problem_sets/problem_set_4/problem_set_4_code.m

```matlab
%{
------------------------------------------------------------------------------------------------------------------------
Purpose: Coding part of problem set 4
Created: Nico Rotundo 2024-11-26 (Adapted from Kiyea's section code)
------------------------------------------------------------------------------------------------------------------------
%}

%% Define parameters and initialize grids

%{
------------------------------------------------------------------------------------------------------------------------
Define parameters
------------------------------------------------------------------------------------------------------------------------
%}

clear all;

% Relative risk aversion coefficient
sigma = 2;

% Income
z_e = 0.2;
z_u = 0.1;
z = [z_e, z_u];

% Transition rates
lambda_e = 1.2;
lambda_u = 1.2;
lambda = [lambda_e, lambda_u];

% Discount rate
rho = 0.05;

%{
------------------------------------------------------------------------------------------------------------------------
Define asset grid
------------------------------------------------------------------------------------------------------------------------
%}

% Borrowing constraint minimum
a_min = -0.15;

% Borrowing constraint maximum
a_max = 5;

% Number of grid points
num_points = 1000;

%{
------------------------------------------------------------------------------------------------------------------------
Define utility function and its derivative
------------------------------------------------------------------------------------------------------------------------
%}

% Utility function (CRRA), handling vector inputs
U = @(c) (c.^(1 - sigma)) / (1 - sigma);

% Derivative of utility, handling vector inputs
U_prime = @(c) c.^(-sigma);

% Inverse of derivative of utility
U_prime_inv = @(Vp) (Vp).^(-1 / sigma);

%{
------------------------------------------------------------------------------------------------------------------------
Tuning parameters (general)
------------------------------------------------------------------------------------------------------------------------
%}

% Step size: can be arbitrarily large in implicit method
Delta = 1000;

% The maximum number of value function iterations
max_iterations_vf = 100;

% Tolerance for value function iterations
tolerance = 10^(-6);

 %{
------------------------------------------------------------------------------------------------------------------------
Tuning parameters (interest rate iteration)
------------------------------------------------------------------------------------------------------------------------
%}

% The maximum number of interest rate iterations
num_iterations_r = 1000;

% Tolerance for interest rate iterations
tolerance_S = 10^(-5);

a_grid = linspace(a_min, a_max, num_points)';
da = (a_max-a_min)/(num_points-1);

aa = [a_grid, a_grid]; % I*2 matrix

%% Initialize matrices before iterating

%{
------------------------------------------------------------------------------------------------------------------------
Define differential operator
------------------------------------------------------------------------------------------------------------------------
%}


% Forward; satisfies Df*V=dVf
Df = zeros(num_points, num_points);
for i = 1:num_points-1
    Df(i,i) = -1/da; Df(i,i+1) = 1/da;
end
Df = sparse(Df);

% Backward; satisfies Db*V=dV
Db = zeros(num_points, num_points);
for i = 2:num_points
    Db(i,i-1) = -1/da; Db(i,i) = 1/da;
end
Db = sparse(Db);

%{
------------------------------------------------------------------------------------------------------------------------
Define A-switch matrix
------------------------------------------------------------------------------------------------------------------------
%}

A_switch = [speye(num_points).*(-lambda(1)), speye(num_points).*lambda(1);
            speye(num_points).*lambda(2), speye(num_points).*(-lambda(2))];

%{
------------------------------------------------------------------------------------------------------------------------
Define initial guesses
------------------------------------------------------------------------------------------------------------------------
%}

% Guess for initial value of the interest rate
r_0_guess = 0.03;

% Set bounds on interest rate
r_min = 0.01;
r_max = 0.04;

% Define the number of points * 2 matrix
z = ones(num_points, 1).*z;

% The value function of "staying put"
r = r_0_guess;

% Initial guess for value function
V_0 = U(z + r.*aa)./rho;
V = V_0;

%% Iterate over interest rates and value function

% Loop over number of iterations for the interest rate
for nr=1:num_iterations_r

    % Set interest rate equal to current
    r_r(nr) = r;

    % Set bounds on the interest rate
    rmin_r(nr) = r_min;
    rmax_r(nr) = r_max;

    % Use the value function solution from the previous interest rate iteration
    % as the initial guess for the next iteration
    if nr>1
        V_0 = V_r(:,:,nr-1);
        V = V_0;
    end

    %% Value function iteration

    % Loop over number of interations for the value function
    for n=1:max_iterations_vf

        % Derivative of the forward value function
        dVf = Df*V;

        % Derivative of the backward value function
        dVb = Db*V;

        % Boundary condition on backwards value function i.e., the borrowing constraint; a>=a_min
        dVb(1,:) = U_prime(z(1,:) + r.*aa(1,:));

        % Boundary condition on forward value function; a<=a_max
        dVf(end,:) = U_prime(z(end,:) + r.*aa(end,:));

        % Indicator whether value function is concave; for stability purposes
        I_concave = dVb > dVf;

        % Compute optimal consumption using forward derivative
        cf = U_prime_inv(dVf);

        % Compute optimal consumption using backward derivative
        cb = U_prime_inv(dVb);

        % Compute optimal savings using forward derivative
        sf = z + r.*aa - cf;

        % Compute optimal savings using backward derivative
        sb = z + r.*aa - cb;

        % Upwind scheme
        If = sf>0;
        Ib = sb<0;
        I0 = 1-If-Ib;
        dV0 = U_prime(z + r.*aa); % If sf<=0<=sb, set s=0

        dV_upwind = If.*dVf + Ib.*dVb + I0.*dV0;

        c = U_prime_inv(dV_upwind);

        % Update value function
        V_stacked = V(:);

        % Update consumption function
        c_stacked = c(:);

        % A = SD
        SD_u = spdiags(If(:,1).*sf(:,1), 0, num_points, num_points)*Df + spdiags(Ib(:,1).*sb(:,1), 0, num_points, num_points)*Db;
        SD_e = spdiags(If(:,2).*sf(:,2), 0, num_points, num_points)*Df + spdiags(Ib(:,2).*sb(:,2), 0, num_points, num_points)*Db;
        SD = [SD_u, sparse(num_points, num_points);
            sparse(num_points, num_points), SD_e];

        % P = A + A_switch
        P = SD + A_switch;

        % B = [(rho + 1/Delta)*I - P]
        B = (rho + 1/Delta)*speye(2*num_points) - P;

        % b = u(c) + 1/Delta*V
        b = U(c_stacked) + (1/Delta)*V_stacked;

        % V = B\b;
        V_update = B\b;
        V_change = V_update - V_stacked;
        V = reshape(V_update, num_points, 2);

        % Convergence criterion
        dist(n) = max(abs(V_change));

        if dist(n)<tolerance
            disp('Value function converged. Iteration = ')
            disp(n)
        break

        end
    end

    %% KF Equation

    % Solve for 0=gdot=P'*g
    PT = P';
    gdot_stacked = zeros(2*num_points,1);

    % Fix one value to obtain a non-singular matrix, otherwise matrix is singular
    i_fix = 1;
    gdot_stacked(i_fix)=.1;

    row_fix = [zeros(1,i_fix-1),1,zeros(1,2*num_points-i_fix)];
    AT(i_fix,:) = row_fix;

    g_stacked = PT\gdot_stacked;

    % Normalization
    g_sum = g_stacked'*ones(2*num_points,1)*da;
    g_stacked = g_stacked./g_sum;

    % Reshape
    gg = reshape(g_stacked, num_points, 2);

    % Compute interest-rate dependent variables for this iteration
    g_r(:,:,nr) = gg;
    adot(:,:,nr) = z + r.*aa - c;
    V_r(:,:,nr) = V;
    dV_r(:,:,nr) = dV_upwind;
    c_r(:,:,nr) = c;

    S(nr) = gg(:,1)'*a_grid*da + gg(:,2)'*a_grid*da;

    %% Update interest rate using Newton's method

    % Only if iteration is past 1
    if nr > 1

        % Change in S
        dS = S(nr) - S(nr-1);

        % Change in r
        dr = r - r_old;

        % Derivative of S matrix
        S_prime = dS/dr;

        % Save current iteration interest rate
        r_old = r;

        % Ppdate interest rate
        r = r - S(nr)/S_prime;

    else
            r_old = r;
            r = r + 1e-4;
    end

    % Check that convergence
    if sqrt(sum((r-r_old).^2))<tolerance
        disp(r)
        break
    end

end

%% Plots

%{
------------------------------------------------------------------------------------------------------------------------
Question 1: Optimal consumption
------------------------------------------------------------------------------------------------------------------------
%}

% Initialize plot
set(gca, 'FontSize', 18)

    % Employed
    plot(a_grid, c_r(:,1,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', [41/255, 182/255, 164/255])
    hold on

    % Unemployed
    plot(a_grid, c_r(:,2,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', [250/255, 165/255, 35/255])

    % Blank line for legend
    plot(NaN, NaN, 'LineStyle', 'none')

    hold off

    grid
        xlabel('Wealth, a','FontSize', 14)

        set(gca, 'TickDir', 'out');
            box off;

        ylabel('Consumption, c_j(a)','FontSize', 14)

        xlim([a_min a_max])

        legend(sprintf('Employed'), ...
            sprintf('Unemployed'), ...
            sprintf('r=%.4f', r), 'Location', 'best', 'FontSize', 14)

% Export graph
exportgraphics(gcf, '/Users/nicorotundo/Documents/GitHub/DynamicProgramming2024/problem_sets/problem_set_4/output/question_1.pdf', 'ContentType', 'vector');

%{
------------------------------------------------------------------------------------------------------------------------
Question 2: Optimal savings
------------------------------------------------------------------------------------------------------------------------
%}

% Initialize plot
set(gca, 'FontSize', 18)

    % Employed
    plot(a_grid, adot(:,1,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', [41/255, 182/255, 164/255])
    hold on

    % Unemployed
    plot(a_grid, adot(:,2,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', [250/255, 165/255, 35/255])

    % Blank line for legend
    plot(NaN, NaN, 'LineStyle', 'none')
    hold off

    grid
        xlabel('Wealth, a', 'FontSize', 14)

        ylabel('Saving, s_j(a)', 'FontSize', 14)

        set(gca, 'TickDir', 'out');
            box off;

        xlim([a_min a_max])

        legend(sprintf('Employed'), ...
            sprintf('Unemployed'), ...
            sprintf('r=%.4f', r), 'Location', 'best', 'FontSize', 14)

% Export graph
exportgraphics(gcf, '/Users/nicorotundo/Documents/GitHub/DynamicProgramming2024/problem_sets/problem_set_4/output/question_2.pdf', 'ContentType', 'vector');

%{
------------------------------------------------------------------------------------------------------------------------
Question 3: Wealth distribution
------------------------------------------------------------------------------------------------------------------------
%}

% Initialize plot
set(gca, 'FontSize', 14)

    % Employed
    plot(a_grid, g_r(:,1,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', [41/255, 182/255, 164/255])
    hold on

    % Unemployed
    plot(a_grid, g_r(:,2,nr), 'LineWidth', 2, 'LineStyle', '-', 'Color', [250/255, 165/255, 35/255])

    % Blank line for legend
    plot(NaN, NaN, 'LineStyle', 'none')

    hold off

    grid
        xlabel('Wealth, a', 'FontSize', 14)

        set(gca, 'TickDir', 'out');
            box off;

        ylabel('Densities, g_j(a)', 'FontSize', 14)

        yy = get(gca, 'yLim');

        hold on

        plot([a_min, a_min], yy, '--k', 'LineWidth', 2)

        hold off

        text(-0.15, yy(1)-0.02*(yy(2) - yy(1)), '$\underline{a}$', 'HorizontalAlignment', 'center', 'FontSize', 15, 'Interpreter', 'latex')

        xlim([-0.2 1])

        legend(sprintf('Employed'), ...
            sprintf('Unemployed'), ...
            sprintf('r=%.4f', r), 'Location', 'best', 'FontSize', 14)

% Export graph
exportgraphics(gcf, '/Users/nicorotundo/Documents/GitHub/DynamicProgramming2024/problem_sets/problem_set_4/output/question_3.pdf', 'ContentType', 'vector');
```