Intro to Numerical Methods — Homework 3 Guide

Class III: Nominal Rigidities, Technology Shocks, and Projection

Alessandro T. Villa

Fall 2025

Learning Goals

In this assignment you will:

1. Formulate a dynamic model with sticky prices (Rotemberg adjustment costs).
2. Derive and use a system of FOCs including the NKPC residual in levels.
3. Implement *projection time iteration*: pointwise policy solves + coefficient updates.
4. Simulate the economy under a Markov TFP process and produce IRFs.

1 Economic Environment

Representative household with CRRA utility over consumption and disutility of labor:

$$u(C, N) = \frac{C^{1-\gamma} - 1}{1-\gamma} - \chi \frac{N^{1+\eta}}{1+\eta}.$$

Firm technology (Cobb–Douglas) and factors:

$$Y_t = Z_t K_t^{\alpha} N_t^{1-\alpha}, \quad W_t = \Gamma_t Z_t K_t^{\alpha} (1-\alpha) N_t^{-\alpha}.$$

Capital evolves with investment I_t and depreciation δ:

$$K_{t+1} = (1-\delta)K_t + I_t, \qquad C_t = Y_t - I_t - \mathcal{R}(\Pi_t, Y_t).$$

Rotemberg price adjustment (in levels):

$$\mathcal{R}(\Pi_t, Y_t) = \frac{\varphi}{2}(\Pi_t - \bar{\Pi})^2 Y_t, \qquad \mathcal{R}_{\Pi}(\Pi_t, Y_t) = \varphi(\Pi_t - \bar{\Pi}) Y_t.$$

Policy interest (one-period bond price) via Taylor rule in levels:

$$q_t \equiv \frac{\beta}{\bar{\Pi}} \left(\frac{\Pi_t}{\bar{\Pi}}\right)^{-\phi_\pi} \left(\frac{Y_t}{\bar{Y}}\right)^{-\phi_y}.$$

TFP follows a two-state Markov chain $Z \in \{Z_L, Z_H\}$ with transition P_Z.

Parameters (use these):

$$\beta = 0.96,\ \delta = 0.1,\ \alpha = \tfrac{1}{3},\ \gamma = 2,\ \chi = 11.6,\ \eta = 2,\ \bar{\Pi} = 1.02,\ \varphi = 10,\ \theta = 7,\ \phi_\pi = 1.5,\ \phi_y = 0.3.$$

Price markup inverse at steady state: $\Gamma^{\mathrm{TSS}} = (\theta - 1)/\theta$.

2 Equilibrium Conditions (Residual Form)

Let $C, Y, W, \Pi, \Gamma, K', N$ denote current variables at (K, Z). Define expectations over Z' using P_Z and projected policies for (K', Z').

(i) **Capital Euler (physical investment):**

$$1 = \mathbb{E}\Big[\beta \frac{u_C(C')}{u_C(C)}(1 + \Gamma' \,\mathrm{MPK}' - \delta)\Big].$$

(ii) **Labor intratemporal condition:**

$$u_C(C)\, W = v_N(N).$$

(iii) **NKPC in levels (Rotemberg)** (only if $\varphi > 0$):

$$(1 - \theta + \theta\Gamma)Y - \mathcal{R}_\Pi(\Pi, Y)\,\Pi + \mathbb{E}\left[\beta \frac{u_C(C')}{u_C(C)}\,\mathcal{R}_\Pi(\Pi', Y')\,\Pi'\right] = 0.$$

(iv) **Bond pricing vs policy (consistency):**

$$q(\Pi, Y) - \mathbb{E}\Big[\beta \frac{u_C(C')}{u_C(C)}\,\frac{1}{\Pi'}\Big] = 0.$$

These yield a system of 4 residuals.

3 Steady State (for centering)

With $\Gamma = \Gamma^{\mathrm{TSS}}$ and $\Pi = \bar{\Pi}$, solve for N^{SS} from static conditions, then compute $K^{\mathrm{SS}}, W^{\mathrm{SS}}, Y^{\mathrm{SS}}, C^{\mathrm{SS}}$.[1]

4 Discretization and Basis (Projection)

- **State grid:** capital grid $K \in [K^{\mathrm{SS}}(1-\texttt{explore}),\ K^{\mathrm{SS}}(1+\texttt{explore})]$ with `g.numK=11`.
- **TFP states:** $Z\_L = 0.99$, $Z\_H = 1.01$, transition $P_Z = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$.
- **Policies to approximate:**

$$\{\, K'(K,Z),\ \Gamma(K,Z),\ \Pi(K,Z),\ N(K,Z)\,\}.$$

- **Basis:** quadratic polynomial in K per Z (linear regression for coefficients):

$$\hat{g}(K;\alpha) = \alpha_0 + \alpha_1 K + \alpha_2 K^2.$$

[1] If needed, you may solve N^{SS} by a scalar root finder using the SS FOCs; then back out K^{SS} from the capital Euler in SS.

5 Algorithm — Projection Time Iteration

We iterate on policy surfaces rather than on the value function.

High-level loop

1. Initialize policy guesses near SS: $K' = K^{\text{SS}}$, $\Pi = \bar{\Pi}$, $\Gamma = \Gamma^{\text{TSS}}$, $N = N^{\text{SS}}$ for all grid points and both Z states.
2. **Pointwise solves:** for each grid point (K_i, Z_j), solve the FOCs *at that point* for the unknown controls (sticky: $[K', \Gamma, \Pi, N]$; flex: $[K', N]$), *taking expectations through the current projected policies.*
3. **Update policy arrays** with dampening.
4. **Refit projection coefficients** (per Z and per policy) by OLS on the basis.
5. Check convergence via relative sup-norm of policies; repeat.

Pointwise nonlinear system (pseudocode)

```
% Unknowns at (K,Z): x = [Knext, Gammatilde, PI, N]   % (sticky-price case)
obj = @(x) FOCs_dyn( K, x(1), x(2), x(3), x(4), Z, Z_index, p, f, Proj );
x0  = [Kp_guess, Gamma_guess, PI_guess, N_guess];      % from previous iter
x   = fsolve(obj, x0, options);                        % solve residuals=0
```

Listing 1: Pointwise FOCs at (K,Z)

Expectations via projection

Inside `FOCs_dyn`, when constructing expectations at (K', Z'), evaluate next-period controls from the current projection:

$$K'' = \widehat{K'}(K'; Z'), \quad \Gamma' = \widehat{\Gamma}(K'; Z'), \quad \Pi' = \widehat{\Pi}(K'; Z'), \quad N' = \widehat{N}(K'; Z').$$

Use these to compute Y', C' and the expected terms in the Euler/NKPC/bond residuals.

Dampening and error metric

Update policies with a damping factor $\lambda \in (0, 1)$:

$$g^{\text{new}} \leftarrow \lambda\, g^{*} + (1 - \lambda)\, g^{\text{old}}.$$

Track the maximum relative change across all policy arrays as the convergence criterion.

6 Implementation Hints

- **Numerical stability:** cast unknowns to `real` before using them; guard divisions by small numbers; keep $N \in (0, 1)$.
- **FSOLVE:** Levenberg–Marquardt with tight tolerances typically works well here. Use previous iteration's solution as `x0`. Read the MATLAB documentation for a brief overview of the methods that `fsolve` uses to solve a system of N nonlinear equations.

- **Projection refit:** per Z, run OLS of each policy array on the basis matrix $\mathbf{X} = [1, K, K^2]$ to refresh coefficients.
- **Taylor-rule consistency:** remember the bond pricing residual equates the *policy* price $q(\Pi, Y)$ to the *asset-pricing* EMR.

7 Suggested File Structure

- `HW3_main.m` (driver: parameters, grids, initial policies, call solver, simulate, plot)
- `Functions/FOCs_dyn.m` (build residual vector given (K, Z) and guesses of the future policy functions)
- `Functions/ProjectionTimeIteration.m` (outer iteration loop)
- `Functions/Project.m` (refit projection coefficients by OLS)
- `Utils/goldenx.m` (if you reuse it) and any small helpers

8 Simulation and IRFs

After convergence:

a) Generate a length-T Markov chain for Z_t with the given P_Z.

b) Initialize $K_1 = K^{\text{SS}}$ and iterate forward using the projected policies:

$$\{N_t, K_{t+1}, \Gamma_t, \Pi_t\} = \{\widehat{N}, \widehat{K'}, \widehat{\Gamma}, \widehat{\Pi}\}(K_t; Z_t).$$

c) Recover W_t, Y_t, C_t from definitions (including Rotemberg cost in C_t).

d) Select a shock window (e.g., $t_0 : t_{\text{end}}$) and plot IRFs for $Z, Y, C, K, W, N, \Gamma, \Pi$ against their SS lines.

Pseudo-MATLAB Snippets (for reference)

Driver. (`main.m`)

```
% Params, functions f.*, steady state (NSS,KSS, etc.)
% Grid and basis: g.k_grd, X = [1, K, K.^2]

% Initial policy guesses near SS for both Z states
kprime0 = KSS*ones(g.numK, p.nz);
PI0     = p.Pibar*ones(g.numK, p.nz);
Gamma0  = p.GammaTSS*ones(g.numK, p.nz);
N0      = p.NSS*ones(g.numK, p.nz);

% Projection container
Proj.numPolicy = 4;             % [K', Gamma~, PI, N]
Proj.Poly = @(K,alphas,z,pol) alphas(1,z,pol) + alphas(2,z,pol).*K + alphas(3,
    z,pol).*K.^2;

% Iterate policies by projection
[err, kpol, PIpol, Gampol, Npol, Proj] = ...
ProjectionTimeIteration(p,g,f,kprime0,Gamma0,PI0,N0,Proj);
```

Projection refit per Z and policy (`Project.m`).

```
    for z = 1:p.nz
    Proj.alphas(:,z,1) = regress(kpol(:,z), g.X);    % K'
    Proj.alphas(:,z,2) = regress(Gampol(:,z), g.X); % Gamma~
    Proj.alphas(:,z,3) = regress(PIpol(:,z), g.X);   % PI
    Proj.alphas(:,z,4) = regress(Npol(:,z), g.X);    % N
    end
```

Residuals builder (`FOCs_dyn.m`).

```
    % Within-period objects
    W = Gammatilde * Z * K^p.alpha * (1-p.alpha) * N^(-p.alpha);
    Y = f.PI(Z, 1, K, N);
    Inv = Knext - (1 - p.delta) * K;
    C = Y - Inv - f.Phi(PI, Y);

    % Expectations via projected next-period policies at (Knext, Z')
    ENKPC = 0; EMR = 0; EMPK = 0;
    for zp = 1:p.nz
    % Pull K'', Gamma', PI', N' from projection:
    Knn = Proj.Poly(Knext, Proj.alphas, zp, 1);
    Gamn = Proj.Poly(Knext, Proj.alphas, zp, 2);
    PIp  = Proj.Poly(Knext, Proj.alphas, zp, 3);
    Np   = Proj.Poly(Knext, Proj.alphas, zp, 4);

    Yp = f.PI(p.z(zp),1,Knext,Np);
    Invp = Knn - (1 - p.delta)*Knext;
    Cp = Yp - Invp - f.Phi(PIp, Yp);

    w = p.Pz(Z_index, zp);                          % transition prob
    MR = p.beta * f.du(Cp,p.gamma)/f.du(C,p.gamma) / PIp;
    EMR  = EMR  + w * MR;
    ENKPC= ENKPC+ w * ( p.beta * f.du(Cp,p.gamma)/f.du(C,p.gamma) * f.Phiprime(PIp
        ,Yp) * PIp );
    EMPK = EMPK + w * ( p.beta * f.du(Cp,p.gamma)/f.du(C,p.gamma) * (1 + Gamn*f.
        PIk(p.z(zp),1,Knext,Np) - p.delta) );
    end

    % Residuals (sticky case)
    err(1) = 1 - EMPK;
    err(2) = f.du(C,p.gamma)*W - f.dv(N,p.chi,p.eta);
    err(3) = (1 - p.theta + p.theta*Gammatilde)*Y - f.Phiprime(PI,Y)*PI + ENKPC;
    Q = (p.beta/p.Pibar) * (PI/p.Pibar)^(-p.phipi) * (Y/p.YSS)^(-p.phiy);
    err(4) = Q - EMR;
```

Projection Time Iteration (`ProjectionTimeIteration.m`).

```
  function [err, kprime, PI, Gamma, N, Proj] = ProjectionTimeIteration(p,g,f,
      kprime,Gamma,PI,N,Proj)
  % Iterates on projected policy surfaces until convergence.
  damp = 0.5;
  tol  = 1e-5;
  options = optimoptions(@fsolve,'Algorithm','Levenberg-Marquardt','Display','off'
      );

  err = 1;
  while err > tol
  for iK = 1:g.numK
```

```
  for iZ = 1:p.nz
  K = g.k_grd(iK);  Z = p.z(iZ);
  x0 = [kprime(iK,iZ), Gamma(iK,iZ), PI(iK,iZ), N(iK,iZ)];
  obj = @(x) FOCs_dyn(K, x(1), x(2), x(3), x(4), Z, iZ, p, f, Proj);
  x = fsolve(obj, x0, options);
  k_temp(iK,iZ)=x(1); G_temp(iK,iZ)=x(2); PI_temp(iK,iZ)=x(3); N_temp(iK,iZ)=x(4);
  end
  end

  % Dampened update and convergence check
  kprime = damp*k_temp + (1-damp)*kprime;
  Gamma  = damp*G_temp + (1-damp)*Gamma;
  PI     = damp*PI_temp + (1-damp)*PI;
  N      = damp*N_temp + (1-damp)*N;
  err = max([max(abs(k_temp(:)-kprime(:))./kprime(:)), ...
  max(abs(PI_temp(:)-PI(:))./PI(:))]);
  end
  end
```