An Introduction to Nonlinear Solution and Estimation Techniques

Alexander W. Richter
Federal Reserve Bank of Dallas

Nathaniel A. Throckmorton
College of William & Mary

TOOLBOX FUNCTIONS

- `script.m`: assigns options to `O` and runs the algorithm
- `parameters.m`: assigns model parameters to `P`
- `steadystate.m`: assigns steady state values to `S(P)`
- `variables.m`: outputs a structure, `V`, containing indices of variables, forecast errors, and shocks and variable titles
- `grids.m`: assigns the discretized state space to `G(O, P)`
- `guess.m`: assigns the initial conjectures to `pf(O,P,S,G)`
- `linmodel.m`: outputs the linear transition matrix, `T`, the impact matrix, `M`, and a 2-element vector of flags, `eu`, indicating existence and uniqueness of the linear solution
- `eqm.m`: outputs a vector, `R`, containing the residuals to a subsystem of expectational equations that are constrained by all of the other equations in the equilibrium system

EXAMPLE: REAL BUSINESS CYCLE MODEL

A social planner chooses $\{c_t, k_{t+1}\}_{t=0}^{\infty}$ to maximize:

$$E_0 \sum_{t=0}^{\infty} \beta^t \frac{c_t^{1-\sigma}}{1-\sigma}$$

subject to

$$c_t + k_{t+1} = z_t k_t^{\alpha} + (1-\delta)k_t$$
$$z_t = (1-\rho)\bar{z} + \rho z_{t-1} + \varepsilon_t$$

Optimality condition:

$$1 = \beta E_t[\underbrace{(c_t/c_{t+1})^{\sigma}(\alpha z_{t+1} k_{t+1}^{\alpha-1} + 1 - \delta)}_{\equiv \Phi(z_{t+1})}]$$

Discretized State Space

- State variables: k_t, z_t
- Number of grid points: N_k, N_z
- Grid boundaries: $[k_{\min}, k_{\max}]$ and $[z_{\min}, z_{\max}]$
- Create evenly spaced grids:

$$x_{grid} = \texttt{linspace}(x_{\min}, x_{\max}, N_x), \quad x \in \{k, z\}$$

- State space contains $N = N_k \times N_z$ independent nodes
- Create an array for each state variable, where every position is a unique permutation of the state space:

$$[k_{gr}, z_{gr}] = \texttt{ndgrid}(k_{grid}, z_{grid})$$

Functional Approximation

- True RE solution only exists in special cases (e.g., $\delta = 1$)
- Goal: Find an approximating function that maps the state space to the optimal decision rule for consumption:

$$\underbrace{c(k,z)}_{\text{True RE Solution}} \approx \underbrace{\mathcal{P}_c(k,z)}_{\text{Approximating Function}}$$

- Basic elements of the algorithm:
  1. Interpolation: Linear, Least squares
  2. Integration: Gauss-Hermite, Trapezoid, Rouwenhorst
  3. Iteration: Time, Fixed-point

Initial Conjecture

Use the linear solution as a guess for $\mathcal{P}_c(k, z)$:

- Linear solution from `gensys.m` takes the form:

$$\hat{Y}' = T\hat{Y} + M\varepsilon$$

  where $\hat{Y} = [\hat{k}, \hat{z}, \hat{c}]^T$, $\hat{x} \equiv (x_t - \bar{x})/\bar{x}$, and $\varepsilon \sim N(0, \sigma^2)$.
- Convert the state space to deviations from steady state
- Compute an initial conjecture for all nodes ($i = 1, \ldots, N$):

$$\hat{\mathcal{P}}_c = \underbrace{T(c_{idx}, [k_{idx}, z_{idx}])}_{1\times 2} \underbrace{[\text{vec}(\hat{k}_{gr}), \text{vec}(\hat{z}_{gr})]^T}_{2\times N}$$

- Convert $\hat{\mathcal{P}}_c$ to levels ($\mathcal{P}_c = \bar{c}(1 + \hat{\mathcal{P}}_c)$) and assign to `pf.c`

Local Approximation

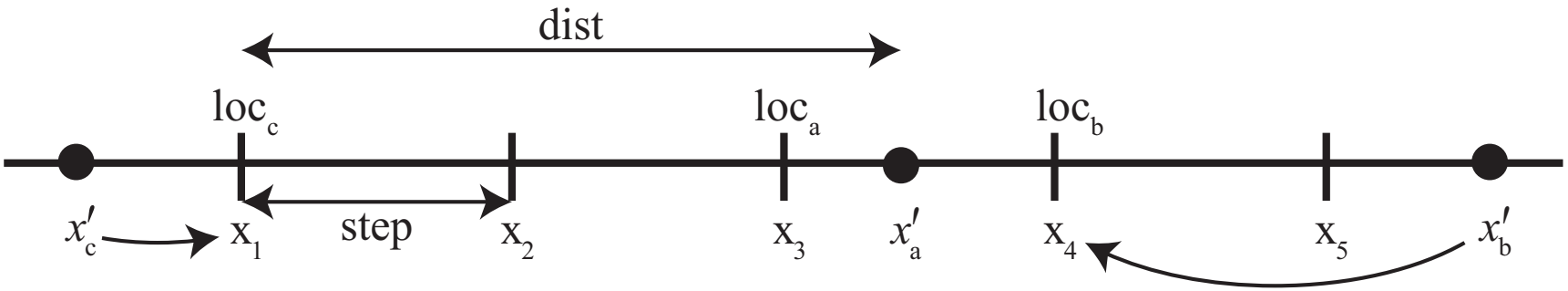
- Piecewise Linear Interpolation: 2 state variables (k, z)
- Goal: Find the policy function value $\mathcal{P}_c(k', z')$
- We have policy function values on nearest nodes

$$[\mathcal{P}_c(k_i, z_j), \mathcal{P}_c(k_i, z_{j+1}), \mathcal{P}_c(k_{i+1}, z_j), \mathcal{P}_c(k_{i+1}, z_{j+1})]$$

  once we determine the grid indices, i, j
- Locate the grid point to left of x', $x \in \{k, z\}$

$$\text{step} = x_2 - x_1, \qquad \text{dist} = x' - x_1$$

$$\text{loc} = \min(N_x - 1, \max(1, \text{floor}(\text{dist}/\text{step}) + 1))$$



Local Approximation

- Interpolate in the k direction:

$$\begin{aligned}\mathcal{P}_c(k', z_j) &= \mathcal{P}_c(k_i, z_j) + (k' - k_i)\frac{\mathcal{P}_c(k_{i+1}, z_j) - \mathcal{P}_c(k_i, z_j)}{k_{i+1} - k_i} \\ &= \underbrace{\frac{k_{i+1} - k'}{k_{i+1} - k_i}}_{\omega_{k_i}} \mathcal{P}_c(k_i, z_j) + \underbrace{\frac{k' - k_i}{k_{i+1} - k_i}}_{\omega_{k_{i+1}}} \mathcal{P}_c(k_{i+1}, z_j)\end{aligned}$$

- Then interpolate in the z direction:

$$\mathcal{P}_c(k', z') = \underbrace{\frac{z_{j+1} - z'}{z_{j+1} - z_j}}_{\omega_{z_j}} \mathcal{P}_c(k', z_j) + \underbrace{\frac{z' - z_j}{z_{j+1} - z_j}}_{\omega_{z_{j+1}}} \mathcal{P}_c(k', z_{j+1})$$

- Combine these two equations:

$$\mathcal{P}_c(k', z') = \sum_{a=0}^{1} \sum_{b=0}^{1} \omega_{k_{i+a}} \omega_{z_{j+b}} \mathcal{P}_c(k_{i+a}, z_{j+b})$$

Local Approximation

- Use a nested loop or write out all of the terms in the sum to calculate the interpolated value of the policy function:

```
nestedsum = 0;  %initialize
for a = 0:1     %loop for k
  for b = 0:1   %loop for z
    nestedsum = nestedsum + ...
      wk(1+a)*wz(1+b)*pf.c(kloc+a,zloc+b);
  end
end
```

- Must calculate the interpolated value for each realization of the stochastic variable(s), each of which requires calculating a different set of locations and weights
- Number of loops equals the number of exogenous states

Global Approximation

- A general class of polynomials can be written as:

$$\mathcal{P}(x;\eta) = \sum_{i=0}^{n} \eta_i \varphi_i(x).$$

- Linear interpolation is a special case of this general class (i.e., $n = 1$, $\varphi_i(x) = x^i$, and α is chosen appropriately)
- For $n > 1$, $\varphi_i(x) = x^i$ is a collection of monomials and

$$\mathcal{P}(x;\eta) = \eta_0 + \eta_1 x + \eta_2 x^2 + \cdots + \eta_p x^p$$

- This set of monomials may lead to multicollinearity (i.e., near linear dependence among the monomials)
- Bases consisting of orthogonal polynomials fix this problem (e.g., Chebyshev and Hermite Polynomials)

EXAMPLE: MONOMIALS

- Consider the complete set of basis functions of order 2:

$$\mathcal{P}(k,z) = \eta_0 + \eta_k k + \eta_z z + \eta_{kk} k^2 + \eta_{kz} kz + \eta_{zz} z^2$$

- Regressor matrix (subscripts denote grid indices):

$$X = \begin{bmatrix} 1 & k_1 & z_1 & k_1^2 & k_1 z_1 & z_1^2 \\ 1 & k_2 & z_2 & k_2^2 & k_2 z_2 & z_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & k_N & z_N & k_N^2 & k_N z_N & z_N^2 \end{bmatrix}$$

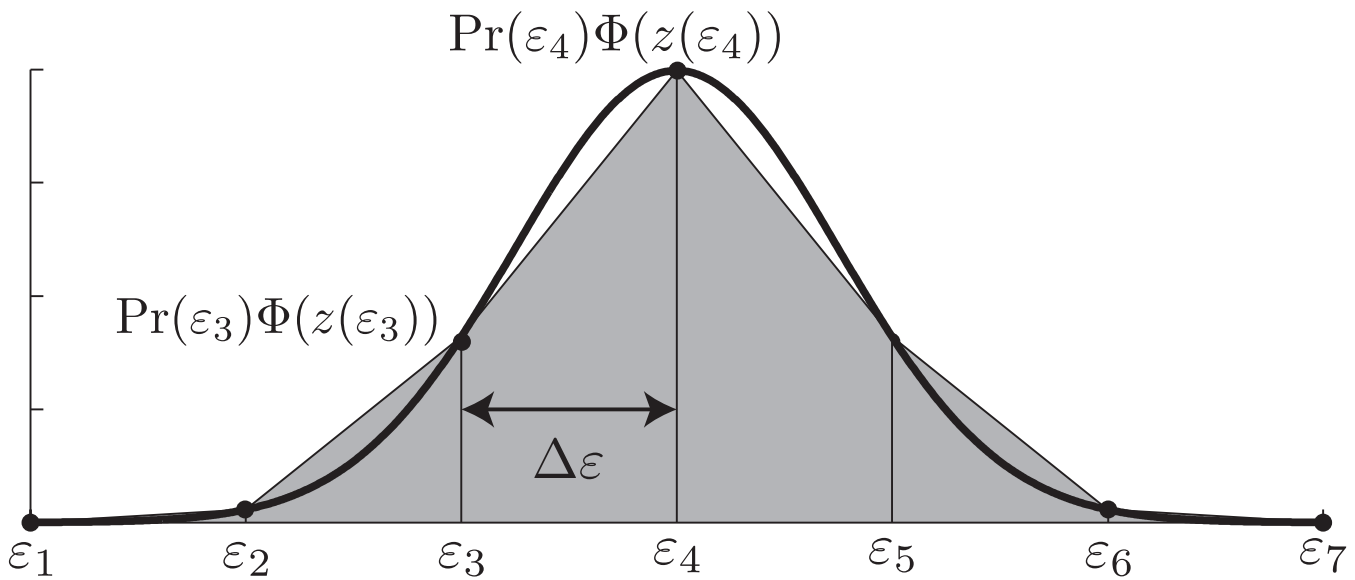
- Obtain coefficients using OLS:

$$\hat{\eta} = (X^T X)^{-1} X^T \operatorname{vec}(\mathcal{P}_c(k,z))$$

$$\mathcal{P}_c(k',z') = X'\hat{\eta}$$

Integration: Trapezoid Rule

$$
\begin{aligned}
E[\Phi(z)] \approx & \frac{\Pr(\varepsilon_1)\Phi(z(\varepsilon_1)) + \Pr(\varepsilon_2)\Phi(z(\varepsilon_2))}{2}\Delta\varepsilon \\
& + \frac{\Pr(\varepsilon_2)\Phi(z(\varepsilon_2)) + \Pr(\varepsilon_3)\Phi(z(\varepsilon_3))}{2}\Delta\varepsilon + \cdots \\
& + \frac{\Pr(\varepsilon_{m-1})\Phi(z(\varepsilon_{m-1})) + \Pr(\varepsilon_m)\Phi(z(\varepsilon_m))}{2}\Delta\varepsilon \\
& = \frac{\Delta\varepsilon}{2}\left[2\sum_{i=1}^{m}\Pr(\varepsilon_i)\Phi(z(\varepsilon_i)) - \Pr(\varepsilon_1)\Phi(z(\varepsilon_1)) - \Pr(\varepsilon_m)\Phi(z(\varepsilon_m))\right]
\end{aligned}
$$



Integration: Gauss-Hermite

- Given a shock, $\varepsilon \sim N(\mu, \sigma^2)$,

$$E[\Phi(z(\varepsilon))] = (2\pi\sigma^2)^{-1/2} \int_{-\infty}^{\infty} \Phi(z(\varepsilon)) e^{-(\varepsilon-\mu)^2/(2\sigma^2)} d\varepsilon$$

- Apply change of variables, $\upsilon = (\varepsilon - \mu)/(\sqrt{2}\sigma)$,

$$\begin{aligned} E[\Phi(z(\upsilon))] &= \pi^{-1/2} \int_{-\infty}^{\infty} \Phi(z(\sqrt{2}\sigma\upsilon + \mu)) e^{-\upsilon^2} d\upsilon \\ &\approx \pi^{-1/2} \sum_{i=1}^{n} \omega_i \Phi(z(\sqrt{2}\sigma\upsilon_i + \mu)) \end{aligned}$$

- ω_i and υ_i are Gauss-Hermite weights and nodes:

$$\omega_i = 2^{n+1} n! \sqrt{\pi} [H_{n+1}(\upsilon_i)]^{-2}$$

H_{n+1} is the physicists' Hermite polynomial of order $n+1$.

Example: Time Iteration

On iteration q, solve for the $\mathcal{P}_c^q(k,z)$ that *satisfies* equilibrium

1. Use log-linear solution on each node to obtain $\mathcal{P}_c^0$
   - Local: $\mathcal{P}_c^1 = \mathcal{P}_c^0$
   - Global: $\hat{\eta}^0 = (X^T X)^{-1} X^T \operatorname{vec}(\mathcal{P}_c^0)$ so $\mathcal{P}_c^1 = X\hat{\eta}^0$
2. Solve for k' and z', given ε'
3. Find $\mathcal{P}_c^q(k',z')$ given the updated state
   - Local: use piecewise linear interpolation
   - Global: update the basis so $\mathcal{P}_c^q(k',z') = X'\hat{\eta}^{q-1}$
4. Evaluate expectations (Trapezoid rule or Gauss Hermite)

$$E[\Phi(z')] = \beta E[\mathcal{P}_c^q(k',z')^{-\sigma}(\alpha z' k'^{\alpha-1} + 1 - \delta)]$$

Example: Time Iteration

5. Use nonlinear solver to find a $\mathcal{P}_c^q(k,z)$ that satisfies the consumption Euler equation, $\mathcal{P}_c^q(k,z)^{-\sigma} = E[\Phi(z')]$.
6. Update policy function
   - Local: $\mathcal{P}_c^{q+1} = \mathcal{P}_c^q$
   - Global: $\hat{\eta}^q = (X^T X)^{-1} X^T \operatorname{vec}(\mathcal{P}_c^q(k,z))$, $\mathcal{P}_c^{q+1}(k,z) = X\hat{\eta}^q$
7. Calculate distance between updates
   - Local: $\text{dist} = \mathcal{P}_c^q(k,z) - \mathcal{P}_c^{q-1}(k,z)$
   - Global: $\text{dist} = \hat{\eta}^q - \hat{\eta}^{q-1}$
8. If $|\text{dist}| < \text{tol}$, then stop. If not, then set $q = q + 1$ and repeat steps 2-7 using $\mathcal{P}_c^{q+1}$ as the new initial conjecture.

Advantage: Satisfies the equilibrium system on each node and nodes can be run in parallel.

Disadvantage: Nonlinear solver must execute on each node.

EXAMPLE: FIXED-POINT ITERATION

Solve for the $\mathcal{P}_c^q(k,z)$ *implied* by the equilibrium system

1. Obtain an initial conjecture
   (step 1 in the time iteration algorithm)
2. Calculate updated variables and expectations
   (steps 2-4 in the time iteration algorithm)
3. Calculate $\mathcal{P}_c^q(k,z) = (E[\Phi(z')])^{-1/\sigma}$ on each node
4. Execute steps 6-8 in the time iteration algorithm

Advantage: Does not require a loop, since all of the nodes are evaluated simultaneously.

Disadvantage: Algorithm is less stable because it does not solve for the optimal policy function on each node.

Note: To simultaneously evaluate all nodes, it is necessary to replicate across all realizations of the shocks.

Extension 1: Elastic Labor Supply

Social planner now chooses $\{c_t, n_t, k_{t+1}\}_{t=0}^{\infty}$ to maximize:

$$E_0 \sum_{t=0}^{\infty} \beta^t \left\{ \frac{c_t^{1-\sigma}}{1-\sigma} - \chi \frac{n_t^{1+\eta}}{1+\eta} \right\},$$

subject to

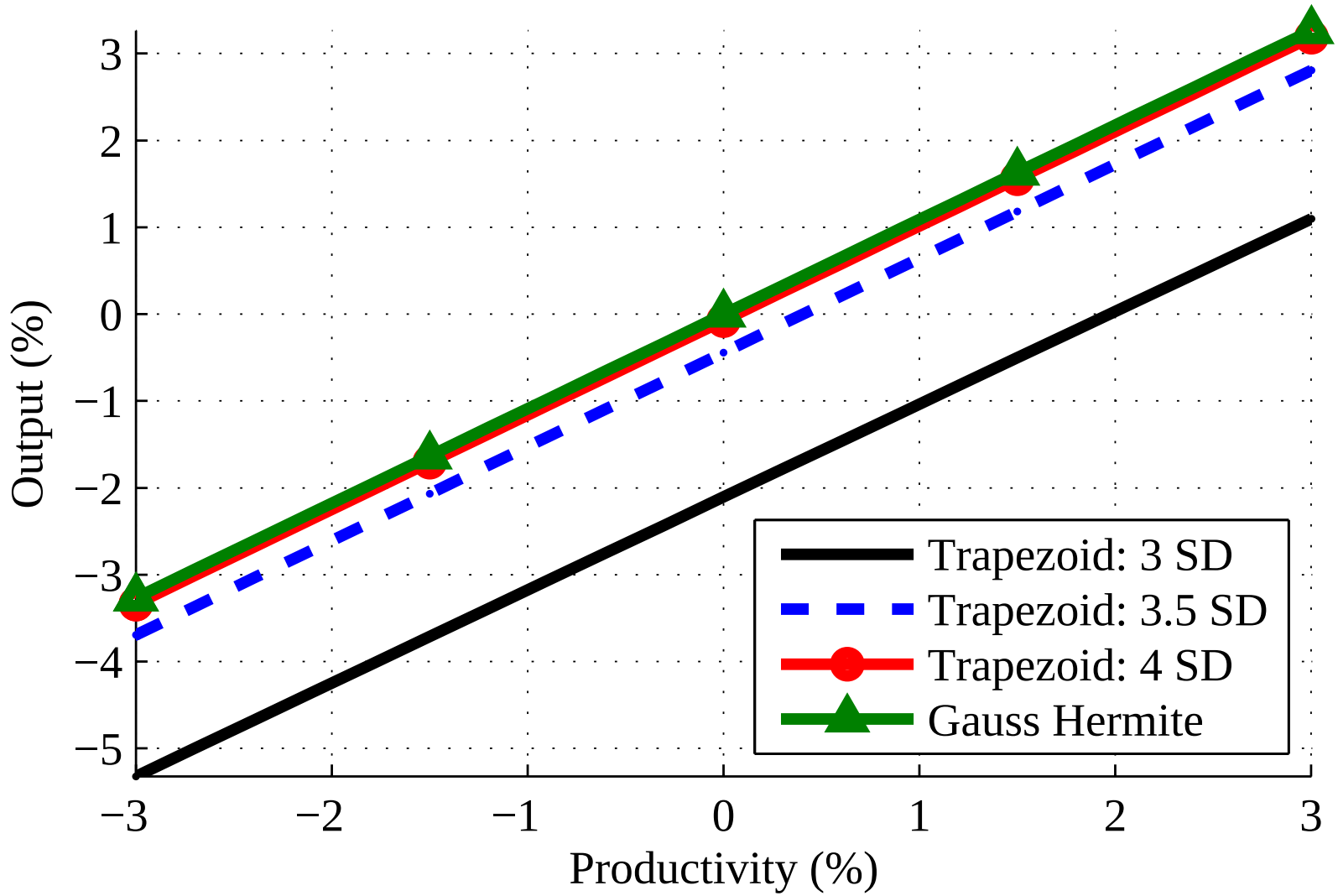
$$c_t + k_{t+1} = z_t k_t^{\alpha} n_t^{1-\alpha} + (1-\delta)k_t$$
$$z_t = (1-\rho)\bar{z} + \rho z_{t-1} + \varepsilon_t$$

Optimality conditions now include:

$$(1-\alpha) z_t k_t^{\alpha} n_t^{-\alpha} = \chi n_t^{\eta} c_t^{\sigma}$$

Static relation does not add a state or policy function, but it is easier to use labor instead of consumption as a policy function.

Policy Function Comparison



Extension 2: Adding Frictions

Introduce investment adjustment costs, variable capital utilization, and habit formation to the textbook RBC model

$$m_{t+1} = \beta((c_t - hc_{t-1})/(c_{t+1} - hc_t))^{\sigma}$$

$$w_t = \chi(c_t - hc_{t-1})^{\sigma} n_t^{\eta}$$

$$q_t = E_t[m_{t+1}(v_{t+1} r_{t+1}^k + q_{t+1}(1 - \delta_{t+1}))]$$

$$1 = q_t[1 - \nu(i_t/i_{t-1} - 1)^2/2 - \nu i_t/i_{t-1}(i_t/i_{t-1} - 1)] + \nu E_t[q_{t+1} m_{t+1}(i_{t+1}/i_t)^2(i_{t+1}/i_t - 1)]$$

$$r_t^k = q_t(\delta_1 + \delta_2(v_t - 1))$$

$$r_t^k = \alpha y_t/(v_t k_{t-1})$$

$$w_t = (1 - \alpha) y_t/n_t$$

$$c_t + i_t = y_t$$

$$k_t = (1 - \delta_t) k_{t-1} + i_t[1 - \nu(i_t/i_{t-1} - 1)^2/2]$$

$$\delta_t = \delta_0 + \delta_1(v_t - 1) + \delta_2(v_t - 1)^2/2$$

$$y_t = z_t(v_t k_{t-1})^{\alpha} n_t^{1-\alpha}$$

$$z_t = (1 - \rho)\bar{z} + \rho z_{t-1} + \varepsilon_t$$

State variables: $\{c, i, k, z\}$, Policy functions: $\{n, v\}$ (not unique)

Extension 3: Productivity Switching

Productivity now has a state-dependent intercept:

$$z_t = (1-\rho)\bar{z}(s_t) + \rho z_{t-1} + \varepsilon_t,$$

where $s_t \in \{1, 2\}$, $z(1) < z(2)$. The state evolves according to:

$$\begin{bmatrix} \Pr[s_{t+1}=1|s_t=1] & \Pr[s_{t+1}=2|s_t=1] \\ \Pr[s_{t+1}=1|s_t=2] & \Pr[s_{t+1}=2|s_t=2] \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix},$$

where $0 \leq p_{ij} \leq 1$ and $\sum_{j=1}^{2} p_{ij} = 1$ for all $i \in \{1, 2\}$

Key Changes:

- Policy functions are dependent on a discrete state variable
- Policy functions account for the expectational effects of the economy switching to the other productivity state

Solving the Model

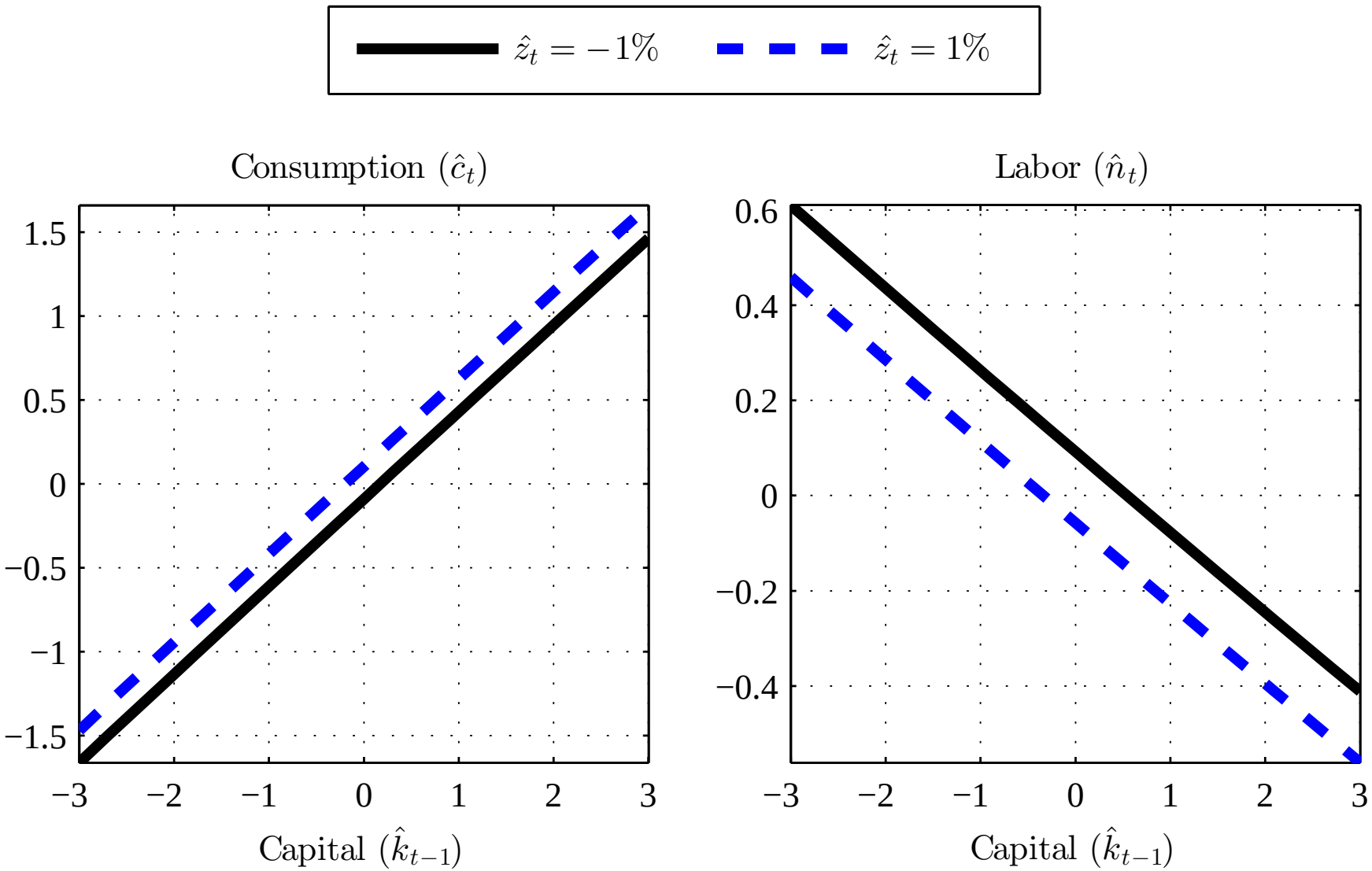
1. Calculate $z(s')$ for each realization of the state
2. Find the updated policy function, $n(s')$, for each state
3. Numerical Integration:
   - First integrate across the continuous random variable, z, conditional on the future realizations of the discrete stochastic variable, s', to obtain:

$$E[\Phi(z')|s'=1], \quad E[\Phi(z')|s'=2]$$

   - Then weight the conditional expectations by their corresponding likelihood. The conditional expectations are:

$$E\left[\Phi(z')|s=i\right] = p_{i1}E[\Phi(z')|s'=1] + p_{i2}E[\Phi(z')|s'=2]$$

State-Dependent Policy Functions



Extension 4: Policy Switching

- Regime dependent reaction coefficients:

$$\phi(s_t) = \begin{cases} \phi & \text{for } s_t = 1, \\ 0 & \text{for } s_t = 2, \end{cases} \qquad \gamma(s_t) = \begin{cases} \gamma & \text{for } s_t = 1, \\ 0 & \text{for } s_t = 2. \end{cases}$$

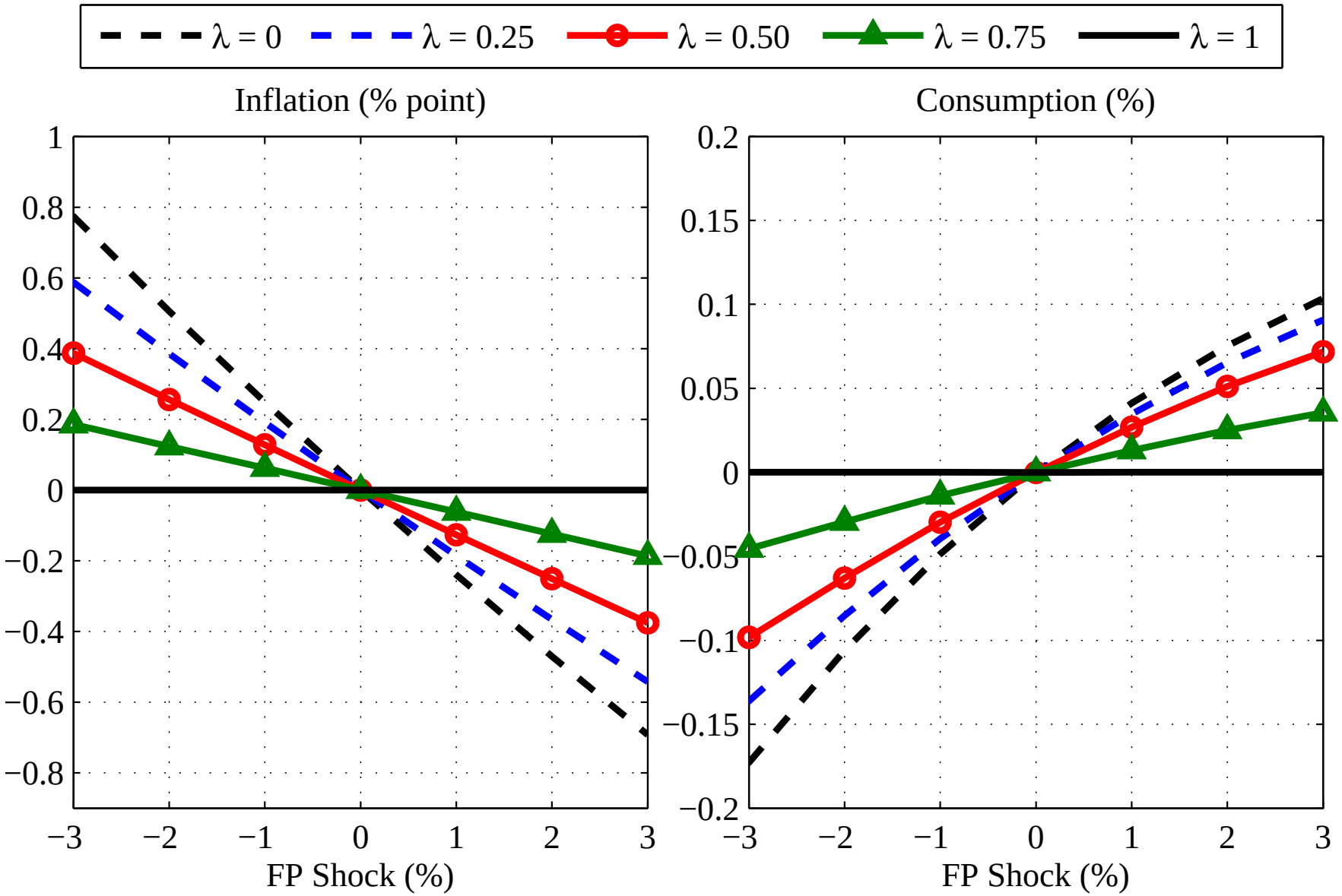
  Regime 1: Active Monetary/Passive Fiscal (AM/PF)
  Regime 2: Passive Monetary/Active Fiscal (PM/AF)

- Transition probabilities:

$$\begin{bmatrix} \Pr[s_t = 1 | s_{t-1} = 1] & \Pr[s_t = 2 | s_{t-1} = 1] \\ \Pr[s_t = 1 | s_{t-1} = 2] & \Pr[s_t = 2 | s_{t-1} = 2] \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}.$$

- Average duration of time spent in the AM/PF regime in the ergodic distribution: $\lambda = (1 - p_{22})/(2 - p_{11} - p_{22})$
- When $\lambda = 1$, there is no chance of moving to the PM/AF

Policy Functions (Lump-Sum Taxes)



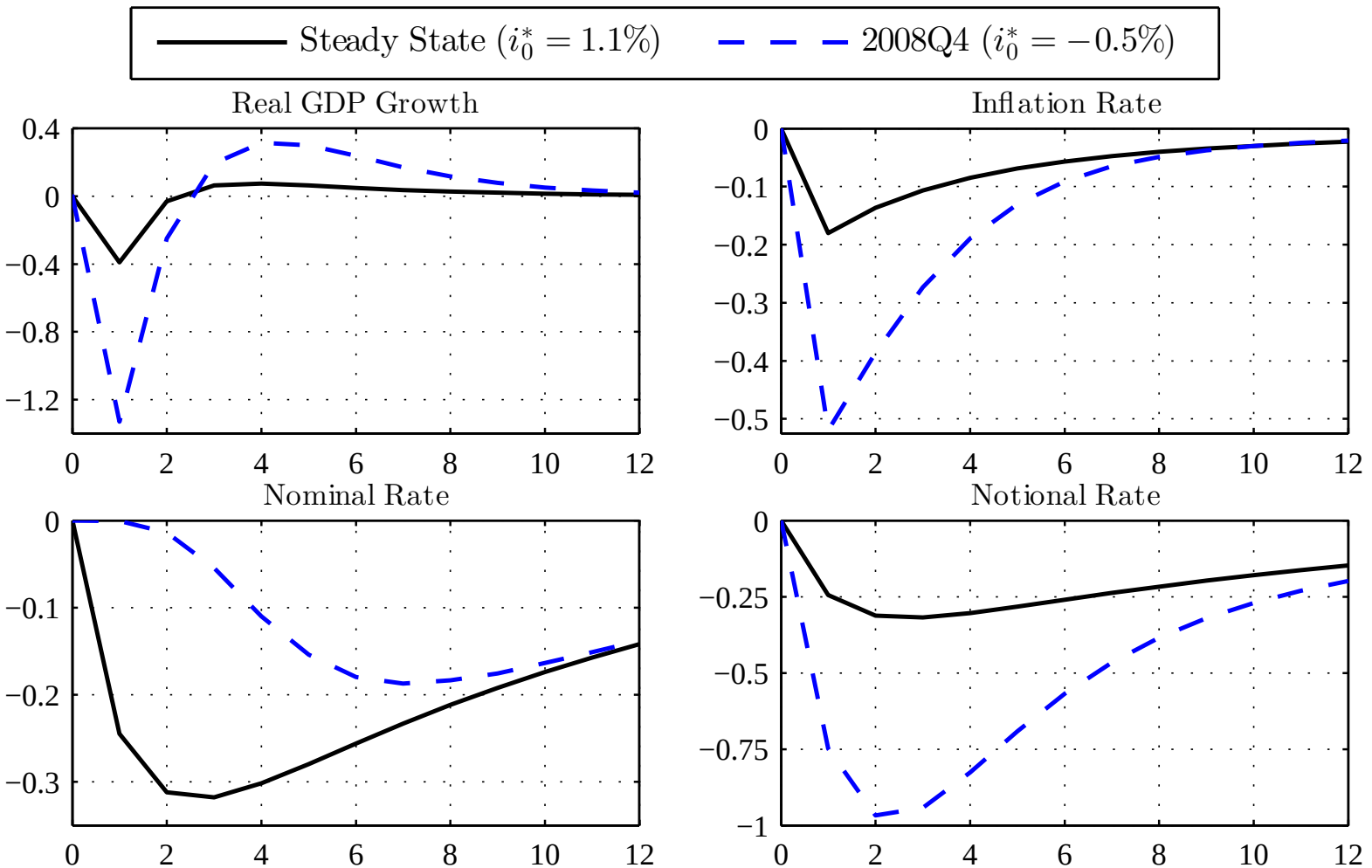
Simulating the Model

- Draw random shocks
  - Turn off shocks to compute the stochastic steady state
- Initialize state variables at the deterministic steady state
- Simulate the exogenous processes to update the state
- Interpolate current period values of the policy functions
- Use the remaining equations in the equilibrium system to simulate the other variables (follow the order in `eqm`)

Generalized Impulse Responses

- Used to study model dynamics away from steady state
- Shocks are consistent with household's expectations
- General Procedure (see Koop, Pesaran, Potter (1996)):
  1. Initialize each simulation at a certain state vector
  2. Calculate the mean of 10,000 simulations of the model conditional on a random shock in the first quarter
  3. Calculate a second mean from another set of 10,000 simulations by replacing the shock in the first quarter with the shock of interest
  4. Compute the percentage change in each period (or difference for rates) between the two means
  5. Repeat 1-4 at an alternative state vector to compare GIRFs

Example: NK Model with a ZLB (Discount Factor Shock)



ROUWENHORST METHOD

- Used to approximate an exogenous $AR(1)$ process
- Kopecky and Suen (2010) show the Rouwenhorst method outperforms other approximations of an $AR(1)$ process
- The approximation is a Markov switching process like the time-varying intercept example, but with n states
- The method determines the bounds of the exogenous state variables, the nodes, and the transition probabilities
- Let $z \sim AR(1)$ with persistence ρ, mean μ_z, and variance

$$\sigma_z^2 = \sigma_\varepsilon^2/(1-\rho^2).$$

Approximation of an $AR(1)$ Process

- The n states for the discretized process z are evenly spaced on $[\mu_z - \sigma_z\sqrt{n-1}, \mu_z + \sigma_z\sqrt{n-1}]$
- The transition matrix from s to s' is computed recursively:
  - For $n = 2$, let $q = (\rho + 1)/2$

$$P_2 = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} = \begin{bmatrix} q & 1-q \\ 1-q & q \end{bmatrix}.$$

  - For $n > 2$,

$$P_n = \underbrace{q \begin{bmatrix} P_{n-1} & \mathbf{0}_{n-1\times 1} \\ \mathbf{0}_{1\times n-1} & 0 \end{bmatrix}}_{\text{Prob of staying in } 1{:}n-1} + \underbrace{(1-q) \begin{bmatrix} \mathbf{0}_{n-1\times 1} & P_{n-1} \\ 0 & \mathbf{0}_{1\times n-1} \end{bmatrix}}_{\text{Prob. of going to } 2{:}n|1{:}n-1}$$

$$+ \underbrace{(1-q) \begin{bmatrix} \mathbf{0}_{1\times n-1} & 0 \\ P_{n-1} & \mathbf{0}_{n-1\times 1} \end{bmatrix}}_{\text{Prob. of going to } 1{:}n-1|2{:}n} + \underbrace{q \begin{bmatrix} 0 & \mathbf{0}_{1\times n-1} \\ \mathbf{0}_{n-1\times 1} & P_{n-1} \end{bmatrix}}_{\text{Prob of staying in } 2{:}n}.$$

Then divide rows 2 through $n-1$ by 2 so they sum to 1.

Integration of an n-state Process

- Conditional expectation of an n-state Markov process:

$$E\left[\Phi(z')|s=i\right] = \sum_{j=1}^{n} p_{ij} E[\Phi(z')|s'=j]$$

- Let $\beta \sim AR(1)$ in addition to z. Computing expectations across multiple exogenous processes generalizes to:

$$E\left[\Phi(z',\beta')|s_z=i_z, s_\beta=i_\beta\right] =$$

$$\sum_{j_\beta=1}^{n_\beta}\sum_{j_z=1}^{n_z} p_{i_\beta j_\beta} p_{i_z j_z} E[\Phi(z',\beta')|s'_z=j_z, s'_\beta=j_\beta],$$

where $i_\beta, j_\beta \in \{1,2,\ldots,n_\beta\}$ and $i_z, j_z \in \{1,2,\ldots,n_z\}$.

IMPLEMENTATION

- Script: Compute transition matrix (2 shocks):

```
e_weightVec = G.e_weight(G.z_gr(inode) == G.z_grid,:)';
u_weightVec = G.u_weight(G.beta_gr(inode) == G.beta_grid,:)';
e_weightMat = e_weightVec(:,ones(O.u_pts,1));
u_weightMat = permute(u_weightVec(:,ones(O.e_pts,1),[2,1]);
weightMat = e_weightMat.*u_weightMat
argzero = csolve('eqm',start,[],crit,itmax,state,...,weightMat);
```

- Eqm: Compute all combinations of shocks

```
EconsMat = weightMat.*(Contents of expectation);
```

and then integrate

```
Econs = sum(EconsMat(:));
```

Benefits of the Rouwenhorst Method

- Improves accuracy:
  - Matches 5 statistics of an AR1 process: the autocorrelation and the conditional and unconditional mean and SD
  - No interpolation or extrapolation of the policy function at future realizations of the exogenous state variables
- Reduces computation time:
  - Requires fewer nodes relative to Gauss-Hermite quadrature
  - Unnecessary to locate and obtain weights for the exogenous state variables if using linear interpolation
  - Reduces the dimension of the nested loop in the linear interpolation step by the number of exogenous states

EXAMPLE: 4 STATES (2 ENDOGENOUS)

- Gauss-Hermite:

```
do i2 = 1,ne2
 do i1 = 1,ne1
   o(i1,i2) = interp(inputs)
 end do
end do
...
do m4 = 0,1
 do m3 = 0,1
  do m2 = 0,1
   do m1 = 0,1
     wtemp = w1(m1+1)*w2(m2+1)*w3(m3+1)*w4(m4+1)
     sum = sum + wtemp*z1(loc1+m1,loc2+m2,loc3+m3,loc4+m4)
   end do
  end do
 end do
end do
```

- Rouwenhorst:

```
do m2 = 0, 1
 do m1 = 0, 1
   o = o + w1(m1+1)*w2(m2+1)*z1(:,:,loc1+m1,loc2+m2)
 end do
end do
```

Introduction to MEX

- Advantages of MATLAB:
  - Many built-in functions with good documentation
  - Easy to debug code
  - Easy to store data in structures
  - Parallel processing easy to implement
- Main drawback: Slow at evaluating loops
- MEX (MATLAB Executable) functions: Allow programmers to write sections of the code using a compiled language (e.g., Fortran) and call it as a function in MATLAB
- Good intermediate step toward full fortran implementation
- Challenge: Users must write a “Gateway” function that allows MATLAB to communicate with compiled code

FORTRAN 90 MEX REQUIREMENTS

- `https://www.mathworks.com/support/compilers`
- Intel Visual Fortran (IVF) Composer XE
  - Basic compiler: very few intrinsic functions
  - IMSL Library: Provides hundreds of additional functions
- Microsoft Visual Studio Professional
- MATLAB default: Fixed-format (f77) Fortran code.
- Our code: Free-format (f90), which is similar to MATLAB.
- To change the default settings, modify the batch files (...\bin\win64\mexopts) by deleting the '/fixed' flag
- Use `mex -setup` to select IVF as the compiler in MATLAB

Writing a Gateway Subroutine

```
1 #include "fintrf.h"
2 subroutine mexFunction(nlhs,plhs,nrhs,prhs)
  ! Declarations
3 implicit none
  ! mexFunc arguments
4 mwPointer plhs(*), prhs(*)
5 integer*4 nlhs, nrhs
```

- Line 1 defines pointer types in the MATLAB interface
- Function arguments:
  - prhs: Pointer to an array which holds the input data
  - plhs: Pointer to an array which will hold the output data
  - nrhs: number of right-hand (input) arguments
  - nlhs: number of left-hand (output) arguments
- Line 3 avoids Fortran's implicit type definitions

KEY MATLAB INTERFACE FUNCTIONS

- `mxGetPr`: Accesses the real data in an mxArray
- `mxGetScalar`: Grabs the value of the first real element of the mxArray (often one element)
- `mxGetM`/`mxGetN`: Determines the number of rows/columns in a specified mxArray
- `mxClassIDFromClassName`: Obtains an identifier for any MATLAB class (e.g., Double)
- `mxCreateNumericArray`: Creates an N-dimensional mxArray in which all data elements have the numeric data type specified by ClassID (7 dimensions max).

Inputting a Policy Function

- Declare variable types and sizes (lines 1-3). If the dimension lengths are variable, use allocatable memory:

```
1 mwpointer c_pr
2 mwSize nk,nz
3 real*8, allocatable, dimension(:,:) :: c
4 nk = mxGetN(prhs(1)) !Capital grid
5 nz = mxGetN(prhs(2)) !Technology grid
6 allocate(c(nk,nz))
7 c_pr = mxGetPr(prhs(3))
8 call mxCopyPtrToReal8(c_pr,c,nk*nz)
```

- Load the dimensions of `pf` from inputs (lines 4 and 5) and allocate the memory (line 6)
- Grab the address of the `pf` (input 3), store in `c_pr` (line 7), and copy to Fortran variable `c` (line 8)

CREATE OUTPUT MATRIX

- Load output size: stochastic realizations (lines 1-2)

```
1 e = mxGetM(prhs(4))
2 allocate(o(e))
  !Create array for return argument
3 cid = mxClassIDFromClassName('double')
4 plhs(1) = mxCreateNumericArray(1,e,cid,0)
5 o_pr = mxGetPr(plhs(1))
  ! Call subroutine and load Fortran array
6 call interpfunction(inputs,o)
7 call mxCopyReal8ToPtr(o,o_pr,e)
```

- Create $1 \times$ e output vector of type double (lines 3-4) and assign address (line 5)
- Call interpolation subroutine (line 6) and copy the data to the output address (line 7)

Parallel Processing in MATLAB

- Any calculations that are not dependent on the results of other calculations can be performed in parallel (e.g., solving for policy values at each node in the state space)
- Requires the Parallel Computing Toolbox (PCT)
- MATLAB 2014a or later: no limit on the number of workers
  MATLAB 2011a-2013b: maximum of 12 workers
  MATLAB 2009a-2010b: maximum of 8 workers
- All available processors are initialized with the function `matlabpool` (`parpool` in MATLAB 2013b and later)
- MATLAB Distributed Computing Server (MDCS) allows parallelization across nodes

Parallel Processing in MATLAB

The PCT requires the following alterations to the code:

- Replace `for` loops with `parfor` loops where applicable. This tells MATLAB to distribute each step in the loop across the specified number of processors
- If a nested `for` loop is used to update policy functions across dimensions, reduce it to one loop by changing to a single index (as opposed to specifying coordinates)
- Remove all global variables and instead use structures/parameter lists and variable arrays as direct inputs into functions called within the `parfor` loop

Parallel Processing in Fortran

OpenMP is a simple way to parallelize a do-loop in Fortran

```
!$omp parallel default(shared) private(g,s,pf)
!$omp do collapse(2)
do i2 = 1,Oz_pts
    do i1 = 1,Ok_pts
        g(1,1) = pf_n(i1,i2)
        s(1,1) = Gk_grid(i1)
        s(2,1) = Gz_grid(i2)
        call csolve(g,s,...,pf)
        pf_n_up(i1,i2) = pf(1,1)
    end do
end do
!$omp end do
!$omp end parallel
```

A New Keynesian Model for Estimation

The representative household chooses $\{c_t, n_t, b_t\}_{t=0}^{\infty}$ to maximize expected lifetime utility given by

$$E_0 \sum_{t=0}^{\infty} \tilde{\beta}_t [\log(c_t - h c_{t-1}^a) - \chi n_t^{1+\eta}/(1+\eta)],$$

where $\tilde{\beta}_0 \equiv 1$ and $\tilde{\beta}_t = \prod_{j=1}^{t} \beta_j$ for $t > 0$ subject to

$$c_t + b_t = w_t n_t + i_{t-1} b_{t-1}/\pi_t + d_t$$

Optimality implies

$$w_t = \chi n_t^{\eta}(c_t - h c_{t-1}^a),$$
$$1 = i_t E_t[q_{t,t+1}/\pi_{t+1}],$$

where $q_{t,t+1} \equiv \beta_{t+1}(c_t - h c_{t-1}^a)/(c_{t+1} - h c_t^a)$ is the pricing kernel.

New Keynesian Model

- Firm optimality condition:

$$\varphi\left(\frac{\pi_t}{\bar{\pi}}-1\right)\frac{\pi_t}{\bar{\pi}}=1-\theta+\theta\frac{w_t}{z_t}+\varphi E_t\left[q_{t,t+1}\left(\frac{\pi_{t+1}}{\bar{\pi}}-1\right)\frac{\pi_{t+1}}{\bar{\pi}}\frac{y_{t+1}}{y_t}\right]$$

- Production Function

$$y_t = z_t n_t$$

- Monetary policy rule

$$i_t = \max\{\underline{\imath}, i_t^*\}$$

$$i_t^* = (i_{t-1}^*)^{\rho_i}(\bar{\imath}(\pi_t/\bar{\pi})^{\phi_\pi}(c_t/(\bar{g}c_{t-1}))^{\phi_c})^{1-\rho_i}\exp(\nu_t),$$

where i^* is the notional interest rate.

New Keynesian Model

- Resource constraint:

$$c_t = [1 - \varphi(\pi_t/\bar{\pi} - 1)^2/2]y_t$$

- Discount factor (β) follows an AR(1) process

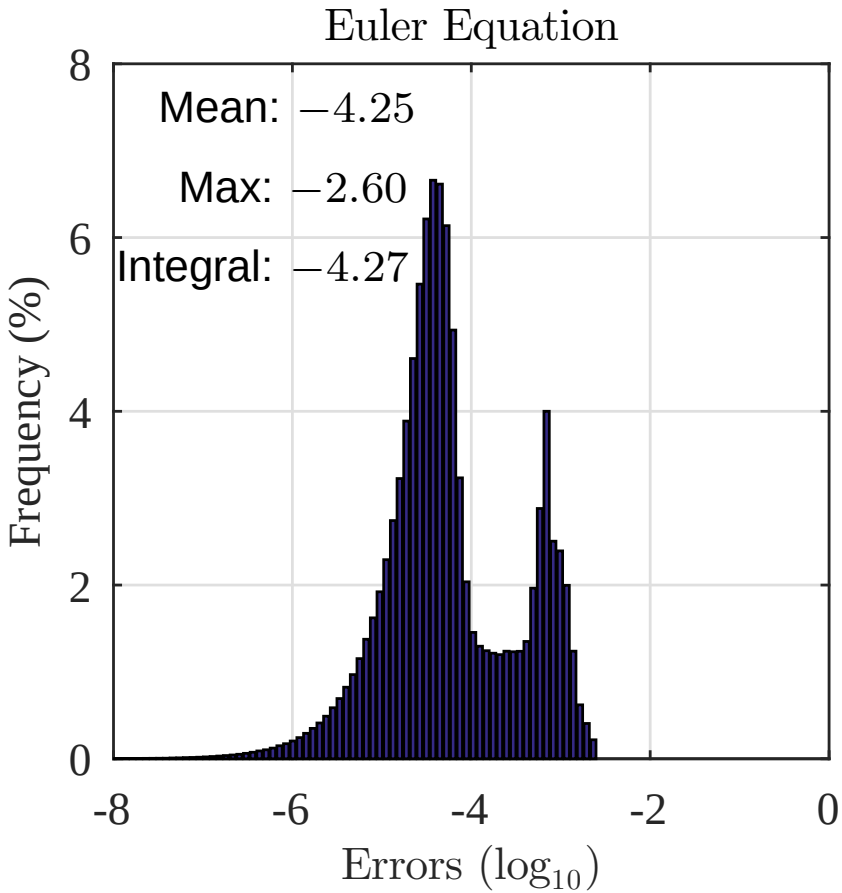
$$\beta_t = \bar{\beta}(\beta_{t-1}/\bar{\beta})^{\rho_\beta} \exp(\varepsilon_t)$$

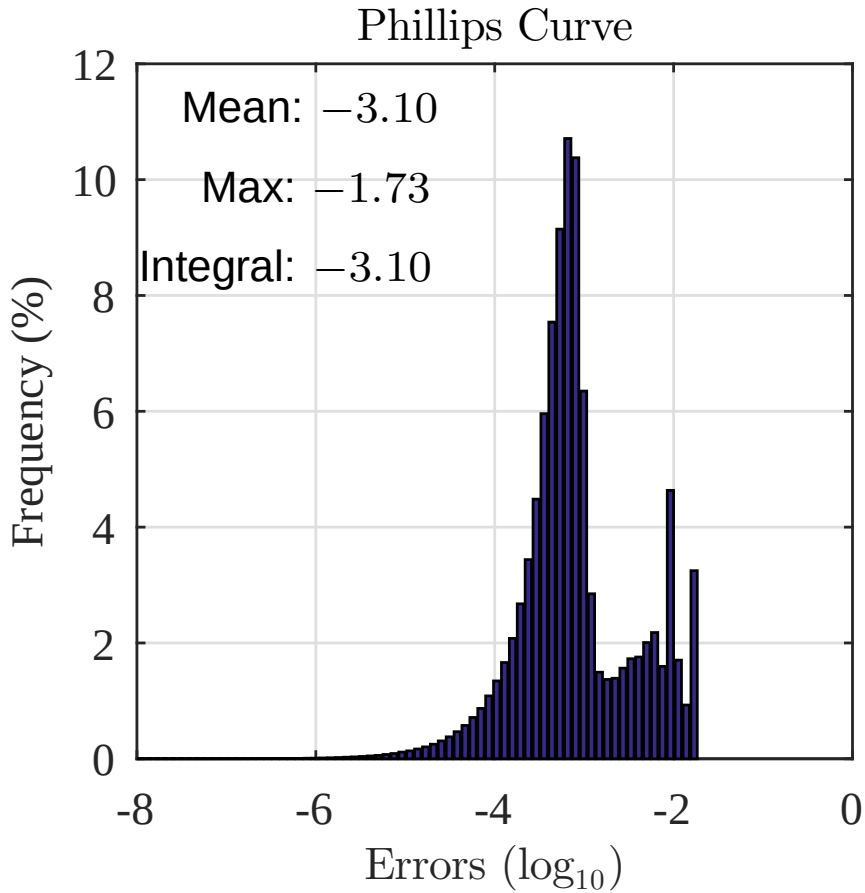
- Technology (z) follows a random walk:

$$z_t = z_{t-1} g_t$$
$$g_t = \bar{g}(g_{t-1}/\bar{g})^{\rho_g} \exp(\upsilon_t)$$

- Exogenous state variables: β_t, g_t, ν_t
- Endogenous state variables: c_{t-1}, i^*_{t-1}
- Policy functions: c_t, π_t

Numerical Error





Estimation Procedure

- Use quarterly data on per capita real GDP, the GDP price deflator, and the Fed Funds Rate from 1986Q1 to 2015Q4
- Use a Metropolis-Hastings algorithm with a particle filter to evaluate the likelihood of the posterior distribution
- Observation equation:

$$\begin{bmatrix} \log\left(\frac{RGDP_t/CNP_t}{RGDP_{t-1}/CNP_{t-1}}\right) \\ \log(DEF_t/DEF_{t-1}) \\ \log(1+FFR_t)/4 \end{bmatrix} = \begin{bmatrix} \log(g_t\tilde{c}_t/\tilde{c}_{t-1}) \\ \log(\pi_t) \\ \log(i_t) \end{bmatrix} + \begin{bmatrix} \xi_{1t} \\ \xi_{2t} \\ \xi_{3t} \end{bmatrix},$$

  where $\xi \sim \mathbb{N}(0, \Sigma)$ is a vector of measurement errors.
- We adapt the particle filter to incorporate the information contained in the current observation, which helps the model better match outliers in the data (e.g., 2008Q4).

Metropolis-Hastings Algorithm

For all $i \in \{0, \ldots, N_d\}$, perform the following steps:

1. Draw a candidate vector of parameters, $\hat{\theta}_i^{cand}$, where

$$\hat{\theta}_i^{cand} \sim \begin{cases} \mathbb{N}(\theta_0, c_0\Sigma) & \text{for } i = 0, \\ \mathbb{N}(\hat{\theta}_{i-1}, c\Sigma) & \text{for } i > 0. \end{cases}$$

2. Compute prior density: $\log \ell_i^{prior} = \sum_{j=1}^{N_p} \log p(\hat{\theta}_{i,j}^{cand} | \mu_j, \sigma_j^2)$
3. Given $\hat{\theta}_i^{cand}$, solve the model. If the algorithm converges, use the particle filter to obtain $\log \ell_i^{model}$, otherwise repeat 1.
4. Accept or reject the candidate draw according to

$$(\hat{\theta}_i, \log \ell_i) = \begin{cases} (\hat{\theta}_i^{cand}, \log \ell_i^{cand}) & \text{if } i = 0, \\ (\hat{\theta}_i^{cand}, \log \ell_i^{cand}) & \text{if } \log \ell_i^{cand} - \log \ell_{i-1} > \hat{u}, \\ (\hat{\theta}_{i-1}, \log \ell_{i-1}) & \text{otherwise,} \end{cases}$$

where $\hat{u} \sim \mathbb{U}[0, 1]$ and $\log \ell_i^{cand} = \log \ell_i^{prior} + \log \ell_i^{model}$.

Adapted Particle Filter

1. Initialize the filter by drawing from the ergodic distribution
2. For all particles $p \in \{1, \ldots, N_p\}$ apply the following steps:
   - 2.1 Draw $\mathbf{e}_{t,p} \sim \mathbb{N}(\bar{\mathbf{e}}_t, I)$, where $\bar{\mathbf{e}}_t$ maximizes $p(\xi_t|\mathbf{z}_t)p(\mathbf{z}_t|\mathbf{z}_{t-1})$
   - 2.2 Obtain $\mathbf{z}_{t,p}$ and the vector of variables, $\mathbf{w}_{t,p}$, given $\mathbf{z}_{t-1,p}$
   - 2.3 Calculate, $\xi_{t,p} = \hat{\mathbf{x}}_{t,p}^{model} - \hat{\mathbf{x}}_t^{data}$. The weight on particle p is

$$\omega_{t,p} = \frac{p(\xi_t|\mathbf{z}_{t,p})p(\mathbf{z}_{t,p}|\mathbf{z}_{t-1,p})}{g(\mathbf{z}_{t,p}|\mathbf{z}_{t-1,p}, \hat{\mathbf{x}}_t^{data})} \propto \frac{\exp(-\xi_{t,p}' H^{-1} \xi_{t,p}/2) \exp(-\mathbf{e}_{t,p}'\mathbf{e}_{t,p}/2)}{\exp(-(\mathbf{e}_{t,p} - \bar{\mathbf{e}}_t)'(\mathbf{e}_{t,p} - \bar{\mathbf{e}}_t)/2)}$$

     The model's likelihood at t is $\ell_t^{model} = \sum_{p=1}^{N_p} \omega_{t,p}/N_p$.

   - 2.4 Normalize the weights, $W_{t,p} = \omega_{t,p}/\sum_{p=1}^{N_p} \omega_{t,p}$. Then use systematic resampling with replacement from the particles.
3. Apply step 2 for $t \in \{1, \ldots, T\}$. $\log \ell^{model} = \sum_{t=1}^{T} \log \ell_t^{model}$.

Particle Adaption

1. Given $\mathbf{z}_{t-1}$ and a guess for $\bar{\mathbf{e}}_t$, obtain $\mathbf{z}_t$ and $\mathbf{w}_{t,p}$
2. Calculate $\hat{\mathbf{x}}_t^{model} = \left[\log(g_t \tilde{y}_t^{gdp} / \tilde{y}_{t-1}^{gdp}), \log(\pi_t), \log(i_t)\right]$.
3. Calculate $\xi_t = \hat{\mathbf{x}}_t^{model} - \hat{\mathbf{x}}_t^{data}$, which is multivariate normal:

$$p(\xi_t|\mathbf{z}_t) = (2\pi)^{-3/2}|H|^{-1/2}\exp(-\xi_t' H^{-1}\xi_t/2)$$
$$p(\mathbf{z}_t|\mathbf{z}_{t-1}) = (2\pi)^{-3/2}\exp(-\bar{\mathbf{e}}_t'\bar{\mathbf{e}}_t/2)$$

$H \equiv \mathrm{diag}(\sigma_{me,\hat{y}}^2, \sigma_{me,\pi}^2, \sigma_{me,i}^2)$ is the ME covariance matrix.

4. Solve for the optimal $\bar{\mathbf{e}}_t$ to maximize

$$p(\xi_t|\mathbf{z}_t)p(\mathbf{z}_t|\mathbf{z}_{t-1}) \propto \exp(-\xi_t' H^{-1}\xi_t/2)\exp(-\bar{\mathbf{e}}_t'\bar{\mathbf{e}}_t/2)$$

We converted MATLAB's `fminsearch` routine to Fortran.

SYSTEMATIC RESAMPLING

- Resampling is the key step in the particle filter.
- Resampling is used to avoid the problem of *degeneracy*: a situation when all but a few of the weights are near zero because the variance of the weights increases over time.
- With resampling, one draws (with replacement) a set of particles from the approximation to the filtering distribution
- Since resampling is done with replacement, a particle with a large weight is likely to be drawn multiple times and particles with small weights are not likely to be drawn at all.
- Resampling effectively deals with the degeneracy problem by getting rid of the particles with very small weights.

Systematic Resampling: Example

```
cdf = cumsum(weights);
Udraws = (rand(nweights,1) +
          (0:(nweights-1))')/nweights;

ipart = 1;
idx = zeros(nweights,1);
for idraw = 1:nweights
    while Udraws(idraw) > cdf(ipart)
        % Reject particle
        ipart = ipart + 1;
    end
    % Resample particle
    idx(idraw) = ipart;
end
```

Programming and Parallelization

- Entire algorithm is programmed in Fortran using Open MPI
- Solve the model by parallelizing the nodes in the state space across all available processors
- Improve filter accuracy by calculating the posterior likelihood on each processor and evaluate whether to accept or reject a draw based on the median likelihood
- With 64 processors, on average it takes 1 second to solve the nonlinear model and 3.3 seconds to filter the data
- In total, we obtain $135{,}000$ draws ($10{,}000$ for the mode search, $25{,}000$ for the initial MH step, and $100{,}000$ for the final MH step), so the total run time is about 1 week.

MESSAGE PASSING INTERFACE (MPI)

- Initialize and finalize MPI only once:

```
call mpi_init(ierr)
call mpi_comm_rank(mpi_comm_world,myid,ierr)
call mpi_comm_size(mpi_comm_world,nprocs,ierr)
...
call mpi_finalize(rc)
```

- Processes do not communicate unless ordered:

```
call mpi_bcast(var,n,type,0,mpi_comm_world,ierr)
```

- Need a way to merge the calculations on each process:

```
call mpi_allreduce(var_temp,var,n,type,&
                   operation,mpi_comm_world,ierr)
```

- Processes may not finish at the same time:

```
call mpi_barrier(mpi_comm_world,ierr)
```

Parallel Processing with openMPI

On a given node, apply the following:

```
do inode = myid + 1,Gnodes,nprocs
  g(1,1) = pf_n(i1,i2)
  s(1,1) = Gk_grid(i1)
  s(2,1) = Gz_grid(i2)
  call csolve(g,s,...,pf)
  pf_n_up(i1,i2) = pf(1,1)
end do
! Impose temporal order
call mpi_barrier(MPI_COMM_WORLD,ierr)
! Combine argzero across processors
call mpi_allreduce(pfn_up_temp,pfn_up,Gnodes,&
                   mpi_double_precision, &
                   mpi_sum,mpi_comm_world,ierr)
```

Cluster Commands

- Open source software:
  - PuTTY (`http://www.putty.org/`):
    Allows users to communicate with the cluster
  - WinSCP (`https://winscp.net/eng/download.php`):
    Allows users to transfer files to the cluster

- Scheduler commands for SLURM:
  - `squeue`: displays all submitted jobs
  - `sinfo`: display cluster usage by queue type
  - `scancel`: cancels a running job
  - `sbatch runscript`: submits job to queue

Example: Run Script

```
#!/bin/bash
#SBATCH --job-name=name
#SBATCH --out=OUT
#SBATCH --partition=compute
#SBATCH --time=hh:mm:ss
#SBATCH --ntasks=processors
#SBATCH --distribution=block:block
#SBATCH --nodes=number of nodes
#SBATCH --ntasks-per-node=16
#SBATCH --mail-type=ALL
#SBATCH --mail-user=email1,email2
mpirun ./a.out
```

Additional Resources

- For more info on the solution method see Richter, Throckmorton & Walker (*Computational Economics*, 2014)
- All of our code is available at: `http://alexrichterecon.com`
- Examples include:
  - Textbook real business cycle model
  - Real business cycle models with real frictions
  - Textbook New Keynesian model
  - NK model with a zero lower bound constraint
  - NK model with Epstein-Zin preferences
  - NK model with monetary and fiscal policy switching
- For more info on nonlinear estimation see Plante, Richter & Throckmorton (*Economic Journal*, 2017)