1) or ARMA(2,2)…you can use LM test for AR or MA or joint and you will have equivalent result( proved earlier).

% First regression

y = yt;

x = ones(length(y),1);

v = y - x\*(x\y);

% Second regression

y = trimr(v,1,0);

x = [ones(length(y),1) trimr(yt,0,1) trimr(v,0,1) ];

disp('Correlation of explanatory variables at second stage')

disp( corrcoef( x(:,[2 3]) ) );

stsm Varma

%=========================================================================

%

% Program to simulate and estimate a VARMA model

%

%=========================================================================

function stsm\_varma( )

clear all

clc

RandStream.setGlobalStream( RandStream('mt19937ar','seed',1234567) );

% Generate the data%nobs means number of observations

nobs = 1000;

mu1 = 0.0; phi111 = 0.6; si111 = 0.2; si112= -0.5; % Equation 1

mu2 = 0.0; phi122 = 0.4; si121 = 0.2; si122 = 0.6; % Equation 2

nobs = nobs + 100; % Allow for startup

y1 = zeros(nobs,1);

y2 = zeros(nobs,1);

v1 = randn(nobs,1);

v2 = randn(nobs,1);

for t = 2:nobs

% these equations follows the format of equation 13.8, book page 474.

y1(t) = mu1 + phi111\*y1(t-1) + v1(t) + si111\*v1(t-1) + si112\*v2(t-1);

y2(t) = mu2 + phi122\*y2(t-1) + v2(t) + si121\*v1(t-1) + si122\*v2(t-1);

end

y = [ trimr(y1,100,0) trimr(y2,100,0) ];

t = length(y);

% Estimate the unrestricted model

bstart = [mu1; phi111; si111; si112; mu2; phi122; si121; si122];

options = optimset('LargeScale','off','Display','iter');

% @ is function handcle creation sign that preeceds a function which is neglogl here into b% .

[ theta,fvalu,~,~,~,[hess](../Theories/Hessian%20matrix%20-%20Wikipedia,%20the%20free%20encyclopedia.pdf) ] = ...

fminunc( @(b) neglogl(b,y),bstart,options );

## Use of Hessian in optimization

[Hessian 1](../Theories/Hessian_page%201.pdf)

[Hessian 2](../Theories/Hessian_page%202.pdf)

Hessian matrices are used in large-scale [optimization](https://en.wikipedia.org/wiki/Optimization_(mathematics)) problems within [Newton](https://en.wikipedia.org/wiki/Newton%27s_method_in_optimization)-type methods because they are the coefficient of the quadratic term of a local [Taylor expansion](https://en.wikipedia.org/wiki/Taylor_expansion) of a function. That is,

y=f(\mathbf{x} + \Delta\mathbf{x})\approx f(\mathbf{x}) + \nabla f(\mathbf{x}) \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^\mathrm T\mathbf{H}(\mathbf{x}) \Delta\mathbf{x}

where ∇f is the [gradient](https://en.wikipedia.org/wiki/Gradient) (∂f/∂x1, ..., ∂f/∂xn). Computing and storing the full Hessian matrix takes [Θ](https://en.wikipedia.org/wiki/Big_O_notation)(n2) memory, which is infeasible for high-dimensional functions such as the [loss functions](https://en.wikipedia.org/wiki/Loss_function) of [neural nets](https://en.wikipedia.org/wiki/Artificial_neural_network), [conditional random fields](https://en.wikipedia.org/wiki/Conditional_random_field), and other [statistical models](https://en.wikipedia.org/wiki/Statistical_model) with large numbers of parameters. For such situations, [truncated-Newton](https://en.wikipedia.org/wiki/Truncated_Newton_method) and [quasi-Newton](https://en.wikipedia.org/wiki/Quasi-Newton_method) algorithms have been developed. The latter family of algorithms use approximations to the Hessian; one of the most popular quasi-Newton algorithms is [BFGS](https://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfarb%E2%80%93Shanno_algorithm).[[4]](https://en.wikipedia.org/wiki/Hessian_matrix#cite_note-4)

Such approximations may use the fact that an optimization algorithm uses the Hessian only as a [linear operator](https://en.wikipedia.org/wiki/Linear_operator)H(v), and proceed by first noticing that the Hessian also appears in the local expansion of the gradient:

\nabla f (\mathbf{x} + \Delta\mathbf{x}) = \nabla f (\mathbf{x}) + \mathbf{H}(\Delta\mathbf{x}) + O(\|\Delta\mathbf{x}\|^2)

Letting Δx = rv for some scalar r, this gives

\mathbf{H}(\Delta\mathbf{x}) = \mathbf{H}(r\mathbf{v}) = r\mathbf{H}(\mathbf{v}) = \nabla f (\mathbf{x} + r\mathbf{v}) - \nabla f (\mathbf{x}) + O(r^2),

i.e.,

\mathbf{H}(\mathbf{v}) = \frac{1}{r} \Bigl[ \nabla f (\mathbf{x} + r\mathbf{v}) - \nabla f (\mathbf{x}) \Bigr] + O(r)

so if the gradient is already computed, the approximate Hessian can be computed by a linear (in the size of the gradient) number of scalar operations. (While simple to program, this approximation scheme is not numerically stable since r has to be made small to prevent error due to the O(r) term, but increasing it loses precision in the first term.[[5]](https://en.wikipedia.org/wiki/Hessian_matrix#cite_note-5))

% Wald test

%Important book pages : Page 144, 145, look into page 129 equation 4.6 to understand why inverse of hessian is used. Page 134 and 135 discusses theoretical part of restriction cases. Page 137, the LM equation shows how we come to derive use of hessian from gradiant and their relationships.

vc = (1/t)\*inv( hess );

r = [0 0 1 0 0 0 0 0 ;

0 0 0 1 0 0 0 0 ;

0 0 0 0 0 0 1 0 ;

0 0 0 0 0 0 0 1 ] ;

q = [ 0; 0; 0; 0 ];

w = (r\*theta - q)'\*inv(r\*vc\*r')\*(r\*theta - q);

disp( ['Wald test = ' num2str(w) ] );

disp( ['p-value = ' num2str(1-chi2cdf(w,4))] );

% Estimate the restricted model

bstart = [ mu1; phi111; mu2; phi122 ];

[ ~,fvalr,~,~,~,~ ] = ...

fminunc( @(b) negloglr(b,y),bstart,options );

% Likelihood Ratio test

fvalr = -fvalr;

fvalu = -fvalu;

lr = -2\*(t-1)\*(fvalr - fvalu);

disp( ['LR test = ' num2str(lr) ] );

disp( ['p-value = ' num2str(1-chi2cdf(lr,4))] );

end

%--------------------------- Subroutines ----------------------------------

%

%--------------------------------------------------------------------------

% Log-likelihood function for an unrestricted VARMA model

%--------------------------------------------------------------------------

function f = neglogl( b,y )

\ [ t,n ] = size( y );

e1 = zeros( t,1 );

e2 = zeros( t,1 );

lf = zeros( t-1,1 );

% First loop over MA part

for i = 2:t

% The structure of equation Simply following the same structure as stsm\_lm and stsm\_gaussn for calculation of LM or log-likelihood function. It also is quite similar to VARMA coding.

e1(i) = y(i,1)-b(1)-b(2)\*y(i-1,1)-b(3)\*e1(i-1)-b(4)\*e2(i-1);

e2(i) = y(i,2)-b(5)-b(6)\*y(i-1,2)-b(7)\*e1(i-1)-b(8)\*e2(i-1);

end

e = [ trimr( e1,1,0 ) trimr( e2,1,0 )] ;

omega = e'\*e/(t-1);

for i = 1:t-1;

lf(i) = -0.5\*n\*log(2\*pi) - 0.5\*log(det(omega)) ...

- 0.5\*e(i,:)\*inv(omega)\*e(i,:)';

end

f = -mean( lf );

end

%--------------------------------------------------------------------------

% Log-likelihood function for a restricted VARMA model

%--------------------------------------------------------------------------

function f = negloglr( b,y )

[ t,n ] = size( y );

e1 = zeros( t,1 );

e2 = zeros( t,1 );

lf = zeros( t-1,1 );

% First loop over MA part

for i = 2:t

e1(i) = y(i,1)-b(1)-b(2)\*y(i-1,1);

e2(i) = y(i,2)-b(3)-b(4)\*y(i-1,2);

end

e = [ trimr( e1,1,0 ) trimr( e2,1,0 )] ;

omega = e'\*e/(t-1);

for i = 1:t-1;

lf(i) = -0.5\*n\*log(2\*pi) - 0.5\*log(det(omega)) ...

- 0.5\*e(i,:)\*inv(omega)\*e(i,:)';

end

f = -mean( lf );

end

First-order

Iteration Func-count f(x) Step-size optimality

0 9 2.84322 0.0898

1 27 2.84029 0.394852 0.022

2 36 2.83988 1 0.0162

3 45 2.83948 1 0.00561

4 54 2.83945 1 0.00331

5 63 2.83943 1 0.0013

6 72 2.83943 1 0.00105

7 81 2.83942 1 0.000542

8 90 2.83942 1 0.000408

9 99 2.83942 1 0.000208

10 108 2.83942 1 0.000117

11 117 2.83942 1 8.33e-05

12 126 2.83942 1 4.89e-05

13 135 2.83942 1 4.19e-05

14 144 2.83942 1 1.36e-05

15 153 2.83942 1 7.87e-06

16 162 2.83942 1 5.19e-06

17 171 2.83942 1 4.77e-06

18 180 2.83942 1 2.06e-06

19 198 2.83942 0.353221 4.47e-07

Local minimum found.

Optimization completed because the size of the gradient is less than

the default value of the function tolerance.

<stopping criteria details>

Computing finite-difference Hessian using user-supplied objective function.

Wald Test=687.3548

p-value = 0

First-order

Iteration Func-count f(x) Step-size optimality

0 5 3.13234 0.371

1 10 3.09615 1 0.257

2 15 3.0667 1 0.0179

3 20 3.06659 1 0.00119

4 25 3.06659 1 0.000683

5 30 3.06659 1 4.18e-05

6 35 3.06659 1 1.32e-05

7 40 3.06659 1 3.9e-06

8 45 3.06659 1 1.13e-06

Local minimum found.

Optimization completed because the size of the gradient is less than

the default value of the function tolerance.

<stopping criteria details>

LR test = 453.883

p-value = 0

>>

stsm varmab

%=========================================================================

%

% Simulate and estimate a VARMA model with multivariate bilinearity

%

%=========================================================================

function stsm\_varmab( )

clear all

clc

RandStream.setGlobalStream( RandStream('mt19937ar','seed',1234567) );

% Generate the data

nobs = 1000;

mu1 = 0.0; phi111 = 0.6; si111 = 0.2; si112= -0.5; % Equation 1

mu2 = 0.0; phi122 = 0.4; si121 = 0.2; si122 = 0.6; % Equation 2

nobs = nobs + 100; % Allow for startup

y1 = zeros(nobs,1);

y2 = zeros(nobs,1);

v1 = randn(nobs,1);

v2 = randn(nobs,1);

for t = 2:nobs % because this is directly specified in book

y1(t) = mu1 + phi111\*y1(t-1) + v1(t) + si111\*v1(t-1) + si112\*v2(t-1);

y2(t) = mu2 + phi122\*y2(t-1) + v2(t) + si121\*v1(t-1) + si122\*v2(t-1);

end

% because we added 100 with nobs in line 18 (allow for startup), here for the data

% generation process, we are trimming 100 from y

y = [ trimr(y1,100,0) trimr(y2,100,0) ];

t = length(y);

% the following part will estimate the unrestricted model, but the data

% generation will take conditions from previous equations.

% Estimate the unrestricted model

bstart = [mu1; phi111; si111; si112; mu2; phi122; si121; si122 ; 0.01 ; 0.01];

options = optimset('LargeScale','off','Display','iter');

[ theta,fvalu,~,~,~,hess ] = ...

fminunc( @(b) neglogl(b,y),bstart,options );

% Wald test

vc = (1/t)\*inv( hess );

r = [0 0 0 0 0 0 0 0 1 0;

0 0 0 0 0 0 0 0 1 1 ];

q = [ 0; 0 ];

w = (r\*theta - q)'\*inv(r\*vc\*r')\*(r\*theta - q);

disp( ['Wald test = ' num2str(w) ] );

disp( ['p-value = ' num2str(1-chi2cdf(w,4))] );

% Estimate the restricted model

bstart = [mu1; phi111; si111; si112; mu2; phi122; si121; si122 ];

[ ~,fvalr,~,~,~,~ ] = ...

fminunc( @(b) negloglr(b,y),bstart,options );

% Likelihood Ratio test

fvalr = -fvalr;

fvalu = -fvalu;

lr = -2\*(t-1)\*(fvalr - fvalu);

disp(' ');

disp(['Unconstrained log-likelihood = ', num2str(fvalu) ]);

disp(['Constrained log-likelihood = ', num2str(fvalr) ]);

disp(' ');

disp( ['LR test = ' num2str(lr) ] );

disp( ['p-value = ' num2str(1-chi2cdf(lr,4))] );

end

%--------------------------- Subroutines ----------------------------------

%

%--------------------------------------------------------------------------

% Log-likelihood function for an unrestricted VARMA model

%--------------------------------------------------------------------------

function f = neglogl( b,y )

[ t,n ] = size( y );

e1 = zeros( t,1 );

e2 = zeros( t,1 );

lf = zeros( t-1,1 );

for i = 2:t

e1(i) = y(i,1)-b(1)-b(2)\*y(i-1,1)-b(3)\*e1(i-1)-b(4)\*e2(i-1) ...

-b(9)\*y(i-1,1)\*e1(i-1);

e2(i) = y(i,2)-b(5)-b(6)\*y(i-1,2)-b(7)\*e1(i-1)-b(8)\*e2(i-1) ...

- b(10)\*y(i-1,2)\*e2(i-1);

end

e = [ trimr( e1,1,0 ) trimr( e2,1,0 )] ;

vc = e'\*e/(t-1);

for i = 1:t-1;

lf(i) = -0.5\*n\*log(2\*pi) - 0.5\*log(det(vc)) ...

- 0.5\*e(i,:)\*inv(vc)\*e(i,:)';

end

f = -mean( lf );

end

%--------------------------------------------------------------------------

% Log-likelihood function for an restricted VARMA model

%--------------------------------------------------------------------------

function f = negloglr( b,y )

[ t,n ] = size( y );

e1 = zeros( t,1 );

e2 = zeros( t,1 );

lf = zeros( t-1,1 );

% First loop over MA part

for i = 2:t

e1(i) = y(i,1)-b(1)-b(2)\*y(i-1,1)-b(3)\*e1(i-1)-b(4)\*e2(i-1);

e2(i) = y(i,2)-b(5)-b(6)\*y(i-1,2)-b(7)\*e1(i-1)-b(8)\*e2(i-1);

end

e = [ trimr( e1,1,0 ) trimr( e2,1,0 )] ;

vc = e'\*e/(t-1);

for i = 1:t-1;

lf(i) = -0.5\*n\*log(2\*pi) - 0.5\*log(det(vc)) ...

- 0.5\*e(i,:)\*inv(vc)\*e(i,:)';

end

f = -mean( lf );

end

stsm\_varmab

%=========================================================================

%

% Simulate and estimate a VARMA model with multivariate bilinearity

%

%=========================================================================

function stsm\_varmab( )

clear all

clc

RandStream.setDefaultStream( RandStream('mt19937ar','seed',1234567) );

% Generate the data

nobs = 1000;

mu1 = 0.0; phi111 = 0.6; si111 = 0.2; si112= -0.5; % Equation 1

mu2 = 0.0; phi122 = 0.4; si121 = 0.2; si122 = 0.6; % Equation 2

nobs = nobs + 100; % Allow for startup

y1 = zeros(nobs,1);

y2 = zeros(nobs,1);

v1 = randn(nobs,1);

v2 = randn(nobs,1);

for t = 2:nobs

y1(t) = mu1 + phi111\*y1(t-1) + v1(t) + si111\*v1(t-1) + si112\*v2(t-1);

y2(t) = mu2 + phi122\*y2(t-1) + v2(t) + si121\*v1(t-1) + si122\*v2(t-1);

end

y = [ trimr(y1,100,0) trimr(y2,100,0) ];

t = length(y);

% Estimate the unrestricted model

bstart = [mu1; phi111; si111; si112; mu2; phi122; si121; si122 ; 0.01 ; 0.01];

options = optimset('LargeScale','off','Display','iter');

[ theta,fvalu,~,~,~,hess ] = ...

fminunc( @(b) neglogl(b,y),bstart,options );

% Wald test

vc = (1/t)\*inv( hess );

r = [0 0 0 0 0 0 0 0 1 0;

0 0 0 0 0 0 0 0 1 1 ];

q = [ 0; 0 ];

w = (r\*theta - q)'\*inv(r\*vc\*r')\*(r\*theta - q);

disp( ['Wald test = ' num2str(w) ] );

disp( ['p-value = ' num2str(1-chi2cdf(w,4))] );

% Estimate the restricted model

bstart = [mu1; phi111; si111; si112; mu2; phi122; si121; si122 ];

[ ~,fvalr,~,~,~,~ ] = ...

fminunc( @(b) negloglr(b,y),bstart,options );

% Likelihood Ratio test

fvalr = -fvalr;

fvalu = -fvalu;

lr = -2\*(t-1)\*(fvalr - fvalu);

disp(' ');

disp(['Unconstrained log-likelihood = ', num2str(fvalu) ]);

disp(['Constrained log-likelihood = ', num2str(fvalr) ]);

disp(' ');

disp( ['LR test = ' num2str(lr) ] );

disp( ['p-value = ' num2str(1-chi2cdf(lr,4))] );

end

%--------------------------- Subroutines ----------------------------------

%

%--------------------------------------------------------------------------

% Log-likelihood function for an unrestricted VARMA model

%--------------------------------------------------------------------------

function f = neglogl( b,y )

[ t,n ] = size( y );

e1 = zeros( t,1 );

e2 = zeros( t,1 );

lf = zeros( t-1,1 );

for i = 2:t

e1(i) = y(i,1)-b(1)-b(2)\*y(i-1,1)-b(3)\*e1(i-1)-b(4)\*e2(i-1) ...

-b(9)\*y(i-1,1)\*e1(i-1);

e2(i) = y(i,2)-b(5)-b(6)\*y(i-1,2)-b(7)\*e1(i-1)-b(8)\*e2(i-1) ...

- b(10)\*y(i-1,2)\*e2(i-1);

end

e = [ trimr( e1,1,0 ) trimr( e2,1,0 )] ;

vc = e'\*e/(t-1);

for i = 1:t-1;

lf(i) = -0.5\*n\*log(2\*pi) - 0.5\*log(det(vc)) ...

- 0.5\*e(i,:)\*inv(vc)\*e(i,:)';

end

f = -mean( lf );

end

%--------------------------------------------------------------------------

% Log-likelihood function for an restricted VARMA model

%--------------------------------------------------------------------------

function f = negloglr( b,y )

[ t,n ] = size( y );

e1 = zeros( t,1 );

e2 = zeros( t,1 );

lf = zeros( t-1,1 );

% First loop over MA part

for i = 2:t

e1(i) = y(i,1)-b(1)-b(2)\*y(i-1,1)-b(3)\*e1(i-1)-b(4)\*e2(i-1);

e2(i) = y(i,2)-b(5)-b(6)\*y(i-1,2)-b(7)\*e1(i-1)-b(8)\*e2(i-1);

end

e = [ trimr( e1,1,0 ) trimr( e2,1,0 )] ;

vc = e'\*e/(t-1);

for i = 1:t-1;

lf(i) = -0.5\*n\*log(2\*pi) - 0.5\*log(det(vc)) ...

- 0.5\*e(i,:)\*inv(vc)\*e(i,:)';

end

f = -mean( lf );

end

stsm finite

%=========================================================================

%

% Simulation demonstration of the finite sample properties of the

% AR(1) estimator- Book page 138 for other available methods. Also the importance of finite distribution…

%

%=========================================================================

function stsm\_finite( )

clear all

clc

RandStream.setDefaultStream( RandStream('mt19937ar','seed',1234567) );

% Generate the data

t = 50;

sigv = 1.0;

phi1 = 0.9;

ndraws = 10000;

theta\_mle = zeros( ndraws,1 );

for i = 1:ndraws

vt = sqrt(sigv)\*randn( t+101,1 );

yt = zeros( length( vt ),1 );

% Simulate the AR(1) model

for j = 2:length( vt )

yt(j) = phi1\*yt(j-1) + vt(j);

end

% Get rid of first 100 observations

yt = trimr( yt,100,0);

% Conditional mle

% This perticular equation is setting condition only...same goes

% for earlier calculations...

theta\_mle(i,:) = trimr( yt,0,1 )\trimr( yt,1,0 );

end

% Compute statistics of sampling distribution

mse\_mle = mean( (theta\_mle - phi1).^2 );

rmse\_mle = sqrt( mse\_mle );

disp( ['Population parameter = ', num2str( phi1 ) ] )

disp( ['Mean (cond. mle) = ', num2str( mean(theta\_mle) ) ] )

disp( ['Bias (cond. mle) = ', num2str( mean( theta\_mle )- phi1 ) ] )

disp( ['Bias (Shenton-Johnson) = ', num2str( -2\*phi1/t ) ] )

disp( ['RMSE (cond. mle) = ', num2str( rmse\_mle ) ] )

end

stsm laglength

%=========================================================================

%

% Simulate an AR(3) model and compute the optimal lag length

%

%==========================================================================

clear all;

clc;

RandStream.setGlobalStream( RandStream('mt19937ar','seed',1) );

% Nrep is the number of repetitions...here it is 2000 times...The

% coefficients are simply received from the equation in the question..We

% always add to the given size T, which is 200 in the question...So we have

% t+100 parameters. Pmax is 7 because in the question we are estimating AR(1) to AR(7).

NRep = 2000;

t = 300;

pmax = 7;

mu = 0.0;

phi1 = 0.2;

phi2 = -0.15;

phi3 = 0.05;

% Now here we have some interesting observation. zeros is creating arrays.

% Each time zeros function seems to be creating random number of 7 rows,

% which might be creating space for input.

acount = zeros( pmax,1 );

scount = zeros( pmax,1 );

hcount = zeros( pmax,1 );

%Steps, generating the data

for k = 1:NRep

% Generate data

% here zeros is again creating random numbers with first few rows with

% 0 scalers...

yt = zeros( t,1 );

% In the question it is mentioned that the true lag structure is 3, so we

% take 3+1....

for i = 4:t

yt(i) = mu+phi1\*yt(i-1)+phi2\*yt(i-2)+phi3\*yt(i-3)+sqrt(0.5)\*randn;

end

%%

%% Lagmatrix: Create matrix of lagged time series ,XLAG = lagmatrix(X,Lags),

% XLAG = lagmatrix(X,Lags) creates a lagged (shifted) version of a time series matrix. The lagmatrix function is useful for creating a regression matrix of explanatory variables for fitting the conditional mean of a return series.

% X is simply a time series data. Lags: Vector of integer lags. lagmatrix applies the first lag to every series in X, then applies the second lag to every series in X, and so forth. To include a time series as is, include a 0 lag. Positive lags correspond to delays, and shift a series back in time. Negative lags correspond to leads, and shift a series forward in time.

%XLags: Lagged transform of the time series X. To create XLAG, lagmatrix shifts each time series in X by the first lag, then shifts each time series in X by the second lag, and so forth. Since XLAG represents an explanatory regression matrix, each column is an individual time series. XLAG has the same number of rows as there are observations in X. Its column dimension is equal to the product of the number of columns in X and the length of Lags. lagmatrix uses a NaN (Not-a-Number) to indicate an undefined observation.

%Example:

% Create a Lag Matrix

%Create a bivariate time series matrix X with five observations each:

%X = [1 -1; 2 -2 ;3 -3 ;4 -4 ;5 -5] % Create a simple

% bivariate series.

%X =

% 1 -1

% 2 -2

% 3 -3

% 4 -4

% 5 -5

%Create a lagged matrix XLAG, composed of X and the first two lags of X:

%XLAG = lagmatrix(X,[0 1 2]) % Create the lagged matrix.

%XLAG =

% 1 -1 NaN NaN NaN NaN

% 2 -2 1 -1 NaN NaN

% 3 -3 2 -2 1 -1

% 4 -4 3 -3 2 -2

% 5 -5 4 -4 3 -3

%The result, XLAG, is a 5-by-6 matrix.

%%

% Set up lags and make sure that T is constant

ylag = lagmatrix(yt,0:pmax);

% here, we are just removing the first 100 observations from lagmatrix....a step we do for all

% random data generation....

ylag = ylag(101:end,:);

% Loop over the lags (yt is first column of ylag)

aic = zeros( pmax,1 );

sic = zeros( pmax,1 );

hic = zeros( pmax,1 );

y = ylag(:,1);

tt = length( y );

for j = 2:pmax+1

% here interestingly ones is simply creating ones matrix in the first

% column

x = [ ones(tt,1) ylag(:,2:j)];

%returns the size of dimension of x only.

%m = size(X,dim).m = size(X,dim) returns the size of the dimension of X specified by scalar dim.

k = size( x,2 );

b = x\y;

e = y-x\*b;

aic(j-1) = log(e'\*e/tt) + 2\*k/tt;

sic(j-1) = log(e'\*e/tt) + k\*log(tt)/tt;

hic(j-1) = log(e'\*e/tt) + 2\*k\*log(log(tt))/tt;

end

% this part is simpky identifying the minimum number representaing the

% laglength

[~,ind]=min(aic);

acount(ind) = acount(ind)+1;

[~,ind]=min(sic);

scount(ind) = scount(ind)+1;

[~,ind]=min(hic);

hcount(ind) = hcount(ind)+1;

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*

%\*\* Generate graph

%\*\*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Switch off TeX interpreter and clear figure

set(0,'defaulttextinterpreter','none');

figure(1);

clf;

bar([acount hcount scount])

colormap gray

xlabel('Lag Order');

laprint(1,'infocrit','options','factory');

Stsm granger

%=========================================================================

%

% Program to perform Granger causality tests on a VAR

%

%=========================================================================

function stsm\_granger( )

clear all

clc

% Read the data: quarterly US data from Jan-1959 to Dec-1998

load sims\_data.mat

% Define variables

r = ytdata(:,1);

lex = log( ytdata(:,2) );

lcp = log( ytdata(:,3) );

lm = log( ytdata(:,4) );

lp = log( ytdata(:,5) );

lo = log( ytdata(:,6) );

sdum = ytdata(:,7:17);

t = length(ytdata);

% Construct variables for use in VAR

% interest rate (is not multiplied with 100 as this is not annual growth rate like the others) and the annual percentage growth rates in money, price and output

yvar = [ r lm lp lo ];

% It is simple annual growth rate in chain multiplication format

% We have 3 columns of data which are the growth rates only...We are

% calculating the growth rate by simply omitting the first and last 12

% data which is 12 months (annual) data and than by

% deductions we are calculating the annual rate of growth.

tmp = 100\*(trimr(yvar(:,2:4),12,0) - trimr(yvar(:,2:4),0,12));

%for the interest rates we are only fixing the dimension as no growth

%rate needs calculating.

y = [ trimr(yvar(:,1),12,0) tmp];

% Granger causality tests (based on the Wald test)

t = length(y);

% Obtain estimates of the VAR(2) by OLS to each equation

% Now, the trimr portions are only doing iteration by number of VAR,this process sets restrictions on the system..

% here upto 2 lags as we are calculating VAR(2). We are using ones only

% for matrix concatenation...

bols = [ones(t-2,1) trimr(y,1,1) trimr(y,0,2)]\trimr(y,2,0);

theta0 = bols(:);

% Estimate model: will converge in one step

ops = [optimset](optimset%20(1).pdf)('LargeScale','off','Display','iter');

[theta1,~,~,~,~,H] = [fminunc](fminunc.pdf)(@(b) neglog( b,y ),theta0,ops);

%The above equation is one estimation where we get the value of theta1 by applying neglog function on theta0 and fminunc and Hessian is calculated with commands set in ops…

please note that, right \ division is used when you are dividing

% between two matrix, and we do / division when we are dividing between

% scalers....this is easier to remember.

% Here H is 36 by 36 matrix...which has been generated with

% optimization option....needs further reading on this. H is simply hessian and hessian is important optimization tool….for all optimization we generally use hessian….

cov1 = H\(1/t);

% Wald test of no causality from money to interest rates

% You can put any restrictions in any of the columns, for 2 by 36

% column, it is good to maintain a sequence, such as 1-5,2-7,3-8 or

% 0-4,1-5,2-7,3-9 etc...please go to all to one variable

r = zeros(2,36);

r(1,3) = 1.0;

r(2,7) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta0 - q)'\*inv(r\*cov1\*r')\*(r\*theta0 - q);

dof = size(r,1);

disp(' ')

disp(['Wald test (money to interest rates) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from prices to interest rates

r = zeros(2,36);

r(1,4) = 1.0;

r(2,8) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

disp(' ')

disp(['Wald test (prices to interest rates) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from output to interest rates

r = zeros(2,36);

r(1,5) = 1.0;

r(2,9) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

disp(' ')

disp(['Wald test (output to interest rates) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% The fun part is when you test for no causality from all, you will keep

% the restrictions that you put in the column vectors earlier exactly the

% same, but number of rows will increase...as if you are concatenating the

% single causalities by rows.

% Wald test of no causality from all to interest rates

r = zeros(2,36);

r(1,3) = 1.0;

r(2,7) = 1.0;

r(3,4) = 1.0;

r(4,8) = 1.0;

r(5,5) = 1.0;

r(6,9) = 1.0;

q = [ 0 ; 0 ; 0 ; 0 ; 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,1);

disp(' ')

disp(['Wald test (all to interest rates) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from interest rates to money

r = zeros(2,36);

r(1,11) = 1.0;

r(2,15) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,2);

disp(' ')

disp(['Wald test (interest rates to money) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from prices to money

r = zeros(2,36);

r(1,13) = 1.0;

r(2,17) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,2);

disp(' ')

disp(['Wald test (prices to money) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from output to money

r = zeros(2,36);

r(1,14) = 1.0;

r(2,18) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,2);

disp(' ')

disp(['Wald test (output to money) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from all to money

r = zeros(2,36);

r(1,11) = 1.0;

r(2,15) = 1.0;

r(3,13) = 1.0;

r(4,17) = 1.0;

r(5,14) = 1.0;

r(6,18) = 1.0;

q = [ 0 ; 0 ; 0 ; 0 ; 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,1);

disp(' ')

disp(['Wald test (all to money) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from interest rates to prices

r = zeros(2,36);

r(1,20) = 1.0;

r(2,24) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,2);

disp(' ')

disp(['Wald test (interest rates to prices) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from money to prices

r = zeros(2,36);

r(1,21) = 1.0;

r(2,25) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,2);

disp(' ')

disp(['Wald test (money to prices) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from output to prices

r = zeros(2,36);

r(1,23) = 1.0;

r(2,27) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,2);

disp(' ')

disp(['Wald test (output to prices) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from all to prices

r = zeros(2,36);

r(1,20) = 1.0;

r(2,24) = 1.0;

r(3,21) = 1.0;

r(4,25) = 1.0;

r(5,23) = 1.0;

r(6,27) = 1.0;

q = [ 0 ; 0 ; 0 ; 0 ; 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,1);

disp(' ')

disp(['Wald test (all to prices) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from interest rates to output

r = zeros(2,36);

r(1,29) = 1.0;

r(2,33) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,2);

disp(' ')

disp(['Wald test (interest rates to output) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from money to output

r = zeros(2,36);

r(1,30) = 1.0;

r(2,34) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,2);

disp(' ')

disp(['Wald test (money to output) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from prices to output \*\*/

r = zeros(2,36);

r(1,31) = 1.0;

r(2,35) = 1.0;

q = [ 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,2);

disp(' ')

disp(['Wald test (prices to output) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

disp(' ')

% Wald test of no causality from all to output

r = zeros(2,36);

r(1,29) = 1.0;

r(2,33) = 1.0;

r(3,30) = 1.0;

r(4,34) = 1.0;

r(5,31) = 1.0;

r(6,35) = 1.0;

q = [ 0 ; 0 ; 0 ; 0 ; 0 ; 0 ];

wd = (r\*theta1 - q)'\*inv(r\*cov1\*r')\*(r\*theta1 - q);

dof = size(r,1);

disp(' ')

disp(['Wald test (all to output) = ',num2str(wd) ]);

disp(['Number of degrees of freedom = ',num2str(dof) ]);

disp(['p-value = ',num2str(1-chi2cdf(wd,dof)) ]);

end

%--------------------------- Functions ----------------------------------

%

%--------------------------------------------------------------------------

% Log-likelihood function for an unrestricted VARMA model

%--------------------------------------------------------------------------

function f = neglog( b,y )

% both t and n are the size of y

[ t,n ] = size( y );

v = zeros( t,4 );

lf = zeros( t-2,1 );

% First loop over MA part

% split each part in 10 lags.. that will maintain consistency with the

% previous restrictions set. Here i is starting with 3v lags as

% mentioned in the question.And yes, please notice that all y matrix is

% going upto a maximum of 4 columns and not more because we set v to be

% a 4 column matrix in the beginning. And so each time we reach 4th

% column we increase lag ,(i-1) turns to (i-2).. Also notice, you had 2

% rows and 36 columns, here, the coefficient ‘b’ will continue to go

% upto 36 columns, so for 4 variables we have 4 v's or for 3 lags we

% have 3+1 =t or 4 v's. we split each v by 10 as we did for the earlier

% part of restrictions in the wald tests, and we end our calculations

% in 36...we also take i upto 4.

for i = 3:t

v(i,1) = y(i,1) - b(1) - b(2)\*y(i-1,1) - b(3)\*y(i-1,2) - ...

b(4)\*y(i-1,3) - b(5)\*y(i-1,4) - b(6)\*y(i-2,1) - ...

b(7)\*y(i-2,2) - b(8)\*y(i-2,3) - b(9)\*y(i-2,4);

v(i,2) = y(i,2) - b(10) - b(11)\*y(i-1,1) - b(12)\*y(i-1,2) - ...

b(13)\*y(i-1,3) - b(14)\*y(i-1,4) - b(15)\*y(i-2,1) - ...

b(16)\*y(i-2,2) - b(17)\*y(i-2,3) - b(18)\*y(i-2,4);

v(i,3) = y(i,3) - b(19) - b(20)\*y(i-1,1) - b(21)\*y(i-1,2) - ...

b(22)\*y(i-1,3) - b(23)\*y(i-1,4) - b(24)\*y(i-2,1) - ...

b(25)\*y(i-2,2) - b(26)\*y(i-2,3) - b(27)\*y(i-2,4);

v(i,4) = y(i,4) - b(28) - b(29)\*y(i-1,1) - b(30)\*y(i-1,2) - ...

b(31)\*y(i-1,3) - b(32)\*y(i-1,4) - b(33)\*y(i-2,1) - ...

b(34)\*y(i-2,2) - b(35)\*y(i-2,3) - b(36)\*y(i-2,4);

end

v = trimr( v,2,0 );

vc = v'\*v/length(v);

for i = 1:t-2;

lf(i) = -0.5\*n\*log(2\*pi) - 0.5\*log(det(vc)) ...

- 0.5\*v(i,:)\*inv(vc)\*v(i,:)';

end

f = -mean( lf );

end

r= (2,36)

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]

Putting restrictions:

Wald test of No Causality

Money -> interest rate ((1,3),(2,7))

[ 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]

Wald test of No Causality

Price-> interest rate ((1,4),(2,8))

[ 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]

Wald test of No Causality

Output-> interest rate ((1,5),(2,9))

[ 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]

Wald test of No Causality

All-> interest rate

r(1,3) = 1.0;

r(2,7) = 1.0;

r(3,4) = 1.0;

r(4,8) = 1.0;

r(5,5) = 1.0;

r(6,9) = 1.0;

[ 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]

Wald test of No Causality

Interest Rate -> Money ((1,11),(2,15))

[ 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]

Wald test of No Causality

Prices -> Money ((1,13),(2,17))

[ 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]

Wald test of No Causality

Output -> Money ((1,14),(2,18))

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]

From all to money

r(1,11) = 1.0;

r(2,15) = 1.0;

r(3,13) = 1.0;

r(4,17) = 1.0;

r(5,14) = 1.0;

r(6,18) = 1.0;

interest rates to prices

r(1,20) = 1.0;

r(2,24) = 1.0;

money to prices

r(1,21) = 1.0;

r(2,25) = 1.0;

output to prices

r(1,23) = 1.0;

r(2,27) = 1.0;

all to prices

r(1,20) = 1.0;

r(2,24) = 1.0;

r(3,21) = 1.0;

r(4,25) = 1.0;

r(5,23) = 1.0;

r(6,27) = 1.0;

interest rates to output

r(1,29) = 1.0;

r(2,33) = 1.0;

money to output

r(1,30) = 1.0;

r(2,34) = 1.0;

prices to output

r(1,31) = 1.0;

r(2,35) = 1.0;

all to output

r(1,29) = 1.0;

r(2,33) = 1.0;

r(3,30) = 1.0;

r(4,34) = 1.0;

r(5,31) = 1.0;

r(6,35) = 1.0;

**Impulse response and variance decomposition**

%=========================================================================

%

% Recursive structural model estimated in four ways

%

%=========================================================================

function stsm\_recursive( )

clear all

clc

% Read the data: quarterly US data from Jan-1959 to Dec-1998

load simsdata.mat

% Define variables

r = ytdata(:,1);

lex = log( ytdata(:,2) );

lcp = log( ytdata(:,3) );

lm = log( ytdata(:,4) );

lp = log( ytdata(:,5) );

lo = log( ytdata(:,6) );

sdum = ytdata(:,7:17);

t = length(ytdata);

% Construct variables for use in VAR

% interest rate and the annual percentage growth rates in money, price and output

yvar = [ r lm lp lo ];

tmp = 100\*(trimr(yvar(:,2:4),12,0) - trimr(yvar(:,2:4),0,12));

yvar = [ trimr(yvar(:,1),12,0) tmp];

lags = [trimr(yvar,1,1) trimr(yvar,0,2) ];

% Structural equation approach ..Book page 493-500…

y = trimr(yvar(:,1),2,0);

x = [ones(length(y),1) lags ];

a1 = x\y;

u1 = y - x\*a1;

y = trimr(yvar(:,2),2,0);

x = [ ones(length(y),1) trimr(yvar(:,1),2,0) lags ];

a2 = x\y;

u2 = y - x\*a2;

y = trimr(yvar(:,3),2,0);

x = [ ones(length(y),1) trimr(yvar(:,[1 2]),2,0) lags ];

a3 = x\y;

u3 = y - x\*a3;

y = trimr(yvar(:,4),2,0);

x = [ ones(length(y),1) trimr(yvar(:,[1 2 3]),2,0) lags ];

a4 = x\y;

u4 = y - x\*a4;

% Structural residuals and covariance

b01 = [ 1 0 0 0;

-a2(2) 1 0 0;

-a3(2) -a3(3) 1 0;

-a4(2) -a4(3) -a4(4) 1 ];

u = [u1 u2 u3 u4 ];

d1 = u'\*u/length(u);

% Reduced form approach

y = trimr(yvar,2,0);

x = [ones(length(yvar)-2,1) lags ];

bar = x\y;

v = y - x\*bar;

u1 = v(:,1);

a2 = v(:,1)\v(:,2);

u2 = v(:,2) - v(:,1)\*a2;

a3 = v(:,[1 2])\v(:,3);

u3 = v(:,3) - v(:,[1 2])\*a3;

a4 = v(:,[1 2 3])\v(:,4);

u4 = v(:,4) - v(:,[1 2 3])\*a4;

% Structural residuals and covariance

b02 = [ 1 0 0 0;

-a2(1) 1 0 0;

-a3(1) -a3(2) 1 0;

-a4(1) -a4(2) -a4(3) 1 ];

u = [u1 u2 u3 u4 ];

d2 = u'\*u/length(u);

% Choleski decomposition approach

y = trimr(yvar,2,0);

x = [ ones(length(y),1) lags ];

bar = x\y;

v = y - x\*bar;

vc = v'\*v/length(v);

s = chol(vc)';

tmp = diag(s);

b0inv = bsxfun(@rdivide,s,tmp');

b03 = inv(b0inv);

d3 = zeros(4);

d3 = diag(tmp.^2);

% MLE approach

ops = optimset('LargeScale','off','Display','iter');

theta0 = 0.1\*ones(10,1);

% Converge in one iteration

%theta0 = [-b01(2,1) ; -b01(3,1) ; -b01(3,2) ; -b01(4,1) ; -b01(4,2) ; -b01(4,3) ; diag(d1) ];

[ theta,fval ] = fminunc(@(theta) neglog(theta,y,v),theta0,ops);

lf = -fval;

disp(['Log-likelihood value = ',num2str(lf) ]);

disp(['T x Log-likelihood value = ' num2str(length(v)\*lf) ]);

b04 = [ 1 0 0 0 ;

-theta(1) 1 0 0 ;

-theta(2) -theta(3) 1 0 ;

-theta(4) -theta(5) -theta(6) 1 ];

d4 = zeros(4);

d4 = diag(theta(7:10));

disp(' ')

disp( 'B0: structural equation approach' )

disp( b01 )

disp( 'B0: reduced form approach' )

disp( b02 )

disp( 'B0: Choleski decomposition approach' )

disp( b03 )

disp( 'B0: maximum likelihood approach' )

disp( b04 )

disp(' ')

disp( 'D: structural equation approach' )

disp( d1 )

disp( 'D: reduced form approach' )

disp( d2 )

disp( 'D: Choleski decomposition approach' )

disp( d3 )

disp( 'D: maximum likelihood approach' )

disp( d4 )

end

%--------------------------- Functions ----------------------------------

%

%--------------------------------------------------------------------------

% Log-likelihood function for the VAR

%--------------------------------------------------------------------------

function f = neglog(b,y,v)

t = length(v);

n = size(y,2);

lnl = zeros(t,1);

b0 = [ 1 0 0 0 ;

-b(1) 1 0 0 ;

-b(2) -b(3) 1 0 ;

-b(4) -b(5) -b(6) 1 ];

% Structural residual variances

d = eye(n);

d = diag(abs(b(7:10)));

vc = inv(b0)\*d\*inv(b0)';

for i=1:t

lnl(i) = -0.5\*n\*log(2\*pi) - 0.5\*log(det(vc)) - 0.5\*v(i,:)\*inv(vc)\*v(i,:)';

end

f = -mean( lnl );

end

%=========================================================================

%

% Recursive structural model estimated in four ways

%Page 494 in book, page 532 in search

%

%=========================================================================

function stsm\_recursive( )

clear all

clc

% Read the data: quarterly US data from Jan-1959 to Dec-1998

load simsdata.mat

% Define variables

r = ytdata(:,1);

lex = log( ytdata(:,2) );

lcp = log( ytdata(:,3) );

lm = log( ytdata(:,4) );

lp = log( ytdata(:,5) );

lo = log( ytdata(:,6) );

sdum = ytdata(:,7:17);

% Here t(length) is simply the length of the rows in total

t = length(ytdata);

% Construct variables for use in VAR

% interest rate and the annual percentage growth rates in money, price and output

yvar = [ r lm lp lo ];

% Interestingly, this is a recursion as we are trimming the first anb

% final and subtracting them we are getting a measure of rate of growth

% only and nothing else

tmp = 100\*(trimr(yvar(:,2:4),12,0) - trimr(yvar(:,2:4),0,12));

% Now we are making the yvar complete by adding the interest rate on

% the front column where no growth rate is calculated.

yvar = [ trimr(yvar(:,1),12,0) tmp];

% we just use the lags later.

lags = [trimr(yvar,1,1) trimr(yvar,0,2) ];

% Structural equation approach

% Now the fun begins. we are starting with y being ineterst rate only

y = trimr(yvar(:,1),2,0);

%here, the first column is the column of ones only.and the rest two

%columns are of lags that we will use for structutal shcoks calculation

%one by one.

x = [ones(length(y),1) lags ];

% its a tricky part. a1 is the structural shocks similar to book page

% 494. The division is giving the impact of interest rate to others and

% we get just one column matrix.

a1 = x\y;

% Now we have the disturbance term by redoing the equation of y. This

% tells us, how deviated y is becoming due to others shocks in the

% system.

u1 = y - x\*a1;

y = trimr(yvar(:,2),2,0);

% the x gets bigger having an additional column as now interest

% rate... is added to the system to calculate impact of interest

% rate... on money. Or its simply following the second euqtion of the

% 4 equation in book page 494, equation 13,40. The structural

% approach.

x = [ ones(length(y),1) trimr(yvar(:,1),2,0) lags ];

a2 = x\y;

u2 = y - x\*a2;

y = trimr(yvar(:,3),2,0);

x = [ ones(length(y),1) trimr(yvar(:,[1 2]),2,0) lags ];

a3 = x\y;

u3 = y - x\*a3;

y = trimr(yvar(:,4),2,0);

x = [ ones(length(y),1) trimr(yvar(:,[1 2 3]),2,0) lags ];

a4 = x\y;

u4 = y - x\*a4;

% Structural residuals and covariance

% Simply check out the page 494 equation 13.41 for all the estimations

% ..from here

b01 = [ 1 0 0 0;

-a2(2) 1 0 0;

-a3(2) -a3(3) 1 0;

-a4(2) -a4(3) -a4(4) 1 ];

u = [u1 u2 u3 u4 ];

% remember, from book page 496, E[u'\*u] = D

d1 = u'\*u/length(u);

% Reduced form approach

% Always look into book page 495 for detailed discussion.

y = trimr(yvar,2,0);

% X like before. Lags is already 8 columns..Just subtracting all by 2

% ..for it is VAR(2).

x = [ones(length(yvar)-2,1) lags ];

bar = x\y;

v = y - x\*bar;

% Now here, in every step the equation increases following the approach

% in page 495. Just take the system of shocks a1,a2,... to the right

% hand side and thats it....very easy....trciks lie in a2 only . Here

% u1,u2,u3 are the structutal residulas we need for estimation of the

% equations.

u1 = v(:,1);

a2 = v(:,1)\v(:,2);

u2 = v(:,2) - v(:,1)\*a2;

a3 = v(:,[1 2])\v(:,3);

u3 = v(:,3) - v(:,[1 2])\*a3;

a4 = v(:,[1 2 3])\v(:,4);

u4 = v(:,4) - v(:,[1 2 3])\*a4;

% Structural residuals and covariance

b02 = [ 1 0 0 0;

-a2(1) 1 0 0;

-a3(1) -a3(2) 1 0;

-a4(1) -a4(2) -a4(3) 1 ];

u = [u1 u2 u3 u4 ];

d2 = u'\*u/length(u);

% Choleski decomposition approach

% Now this is really easy

% step 1: compute V as in the reduced form approach

y = trimr(yvar,2,0);

x = [ ones(length(y),1) lags ];

bar = x\y;

v = y - x\*bar;

% Step 2: E[u'\*u] = D , and V= D\*inv(B0)\*inv(B0)' book page 495 and

% 496..

vc = v'\*v/length(v);

s = chol(vc)';

tmp = diag(s);

% step 3: simple, page 496(book), right arrray matrix divide,

% B0 = (S \ D)', here we are deviding S with diagonal elements of S...

% which is D and than invesring it in b03. d3 is simply doing the

% same.. thing again, finding D by squaring diagonal elements of D.

b0inv = bsxfun(@rdivide,s,tmp');

b03 = inv(b0inv);

% first we are creating a zeros matrix to make space for teh arrays..a

% and than we are creating the diagonal matrix so that we can only

% place the diagonal elements in a zero matrix and have a proper

% structure of diagonal matrix. we have values for only digonal elements...

%good use of zeros.

%

d3 = zeros(4);

d3 = diag(tmp.^2);

% MLE approach

ops = optimset('LargeScale','off','Display','iter');

theta0 = 0.1\*ones(10,1);

% Converge in one iteration

%theta0 = [-b01(2,1) ; -b01(3,1) ; -b01(3,2) ; -b01(4,1) ; -b01(4,2) ; -b01(4,3) ; diag(d1) ];

[ theta,fval ] = fminunc(@(theta) neglog(theta,y,v),theta0,ops);

lf = -fval;

disp(['Log-likelihood value = ',num2str(lf) ]);

disp(['T x Log-likelihood value = ' num2str(length(v)\*lf) ]);

b04 = [ 1 0 0 0 ;

-theta(1) 1 0 0 ;

-theta(2) -theta(3) 1 0 ;

-theta(4) -theta(5) -theta(6) 1 ];

d4 = zeros(4);

d4 = diag(theta(7:10));

disp(' ')

disp( 'B0: structural equation approach' )

disp( b01 )

disp( 'B0: reduced form approach' )

disp( b02 )

disp( 'B0: Choleski decomposition approach' )

disp( b03 )

disp( 'B0: maximum likelihood approach' )

disp( b04 )

disp(' ')

disp( 'D: structural equation approach' )

disp( d1 )

disp( 'D: reduced form approach' )

disp( d2 )

disp( 'D: Choleski decomposition approach' )

disp( d3 )

disp( 'D: maximum likelihood approach' )

disp( d4 )

end

%--------------------------- Functions ----------------------------------

%

%--------------------------------------------------------------------------

% Log-likelihood function for the VAR

%--------------------------------------------------------------------------

function f = neglog(b,y,v)

t = length(v);

n = size(y,2);

lnl = zeros(t,1);

b0 = [ 1 0 0 0 ;

-b(1) 1 0 0 ;

-b(2) -b(3) 1 0 ;

-b(4) -b(5) -b(6) 1 ];

% Structural residual variances

d = eye(n);

d = diag(abs(b(7:10)));

vc = inv(b0)\*d\*inv(b0)';

for i=1:t

lnl(i) = -0.5\*n\*log(2\*pi) - 0.5\*log(det(vc)) - 0.5\*v(i,:)\*inv(vc)\*v(i,:)';

end

f = -mean( lnl );

end

%=========================================================================

%

% Program to reproduce Diebold and Yilmaz (2009) spillover model

%=========================================================================

stsm\_DIY

clc

clear all

load('Raisul-1.mat');

% Choose the data

y = Raisulindex1;

z = bsxfun(@rdivide,bsxfun(@minus,y,mean(y)),std(y));

disp(' Mean Median Max Min Std.dev. Skew Kurt')

disp( [ mean(y)' median(y)' max(y)' min(y)' std(y)' mean(z.^3)' mean(z.^4)'] )

% Estimate a VAR with lag p=2

x = [ones(length(y)-2,1) trimr(y,1,1) trimr(y,0,2) ];

b = x\trimr(y,2,0);

%% Taking total number of rows and columns

% When we put the size command it automatically places the no of rows in element

% 1 of [] and no of columns in element 2 of []

[rcnt, ccnt] = size(b)

%% need to fix values for these variables

mu = trimr(b,0,rcnt-1)'; % Vector of intercepts

phi1 = trimr(b,1,ccnt)'; % Lag 1 parameter estimates

phi2 = trimr(b,ccnt+1,0)'; % Lag 2 parameter estimates

%% Generate VMA (non-orthogonalized) for horizons 1 to 10

%here is si1, we are asking to return the second dimension of y first and

%then we are putting identity matrix eye command ..Easy, if we put size(y)

%only it will retuurn 6701 (rcnt) because it is a 6701 by 8 matrix.but id

%we put size (y,2) it will return the ccnt or 8....now we are converting 8

%into a 8 by 8 identity matrix with eye command....why?... because we are

%making place with 0's to put scalers in the next steps while keeping restrictions

%or 1 in the diagonal so that we can measure the impact of one variable to

%others....

si1 = eye(size(y,2));

% in the next step we continue to create 8 by 8 matrix...the earlier

% commands helpmed me to create dimensions and fix it...

si2 = phi1;

si3 = phi1\*si2 + phi2;

si4 = phi1\*si3 + phi2\*si2;

si5 = phi1\*si4 + phi2\*si3;

si6 = phi1\*si5 + phi2\*si4;

si7 = phi1\*si6 + phi2\*si5;

si8 = phi1\*si7 + phi2\*si6;

si9 = phi1\*si8 + phi2\*si7;

si10 = phi1\*si9 + phi2\*si8;

% Generate VMA (orthogonalized) for horizons 1 to 10

v = trimr(y,2,0) - x\*b; % VAR residuals

% in a 3 by 7 matrix length(X) gives 7 (ccnt).....helps to convert big

% matrix to the number of its columns....

vc = v'\*v/length(v);

% d is giving me the diagonal values of vc only

d = diag(vc);

% as soon as we take cholesky decomposition, it starts to decompose....it

% turns the upper dimension of the diagonal to 0s, restructuring the

% matrix...

s = chol(vc)';

ir1 = si1\*s;

ir2 = si2\*s;

ir3 = si3\*s;

ir4 = si4\*s;

ir5 = si5\*s;

ir6 = si6\*s;

ir7 = si7\*s;

ir8 = si8\*s;

ir9 = si9\*s;

ir10 = si10\*s;

% Compute variance decompositions for horizons 1 to 10

% If A is a matrix, then sum(A) returns a row vector containing the sum of

% each column. 1 by n row matrix

% sum(vd1,2) = X means X returns the sum of all rows of vd1. it becomes 1 column matrix...

% S = sum(A,dim) returns the sum along dimension dim. For example, if A is a matrix, then sum(A,2) is a column vector containing the sum of each row.

%n by 1 matrix

vd1 = ir1.^2;

vd1 = 100\*bsxfun(@rdivide,vd1,sum(vd1,2));

vd2 = ir1.^2 + ir2.^2;

vd2 = 100\*bsxfun(@rdivide,vd2,sum(vd2,2));

vd3 = ir1.^2 + ir2.^2 + ir3.^2;

vd3 = 100\*bsxfun(@rdivide,vd3,sum(vd3'));

vd4 = ir1.^2 + ir2.^2 + ir3.^2 + ir4.^2;

vd4 = 100\*bsxfun(@rdivide,vd4,sum(vd4'));

vd5 = ir1.^2 + ir2.^2 + ir3.^2 + ir4.^2 + ir5.^2;

vd5 = 100\*bsxfun(@rdivide,vd5,sum(vd5'));

vd6 = ir1.^2 + ir2.^2 + ir3.^2 + ir4.^2 + ir5.^2 + ir6.^2;

vd6 = 100\*bsxfun(@rdivide,vd6,sum(vd6'));

vd7 = ir1.^2 + ir2.^2 + ir3.^2 + ir4.^2 + ir5.^2 + ir6.^2 + ir7.^2;

vd7 = 100\*bsxfun(@rdivide,vd7,sum(vd7'));

vd8 = ir1.^2 + ir2.^2 + ir3.^2 + ir4.^2 + ir5.^2 + ir6.^2 + ir7.^2 + ir8.^2;

vd8 = 100\*bsxfun(@rdivide,vd8,sum(vd8'));

vd9 = ir1.^2 + ir2.^2 + ir3.^2 + ir4.^2 + ir5.^2 + ir6.^2 + ir7.^2 + ir8.^2 + ir9.^2;

vd9 = 100\*bsxfun(@rdivide,vd9,sum(vd9'));

vd10 = ir1.^2 + ir2.^2 + ir3.^2 + ir4.^2 + ir5.^2 + ir6.^2 + ir7.^2 + ir8.^2 + ir9.^2 + ir10.^2;

vd10 = 100\*bsxfun(@rdivide,vd10,sum(vd10,2));

%% Change the country list

% 1. Australia

% 2. USA

% 3. Indonesia

% 4. Japan

% 5. Malaysia

% 6. Phillipine

% 7. China

% 8. Thailand

str = [ 'AUS ' ;

'USA ' ;

'INDO ' ;

'JPN ' ;

'MAL ' ;

'PHIL ' ;

'CHN ' ;

'THL ' ];

disp('Variance decomposition at period 10')

disp(vd10);

% easy, take the summation of all rows and remove the diagonal values, it

% will away the contribution of the variable on itself...

tmp=sum(vd10,2)-diag(vd10);

disp('Contribution From Others')

disp( [str num2str(tmp) ]);

% Its itsybitsy....now, we are taking 1 row by summing all elements in

% columns and than subtracting from the diagonal matrix...it gives the

% impact of the vartiable of interet to others....

tmp2=(sum(vd10)-diag(vd10)')';

disp('Contribution To Others')

disp( [str num2str(tmp2) ]);