```
% Script construido baseado nos scripts de B.Moll
% Fonte original em: http://www.princeton.edu/~moll/HACTproject.htm

%% Limpeza de Variaveis
clear all; clc; close all;

rho = 0.05;
r = 0.03;
z1 = 1;
eta = 0.75;

w=1;

I= 150;
amin = -0.15;
amax = 3;
a = linspace(amin,amax,I)';
da = (amax-amin)/(I-1);

maxit= 10000;
crit = 10^(-6);
Delta = 1000;

dVf = zeros(I,1);
dVb = zeros(I,1);
c = zeros(I,1);

options=optimset('Display','off');
x0 = 1;


% Define w e r
AA = 1;
delta = 0.06;
alpha = 0.33;

check = 1;

r_high = rho;
r_low = 0;

while check ==1

    w = (1-alpha) * AA *((r+delta)/(AA*alpha))^(alpha/(alpha-1));


    tic;
    for i=1:I
        params = [a(i),z1,w,r];
        myfun = @(l) SolveLabor(l,params);
        [l01,fval,exitflag] = fzero(myfun,x0,options);

        l0(i,:)=[l01];
    end
    toc
```

```
    v0(:,1) = log(w*z1.*l0(1,1) + r.*a)/rho;

    lmin = l0(1,:);
    lmax = l0(I,:);

    v = v0;

    for n=1:maxit
        V = v;
        V_n(:,n)=V;
        % forward difference
        dVf(1:I-1) = (V(2:I)-V(1:I-1))/da;
        dVf(I) = (w*z1.*lmax + r.*amax).^(-1); %state constraint boundary condition
        % backward difference
        dVb(2:I) = (V(2:I,:)-V(1:I-1,:))/da;
        dVb(1) = (w*z1.*lmin + r.*amin).^(-1); %state constraint boundary condition

        %consumption and savings with forward difference
        cf = dVf.^(-1);
        lf = 1-(dVf.*w.*z1/eta).^(-1);
        ssf = w*z1.*lf + r.*a - cf;
        %consumption and savings with backward difference
        cb = dVb.^(-1);
        lb = 1-((dVb.*w.*z1/eta).^(-1));
        ssb = w*z1.*lb + r.*a - cb;
        %consumption and derivative of value function at steady state
        c0 = w*z1.*l0 + r.*a;
        dV0 = c0.^(-1);

        Ib = ssb < 0; %negative drift --> backward difference
        If = (ssf > 0).*(1-Ib); %positive drift --> forward difference
        I0 = (1-If-Ib); %at steady state

        c = cf.*If + cb.*Ib + c0.*I0;
        l = lf.*If + lb.*Ib + l0.*I0;
        u = log(c) + eta*log(1-l);

        %CONSTRUCT MATRIX
        X = -Ib.*ssb/da;
        Y = -If.*ssf/da + Ib.*ssb/da;
        Z = If.*ssf/da;

        A1=spdiags(Y(:,1),0,I,I)+spdiags(X(2:I,1),-1,I,I)+spdiags([0;Z(1:I-1,1)],1,I,↙
I);

        A = A1;
        B = (1/Delta + rho)*speye(I) - A;

        u_stacked = [u(:)];
        V_stacked = [V(:)];

        b = u_stacked + V_stacked/Delta;
        V_stacked = B\b; %SOLVE SYSTEM OF EQUATIONS
```

```
        Vchange = V_stacked - v;
        v = V_stacked;

        dist(n) = max(abs(Vchange));
        if dist(n)<crit
            disp('Value Function Converged, Iteration = ')
            disp(n)
            break
        end
    end
    toc;



    %% MARKET CLEARING CONDITIONS

    %%%%%%%%%%%%%%%%%%%%%%%%%%
    % FOKKER-PLANCK EQUATION %
    %%%%%%%%%%%%%%%%%%%%%%%%%%
    AT = A';
    b = zeros(I,1);

    %need to fix one value, otherwise matrix is singular
    i_fix = 1;
    b(i_fix)=.1;
    row = [zeros(1,i_fix-1),1,zeros(1,I-i_fix)];
    AT(i_fix,:) = row;

    %Solve linear system
    gg = AT\b;
    g_sum = gg'*ones(I,1)*da;
    gg = gg./g_sum;
    g = gg;

    check1 = g(:,1)'*ones(I,1)*da;

    Asset_Supply = g(:,1)'*a*da;

    if Asset_Supply > crit
        r_high = r;
        r = (r_high + r_low)/2;
    elseif Asset_Supply < -crit
        r_low = r;
        r = (r_high + r_low)/2;
    else
        check = 0;
    end
end
fprintf('\nFunção convergiu.\n')
fprintf('taxa de juros encontrada: %2.3f\n', r)
```