function [f,x,xp,y,yp] = RBCmodel(unitFree)

%% Section 1: Declaring coefficients and variables

%Define the structureal parameters

syms DELTA BETTA B ETAl ETAc THETA ALFA RHOA

%Define the state variables in this period (\_cu) and the next period

%(\_cup). Variables lagged by one period appear as "\_ba1"

syms k\_cu c\_ba1 a\_cu

syms k\_cup c\_ba1p a\_cup

%Define the control variables in this period (\_cu) and the next period (\_cup)

syms c\_cu iv\_cu y\_cu la\_cu n\_cu rk\_cu w\_cu

syms c\_cup iv\_cup y\_cup la\_cup n\_cup rk\_cup w\_cup

%% Section 2: Endogenous model equations

% Eq 1: FOC for consumption

if unitFree == 1

eq1 = -1 +((c\_cu-B\*c\_ba1)^(-ETAc) - BETTA\*B\*(c\_cup-B\*c\_cu)^(-ETAc))/la\_cu;

else

eq1 = -la\_cu +(c\_cu-B\*c\_ba1)^(-ETAc) - BETTA\*B\*(c\_cup-B\*c\_cu)^(-ETAc);

end

% Eq 2: Household's FOC for labor

if unitFree == 1

eq2 = (-THETA\*(1-n\_cu)^-ETAl)/(la\_cu\*w\_cu) + 1;

else

eq2 = -THETA\*(1-n\_cu)^-ETAl + la\_cu\*w\_cu;

end

% Eq 3: FOC for capital

if unitFree == 1

eq3 = -1 + (BETTA\*la\_cup\*(rk\_cup + (1-DELTA)))/la\_cu;

else

eq3 = -la\_cu + BETTA\*la\_cup\*(rk\_cup + (1-DELTA));

end

% EQ 4: Firm's FOC for capital

if unitFree == 1

eq4 = -a\_cu\*(1-ALFA)\*k\_cu^(-ALFA)\*n\_cu^(ALFA) + rk\_cu;

else

eq4 = (-a\_cu\*(1-ALFA)\*k\_cu^(-ALFA)\*n\_cu^(ALFA))/rk\_cu + 1;

end

% EQ 5: Firm's FOC for labor

if unitFree == 1

eq5 = (-a\_cu\*ALFA\*k\_cu^(1-ALFA)\*n\_cu^(ALFA-1))/w\_cu + 1;

else

eq5 = -a\_cu\*ALFA\*k\_cu^(1-ALFA)\*n\_cu^(ALFA-1) + w\_cu;

end

% EQ 6: National income identity

if unitFree == 1

eq6 = (-c\_cu - iv\_cu)/y\_cu + 1;

else

eq6 = -c\_cu - iv\_cu + y\_cu;

end

% EQ 7: Production function

if unitFree == 1

eq7 = -1 + (a\_cu\*k\_cu^(1-ALFA)\*n\_cu^(ALFA))/y\_cu;

else

eq7 = -y\_cu + a\_cu\*k\_cu^(1-ALFA)\*n\_cu^(ALFA);

end

% EQ 8: Law of motion for capital

if unitFree == 1

eq8 = (-k\_cup + (1-DELTA)\*k\_cu)/iv\_cu + 1;

else

eq8 = -k\_cup + (1-DELTA)\*k\_cu + iv\_cu;

end

%% Section 3: Link equations

% EQ 9: Link for lagged consumption

if unitFree == 1

eq9 = -c\_ba1p/c\_cu + 1;

else

eq9 = -c\_ba1p + c\_cu;

end

%% Section 4: The exogenous shocks

% EQ 10: Law of motion for technology

eq10 = -log(a\_cup) + RHOA\*log(a\_cu);

%Create function f

f = [eq1;eq2;eq3;eq4;eq5;eq6;eq7;eq8;eq9;eq10];

%% Section 5: Defining x,xp,y,yp

% Define the vector of states, x and xp

x = [k\_cu c\_ba1 a\_cu ];

xp = [k\_cup c\_ba1p a\_cup];

% Define the vector of controls, y and yp

y = [c\_cu iv\_cu y\_cu la\_cu n\_cu rk\_cu w\_cu ];

yp = [c\_cup iv\_cup y\_cup la\_cup n\_cup rk\_cup w\_cup];

% For the log-approximation: Make f a function of

% the logarithm of the state and control vector

f = subs(f, [x,y,xp,yp], exp([x,y,xp,yp]));

f =[

1/(c\_cu - B\*c\_ba1)^ETAc - la\_cu - (B\*BETTA)/(c\_cup - B\*c\_cu)^ETAc

la\_cu\*w\_cu - THETA/(1 - n\_cu)^ETAl

BETTA\*la\_cup\*(rk\_cup - DELTA + 1) - la\_cu

rk\_cu + (a\_cu\*n\_cu^ALFA\*(ALFA - 1))/k\_cu^ALFA

w\_cu - ALFA\*a\_cu\*k\_cu^(1 - ALFA)\*n\_cu^(ALFA - 1)

y\_cu - iv\_cu - c\_cu

a\_cu\*k\_cu^(1 - ALFA)\*n\_cu^ALFA - y\_cu

iv\_cu - k\_cup - k\_cu\*(DELTA - 1)

c\_cu - c\_ba1p

RHOA\*log(a\_cu) - log(a\_cup)]

f =[

1/(exp(c\_cu) - B\*exp(c\_ba1))^ETAc - exp(la\_cu) - (B\*BETTA)/(exp(c\_cup) - B\*exp(c\_cu))^ETAc

exp(la\_cu)\*exp(w\_cu) - THETA/(1 - exp(n\_cu))^ETAl

BETTA\*exp(la\_cup)\*(exp(rk\_cup) - DELTA + 1) - exp(la\_cu)

exp(rk\_cu) + (exp(a\_cu)\*exp(n\_cu)^ALFA\*(ALFA - 1))/exp(k\_cu)^ALFA

exp(w\_cu) - ALFA\*exp(a\_cu)\*exp(k\_cu)^(1 - ALFA)\*exp(n\_cu)^(ALFA - 1)

exp(y\_cu) - exp(iv\_cu) - exp(c\_cu)

exp(a\_cu)\*exp(k\_cu)^(1 - ALFA)\*exp(n\_cu)^ALFA - exp(y\_cu)

exp(iv\_cu) - exp(k\_cup) - exp(k\_cu)\*(DELTA - 1)

exp(c\_cu) - exp(c\_ba1p)

RHOA\*log(exp(a\_cu)) - log(exp(a\_cup))]

function [DELTA,BETTA,B,ETAl,ETAc,THETA,ALFA,RHOA,... %parameters

k\_cu, c\_ba1 ,a\_cu,... %states at time t

k\_cup,c\_ba1p,a\_cup,... %states at time t+1

c\_cu ,iv\_cu ,y\_cu ,la\_cu ,n\_cu ,rk\_cu ,w\_cu ,... %controls at time t

c\_cup,iv\_cup,y\_cup,la\_cup,n\_cup,rk\_cup,w\_cup,... %controls at time t+1

auxOut,errorMes] = RBCmodel\_ss(params)

%% Section 1:

% The size of the model

ny = 7; %Number of control variables

nx = 3; %Number of state variables

ne = 1; %Number of shocks

mx = 1; %Number of endogenous state variables (must come first in x)

myx = 1; %Number of lagged control variables appearing as states.

%These lagged control variables must come first in y AND must

%appear in x after the endogenous states.

%Unfold the params vector

DELTA = params.DELTA;

BETTA = params.BETTA;

B = params.B;

ETAl = params.ETAl;

ETAc = params.ETAc;

THETA = params.THETA;

ALFA = params.ALFA;

RHOA = params.RHOA;

STDA = params.STDA;

% The eta matrix: nx\*ne

eta = zeros(nx,ne);

eta(3,1) = STDA;

%% Section 2: Solving for the steady state

errorMes = 0; % Initializing errorMes: 0 for no errros, else 1

A = 1; % The value of technology

RK = 1/BETTA - (1-DELTA); % The rental rate of capital

% Capital divided by labor

K\_O\_N = (RK/(A\*(1-ALFA)))^(-1/ALFA);

if K\_O\_N <= 0

errorMes = 1;

end

W = A\*ALFA\*(K\_O\_N)^(1-ALFA); % The wage level

IV\_O\_N = DELTA\*K\_O\_N; % Investment over labor

Y\_O\_N = A\*K\_O\_N^(1-ALFA); % Output over labor

C\_O\_N = Y\_O\_N - IV\_O\_N; % Consumption over labor

if C\_O\_N <= 0

errorMes = 1;

end

% The labor level

if ETAc == 1 && ETAl == 1

% Closed-form solution for N

N = (1-BETTA\*B)\*(C\_O\_N\*(1-B))^-1\*W/THETA/(1+(1-BETTA\*B)\*(C\_O\_N\*(1-B))^-1\*W/THETA);

else

% No closed-form solution and we therefore use a fixed-point algorithm

if errorMes == 0

options = optimset('Display','off','TolX',1e-12,'TolFun',1e-12);

N0 = 1/3;

[N,~,exitflag] = fsolve(@findN,N0,options);

if exitflag <= 0

errorMes = 1;

end

else

N = NaN;

end

end

% Value of remaining variables

C = C\_O\_N\*N;

Y = Y\_O\_N\*N;

IV = IV\_O\_N\*N;

K = K\_O\_N\*N;

LA = (C-B\*C)^(-ETAc)-BETTA\*B\*(C-B\*C)^(-ETAc);

%% Section 3: Reporting the steady state variables for the perturbation approximation

% The value of x and xp

k\_cu = log(K);

c\_ba1 = log(C);

a\_cu = log(A);

k\_cup = log(K);

c\_ba1p = log(C);

a\_cup = log(A);

% The value of y and yp

c\_cu = log(C);

iv\_cu = log(IV);

y\_cu = log(Y);

la\_cu = log(LA);

n\_cu = log(N);

rk\_cu = log(RK);

w\_cu = log(W);

c\_cup = log(C);

iv\_cup = log(IV);

y\_cup = log(Y);

la\_cup = log(LA);

n\_cup = log(N);

rk\_cup = log(RK);

w\_cup = log(W);

% Auxiliary outputs

%The level of the states

auxOut.Xss = exp([k\_cu c\_ba1 a\_cu])';

%The level of the controls

auxOut.Yss = exp([c\_cu iv\_cu y\_cu la\_cu n\_cu rk\_cu w\_cu])';

auxOut.labelx = [{'k\_t'},{'c\_{t-1}'},{'a\_t'}];

auxOut.labely = [{'c\_t'},{'i\_t'},{'y\_t'},{'la\_t'},{'n\_t'},{'rk\_t'},{'w\_t'}];

auxOut.ny = ny;

auxOut.nx = nx;

auxOut.ne = ne;

auxOut.mx = mx;

auxOut.myx = myx;

auxOut.eta = eta;

%1 for a log-transformation, 2 for logistic transformation, 0 for a level approx

auxOut.transformX = ones(1,nx);

auxOut.transformY = ones(1,ny);

%% Section 4: auxiliary function for the steady state

function error = findN(N)

error = THETA\*(1-N)^(-ETAl)\*N^ETAc - (1-BETTA\*B)\*(C\_O\_N\*(1-B))^(-ETAc)\*W;

end

%% Section 3: Reporting the steady state variables for the perturbation approximation

% The value of x and xp

k\_cu = log(K);

c\_ba1 = log(C);

a\_cu = log(A);

k\_cup = log(K);

c\_ba1p = log(C);

a\_cup = log(A);

% The value of y and yp

c\_cu = log(C);

iv\_cu = log(IV);

y\_cu = log(Y);

la\_cu = log(LA);

n\_cu = log(N);

rk\_cu = log(RK);

w\_cu = log(W);

c\_cup = log(C);

iv\_cup = log(IV);

y\_cup = log(Y);

la\_cup = log(LA);

n\_cup = log(N);

rk\_cup = log(RK);

w\_cup = log(W);

% Auxiliary outputs

auxOut.Xss = [k\_cu c\_ba1 a\_cu]';

auxOut.Yss = [c\_cu iv\_cu y\_cu la\_cu n\_cu rk\_cu w\_cu]';

auxOut.labelx = [{'k\_t'},{'c\_{t-1}'},{'a\_t'}];

auxOut.labely = [{'c\_t'},{'i\_t'},{'y\_t'},{'la\_t'},{'n\_t'},{'rk\_t'},{'w\_t'}];

auxOut.ny = ny;

auxOut.nx = nx;

auxOut.ne = ne;

auxOut.mx = mx;

auxOut.myx = myx;

auxOut.eta = eta;

auxOut.logTransY= ones(1,ny); %1 for a log-approx, 0 for a level approx

auxOut.logTransX= ones(1,nx); %1 for a log-approx, 0 for a level approx

function [Ysim,Xsim] = untransformYandX(Ysim,model,Xsim)

[ny,numSim] = size(Ysim);

% Adding the level

Ysim = Ysim + repmat(model.g0,1,numSim);

% undo the transformation

for i=1:ny

if model.transformY(i) == 1

% log-transformation

elseif model.transformY(i) == 2

% logistic transformation

Ysim(i,:) = 1./(1+exp(-Ysim(i,:)));

elseif model.transformY(i) == 0

% Level approximation

end

end

if exist('Xsim','var')

Xsim = Xsim + repmat(model.h0,1,numSim);

nx = size(Xsim,1);

for i=1:nx

if model.transformX(i) == 1

% log-transformation

elseif model.transformX(i) == 2

% logistic transformation

Xsim(i,:) = 1./(1+exp(-Xsim(i,:)));

elseif model.transformX(i) == 0

% Level approximation

end

end

end

end

addpath(genpath(pwd))

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* USER SETTING \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

order\_app = 3; % The order of analytical derivatives which we compute

%% Model specific part

unitFree = 0;

[f,x,xp,y,yp] = RBCmodel(unitFree);

% A string with the output and name of the function computing the steady state

nameSteadyState = '[DELTA,BETTA,B,ETAl,ETAc,THETA,ALFA,RHOA,k\_cu, c\_ba1 ,a\_cu,k\_cup,c\_ba1p,a\_cup,c\_cu ,iv\_cu ,y\_cu ,la\_cu ,n\_cu ,rk\_cu ,w\_cu , c\_cup,iv\_cup,y\_cup,la\_cup,n\_cup,rk\_cup,w\_cup,auxOut,errorMes] = RBCmodel\_ss(params);';

addpath(genpath(pwd))

clear

%% Model specific part

unitFree = 1;

[f,x,xp,y,yp] = RBCmodel(unitFree);

% The parameters of the model

allModelParams = {'DELTA','BETTA','B','ETAl','ETAc','THETA','ALFA','RHOA','STDA'};

% The number of exogenous shocks in the model

ne = 1;

close all

clear

clc

addpath(genpath(pwd))

% This script estimates a basic RBC model by solved by Extended Perturbation by SMM

% using the following moments

% - E[y]

% - E[y(i)\*y(j)] for i=1:ny and j=i,ny

% - E[y(i)\_t\*y(i)\_t-k] for i=1:ny and k=1,2,...autoLagsIdx

% By Martin M. Andreasen, November 2016

%% Section 1: User settings for the SMM estimation

appMethod = 2; % 1 for pruning and 2 for extended perturbation

tau = 5; % model moments computed from a sample of tau\*T observations

orderApp = 2; % Approximation order of DSGE model

T = 100; % Length of the simulated sample path

optim = 2; % 1 for the CMAES

% 2 for a gradient based optimizer

% 3 for the simple algorithm by Nelder-Mead.

numOptimStep1 = 2; % Number of optimizations in step 1

numOptimStep2 = 2; % Number of optimizations in step 1

MaxIter = 5000; % Max number of iterations for the optimizer

MaxEvals = 5000; % Max number of function evaluations for the optimizer

TolFun = 1D-6; % Function tolerance at the objective function

TolX = 1D-6; % Function tolerance for the coefficients

PopSize = 50; % Number of draws per generation i.e. lambda in the CMAES

autoLagsIdx = [1 5]; % Number of lagged covariances included in the GMM estimation

selectY = [1,2,5]'; % The control variables selected for the estimation

qLag = 20; % Number of lags for the Newey-West estimator of W

epsValue = 1e-6; % stepsize for computing standard errors

% Configurations for the Extended Perturnation

setupEPer.Nmax = 200; %Max steps in the perfect foresight solution

setupEPer.Nmin = 20; %Min steps in the perfect foresight solution

setupEPer.maxDistSS = 0.005; %Max distance to SS when determining N

setupEPer.orderAppStart = 3; %Order of approximation for the starting values in the fixed point solver

setupEPer.fixedPointSolver= 1; %1 for the Newton-Raphson solver,

%2 Newton-Raphson solver with optimal delta,

%3 for an LM algorithm that minimizes the residuals

%4 for an LM algorithm that minimizes the weighted residuals

setupEPer.JacobianOption = 3; %1 for using numerical J and to solve the system J\*x = -f

%2 for using analytical J to solve the system J\*x = -f

%3 for using analytical J computed recursively to solve the system J\*x = -f

setupEPer.lambda0 = 1e-6; %Tuining parameter for fixedPointSolver = 3

setupEPer.lambdaBackup = 1e-2; %Tuining parameter for the LM algorithm when used as backup

setupEPer.tolf = 1e-6; %Tolerance level for optimization problem

setupEPer.MaxIter = 1D4; %Maximum number of iterations allowed in the Extended Path algorithm

setupEPer.residualMax = 0.0001; %Max allowed value of a residual - for the hybrid simulator

setupEPer.MexOn = 1; %1 For using MEX-files, else 0

%% Section 2: Model parameters

allModelParams = {'DELTA','BETTA','B','ETAl','ETAc','THETA','ALFA','RHOA','STDA'};

% The variables which we do not select for estimation are calibrated to the assigned values

calibrateParams.ETAl = 1.00;

calibrateParams.THETA = 3.48;

% Starting values

params0.DELTA = 0.025;

params0.BETTA = 0.984;

params0.B = 0.5;

%params0.ETAl = 1;

params0.ETAc = 2;

params0.ALFA = 0.667;

params0.RHOA = 0.979;

params0.STDA = 0.0072;

%% Section 3: We load the data or simulate the model using params0

data = dataForSMM(params0,calibrateParams,orderApp,T,selectY,setupEPer,appMethod);

% Compute empirical moments

[dataMoments,nameMoments] = momentsGMMData(data,autoLagsIdx);

% Transforming params0 for estimation

selectParams = fieldnames(params0);

params0Values = struc2values(params0,selectParams);

% Test of enought moments for estimation

numObs = size(data,2);

numMom = numObs + numObs\*(numObs+1)/2 + numObs\*size(autoLagsIdx,2);

numParams = size(selectParams,1);

disp(['Parameters to estimate = ', num2str(numParams) ,'. Moments for estimation = ', num2str(numMom)]);

if numMom < numParams

error('We must have at least as many moments as parameters for GMM')

end

% For CMAES, the standard deviations for the search

Insigma.DELTA = 0.01;

Insigma.BETTA = 0.01;

Insigma.B = 0.1;

Insigma.ETAl = 1;

Insigma.ETAc = 1;

Insigma.ALFA = 0.05;

Insigma.RHOA = 0.05;

Insigma.STDA = 0.001;

% For CMAES, the lower and upper bounds for the parameters

lowerBounds.DELTA = 0;

lowerBounds.BETTA = 0;

lowerBounds.B = 0;

lowerBounds.ETAl = 0;

lowerBounds.ETAc = 0;

lowerBounds.ALFA = 0;

lowerBounds.RHOA = 0;

lowerBounds.STDA = 0;

upperBounds.DELTA = 1;

upperBounds.BETTA = 1;

upperBounds.B = 1;

upperBounds.ETAl = 10;

upperBounds.ETAc = 10;

upperBounds.ALFA = 1;

upperBounds.RHOA = 1;

upperBounds.STDA = 1;

% Test of all variables names are either estimated or calibrated

paramsTest(allModelParams,params0,calibrateParams)

%% Section 4: Constructing the struct setupStep1

setupStep1.selectParams = selectParams;

setupStep1.calibrateParams = calibrateParams;

setupStep1.data = data;

setupStep1.dataMoments = dataMoments;

setupStep1.autoLagsIdx = autoLagsIdx;

setupStep1.optim = optim;

setupStep1.MaxIter = MaxIter;

setupStep1.MaxEvals = MaxEvals;

setupStep1.TolFun = TolFun;

setupStep1.TolX = TolX;

setupStep1.PopSize = PopSize;

setupStep1.selectY = selectY;

setupStep1.orderApp = orderApp;

setupStep1.Insigma = Insigma;

setupStep1.lowerBounds = lowerBounds;

setupStep1.upperBounds = upperBounds;

setupStep1.optimWeightMat = 0;

setupStep1.qLag = qLag;

setupStep1.nameMoments = nameMoments;

setupStep1.tau = tau;

setupStep1.appMethod = appMethod;

setupStep1.setupEPer = setupEPer;

%% Section 5: Step 1 of the SMM estimation

Wstep1 = getOptimalWeighting(qLag,dataMoments,setupStep1);

setupStep1.Sw = chol(diag(diag(Wstep1)));

for i=1:numOptimStep1

[paramsStep1,setupStep1] = runOptimization(setupStep1,params0);

params0 = paramsStep1;

end

resultsStep1 = getSESMM(struct2array(paramsStep1)',epsValue,setupStep1);

%% Section 6: Step 2 of the SMM estimation

[~,modelMoments] = objectFunc(struct2array(paramsStep1)',setupStep1);

Wopt = getOptimalWeighting(qLag,modelMoments,setupStep1);

setupStep2 = setupStep1;

setupStep2.Sw = chol(Wopt);

setupStep2.optimWeightMat = 1;

disp(['Rank of Wopt = ', num2str(rank(Wopt))])

params0Step2 = paramsStep1;

for i=1:numOptimStep2

[paramsStep2,setupStep2] = runOptimization(setupStep2,params0Step2);

params0Step2 = paramsStep2;

end

% The standard errors

resultsStep2 = getSESMM(struct2array(paramsStep2)',epsValue,setupStep2);

resultsStep1 =

params: [1x1 struct]

paramsSE: [1x1 struct]

Jacobian: [15x8 double]

Q: 0.4853

model: [1x1 struct]

modelMoments: [15x1 double]

dataMoments: [15x1 double]

nameMoments: {1x15 cell}

modelMomentsScaled: [15x1 double]

nameMomentsScaled: {1x15 cell}

dataMomentsScaled: [15x1 double]

resultsStep2 =

Jtest: 5.4194

JtestDf: 7

ProbJtest: 0.6089

params: [1x1 struct]

paramsSE: [1x1 struct]

Jacobian: [15x8 double]

Q: 0.0090

model: [1x1 struct]

modelMoments: [15x1 double]

dataMoments: [15x1 double]

nameMoments: {1x15 cell}

modelMomentsScaled: [15x1 double]

nameMomentsScaled: {1x15 cell}

dataMomentsScaled: [15x1 double]