

problem_sets/problem_set_5/problem_set_5_question_3.m

```
1 %{\n2\n3 Purpose: Coding part of problem set 5\n4 Created: Nico Rotundo 2024-12\n5\n6 %}\n7\n8 %% Define parameters and initialize grids\n9\n10 %{\n11\n12 Define parameters\n13\n14 %}\n15\n16 clear all;\n17\n18 % Relative risk aversion coefficient\n19 sigma = 2;\n20\n21 % Productivity\n22 z_u = 1;\n23 z_e = 2;\n24 z = [z_u, z_e];\n25\n26 % Transition rates\n27 lambda_u = 1/3;\n28 lambda_e = 1/3;\n29 lambda = [lambda_u, lambda_e];\n30\n31 % Discount rate\n32 rho = 0.05;\n33\n34 % Capital depreciation rate\n35 delta = 0.05;\n36\n37 % Capital share in production\n38 alpha = 1/3;\n39\n40 % TFP\n41 A = .1;\n42\n43 % Interest rate for partial equilibrium part of the problem set\n44 r = 0.04;\n45\n46 %{\n47\n48 Define capital grid\n49\n50 %}\n51\n52 % Borrowing constraint minimum\n53 k_min = 0;\n54\n55 % Borrowing constraint maximum\n56 k_max = 20;\n57\n58 % Number of grid points\n59 num_points = 1000;\n60\n61 k_grid = linspace(k_min, k_max, num_points)';\n62 dk = (k_max-k_min)/(num_points-1);\n63\n64 kk = [k_grid, k_grid]; % I*2 matrix\n65\n66 %{\n67\n68 Define utility function and its derivative\n69\n70 %}\n71\n72 % Utility function (CRRA), handling vector inputs\n73 U = @(c) (c.^(1 - sigma)) / (1 - sigma);\n74\n75 % Derivative of utility, handling vector inputs\n76 U_prime = @(c) c.^(-sigma);\n77\n78 % Inverse of derivative of utility\n79 U_prime_inv = @(Vp) (Vp).^(-1 / sigma);\n80\n81 %{\n82\n83 Firm production technology and FOCs\n84\n85 %}\n86\n87 % Production technology Y = A K^alpha L^(1-alpha)\n88 Y = @(K,L) K.^alpha*L.^(1-alpha);\n89\n90 % F.O.C.: \n91 w_foc = @(K,L) (1-alpha)*A*(K/L).^alpha;\n92 r_foc = @(K,L) alpha*A*(L/K).^(1-alpha);\n93 K_partial = @(L,r) L*((alpha*A/(r+delta)).^(1/(1-alpha))) ;\n94\n95 %{\n96\n97 Tuning parameters (general)\n98\n99 %}\n100\n101 % Step size: can be arbitrarily large in implicit method\n102 Delta = 1000;\n103\n104 % The maximum number of value function iterations\n105 max_iterations_vf = 100;\n106\n107 % Tolerance for value function iterations\n108 tolerance = 10^(-6);\n109\n110 %{\n111\n112 Tuning parameters (interest rate iteration)\n113\n114 %}\n115\n116 % The maximum number of interest rate iterations\n117 num_iterations_r = 1000;\n118\n119 % Tolerance for interest rate iterations\n120 tolerance_5 = 10^(-5);\n121\n122 %% Initialize matrices before iterating\n123\n124 %{\n125\n126 Define differential operator\n127\n128 %}\n129\n130 % Forward; satisfies Df*V=dVf\n131 Df = zeros(num_points, num_points);\n132 for i = 1:num_points-1\n133     Df(i,i) = -1/dk; Df(i,i+1) = 1/dk;\n134 end\n135 Df = sparse(Df);\n136\n137 % Backward; satisfies Db*dV=dV\n138 Db = zeros(num_points, num_points);\n139 for i = 2:num_points\n140     Db(i,i-1) = -1/dk; Db(i,i) = 1/dk;\n141 end\n142 Db = sparse(Db);\n143\n144 %{\n145\n146 Define A-switch matrix\n147\n148 %}\n149\n150\n151 A_switch = [speye(num_points).*(-lambda(1)), speye(num_points).*lambda(1);\n152             speye(num_points).*lambda(2), speye(num_points).*(-lambda(2))];\n153\n154\n155 %{\n156\n157 Calculate labor, capital, and wage levels\n158\n159 %}\n160\n161 % Labor\n162 L = (z_e*lambda_u + z_u*lambda_e)/(lambda_e+lambda_u);\n163\n164 % Capital\n165 K = K_partial(L,r);\n166\n167 % Wage\n168 w = w_foc(K,L);\n169\n170 %{\n171\n172 Define initial guesses\n173\n174 %}\n175\n176 % Guess for initial value of the interest rate\n177 r_0_guess = 0.03;\n178\n179 % Set bounds on interest rate\n180 r_min = 0.01;\n181 r_max = 0.04;\n182\n183 % Define the number of points * 2 matrix\n184 z = ones(num_points, 1).*z;\n185\n186 % Initial guess for value function\n187 V_0 = U(w.*z + r.*kk)./rho;\n188 V = V_0;\n189\n190 %% Value function iteration\n191\n192 % Loop over number of iterations for the value function\n193 for n=1:max_iterations_vf\n194\n195     % Derivative of the forward value function\n196     dVf = Df*V;\n197\n198     % Derivative of the backward value function\n199     dVb = Db*V;\n200\n201     % Boundary condition on backwards value function i.e., the borrowing constraint; a=a_min\n202     dVb(1,:) = U_prime(w.*z(1,:) + r.*kk(1,:));\n203\n204     % Boundary condition on forward value function; a=a_max\n205     dVf(end,:) = U_prime(w.*z(end,:) + r.*kk(end,:));\n206\n207     % Indicator whether value function is concave; for stability purposes\n208     I_concave = dVb > dVf;\n209\n210     % Compute optimal consumption using forward derivative\n211     cf = U_prime_inv(dVf);\n212\n213     % Compute optimal consumption using backward derivative\n214     cb = U_prime_inv(dVb);\n215\n216     % Compute optimal savings using forward derivative\n217     sf = w.*z + r.*kk - cf;\n218\n219     % Compute optimal savings using backward derivative\n220     sb = w.*z + r.*kk - cb;\n221\n222     % Upwind scheme\n223     If = sf>0;\n224     Ib = sb<0;\n225     I0 = 1-If-Ib;\n226     dV0 = U_prime(w.*z + r.*kk); % If sf<=0<=sb, set s=0\n227\n228     dV_upwind = If.*dVf + Ib.*dVb + I0.*dV0;\n229\n230     c = U_prime_inv(dV_upwind);\n231\n232     % Update value function\n233     V_stacked = V(:);\n234\n235     % Update consumption function\n236     c_stacked = c(:);\n237\n238     % A = SD\n239     SD_u = spdiags(If(:,1).*sf(:,1), 0, num_points, num_points)*Df + spdiags(Ib(:,1).*sb(:,1), 0, num_points, num_points)*Db;\n240     SD_e = spdiags(If(:,2).*sf(:,2), 0, num_points, num_points)*Df + spdiags(Ib(:,2).*sb(:,2), 0, num_points, num_points)*Db;\n241     S = [SD_u, sparse(num_points, num_points);\n242          sparse(num_points, num_points), SD_e];\n243\n244     % P = A + A_switch\n245     P = SD + A_switch;\n246\n247     % B = [(rho + 1/Delta)*I - P]\n248     B = (rho + 1/Delta)*speye(2*num_points) - P;\n249\n250     % b = u(c) + 1/Delta*v\n251     b = U(c_stacked) + (1/Delta)*V_stacked;\n252\n253     % V = B\\b;\n254     V_update = B\\b;\n255     V_change = V_update - V_stacked;\n256     V = reshape(V_update, num_points, 2);\n257\n258     % Convergence criterion\n259     dist(n) = max(abs(V_change));\n260\n261     if dist(n)<tolerance\n262         disp('Value function converged. Iteration = ')\n263         disp(n)\n264     end\n265\n266 end\n267\n268 %% KF Equation\n269\n270 % Solve for 0=gdot=P'*g\n271 PT = P';\n272 PT_eigs = PT;\n273 gdot_stacked = zeros(2*num_points,1);\n274\n275 % Fix one value to obtain a non-singular matrix, otherwise matrix is singular\n276 i_fix = 1;\n277 gdot_stacked(i_fix)=1;\n278\n279 row_fix = [zeros(1,i_fix-1),1,zeros(1,2*num_points-i_fix)];\n280 PT(i_fix,:) = row_fix;\n281\n282 g_stacked = PT\\gdot_stacked;\n283\n284 % Normalization\n285 g_sum = g_stacked'*ones(2*num_points,1)*dk;\n286 g_stacked = g_stacked./g_sum;\n287\n288 % Reshape\n289 gg = reshape(g_stacked, num_points, 2);\n290\n291 % Solve KF equation\n292 [g_stacked_eigs, eigval] = eigs(PT_eigs, 1, 0);\n293 g_sum_eigs = g_stacked_eigs'*ones(2*num_points,1)*dk;\n294 g_stacked_eigs = g_stacked_eigs./g_sum_eigs;\n295\n296 % Plots\n297\n298 %{\n299\n300 Question 3: Stationary distribution at r = 0.04\n301\n302 %}\n303\n304\n305 % Initialize plot\n306 set(gca, 'FontSize', 18)\n307\n308 % Employed\n309 plot(k_grid, gg(:,2), 'LineWidth', 2, 'LineStyle', '-', 'Color', [41/255, 182/255, 164/255])\n310 hold on\n311\n312 % Unemployed\n313 plot(k_grid, gg(:,1), 'LineWidth', 2, 'LineStyle', '-', 'Color', [250/255, 165/255, 35/255])\n314\n315 hold off\n316\n317 grid\n318 xlabel('Capital, k', 'FontSize', 14)\n319\n320 ylabel('Densities, g_j(k)', 'FontSize', 14)\n321\n322 set(gca, 'TickDir', 'out');\n323 box off;\n324\n325 xlim([-0.1 1])\n326\n327 legend(sprintf('Employed'), ... \n328         sprintf('Unemployed'), ... \n329         sprintf('r=%4f', r), 'Location', 'best', 'FontSize', 14)\n330\n331 % Export graph\n332 exportgraphics(gcf, '/Users/nicorotundo/Documents/GitHub/DynamicProgramming2024/problem_sets/problem_set_5/output/question_3.pdf', 'ContentType', 'vector');
```