function OUT = Global(param_vec)
% This Code Solves the Model Documented in "Global Sunspots and Asset
% Prices in a Lifecycle Economy" by Roger E. A. Farmer (c) January 2015
% Updated January 29th 2015

reset(symengine);
syms pk pkP b bP m mP

% m_app is the degree of approximation of the Chebyshev polynomial.
global m_app
m_app = 4;

Diagnostic_Graph = 0; % Set to 1 to see real-time graph of solution steps
if nargin ==0
    % Choose values for the structural parameters if none are given
    bet1 = 0.98;
    bet2 = 0.9;
    pye = 0.98;
    tau = 0.02;
else
    bet1 = param_vec(1);
    bet2 = param_vec(2);
    pye = param_vec(3);
    tau = param_vec(4);
k = param_vec(5);
end

A = 1./((1-bet1*pye));
B = 1./((1-bet2*pye));
mu = 0.5;
k = 4;

C1Y(b,pk) = A^-(1)*(1-tau)*pk*mu*(1-pye);
C2Y(b,pk) = B^-(1)*pk*(1-tau)*(1-mu)*(1-pye);
C1(b,pk) = (A-B)^-(1)*(pk*(1-tau) + b - B);
C2(b,pk) = (A-B)^-(1)*(A - pk*(1-tau) - b);
C1O(b,pk) = (C1(b,pk) - C1Y(b,pk));
C2O(b,pk) = (C2(b,pk) - C2Y(b,pk));
c1O(b,pk) = C1O(b,pk)/pye;
c2O(b,pk) = C2O(b,pk)/pye;

% Using these definitions I compute expressions for the marginal rate of
% substitution of an agent of each type as functions of asset values at
% consecutive dates

MRS1(b,pk,bP,pkP) = pye*bet1*C1(b,pk)./C1O(bP,pkP);
MRS2(b,pk,bP,pkP) = pye*bet2*C2(b,pk)./C2O(bP,pkP);
% Next, I find an expression for the future price of long term debt as a % function of b, pk and bP by equating the marginal rates of substitution % of the two types in every state and I use this to write an expression for % the pricing kernel as a function of b, pk and bP

```
psifn(b,pk,bP) = solve(MRS1(b,pk,bP,pkP)==MRS2(b,pk,bP,pkP),pkP);
phifn(b,pk,bP) = MRS1(b,pk,bP,psifn(b,pk,bP));
```

%H This step finds expressions for pk, pkP and bP as functions of the % pricing kernel that I define as a new variable mP

```
T1 = solve(pkP-psifn(b,pk,bP),b-phifn(b,pk,bP)*bP-\tau, ...
     mP-phifn(b,pk,bP),pkP,pk,b);
 pkPfn(mP,bP) = T1.pkP;
% This is the function xi(m',b') introduced a Equation (48) on Page 33
 pkfn(mP,bP) = T1.pk;
```

% Next, I find a difference equation that maps \{b,m\} to \{bP,mP\}

```
T2 = solve(pkPfn(mP,bP)-psifn(b,pkPfn(m,b),bP),b-mP*bP-\tau,bP,mP);
f(m,b) = T2.mP;
g(m,b) = T2.bP;
```

% Now I solve for the steady state

```
xs = solve([f(m,b)-m,g(m,b)-b],[m,b]);
ms1 = real(double(xs.m));
bs1 = real(double(xs.b));
```

% Now define the value of the consumption of old agents of each type as % functions of the state

```
C2OFN(m,b) = c2O(b,pkPfn(m,b));
C1OFN(m,b) = c1O(b,pkPfn(m,b));
```

% The following two functions are used to construct the endpoints of the % set D. See Equation (47) on Page 30

```
C2FN(m,b) = C2(b,pkPfn(m,b));
C1FN(m,b) = C1(b,pkPfn(m,b));
```

% ... and use these functions to eliminate infeasible steady states

```
ms = ms1(C2OFN(ms1,bs1)>0 & C1OFN(ms1,bs1)>0 & pkPfn(ms1,bs1)>0 ... 
     & bs1>0 & ms1>0);
bs = bs1(C2OFN(ms1,bs1)>0 & C1OFN(ms1,bs1)>0 & pkPfn(ms1,bs1)>0 ... 
     & bs1>0 & ms1>0);
ms = real(double(ms));
bs = real(double(bs));
```

% Store the values of the steady state to the output structure

```
OUT.ms = ms;
OUT.bs = bs;
```

% Next I define numerical functions that I will use later to solve for the % stable manifold using Chebyshev polynomials
Fn = @(m,b,mP,bP) double(mP-f(m,b));
Gn = @(m,b,mP,bP) double(bP-g(m,b));

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ANALYSIS OF THE PROPERTIES OF THE MODEL AROUND THE STEADY STATE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% In this section I compute the stable eigenvector around the steady state
% This will be used to provide an initial guess for the policy function and
% I will also use it to provide an initial guess for bounds on the domain
% over which the equilibrium is well defined
x = [m b];
y = [mP bP];

% First I compute analytic derivatives
HF(x,y) = [y(1)-f(x(1),x(2)),y(2)-g(x(1),x(2))];
JF1(x,y) = -jacobian(HF,x);
JF2(x,y) = jacobian(HF,y);

% Then I evaluate them numerically at the steady state and compute an
% eigenvalue decomposition
xys = [m,b,mP,bP];
xysn = [ms,bs,ms,bs];
JF1n = double(subs(JF1,xys,xysn));
JF2n = double(subs(JF2,xys,xysn));
J = JF2n/JF1n;
[Q,Ev] = eig(J);

% Here I display the values of the Eigenvalues and I store the eigenvectors
% and the eigenvalues in the output structure
disp({'Eigenvalues    ' num2str(diag(Ev)' )});
OUT.Q = Q; OUT.EV = Ev;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Now I Use the stable eigenvector to form initial guesses for the policy
% function and for the dynamic equation for m
Q1 = inv(Q);
q11 = Q1(1,1);q12 = Q1(1,2);
fun1(m) = vpa(bs + (-q11/q12)*(m - ms)); % fun1 is the policy function
fun2(m) = vpa(ms + Ev(2,2)*(m - ms)); % fun2 is the dynamics of m

% Now I Compute a guess for domain of the Eigenfunction by finding values
% of m for which the consumption of the old of each type goes to zero
UB = fsolve(@(m) double(C2OFN(m,fun2(m))),0.7);
LB = fsolve(@(m) double(C1OFN(m,fun2(m))),0.7);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% NONLINEAR SOLUTION FOR THE POLICY FUNCTION USING CHEBYSHEV POLYNOMIALS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% X is a grid defined over [-1,1] that contains the Chebyshev nodes for an
% approximation of degree m_app. I use the Gauss-Lobatto nodes which
% correspond to extrema of the Chebyshev polynomials
% Y is the map of the X grid to the domain of m
% I choose bounds LB and UB for the grid where LB and UB are the values
% where the approximate linear solution delivers zero consumption for one
% of the types
X = setgrid(1,m_app)';
Y = ITx(X,LB,UB);
% fun1a and fun1b generate numerical Matlab functions for my guess of
% the policy function and for the dynamic equation
% The functions Tx and ITx map between the domain of Y and the domain of X
fun1a = @(X) double(fun1(ITx(X,LB,UB)));
fun2a = @(X) double(fun2(ITx(X,LB,UB)));
% The vectors a0 and b0 are the coefficients that provide an m_app'th order
% approximation to the functions f1a and f2a
[a0, ~] = Fappr(fun1a,1,m_app);
b0 = double(b0);

% At this stage I stack the coefficient vectors into a single parameter
% vector Z0 which has dimension 2K x 1. The first K elements of Z0 describe
% the policy function and the second K elements define the dynamic equation
% governing the behavior of the pricing kernel
% Z0 is my initial guess for the unknown functions
Z0 = [a0; b0];

% The function m10, b10, m1F and b1F compute the Chebyshev approximation
% of a function defined by the parameter vector Z at an arbitrary point X
% The function FREP replicates points in the domain of the function being
% approximated. This allows a solution algorithm to compute approximations
% to the jacobian by evaluating the values of functions that are near a
% candidate function

% The functions m10 and b10 have domain X
% The functions m1F and b1F have domain Y
% The function Id is the identity function
% The functions F1 and F2 contain the operator equations that we are trying
% to solve. The function Fn is my approximation to
% m-f(m,b)
% and Gn is my approximation to
% b-g(m,b);
% The first argument of Fn (and Gn) is m
% The second argument is b which I express as an unknown function b1F(m)
% The third argument is mP which I express as a different unknown
% function m1F(m)
% The fourth argument is bP which is the composition of the two functions
% b1F and m1F
m10 = @(Z,X) FREP(Z(1:n,:), X);
b10 = @(Z,X) FREP(Z(n+1:end,:), X);
m1F = @(Z,Y) m10(Z, Tx(Y, LB, UB));
b1F = @(Z,Y) b10(Z, Tx(Y, LB, UB));
Id = @(Z,Y) repmat(Y, 1, size(Z, 2));
F1 = @(Z,Y) Gn(Id(Z,Y), b1F(Z,Y), m1F(Z,Y), b1F(Z,m1F(Z,Y)));
F2 = @(Z,Y) Fn(Id(Z,Y), b1F(Z,Y), m1F(Z,Y), b1F(Z,m1F(Z,Y)));
FUN = @(Z) [F1(Z,Y); F2(Z,Y)];
% Here I solve for a function that approximates Fn and Gn exactly at the
% Chebyshev nodes
if Diagnostic_Graph == 1
    options = optimoptions(@fsolve,'Display','off','MaxIter', ... 
        1000,'PlotFcns',@optimplotfval);
else
    options = optimoptions(@fsolve,'Display','off','MaxIter', ... 
        1000);
end

ZS = fsolve(FUN,Z0,options);

% Finally, I copy variables to the output structure
OUT.m_app = m_app;
OUT.ZS = ZS;
OUT.X = X;
OUT.Y = Y;
OUT.m1F = m1F;
OUT.b1F = b1F;
OUT.m10 = m10;
OUT.b10 = b10;
OUT.LB = LB;
OUT.UB = UB;
OUT.C2OFN(m,b) = C2OFN(m,b);
OUT.C1OFN(m,b) = C1OFN(m,b);
OUT.C2FN(m,b) = C2FN(m,b);
OUT.C1FN(m,b) = C1FN(m,b);
OUT.pkfn(mP,bP) = pkfn(mP,bP);
OUT.pkPfn(mP,bP) = pkPfn(mP,bP);
OUT.k = k;
OUT.ms = ms;
OUT.bs = bs;
OUT.pye = pye;
OUT.Id = Id;
OUT.tau = tau;

% These two lines generate figures for the paper
OUT = plot_graphs(OUT);
plot_graph_5;
end

% END OF MAIN PROGRAM
% BEGIN SUBROUTINES

function OUT = plot_graphs(OUT,Runs)

% This Code simulates data and produces graphs for the Model Documented
% in "Global Sunspots and Asset Prices in a Lifecycle Economy"
% by Roger E. A. Farmer (c) 2015

if nargin ==1
    Runs =1;
end

rng(12);  % Set this seed for random number generator for Figure 4
% rng(22);  % Set this seed for random number generator for Figure 6

% Graphs for Figure 6 were produced in Eviews using the series OUT.xc1
% OUT.xc2 OUT.RR OUT.RS and OUT.xpi
% The random seed was set to rng(22)

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

% Set Runs greater than 9 to produce statistics for the Sharpe ratio
% averaged over multiple draws of length T

T = 60;
% T is the sample length for simulations
RN = 1.05;
% RN is the gross nominal interest rate

fnum = 1;
LB = OUT.LB;
UB = OUT.UB;
C2FN = OUT.C2FN;
C1FN = OUT.C1FN;
ms = OUT.ms;
bs = OUT.bs;
b1F = OUT.b1F;
m1F = OUT.m1F;
ZS = OUT.ZS;
pkfn = OUT.pkfn;
pkPfn = OUT.pkPfn;
k = OUT.k;
pye = OUT.pye;

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

% This code generates Figure 3 from the paper

figure(fnum+2);
t = LB:0.001:UB;
subplot(2,2,1);
plot(t,[t*m1F(ZS,t')'-t],'LineWidth',2);
xlim([LB UB]);
xlabel('Dynamics of the Pricing Kernel','FontSize',16);
ylabel('Change in the Kernel','FontSize',16);
subplot(2,2,2);
plot(t, 100*double(C2FN(t', b1F(ZS, t'))), 'LineWidth', 2);
xlim([LB UB]);
xlabel('Consumption of Each Type', 'FontSize', 16);
ylabel('Percent of GDP', 'FontSize', 16);
ylim([0 100]);
subplot(2, 2, 3);
plot(t, 100*b1F(ZS, t'), 'LineWidth', 2);
xlim([LB UB]);
xlabel('Govt Debt as a % of GDP', 'FontSize', 16);
ylabel('Percent of GDP', 'FontSize', 16);
subplot(2, 2, 4);
plot(t, pkPfn(m1F(ZS, t'), b1F(ZS, t'))', 'LineWidth', 2);
xlim([LB UB]);
xlabel('The Price of a Tree', 'FontSize', 16);
ylabel('Price', 'FontSize', 16);

%%
BigS = zeros(5, Runs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Here I generate a function to compute the price of a tree
% See Equation (49) on Page 33
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
infun = @(s, alp, bet) ((UB-LB)*s+LB).*pdf('beta', s, alp, bet);
in2fun = @(s, alp, bet) double(pkPfn(((UB-LB)*s+LB),
  double(b1F(ZS, ((UB-LB)*s+LB)').')));

pkFF = @(alp, bet) 1 + pye*integral( @(s) in2fun(s, alp, bet).*
infun(s, alp, bet), 0, 1, 'ArrayValued', true);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If Runs >1 the code produces smoothed histograms of Sharpe ratios
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:Runs
  disp(['Run Number ' num2str(i)]);
  T = T+1;
  xp = ones(1, T)*double(pkfn(ms, bs));
  RS = ones(1, T);
  alphparam = ones(1, T);
  betparam = ones(1, T);
  Mp1 = ones(1, T);
  xb = ones(size(xp))*bs;

  % This loop generates T years of simulated data
  for t = 2:T
    disp([num2str(i) '. ' num2str(t)]);
    Mp1(1, t) = (m1F(ZS, xm(1, t-1)) - LB)/(UB-LB);
    V = max(k/Mp1(1, t), k/(1-Mp1(1, t)));
    alphparam(1, t) = V*Mp1(1, t);
    betparam(1, t) = V*(1-Mp1(1, t));
    xm(1, t) = random('beta', alphparam(1, t), betparam(1, t))*(UB-LB) + LB;
    RS(1, t) = (1./((UB-LB)*Mp1(1, t)+LB)-1)*100;
  end
end
xb(1,t) = double( b1F(ZS,xm(1,t)) );
xp(1,t) = double(pkFF(alphparam(1,t),betparam(1,t)));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Here I produce various functions of m and b
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xpi = (double(RN*xm)-1)*100;
RR  = (pye*xp(2:end)./(xp(1:end-1)-1)-1)*100;
xpi = xpi(2:end);
xm = xm(2:end);
RS = RS(2:end);
xp = xp(2:end);
xb = xb(2:end);
xc2 = double(C2FN(xm,xb));
xc1 = double(C1FN(xm,xb));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This code generates Figure 4 from the paper
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(fnum+3);
time = 1:T-1;
subplot(1,2,1);
hold off;
plot(time,RS,'-','LineWidth',2);
hold on;
plot(time,RR,'-.','color',[0 0.5 0],'LineWidth',2);
plot(time,xpi,'-.','color',[0.5 0 0],'LineWidth',2);
hold off;
xlabel('Date in Years','FontSize',20);
ylabel('Percent Per Year','FontSize',20);
legend({'Safe Rate','Risky Rate','Expected Inflation'},...
           'Location','northoutside','Orientation','horizontal','FontSize',15);
legend BOXOFF;

subplot(1,2,2);
plot(time,xp,'-','LineWidth',2);
xlabel('Date in Years','FontSize',20);
ylabel('Price of a Tree','FontSize',20);
legend({'PE Ratio '},...
           'Location','northoutside','Orientation','horizontal','FontSize',15);
legend BOXOFF;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This code writes the mean of T years of data of the risky and safe
% rates and it computes the Sharpe ratio
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
BigS(:,i) = [mean(RR); mean(RS); mean(xpi); mean(RR-RS)/std(RR) ;std(xpi)];
OUT.BigS = BigS;
disp(['Mean of Risky Rate ' num2str(mean(RR))]);
disp(['Mean of Safe Rate ' num2str(mean(RS))]);
disp(['Mean of Expected Inflation Rate ' num2str(mean(xpi))]);
disp(['Sharpe Ratio ' num2str(mean(RR-RS)/std(RR))]);
end

save('savefile.mat','BigS');
% This code generates Figure 2 from the paper
figure(fnum + 1);

t1 = [LB+(0.1*(UB-LB)) (LB+UB)/2 UB-0.1*(UB-LB)];
avL = ones(1,numel(t1));
bvL = ones(1,numel(t1));
for i = 1:numel(t1)
    Mp1 = (m1F(ZS,t1(i))-LB)/(UB-LB);
    V = k*max(k/Mp1,k/(1-Mp1));
    avL(1,i) = V*Mp1;
    bvL(1,i) = V*(1-Mp1);
end

x = 0:0.001:1;
xa = LB + x*(UB-LB);
hold off;
for i = 1:numel(t1)
    plot(xa,pdf('beta',x,avL(1,i),bvL(1,i)),'LineWidth',2);
hold on;
end
toplace = m1F(ZS,t1')';
fromplace = t1;
maxp=16;
line([fromplace;fromplace],[0 0 0;maxp maxp maxp ],'Color',[0.75 0 0],...
    'LineWidth',1,'LineStyle','--');
line([toplace;toplace],[0 0 0;maxp maxp maxp ],'Color',[0 0.5 0],...
    'LineWidth',1,'LineStyle','-.');
xlim([min(xa) max(xa)]);
xlabel('Period t Discount Factor','FontSize',24);
ylabel('Pdf of Period t+1 Discount Factor','FontSize',22);

% This code generates Figure 5 from the paper
% The figure is only generated for values of Runs great than or equal to 10
if Runs>9
    figure(fnum+3);
    subplot(2,1,1);
    [x, y] = ksdensity(OUT.BigS(4,:));
    plot(y,x,'LineWidth',2);
    xlabel('Sharpe Ratio','FontSize',24);
    ylabel('Sharpe Ratio','FontSize',24);
    subplot(2,1,2);
    [x1, y1] = ksdensity(OUT.BigS(1,:));
    [x2, y2] = ksdensity(OUT.BigS(2,:));
    plot(y1,x1,y2,x2,'LineWidth',2);
end

OUT.xm = xm;
OUT.xp = xp;
OUT.xb = xb;
OUT.BigS = BigS;
OUT.RR = RR;
OUT.RS = RS;
OUT.xpi = xpi;
OUT.xc1 = xc1;
OUT.xc2 = xc2;
OUT.xp = xp;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This Code produces Figure 5 for the Model Documented
% in "Global Sunspots and Asset Prices in a Lifecycle Economy"
% by Roger E. A. Farmer (c) 2015
% The code imports the matrix Data_Figure_for_5.mat
% This matrix is generated by the function plot_graphs by setting Runs =
% 6,000.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plot_graphs_5
figure(5);
Data = xlsread('DataFile');
O2 = Data;
 subplot(2,1,1);
 [x, y] = ksdensity(O2(4,:));
 plot(y,x,'LineWidth',2);
 xlabel('Sharpe Ratio','FontSize',24);
 legend({'Sharpe Ratio'},...
 'Location','northoutside','Orientation','horizontal','FontSize',15);
 legend BOXOFF;
 subplot(2,1,2);
 [x1, y1] = ksdensity(O2(1,:));
 [x2, y2] = ksdensity(O2(2,:));
 plot(y1,x1,y2,x2,'--r','LineWidth',2);
 xlabel('The Safe and Risky Rates','FontSize',24);
 legend({'Risky Rate','Safe Rate'},...
 'Location','northoutside','Orientation','vertical','FontSize',15);
 legend BOXOFF;
 mean(O2,2)
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function z = FREP(a,x)
% This function replicates a given chebyshev polynomial so that Fsolve may
% evaluate alternative polynomials at the same grid points
global m_app;
n2 = size(a,2);
m2 = size(x,2);
 if m2==1 && n2>1
   X = repmat(x,1,n2);
 else
   X=x;
 end
z = zeros(size(x));
for i = 1:size(a,2)
z(:,i) = Fsim(a(:,i),X(:,i)',m_app)';
function [p,c] = cheb(x,r,i)
% This program calculates the Chebyshev polynomials of degree 1 through r,
% evaluated at x: if i is missing, the program returns the first r polynomials
% otherwise, for i <= r, the program returns the i'th polynomial. x must be
% between -1 and 1. i and r are positive integers with i <=r
% c is a matrix where column j is the chebychev coefficient on x^j on term i
% (c) Roger Farmer October 29th 2014

%%
if x<-1 || x>1
    disp('Invalid input: x must be between -1 and 1');
    p = [];
    return
elseif nargin>2
    if i > r;
        disp('Invalid input: i must be less than or equal to r');
        p = [];
        return;
    end
end
p = zeros(r,size(x,2));
p(1,:) = 1;
if r >1
    p(2,:) = x;
end
for t = 3:r
    p(t,:) = 2.*x.*p(t-1,:)-p(t-2,:);
end,
if nargin>2;
    p = p(i,:);
end
c = zeros(r,r);
c(1,1) = 1;
c(2,2) = 1;
for t = 3:r
    c(t,:) = 2*[0 c(t-1,1:r-1)] - c(t-2,:);
end
end

function [csi] = coloc(mu,~)
% Calculate the Chebyshev colocation points using the Gauss-Lobotto nodes
% These are the extrema of the polynomials: mu is a positive integer which
% represents the degree of the approximnination.
% csi are the colocation points
% (c) Roger Farmer October 29th 2014
% RF Updated December 29th 2014: added a second argument
% If function is one-dimensional add any integer as a second argument

%%
big = 10^5;
if mu == 0
    csi = 0;
    return
else
    if nargin > 1
        n = mu;
    else
        n = 2^mu+1;
    end
    s = 1:n;
    csi = (-cos(pi*(s-1)./(n-1)))';
    csi = sign(csi).*floor(abs(csi*big))/big;
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [b, p] = Fappr(FUN, d, mu)
% Generate an approximation to the function FUN where
% FUN maps [-1,1]^d and mu is the degree of approximation
% b completely characterizes the approximation to b
% p is a matrix that contains indices of the polynomial coefficients at each
% grid point
% (c) Roger Farmer October 29th 2014

% [g, p] = setgrid(d, mu);
B = ScriptB(d, mu);
b = B\(FUN(g)');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function z = Fsim(b, x, mu)
% Compute the value of a Chebyshev polynomial evaluated at x where
% rows(x)=d is the dimension of the domain and cols(x)=n is the number of
% points at which the polynomial is evaluated. The vector b an
% approximation to a function F: [-1,1]^d -> R. The choice of grid
% points and polynomials is determined by Maliar and Maliar's implementation
% of the Smolyak sparse grid method. mu is the degree of approximation
% (c) Roger Farmer October 29th 2014

% d = size(x, 1);
% n = size(x, 2);
% [~, pc] = setgrid(d, mu);
% N = max(max(pc));
% K = size(pc, 2);
% z = ones(1, n);
% X = ones(K, n);
for i = 1:n
    for k = 1:K
        X(k, i) = 1;
        for j = 1:d
            y = cheb(x(j, i), N);
            X(k, i) = X(k, i) * y(pc(j, k));
        end
    end
end
\[ z(1, i) = b' * X(:, i); \]

end
end

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

function X = inds(d, mu)
% Computes all vectors of integers of length d for which the sum of the
% elements is less than or equal to d+mu and greater than or equal to d.
% (c) Roger Farmer October 29th 2014
% q = max(d,1+mu); [removed 10-29]

%%
x = ones(1, d);
X = x;
while x(1) < mu + 1 || sum(x) < mu + d
    while sum(x) < mu + d
        k = d;
        x(k) = x(k) + 1;
        X = [X; x]; %#ok<AGROW>
    end
    k = k - 1;
    x(k) = x(k) + 1;
    x(k+1:end) = 1;
    X = [X; x]; %#ok<AGROW>
end
end

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

function x = ITx(xhat, xL, xU, S)
% This function maps [-1,1] to [xL,xIU]
% S is a shape parameter used in the infinite case
% (c) Roger Farmer October 29th 2014

%%
% If necessary switch bounds so that xU is upper bound
if nargin < 4;
    S = 1;
end
if xL > xU
    t = xU;
    xU = xL;
    xL = t;
end
xhat(xhat< -1) = -1;
xhat(xhat> 1) = 1;
if isfinite(xL) && isfinite(xU)
    x = (1/2) * (xhat * (xU - xL) + xU + xL);
elseif ~isfinite(xL) && ~isfinite(xU)
    x = log((1 + xhat)./(1 - xhat))./S;
elseif ~isfinite(xL) && isfinite(xU)
    disp('Cannot handle this case yet')
    x = [];
    return;
elseif isfinite(xL) && ~isfinite(xU)
    disp('Cannot handle this case yet')
    x = [];
    return;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function B = ScriptB(d,mu)
% Construct the matrix ScriptB
% Reference Judd Maliar Maliar Valero Page 16
% The approximation coefficients to a function F satisfy the linear
% equations b = B^{-1}*F(g) where g are the grid points
% (c) Roger Farmer October 29th 2014

% [g, pc] = setgrid(d,mu);
% n = size(pc,2);
% B = ones(n,n);
% N = max(max(pc));
% for i = 1: n
%    x = g(:,i);
%    for k = 1:n
%        for j = 1:d
%            y = cheb(x(j),N);
%            B(i,k) = B(i,k)*y(pc(j,k));
%        end
%    end
%end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [points, polychoose] = setgrid(d,mu)
% points is a set of grid points for a sparse Smolyak grid with dimension d
% and approximation level mu
% polychoose is a set of indices of the chebychev polynomials that occur at
% each grid point
% (c) Roger Farmer October 29th 2014

% if d==1
%    points = coloc(mu,1)';
%    polychoose = 1:length(points);
%    return
end
s = cell(mu+1,1);
a = cell(mu+1,1);
polyind = cell(mu+1,1);
s{1} = 0;
a{1} = s{1};
polyind{1} = 1;
for i = 2:mu+1
    s{i} = coloc(i-1);
a{i} = setdiff(s{i},s{i-1});
polyind{i} = (polyind{i-1}(end)+1:polyind{i-1}(end)+size(a{i},1))';
end
p1 = inds(d,mu);
points = [];
polychoose = [];
for i = 1:size(p1,1);
temp1 = a{p1(i,1)}';
temp2 = polyind{p1(i,1)}';
for j = 2:d
    temp1 = combvec(temp1,a{p1(i,j)}');
temp2 = combvec(temp2,polyind{p1(i,j)}');
end
points = [points temp1]; %#ok<AGROW>
polychoose = [polychoose temp2]; %#ok<AGROW>
end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function xhat = Tx(x,xL,xU,S)
% This function maps [xL,xIU] to [-1,1]
% S is a shape parameter used in the infinite case
% (c) Roger Farmer October 29th 2014
%
if nargin < 4;
    S=1;
end
if xL>xU
    t = xU;
xU = xL;
xL = t;
end

x(x<xL) = xL;
x(x>xU) = xU;

if isfinite(xL) && isfinite(xU)
    xhat = (2*x -(xU+xL))/(xU-xL);
elseif ~isfinite(xL) && ~isfinite(xU);
    xhat = -(1 - exp(S*x))/(1+exp(S*x));
elseif ~isfinite(xL) && isfinite(xU)
    disp('Cannot handle this case yet')
x = [];
    return;
elseif isfinite(xL) && ~isfinite(xU)
    disp('Cannot handle this case yet')
x = [];

end

    return;
end
xhat(xhat<1) = -1;
xhat(xhat>1) = 1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END OF SUBROUTINES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%