ECON 6130

Problem Set 8

Gabe Sekeres

December 8, 2024

1. **Projection Method:**

   (a) I completed this step using code from the previous problem set, and attained the same matrices g_x and h_x as in the previous problem set.

   (b) We know that productivity evolves according to the AR(1) process

   $$\log(A_t) = \rho \log(A_{t-1}) + \sigma_a \varepsilon_t$$

   This implies that $\text{Var}(A_t) = \frac{\sigma_a^2}{1-\rho^2}$, so we have that $\text{std}(A_t) = \sqrt{\frac{\sigma_a^2}{1-\rho^2}}$. Solving in our model, we get that `stda` $= 0.032026$, which is the same as what Ryan got.

   (c) Using the given Gaussian-Hermite Quadrature function, I created the grid of shocks and probabilities, and got that $\mathbb{E}[\varepsilon_{t+1}^2] = 1 \cdot 10^{-4} = (1 \cdot 10^{-2})^2 = \sigma_a^2$.

   (d) I created the grid of possible future realizations of $\log(A_t)$, and got that

   $$A' = \begin{bmatrix} 0.9583 & 0.9776 & 0.9972 & 1.0173 & 1.0378 & 1.0587 & 1.0801 \\ 0.9410 & 0.9602 & 0.9799 & 1.0000 & 1.0205 & 1.0414 & 1.0627 \\ 0.9236 & 0.9429 & 0.9626 & 0.9827 & 1.0032 & 1.0241 & 1.0454 \end{bmatrix}$$

   (e) I created the capital and labor grids, using the same strategy as in Problem Set 7.

   (f) I did this

   (g) Yup did this too

   (h) I created the function `pset8_residual`, which is below in the code section. For completeness, I will now proceed to list parts (i) to (m) individually.

   (i) Part (i)!

   (j) Did part (j)

   (k) Yeah this too

   (l) I really dislike gradescope.

   (m) Also LaTeX enumerate environment.

   (n) I ran the residual function with the initial policy guesses. Annoyingly, I got an initial residual result of 0.0340 rather than 0.0341. I've tried debugging this a number of times, and cannot seem to find the error. It's close to the true value, but far enough that it's clearly not a rounding error or a floating point issue. I'm not sure what's happening, I'll complete the rest of the problem set assuming that my code is correct.

   (o) Using `fsolve`, I got that 3 iterations were required. The employment policy function when A, K, and N were minimized was

   `npol(1, 1, 1)` $= 16.0896$

(p) I created this plot, which is Figure 1. We can see that the capital policy functions are almost exactly the same as the linear model – so much so that their difference is not noticeable (trust me that the green line was plotted correctly, I checked). However, the labor policy function is noticeably different.

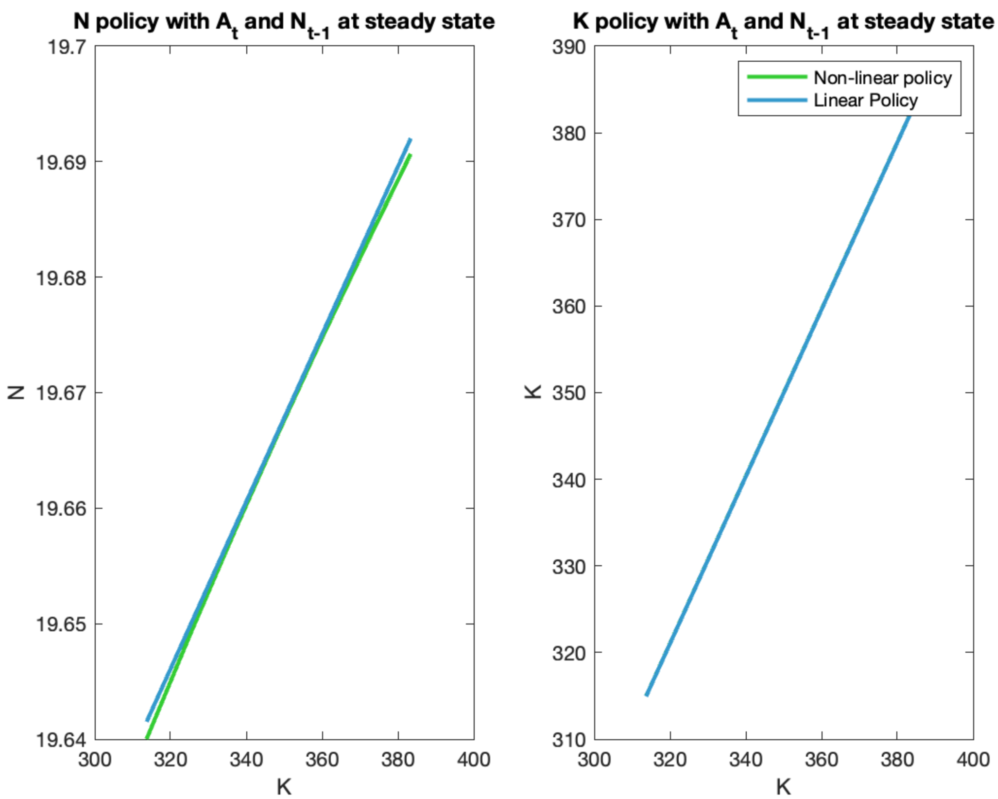

Figure 1: Linear and Projected Policy Functions

(q) I simulated the economy for 5,000 periods, and generated the following table:

Moment	**Linear Model Value**	**Value Function Model Value**
std log(Y)	0.0522	0.00099
std log(C)	0.0391	0.00979
std log(I)	0.0975	0.02195
std log(N)	0.0119	0.00056

2. **Matlab Code:** The Matlab code runs in 17.709 seconds. As a winter break project, I'll be attempting to transition the code to Julia and optimize it. I created the files `pset_solve_projection.m` and `pset8_residual.m`, and used the usual helper functions and parameter function.

   (a) `pset_solve_projection.m`

```
% Solve for the linear model
% (code from previous pset)

clear;
format long;
tic;

% Add helper functions
addpath('/Users/gabesekeres/Dropbox/Notes/Cornell_Notes/Fall_2024/
    Macro/Matlab/pset8_helper_functions')

% Load parameters
param = pset8_parameters;

bet = param.bet;
sig = param.sig;
alpha = param.alpha;
deltak = param.deltak;
deltan = param.deltan;
phin = param.phin;
chi = param.chi;
eps = param.eps;
rho = param.rho;
siga = param.siga;

% Solve the linear model
pset7_linear_model;
rehash;
pval  = struct2array(param);

[fy,fx,fyp,fxp,ftest,yxss] = pset7_model_df(pval');
[gx,hx] = gx_hx_alt(fy,fx,fyp,fxp);
eta = [0 ; 1];
disp('gx:');
disp(gx);
disp('hx:');
disp(hx);
disp('yxss:');
disp(yxss);

% Put parameter value in memory
passign(param);

% Steady state values:
abar = yxss(a_idx);
kbar = yxss(k_idx);
cbar = yxss(c_idx);
```

```
nbar = yxss(n_idx);
vbar = yxss(val_idx);

% Set grid of shocks
na = 7;
stda = sqrt(siga^2/(1-rho^2));
disp('stda:');
disp(stda);

agrid = exp(linspace(-2*stda,2*stda,na));
[~, epsi, pw] = GH_Quadrature(3,1,1);
egrid = epsi .* siga;

disp('egrid^2:');
disp(egrid'.^2 * pw);

% Create grid for potential realizations of next period's shock
aprime = exp(rho * log(agrid)) + egrid;
disp('aprime:');
disp(aprime);

% Capital grid
nk = 7;
kgrid = linspace(0.9*kbar, 1.1*kbar, nk);

% Labor grid
nn = 21;
ngrid = linspace(0.8*nbar, 1.2*nbar, nn);

% Combination grid
[aagr, kkrg, nngr] = ndgrid(agrid, kgrid, ngrid);
aagr = aagr(:)';
kkrg = kkrg(:)';
nngr = nngr(:)';

state_grid = [aagr; kkrg; nngr];

% Initial policy function guesses
kinit = kbar + 1 * hx(2,:) * [[aagr - abar]; [kkrg - kbar]; [nngr -
    nbar]];
ninit = nbar + 1 * gx(n_idx,:) * [[aagr - abar]; [kkrg - kbar]; [nngr
    - nbar]];

kinit = reshape(kinit, [nk, na, nn]);
ninit = reshape(ninit, [nn, na, nk]);

% For residual function
allgrid = {agrid, kgrid, ngrid};

% Initial residual:
resid0 = pset8_residual(kinit(:)', ninit(:)', state_grid, allgrid,
```

```
    aprime, pw, param);
disp('Initial residual:');
disp(sum(abs(resid0(:))));


% Solve for policy functions using projection method
ns = size(state_grid,2);
obj = @(x) pset8_residual(x(1:ns),x(ns+1:2*ns),state_grid,allgrid,
    aprime,pw,param);
options         = optimoptions('fsolve');
options.Display = 'iter';

xout = fsolve(obj,[kinit(:)',ninit(:)'],options);

% Reshape the output to get the final policy functions
kpol = reshape(xout(1:ns), [na, nk, nn]);
npol = reshape(xout(ns+1:end), [na, nk, nn]);

% Display the value of the final employment policy function (npol) at
    the lowest levels of K, A, and N
disp('Final employment policy function (npol) at the lowest levels of
    K, A, and N:');
disp(npol(1, 1, 1));

% Linear policy functions
kinitpol = reshape(kinit,[na,nk,nn]);
ninitpol = reshape(ninit,[na,nk,nn]);

% Define colors
calm_blue = [0.2, 0.6, 0.8];
calm_green = [0.2, 0.8, 0.2];

% Fix A_t and N_{t-1} at their steady state values
a_fixed_idx = find(agrid == abar, 1);
n_fixed_idx = find(ngrid == nbar, 1);

% Get policy functions at steady state
npol_ss = npol(a_fixed_idx, :, n_fixed_idx);
kpol_ss = kpol(a_fixed_idx, :, n_fixed_idx);
ninitpol_ss = ninitpol(a_fixed_idx, :, n_fixed_idx);
kinitpol_ss = kinitpol(a_fixed_idx, :, n_fixed_idx);

figure;
subplot(1,2,1);
plot(kgrid, npol_ss, 'linewidth', 2, 'Color', calm_green);
ylabel('N'); xlabel('K'); title('N policy with A_t and N_{t-1} at
    steady state');
hold on;
plot(kgrid, ninitpol_ss, 'LineWidth', 2, 'Color', calm_blue);

subplot(1,2,2);
```

```
plot(kgrid, kpol_ss, 'linewidth', 2, 'Color', calm_green);
ylabel('K'); xlabel('K'); title('K policy with A_t and N_{t-1} at
   steady state');
hold on;
plot(kgrid, kinitpol_ss, 'LineWidth', 2, 'Color', calm_blue);

legend('Non-linear policy', 'Linear Policy');

saveas(gcf, '/Users/gabesekeres/Dropbox/Notes/Cornell_Notes/Fall_2024/
   Macro/Matlab/pset8_policy_functions_steady_state.png');


% Simulate over 5000 periods using projection solutions:
% Initialize the Markov chain for the TFP shocks
% Get markov transition matrix
na = 7;
[~, theta, ~] = AR1_rouwen(na,rho,0,siga);
mc = dtmc(theta);
x = simulate(mc, 5000);

% Initialize capital and labor at their steady state values
ks = kbar;
ns = nbar;

% Reshape policy functions
npol = reshape(npol, na, nk, nn);
kpol = reshape(kpol, na, nk, nn);

% Initialize a matrix to store simulation results
vect = zeros(5000, 4);

% Simulate the economy over 5000 periods
for u = 1:5000
    % Find the indices for the current state
    a_idx = x(u);
    k_idx = find(kgrid == ks, 1);
    n_idx = find(ngrid == ns, 1);

    % Ensure indices are within bounds
    if isempty(k_idx)
        [~, k_idx] = min(abs(kgrid - ks));
    end
    if isempty(n_idx)
        [~, n_idx] = min(abs(ngrid - ns));
    end

    % Get the policy functions for the current state
    nc = npol(a_idx, k_idx, n_idx);
    kc = kpol(a_idx, k_idx, n_idx);

    % Calculate economic variables
```

```
    y = ks^alpha * nc^(1-alpha);
    i = kc - (1-deltak) * ks;
    v = ((nc - (1-deltan) * ns) / chi)^(1/eps);
    c = y - i - phin * v;

    % Store the results
    vect(u, :) = [y, c, i, nc];

    % Update state variables for the next period
    ns = nc;
    ks = kc;
end

% Take the logarithm of the results
lvect = log(vect);

% Standard deviations projection model

disp("Standard deviations projection model:");
disp(['Y: ', num2str(std(lvect(:,1)))]);
disp(['C: ', num2str(std(lvect(:,2)))]);
disp(['I: ', num2str(std(lvect(:,3)))]);
disp(['N: ', num2str(std(lvect(:,4)))]);









toc;
```

(b) pset8_residual.m

```
function out = pset8_residual(kpol,npol,state_grid,allgrid,aprime,pw,
   param)

    ns = size(state_grid,2);

    bet = param.bet;
    sigma = param.sig ;
    alph = param.alpha ;
    deltak = param.deltak;
    deltan = param.deltan;
    phin = param.phin;
    chi = param.chi;
    eps = param.eps;

    %Time-t states
    A   = state_grid(1,:);
    K   = state_grid(2,:);
    N   = state_grid(3,:);

    % Policy guesses to compute H and future state (need to repeat
       future states for each possible shock between now and then)
    A_p = aprime;
    K_p = kpol;
    K_p = repmat_row(K_p,3);
    N_p = npol;
    N_p = repmat_row(N_p,3);

    %Subsitute out some things to get time T vars
    GDP = A.*K.^alph.*N_p(1,:).^(1-alph);
    I   = K_p(1,:) - (1-deltak)*K;
    V   = ((N_p(1,:) - (1-deltan)*N)/chi).^(1/eps);
    C   = GDP-I-phin*V;

    %Time t+1 policy

    state_grid_p = [repmat(A_p(:),7*21,1)';K_p(:)';N_p(:)'];

    N_pp  = ndim_simplex_eval(allgrid,state_grid_p,npol(:));
    K_pp = ndim_simplex_eval(allgrid,state_grid_p,kpol(:));

    K_pp = reshape(K_pp,[3,ns]); %Rows are different TFP shocks
    N_pp  = reshape(N_pp,[3,ns]); %Rows are different TFP shocks

    A_p = repmat(A_p(:)',7*21,1)';
    A_p = reshape(A_p,[3,ns]);

    %To get C_p, we need to follow the same steps at time t+1
    GDP_p  = A_p.*K_p.^alph.*N_pp.^(1-alph);
    I_p    = K_pp - (1-deltak)*K_p;
    V_p   = ((N_pp - (1-deltan)*N_p)/chi).^(1/eps);
```

```
    C_p    = GDP_p-I_p-phin*V_p;

    %Main equations
    out = zeros(2,size(state_grid,2));
    out(1,:) = C.^(-sigma)-bet.*pw'*(C_p.^(-sigma).*(A_p.*alph.*(K_p./
       N_pp).^(alph-1)+(1-deltak)));
    out(2,:) = C.^(-sigma).*(phin./(eps.*chi.*V.^(eps-1)) - A.*(1-alph
       ).*(K./N_p(1,:)).^(alph))-bet*pw'*(C_p.^(-sigma).*((1-deltan).*
       phin)./(eps.*chi.*V_p.^(eps-1)));



end
```