Problem 1

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

%

% Program to do LR, Wald and LM tests of a simple regression model

%

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

clear all;

clc;

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

t = 5;

x = [1,2,4,5,8]';

beta = 1;

sig = 2;

% Generate data

u = sig\*randn(t,1);

y = x\*beta + u;

% Constrained estimation

% its easy as hell. bhat0 is the restriction assigned on beta-hat-0. Page 122 case 1.

%the value of sigma calculation is given in book page 125, under

%unrestricted maximum likelihood estimators. The log-likelihood function is

%also given in the same page. Also turn to page 128.

bhat0 = 0.0;

sig20 = mean((y - x\*bhat0).^2);

logL0 = - 1/2\*log(2\*pi) - 1/2\*log(sig20)...

- 1/(2\*sig20\*t)\*sum((y - x\*bhat0).^2);

%fprintf function transforms the reults in text format and saves it.

fprintf('Constrained estimation results \n')

fprintf('beta\_hat\_0 = %3.4f \n',bhat0)

fprintf('sigma2\_hat\_0 = %3.4f \n',sig20)

fprintf('logL\_0 = %3.4f \n',logL0)

fprintf('\n')

% Unconstrained estimation

% its the basic beta equation derives from y=x\*beta+error. This is left

% array division. x = B.\A divides each element of A by the corresponding element of B.

bhat1 = x\y;

err2 = (y - x\*bhat1).^2;

sig21 = mean(err2);

logL1 = - 1/2\*log(2\*pi) - 1/2\*log(sig21)...

- 1/(2\*sig21\*t)\*sum((y - x\*bhat1).^2);

fprintf('Unconstrained estimation results \n')

fprintf('beta\_hat\_1 = %3.4f \n',bhat1)

fprintf('sigma2\_hat\_1 = %3.4f \n',sig21)

fprintf('logL\_1 = %3.4f \n',logL1)

fprintf('\n')

% Likelihood ratio test

%page 126

lr = -2\*t\*(logL0 - logL1);

pv = 1-chi2cdf(lr,1);

disp('Likelihood ratio test')

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

disp( ['p-value = ' num2str(pv) ] );

disp(' ')

% Wald test

r = [ 1 0 ];

q = 0 ;

theta1 = [bhat1; sig21];

%The following is information matrix. The identity mnatrix equation is found

%in page 121, the last equation of the page.

%omega1 = sig21\*inv(x'\*x);

i = [ (x'\*x)/(t\*sig21) 0; 0 1/(2\*(sig21)^2) ];

w = t\*(r\*theta1 - q)'\*inv(r\*i\*r')\*(r\*theta1 - q);

pv = chi2cdf(w,1);

disp('Wald test')

disp( ['Unconstrained estimates = ' num2str( [bhat1 sig21 ] ) ] );

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

disp( ['p-value = ' num2str(pv) ] );

disp(' ')

% LM test analytic derivatives

%Following is the gradient vector. The equation is found in book page 19.

%Replace the yt-mue values with y-xbhat0....

g = [ x'\*(y-x\*bhat0)/(t\*sig20); ...

-(1/(2\*sig20))+(1/(2\*(sig20)^2\*t))\*sum( (y - x\*bhat0).^2 ) ];

%Following is the information matrix

i = [ (x'\*x)/(t\*sig20) 0; 0 1/(2\*(sig20)^2) ];

% the lagrangean multiplier equation can be found in page 131, equation 4,12.

lm = t\*g'\*inv(i)\*g;

pv = chi2cdf(lm,1);

disp('LM test')

disp( ['Constrained estimates = ' num2str( [0 sig20 ] ) ] );

disp( ['LM stat = ' num2str(lm) ] );

disp( ['p-value = ' num2str(pv) ] );

disp(' ')

Problem 2

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

%

% Program to do LR, Wald and LM test of Weibull Distribution

%

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

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

%

% Program to do LR, Wald and LM test of Weibull Distribution

%

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

function test\_weibull( )

clear all

clc

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

% Simulate data

t = 20;

alpha = 1;

beta = 2;

%wblrnd: Weibull random numbers.

%R = wblrnd(A,B) generates random numbers for the Weibull distribution with scale parameter, A and shape parameter, B. The input arguments A and B can be either scalars or matrices. A and B, can be vectors, matrices, or multidimensional arrays that all have the same size. A scalar input is expanded to a constant array of the same size as the other input.

%R = wblrnd(A,B,m,n,...) or R = wblrnd(A,B,[m,n,...]) generates an m-by-n-by-... array. The A, B parameters can each be scalars or arrays of the same size as R.

%y = wblrnd(alpha,beta,t,1);

%we get t rows and 1 column of weibull random numbers.

% or load data

y = [0.293, 0.589, 1.374, 0.954, 0.608, 1.199, 1.464, ...

0.383, 1.743, 0.022, 0.719, 0.949, 1.888, 0.754, ...

0.873, 0.515, 1.049, 1.506, 1.090, 1.644]';

% Here we are preparing for unrestricted maximum likelihood estiamates and the value of the log likelihood function.

bstart = [alpha; beta];

options = optimset('Display','off',...

'LargeScale','off');

% Unconstrained estimation

[bu,logL1] = fminunc(@(b) loglu(b,y),bstart,options);

logL1 = -logL1;

disp(' ');

disp('Unconstrained estimation results');

disp( ['alpha = ' num2str(bu(1)) ] );

disp( ['beta = ' num2str(bu(2)) ] );

disp( ['log L = ' num2str(t\*logL1) ] );

fprintf('\n')

% Constrained estimation

% instead of beta we use 1, which is the crestriction assigned.

[b0,logL0] = fminunc(@(b) loglc(b,y),1,options);

logL0 = -logL0;

disp(' ')

disp('Constrained estimation results');

disp( ['alpha = ' num2str(b0) ] );

disp( ['beta = ' num2str(1) ] );

disp( ['log L = ' num2str(t\*logL0) ] );

% Likelihood ratio test

lr = -2\*t\*(logL0 - logL1);

p = 1 - chi2cdf(lr,1);

disp('Likelihood ratio test')

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

disp( ['p-value = ' num2str(p) ] );

disp( ' ' );

% Wald test

%numhess computes finite difference hessian.

h = numhess(@loglu,bu,y);

disp('Hessian evaluated at unconstrained estimates');

disp( h );

disp( ' ' );

r = [ 0 1 ];

q = 1;

% bu and hess has been evaluated using optimzation or likelihood

% estimation method.

w = t\*(r\*bu - q)'\*inv(r\*h\*r')\*(r\*bu - q);

p = 1 - chi2cdf(w,1);

disp('Wald test')

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

disp( ['p-value = ' num2str(p) ] );

disp( ' ' );

% LM test

th0 = [b0; 1];

%numgrad Computes numerical gradient at each observation...page

%132...equation 4.15...

gmat = numgrad(@logltu,th0,y);

g = mean(gmat)';

j = gmat'\*gmat/t;

lm = t\*g'\*inv(j)\*g;

p = 1 - chi2cdf(lm,1);

disp('LM test')

disp( ['LM stat = ' num2str(lm) ] );

disp( ['p-value = ' num2str(p) ] );

end

%

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

%

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

% Unconstrained likelihood function

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

%here we take the mean of unrestricted log-likelihood function at each

%observation.

function logl = loglu(b,y)

logl = -mean( logltu(b,y) );

end

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

% Unconstrained likelihood function at each observation

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

%the unrestricted log likelihood function is found in page 128 the first

%equation..

function lt = logltu(b,y)

alpha = b(1);

beta = b(2);

lt = log(alpha)+log(beta)+(beta-1)\*log(y)- alpha\*y.^(beta);

end

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

% Constrained likelihood function

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

function logl = loglc(b,y)

logl = -mean(logltc(b,y));

end

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

% Constrained likelihood function at each observation

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

% we are just changing the value of beta from 2 to 1.

function lt = logltc(alpha,y)

beta = 1;

lt = log(alpha)+log(beta)+(beta-1)\*log(y)- alpha\*y.^(beta);

end

Problem 3

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

%

% Program to perform tests of the stationary distribution of the

% interest rate based on the gamma distribution

%

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

function test\_interest( )

clear all

clc

% Load data (5505x4 array called eurodata, 1 Jun 1973 - 25 Feb 1995)

% 1. year

% 2. day

% 3. date stamp

% 4. interest rates

load eurodollar.mat

r = eurodata(:,4);

t = length(r);

% Estimate the model

start = [1 ; 1];

ops = optimset('LargeScale', 'off', 'Display', 'off');

[bhat,~,~,~,~,hess] = fminunc(@(b) neglog(b,r),start,ops);

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

nu = bhat(1);

omega = bhat(2);

disp( 'Parameter estimates')

disp( ['nu = ',num2str(nu) ]);

disp( ['omega = ',num2str(omega) ]);

disp(' ')

disp( 'Hessian matrix')

disp( hess );

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

% Estimate the mean and its standard error

mu = nu/omega;

d = [(1/omega) (-nu/omega^2)];

vr = d\*vc\*d';

se = sqrt(vr);

disp(' ')

disp(['Mean = ',num2str(mu) ]);

disp(['Std Error of Mean = ',num2str(se) ]);

% Wald test of the mean

c = mu;

q = 0.1;

wd = t\*(c - q)'\*inv(d\*(inv(hess))\*d')\*(c - q);

disp(' ')

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

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

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

% Estimate the variance and its standard error

s2 = nu/omega^2;

d = [(1/omega^2) (-2\*nu/omega^3)];

vr = d\*vc\*d';

se = sqrt(vr);

disp(' ')

disp(['Variance = ',num2str(s2) ]);

disp(['Std Error of Variance = ',num2str(se) ]);

% Wald test of the variance

c = s2;

q = 0.001;

wd = t\*(c - q)'\*inv(d\*(inv(hess))\*d')\*(c - q);

disp(' ')

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

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

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

% Estimate skewness and its standard error

k3 = 2/sqrt(nu);

d = [(-1/nu^(3/2)) 0];

vr = d\*vc\*d';

se = sqrt(vr);

disp(' ')

disp(['Skewness = ',num2str(k3) ]);

disp(['Std Error of skewness = ',num2str(se) ]);

% Wald test of the skewness

c = k3;

q = 0.0;

wd = t\*(c - q)'\*inv(d\*(inv(hess))\*d')\*(c - q);

disp(' ')

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

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

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

% Estimate kurtosis and its standard error

k4 = 3 + 6/nu;

d = [-6/nu^2 0];

vr = d\*vc\*d';

se = sqrt(vr);

disp(' ')

disp(['Kurtosis = ',num2str(k4) ]);

disp(['Std Error of kurtosis = ',num2str(se) ]);

% Wald test of the kurtosis

c = k4;

q = 3;

wd = t\*(c - q)'\*inv(d\*(inv(hess))\*d')\*(c - q);

disp(' ')

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

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

end

%

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

%

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

% Likelihood function for stationary distribution of CIR model

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

function lf = neglog(b,y)

nu = abs(b(1));

om = abs(b(2));

f = nu\*log( om ) - gammaln( nu ) + (nu-1)\*log(y) - om\*y;

lf = -mean(f);

end

Problem 4

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

%

% Program to generate asymptotic distribution of the Wald test

% applied to the regression model.

%

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

function test\_asymptotic( )

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',1234) )

% Set parameter values

beta0 = 1.0;

beta1 = 0.0;

beta2 = 0.0;

beta3 = 0.0;

sig2 = 0.1;

sig = sqrt(sig2);

t = 1000;

ndraws = 10000;

x1 = rand(t,1);

x2 = randn(t,1);

x3 = randn(t,1).^2;

% Arrays to hold results

wd1 = zeros(ndraws,1);

wd2 = zeros(ndraws,1);

wd3 = zeros(ndraws,1);

% Loop over number of replications

theta0 = [beta0 ; beta1 ; beta2 ; beta3 ; sig2 ];

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

for i = 1:ndraws

u = sig\*randn(t,1);

y = beta0 + beta1\*x1 + beta2\*x2 + beta3\*x3 + u;

[p] = fminsearch(@(p) neglog(p,y,x1,x2,x3),theta0,options);

H = numhess(@neglog,p,y,x1,x2,x3);

theta = p;

cov = inv(H);

% One restriction

R = [0 1 0 0 0];

Q = 0;

wd1(i) = t\*(R\*theta - Q)'\*inv(R\*cov\*R')\*(R\*theta - Q);

% Two restrictions

R = [ 0 1 0 0 0 ;

0 0 1 0 0 ];

Q = [0 ; 0 ];

wd2(i) = t\*(R\*theta - Q)'\*inv(R\*cov\*R')\*(R\*theta - Q);

% Three restrictions

R = [ 0 1 0 0 0 ;

0 0 1 0 0 ;

0 0 0 1 0 ];

Q = [ 0 ; 0 ; 0 ];

wd3(i) = t\*(R\*theta - Q)'\*inv(R\*cov\*R')\*(R\*theta - Q);

end

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

%\*\*\*

%\*\*\* Generate graph

%\*\*\*

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

% Switch off TeX interpreter and clear figure

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

figure(1);

clf;

[fcdf,x] = ecdf( wd3 );

[f,bins] = ecdfhist(fcdf,x,21);

bar(bins,f,'hist');

%[n,xout]=hist(rt,51);

%bar(xout,n/t)

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

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

axis([0,15,-Inf,Inf])

box off

hold on

ygrid = 0.0001:0.1:15;

plot(ygrid,chi2pdf(ygrid,3),'-k','LineWidth',0.75)

ylabel('$f(W)$');

xlabel('$W$');

%set(gca,'YTick',[] );

hold off

% Print the tex file to the relevant directory

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

end

%

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

%

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

% Log-likelihood function of unconstrained model

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

function lf = neglog(theta,y,x1,x2,x3)

m = theta(1) + theta(2)\*x1 + theta(3)\*x2 + theta(4)\*x3;

s2 = abs(theta(5));

lf = -mean( -0.5\*log(2\*pi) - 0.5\*log(s2) - 0.5\*(y - m).^2/s2 ) ;

end

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

% % Computes finite difference Hessian

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

%

% function H = numhess( f,x,varargin )

%

% k = length( x );

% f0 = feval( f, x, varargin{:} );

%

% % Compute the stepsize (h)

% dx = eps.^( 1/3 )\*( abs(x) + eps );

% xh = x + dx;

% dx = xh - x;

% ee = diag( dx );

%

% % Compute forward and backward steps

% fplus = zeros( k,1 );

%

% for i=1:k

%

% fplus(i) = feval( f, x+ee(:,i), varargin{:} );

% end

%

% H = zeros( k );

% for j = 1:k

%

% for l = 1:j;

%

% H(j,l) = feval( f, x+ee(:,j)+ee(:,l), varargin{:} );

% end

% end

% H = H + tril( H,-1 )';

%

% fpp = bsxfun( @plus, fplus, fplus' );

% H = H - fpp + f0;

% H = H./( dx\*dx' );

%

% end

%

**Problem 5**

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

%

% Simulating the size of the wald test using exponential regression

% model

%

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

function test\_size( )

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',123457) )

beta0 = 1;

beta1 = 0;

t = [5,10,25,100];

ndraws = 10000;

wd = zeros(ndraws,1);

chi2 = chi2inv(0.95,1);

output = zeros(2,length(t));

size = output(1,:);

cv = output(2,:);

for l = 1:length(t)

j = 0;

x = randn(t(l),1); % Explanatory variable (fixed in repeated samples)

for i = 1:ndraws % Main do loop to generate Monte Carlo results

mue = exp(beta0 + beta1\*x); %mean

u = rand(t(l),1);

y = -mue.\*log(1 - u);

theta0 = [beta0, beta1];

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

theta = fminunc(@(theta) lnl(theta,y,x),theta0,options);

H = Hess(theta,y,x);

% One restriction

R = [0 1];

Q = 0;

wd(i,1) = t(l)\*(R\*theta' - Q)'\*inv( R\*inv(-H)\*R' )\*(R\*theta' - Q);

if wd(i)>chi2

j = j+1;

end

end

size(l) = j/ndraws;

wd\_sort = sort(wd,1);

cv(l) = wd\_sort(ndraws\*0.95);

end

disp(' 5 10 25 100 ');

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

disp(['size: ' num2str(size)]);

disp(['Critical value (5%): ' num2str(cv)]);

end

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

% Unconstrained log-likelihood function at each observation

function ln1 = lnlt(theta,y,x)

beta0 = theta(1);

beta1 = theta(2);

mue = exp(beta0 + beta1\*x);

ln1 = -log(mue) - y./mue;

end

% Unconstrained log-likelihood function

function ln = lnl(theta,y,x)

lnlt1 = lnlt(theta,y,x);

ln = -mean(lnlt1);

end

% Hessian

function H = Hess(theta,y,x)

beta0 = theta(1);

beta1 = theta(2);

mue = exp(beta0 + beta1\*x);

H = zeros(2,2);

H(1,1) = -sum(y./mue);

H(1,2) = -sum((x.\*y)./mue );

H(2,1) = H(1,2);

H(2,2) = -sum( ((x.^2).\*y)./mue );

end

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

%

% Simulating the power of the Wald test using exponential regression

% model

%

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

function test\_power( )

clear all;

clc;

RandStream.setDefaultStream( RandStream('mt19937ar','seed',123457) )

beta0 = 1; % Intercept parameter

t = 5; % Sample size

ndraws = 10000;

wd = zeros(ndraws,1);

beta1\_range = [-4, -3, -2, -1, 0, 1, 2, 3, 4];

power = zeros(1,length(beta1\_range));

for k = 1:length(beta1\_range);

beta1 = beta1\_range(k);

j = 0;

x = randn(t,1); % Explanatory variable (fixed in repeated samplea)

for i = 1:ndraws % Main do loop to generate Monte Carlo results

mue = exp(beta0 + beta1\*x); %mean

u = rand(t,1);

y = -mue.\*log(1 - u);

theta0 = [beta0, beta1];

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

theta = fminunc(@(theta) lnl(theta,y,x),theta0,options);

H = analytic\_hessian(theta,y,x);

% One restriction

R = [0 1];

Q = 0;

wd(i,1) = t\*(R\*theta' - Q)'\*inv( R\*inv(-H)\*R' )\*(R\*theta' - Q);

if wd(i)>4.288

j = j+1;

end

end

power(k) = j/ndraws;

end

disp(['beta1 = ' num2str(beta1\_range)]);

disp(['size for =' num2str(power)]);

end

%

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

%

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

% Define the unconstrained log of the likelihood at each observations

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

function ln1 = lnlt(theta,y,x)

beta0 = theta(1);

beta1 = theta(2);

mue = exp(beta0 + beta1\*x);

ln1 = -log(mue) - y./mue;

end

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

% Define the unconstrained log of the likelihood

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

function ln = lnl(theta,y,x)

lnlt1 = lnlt(theta,y,x);

ln = -mean(lnlt1);

end

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

% Define the analytic Hessian

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

function H = analytic\_hessian(theta,y,x)

beta0 = theta(1);

beta1 = theta(2);

mue = exp(beta0 + beta1\*x);

H = zeros(2,2);

H(1,1) = -sum(y./mue);

H(1,2) = -sum((x.\*y)./mue );

H(2,1) = H(1,2);

H(2,2) = -sum( ((x.^2).\*y)./mue );

end

**Problem 6**

% Exercise 4.7

function exp\_reg()

rand('state',0)

% Part (a) - simulation

beta0 = 1;

beta1 = 2;

T = 2000;

x\_t = rand(T,1);

mu\_t = beta0 + beta1.\*x\_t;

u\_t = rand(T,1);

y\_t = -mu\_t.\*log(1 - u\_t);

% Part (b) - unconstrained estimation

th0 = [1; 1];

options = optimset('GradObj','on','Display','off');

[theta\_1,logL\_1] = fminsearch(@logl\_u,th0,options,y\_t,x\_t);

fprintf('\n')

fprintf('Unconstrained estimation results \n')

fprintf('beta0\_hat\_1 = %3.4f \n',theta\_1(1))

fprintf('beta1\_hat\_1 = %3.4f \n',theta\_1(2))

fprintf('logL\_1 = %3.4f \n',-logL\_1)

fprintf('\n')

% Part (c) - constrained estimation

th0 = [1; 1];

options = optimset('GradObj','off','Display','off');

[theta\_0,logL\_0] = fminsearch(@logl\_c,th0,options,y\_t,x\_t);

theta\_0(2) = 0;

fprintf('\n')

fprintf('Constrained estimation results \n')

fprintf('beta0\_hat\_0 = %3.4f \n',theta\_0(1))

fprintf('beta1\_hat\_0 = %3.4f \n',theta\_0(2))

fprintf('logL\_0 = %3.4f \n',-logL\_0)

fprintf('\n')

% Part (d) - tests

% (i) Likelihood ratio test

LR = 2\*(logL\_0 - logL\_1);

p = 1 - chi2cdf(LR,1);

fprintf('Likelihood ratio test \n')

fprintf('LR stat = %3.4f \n',LR)

fprintf('p-value = %3.4f \n',p)

fprintf('\n')

% (ii) Wald test with analytic derivatives

H\_1 = HExp(theta\_1,y\_t,x\_t);

I\_1 = IExp(theta\_1,y\_t,x\_t);

Gt = GtExp(theta\_1,y\_t,x\_t);

J\_1 = Gt'\*Gt;

fprintf('Analytically determined matrices for covariance \n')

fprintf('Hessian evaluated at theta\_hat\_1 \n')

fprintf('H(th\_hat\_1) = \n')

fprintf(' %3.4f %3.4f \n',H\_1)

fprintf('Information matrix evaluated at theta\_hat\_1 \n')

fprintf('I(th\_hat\_1) = \n')

fprintf(' %3.4f %3.4f \n',I\_1)

fprintf('OPG evaluated at theta\_hat\_1 \n')

fprintf('J(th\_hat\_1) = \n')

fprintf(' %3.4f %3.4f \n',J\_1)

fprintf('\n')

R = [0 1];

Q = 0;

WH = (R\*theta\_1 - Q)'\*inv( R\*inv(-H\_1)\*R' )\*(R\*theta\_1 - Q);

WI = (R\*theta\_1 - Q)'\*inv( R\*inv( I\_1)\*R' )\*(R\*theta\_1 - Q);

WJ = (R\*theta\_1 - Q)'\*inv( R\*inv( J\_1)\*R' )\*(R\*theta\_1 - Q);

pH = 1 - chi2cdf(WH,1);

pI = 1 - chi2cdf(WI,1);

pJ = 1 - chi2cdf(WJ,1);

fprintf('Wald tests with analytical derivatives \n')

fprintf('Using Hessian \n')

fprintf('Wald stat = %3.4f \n',WH)

fprintf('p value = %3.4f \n',pH)

fprintf('Using information matrix \n')

fprintf('Wald stat = %3.4f \n',WI)

fprintf('p value = %3.4f \n',pI)

fprintf('Using OPG \n')

fprintf('Wald stat = %3.4f \n',WJ)

fprintf('p value = %3.4f \n',pJ)

fprintf('\n')

% (ii) Wald test with numerical derivatives

Gt = numgrad(@loglt\_u,theta\_1,y\_t,x\_t);

J\_1 = Gt'\*Gt;

H\_1 = -numhess(@logl\_u,theta\_1,y\_t,x\_t);

fprintf('Numerically determined matrices for covariance \n')

fprintf('Hessian evaluated at theta\_hat\_1 \n')

fprintf('H(th\_hat\_1) = \n')

fprintf(' %3.4f %3.4f \n',H\_1)

fprintf('OPG evaluated at theta\_hat\_1 \n')

fprintf('J(th\_hat\_1) = \n')

fprintf(' %3.4f %3.4f \n',J\_1)

fprintf('\n')

WH = (R\*theta\_1 - Q)'\*inv( R\*inv(-H\_1)\*R' )\*(R\*theta\_1 - Q);

WJ = (R\*theta\_1 - Q)'\*inv( R\*inv( J\_1)\*R' )\*(R\*theta\_1 - Q);

pH = 1 - chi2cdf(WH,1);

pJ = 1 - chi2cdf(WJ,1);

fprintf('Wald tests with numerical derivatives \n')

fprintf('Using Hessian \n')

fprintf('Wald stat = %3.4f \n',WH)

fprintf('p value = %3.4f \n',pH)

fprintf('Using OPG \n')

fprintf('Wald stat = %3.4f \n',WJ)

fprintf('p value = %3.4f \n',pJ)

fprintf('\n')

% (iii) LR tests with analytic derivatives

H\_0 = HExp(theta\_0,y\_t,x\_t);

I\_0 = IExp(theta\_0,y\_t,x\_t);

Gt = GtExp(theta\_0,y\_t,x\_t);

J\_0 = Gt'\*Gt;

G\_0 = GExp(theta\_0,y\_t,x\_t)';

fprintf('Analytically determined results \n')

fprintf('gradient evaluated at theta\_hat\_0 \n')

fprintf('G(th\_hat\_0) = \n')

fprintf(' %3.4f \n',G\_0)

fprintf('\n')

fprintf('Hessian evaluated at theta\_hat\_0 \n')

fprintf('H(th\_hat\_0) = \n')

fprintf(' %3.4f %3.4f \n',H\_0)

fprintf('Information matrix evaluated at theta\_hat\_0 \n')

fprintf('I(th\_hat\_0) = \n')

fprintf(' %3.4f %3.4f \n',I\_0)

fprintf('OPG evaluated at theta\_hat\_0 \n')

fprintf('J(th\_hat\_0) = \n')

fprintf(' %3.4f %3.4f \n',J\_0)

fprintf('\n')

LMH = G\_0'\*inv(-H\_0)\*G\_0;

LMI = G\_0'\*inv(I\_0)\*G\_0;

LMJ = G\_0'\*inv(J\_0)\*G\_0;

pH = 1 - chi2cdf(LMH,1);

pI = 1 - chi2cdf(LMI,1);

pJ = 1 - chi2cdf(LMJ,1);

fprintf('LM tests with analytical derivatives \n')

fprintf('Using Hessian \n')

fprintf('LM stat = %3.4f \n',LMH)

fprintf('p value = %3.4f \n',pH)

fprintf('Using information matrix \n')

fprintf('LM stat = %3.4f \n',LMI)

fprintf('p value = %3.4f \n',pI)

fprintf('Using OPG \n')

fprintf('LM stat = %3.4f \n',LMJ)

fprintf('p value = %3.4f \n',pJ)

fprintf('\n')

% (iii) LR tests with numerical derivatives

Gt = numgrad(@loglt\_u,theta\_0,y\_t,x\_t);

J\_0 = Gt'\*Gt;

H\_0 = -numhess(@logl\_u,theta\_0,y\_t,x\_t);

G\_0 = sum(Gt)';

fprintf('Numerically determined results \n')

fprintf('gradient evaluated at theta\_hat\_0 \n')

fprintf('G(th\_hat\_0) = \n')

fprintf(' %3.4f \n',G\_0)

fprintf('\n')

fprintf('Hessian evaluated at theta\_hat\_0 \n')

fprintf('H(th\_hat\_0) = \n')

fprintf(' %3.4f %3.4f \n',H\_0)

fprintf('OPG evaluated at theta\_hat\_0 \n')

fprintf('J(th\_hat\_0) = \n')

fprintf(' %3.4f %3.4f \n',J\_0)

fprintf('\n')

LMH = G\_0'\*inv(-H\_0)\*G\_0;

LMJ = G\_0'\*inv(J\_0)\*G\_0;

pH = 1 - chi2cdf(LMH,1);

pJ = 1 - chi2cdf(LMJ,1);

fprintf('LM tests with numerical derivatives \n')

fprintf('Using Hessian \n')

fprintf('LM stat = %3.4f \n',LMH)

fprintf('p value = %3.4f \n',pH)

fprintf('Using OPG \n')

fprintf('LM stat = %3.4f \n',LMJ)

fprintf('p value = %3.4f \n',pJ)

fprintf('\n')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Sub-routines

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Define the unconstrained log-likelihood at each observation

function LogLt = loglt\_u(theta,y\_t,x\_t)

beta0 = theta(1);

beta1 = theta(2);

mu\_t = beta0 + beta1\*x\_t;

LogLt = -log(mu\_t) - y\_t./mu\_t;

% -log likelihood summed over all observations

function LogL = logl\_u(theta,y\_t,x\_t)

LogLt = loglt\_u(theta,y\_t,x\_t);

LogL = -sum(LogLt);

% analytic gradient at each observation

function Gt = GtExp(theta,y\_t,x\_t)

T = size(y\_t,1);

beta0 = theta(1);

beta1 = theta(2);

mu\_t = beta0 + beta1\*x\_t;

Gt = zeros(T,2);

Gt(:,1) = -mu\_t.^(-1) + y\_t.\*mu\_t.^(-2);

Gt(:,2) = x\_t.\*Gt(:,1);

% analytic gradient summed over all observations

function G = GExp(theta,y\_t,x\_t)

Gt = GtExp(theta,y\_t,x\_t);

G = sum(Gt);

% analytic hessian

function H = HExp(theta,y\_t,x\_t)

beta0 = theta(1);

beta1 = theta(2);

mu\_t = beta0 + beta1\*x\_t;

H = zeros(2,2);

H(1,1) = sum(mu\_t.^(-2) - 2.\*y\_t.\*mu\_t.^(-3));

H(1,2) = sum( (mu\_t.^(-2) - 2.\*y\_t.\*mu\_t.^(-3)).\*x\_t );

H(2,1) = H(1,2);

H(2,2) = sum( (mu\_t.^(-2) - 2.\*y\_t.\*mu\_t.^(-3)).\*x\_t.^2 );

% analytic information matrix

function I = IExp(theta,y\_t,x\_t)

beta0 = theta(1);

beta1 = theta(2);

mu\_t = beta0 + beta1\*x\_t;

I = zeros(2,2);

I(1,1) = sum(mu\_t.^(-2));

I(1,2) = sum( (mu\_t.^(-2)).\*x\_t );

I(2,1) = I(1,2);

I(2,2) = sum( (mu\_t.^(-2)).\*x\_t.^2 );

% constrained log-likelihood at each observation

function LogLt = loglt\_c(theta,y\_t,x\_t)

beta0 = theta(1);

beta1 = 0;

mu\_t = beta0 + beta1\*x\_t;

LogLt = -log(mu\_t) - y\_t./mu\_t;

% -log likelihood summed over all observations

function LogL = logl\_c(theta,y\_t,x\_t)

LogLt = loglt\_c(theta,y\_t,x\_t);

LogL = -sum(LogLt);

% Exercise 4.8

function ch4\_8

rand('state',0)

% Part (a) - simulation

beta0 = 1;

beta1 = 2;

T = 2000;

x\_t = rand(T,1);

mu\_t = beta0 + beta1.\*x\_t;

u\_t = rand(T,1);

y\_t = -mu\_t.\*log(1 - u\_t);

% Part (b) - unconstrained estimation

th0 = [1;1;1];

options = optimset('GradObj','off','Display','off');

[theta\_1,logL\_1] = fminsearch(@logl\_u,th0,options,y\_t,x\_t);

fprintf('\n')

fprintf('Unconstrained estimation results \n')

fprintf('beta0\_hat\_1 = %3.4f \n',theta\_1(1))

fprintf('beta1\_hat\_1 = %3.4f \n',theta\_1(2))

fprintf('rho\_hat\_1 = %3.4f \n',theta\_1(3))

fprintf('logL\_1 = %3.4f \n',-logL\_1)

fprintf('\n')

% Part (c) - testing

% (i) Likelihood ratio test

th0 = [1;1;1];

options = optimset('GradObj','off','Display','off');

[theta\_0,logL\_0] = fminsearch(@logl\_c,th0,options,y\_t,x\_t);

theta\_0(3) = 1;

fprintf('\n')

fprintf('Constrained estimation results \n')

fprintf('beta0\_hat\_0 = %3.4f \n',theta\_0(1))

fprintf('beta1\_hat\_0 = %3.4f \n',theta\_0(2))

fprintf('rho\_hat\_1 = %3.4f \n',theta\_0(3))

fprintf('logL\_0 = %3.4f \n',-logL\_0)

fprintf('\n')

LR = 2\*(logL\_0 - logL\_1);

p = 1 - chi2cdf(LR,1);

fprintf('Likelihood ratio test \n')

fprintf('LR stat = %3.4f \n',LR)

fprintf('p-value = %3.4f \n',p)

fprintf('\n')

% (ii) Wald test with Hessian computed from numerical derivatives

H\_1 = -numhess(@logl\_u,theta\_1,y\_t,x\_t);

R = [0 0 1];

Q = 1;

fprintf('Hessian evaluated at theta\_hat\_1, determined numerically \n')

fprintf('H(th\_hat\_1) = \n')

fprintf(' %4.4f %4.4f %4.4f \n',H\_1)

fprintf('\n')

WH = (R\*theta\_1 - Q)'\*inv( R\*inv(-H\_1)\*R' )\*(R\*theta\_1 - Q);

pH = 1 - chi2cdf(WH,1);

fprintf('Wald tests with numerical derivatives \n')

fprintf('Using Hessian \n')

fprintf('Wald stat = %3.4f \n',WH)

fprintf('p value = %3.4f \n',pH)

fprintf('\n')

% (iii) LM test with OPG computed from numerical derivatives

Gt = numgrad(@loglt\_u,theta\_0,y\_t,x\_t);

J\_0 = Gt'\*Gt;

fprintf('OPG evaluated at theta\_hat\_0, determined numerically \n')

fprintf('J(th\_hat\_0) = \n')

fprintf(' %4.4f %4.4f %4.4f \n',J\_0)

fprintf('\n')

G\_0 = sum(Gt)';

fprintf('gradient evaluated at theta\_hat\_0 \n')

fprintf('G(th\_hat\_0) = \n')

fprintf(' %4.4f \n',G\_0)

fprintf('\n')

LMJ = G\_0'\*inv(J\_0)\*G\_0;

pJ = 1 - chi2cdf(LMJ,1);

fprintf('LM test with OPG matrix computed from numerical derivatives \n')

fprintf('LM stat = %3.4f \n',LMJ)

fprintf('p value = %3.4f \n',pJ)

fprintf('\n')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Sub-routines

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Define the unconstrained log-likelihood at each observation

function LogLt = loglt\_u(theta,y\_t,x\_t)

beta0 = theta(1);

beta1 = theta(2);

rho = theta(3);

mu\_t = beta0 + beta1\*x\_t;

LogLt = -log(gamma(rho)) - rho.\*log(mu\_t) + (rho-1).\*log(y\_t) -...

y\_t./mu\_t;

% -log likelihood summed over all observations

function LogL = logl\_u(theta,y\_t,x\_t)

LogLt = loglt\_u(theta,y\_t,x\_t);

LogL = -sum(LogLt);

% constrained log-likelihood at each observation

function LogLt = loglt\_c(theta,y\_t,x\_t)

beta0 = theta(1);

beta1 = theta(2);

rho = 1;

mu\_t = beta0 + beta1\*x\_t;

LogLt = -log(gamma(rho)) - rho.\*log(mu\_t) + (rho-1).\*log(y\_t) -...

y\_t./mu\_t;

% -log likelihood summed over all observations

function LogL = logl\_c(theta,y\_t,x\_t)

LogLt = loglt\_c(theta,y\_t,x\_t);

LogL = -sum(LogLt);

**Problem 7**

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

%

% Program for Neyman's Smooth Goodness of Fit Test

%

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

function test\_smooth( )

clear all;

clc;

% Initialise random number generator

RandStream.setDefaultStream( RandStream('mt19937ar','seed',123457) )

% Simulate data

t = 1000;

y = randn(t,1); % normal (simulate under H0)

% y = randn(t,1).^2; % chisquared with 1 degree of freedom (simulate under H1)

% Transform the data to uniform values and estimate my maximum likelihood

u = normcdf(y);

theta0 = 0.1\*ones(4,1);

% Estimate model

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

[theta,lnlu,~,~,~,H] = fminunc(@neglog,theta0,options,u);

%[theta,lnlu] = fminsearch(@neglog,theta0,options,u)

% LR test

lnl1 = -lnlu;

lnl0 = 0.0; % Under the null lnl is zero as f under H0 is 1 so lnl=0

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

dof = 4;

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

disp(['Degrees of freedom = ' num2str(dof) ]);

disp(['P-value = ' num2str(1-chi2cdf(lr,dof)) ]);

disp(' ');

% Wald test

wd = t\*theta'\*(inv(H))\*theta;

disp(['Wald statistic = ' num2str(wd) ]);

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

disp(' ')

% LM test

z = u - 0.5;

phi1 = sqrt(3)\*2\*z;

phi2 = sqrt(5)\*(6\*z.^2 - 0.5);

phi3 = sqrt(7)\*(20\*z.^3 - 3\*z);

phi4 = 3\*(70\*z.^4 - 15\*z.^2 + 3/8);

lm = (sum(phi1)/sqrt(t))^2 + (sum(phi2)/sqrt(t))^2 ...

+ (sum(phi3)/sqrt(t))^2 + (sum(phi4)/sqrt(t))^2;

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

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

end

%

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

%

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

% Program used to compute normalizing constant numerically

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

function y = normconst(u,theta)

z = u - 0.5;

t = theta(1)\*sqrt(3)\*2\*z + theta(2)\*sqrt(5)\*(6\*z.^2 - 0.5) ...

+ theta(3)\*sqrt(7)\*(20\*z.^3 - 3\*z) + theta(4)\*3\*(70\*z.^4 - 15\*z.^2 + 3/8);

y = exp(1 + t);

end

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

% Log-likelihood function

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

function lf = neglog(b,u)

z = u - 0.5;

c = quad(@(u) normconst(u,b),0,1);

f = 1 + b(1)\*sqrt(3)\*2\*z + b(2)\*sqrt(5)\*(6\*z.^2 - 0.5) ...

+ b(3)\*sqrt(7)\*(20\*z.^3 - 3\*z) + b(4)\*3\*(70\*z.^4 - 15\*z.^2 + 3/8) - log(c);

lf = -mean( f );

end

**Problem 8**

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

%

% Program to estimate a bivariate gaussian copula for asset returns

% Asset price data from 6 August 2010 to 2 January 2001 (note that the

% data are in reverse order ie from recent to past)

%

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

function test\_copula( )

clear all

clc

% Load data

load diversify.mat

t = 2413;

% Select appropriate sample

pt\_apple = pt\_apple(1:t);

pt\_ford = pt\_ford(1:t);

% Compute percentage returns

r\_apple = 100\*diff(log(pt\_apple));

r\_ford = 100\*diff(log(pt\_ford));

y = [r\_apple r\_ford];

t =length(y);

% Compute statistics

m = mean(y);

s = std(y);

c = corrcoef(y);

r = c(1,2);

% Estimate parameters of the copula

start = [ m' ; s' ; r ];

ops = optimset('LargeScale','off','Display','off');

[ bhat,~,~,~,~,hess] = fminunc(@(b) neglog( b,y ),start,ops );

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

% Wald test of independence

wd = (bhat(5) - 0)^2/vc(5,5);

disp(' ');

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

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

end

%

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

%

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

% Copula log-likelihood function

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

function lf = neglog(b,y)

m1 = b(1); % Asset 1

s1 = abs(b(3));

z1 = (y(:,1) - m1)/s1;

f1 = normpdf(z1)/s1;

m2 = b(2); % Asset 2

s2 = abs(b(4));

z2 = (y(:,2) - m2)/s2;

f2 = normpdf(z2)/s2;

r = b(5); % Dependence

lt = -log(2\*pi) - log(s1\*s2) - 0.5\*log(1 - r^2) ...

- 0.5\*(z1.^2 - 2\*r\*z1.\*z2 + z2.^2)/(1 - r^2) + log(f1) + log(f2);

lf = -mean(lt);

end