

SOLVING, SIMULATING AND ESTIMATING NONLINEAR DSGE MODELS

Alexander W. Richter
Federal Reserve Bank of Dallas

November 2025

The views expressed in this presentation are my own and do not necessarily reflect the views of the Federal Reserve Bank of Dallas or the Federal Reserve System

OUTLINE

1. BASIC ALGORITHM

2. APPLICATIONS

3. GIRFs & ESTIMATION

4. MEX & PARALLELIZATION

5. HPC COMMANDS

RBC MODEL: FIRMS

A firm chooses $\{n_t, k_t\}$ to maximize:

$$y_t - w_t n_t - r_t^k k_t$$

subject to

$$y_t = z_t k_t^\alpha n_t^{1-\alpha}$$

Optimality conditions:

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

$$r_t^k = \alpha y_t/k_t$$

RBC MODEL: HOUSEHOLDS

A household chooses $\{c_t, n_t, i_t, k_{t+1}\}$ 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

$$c_t + i_t = w_t n_t + r_t^k k_t$$

$$k_{t+1} = (1 - \delta)k_t + i_t$$

Optimality conditions:

$$w_t = \chi n_t^{\eta} c_t^{\sigma}$$

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

RBC MODEL: EQUILIBRIUM

Nonlinear system of equations:

$$w_t = \chi n_t^\eta c_t^\sigma$$

$$1 = \beta E_t[(c_t/c_{t+1})^\sigma (r_{t+1}^k + 1 - \delta)]$$

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

$$r_t^k = \alpha y_t/k_t$$

$$y_t = z_t k_t^\alpha n_t^{1-\alpha}$$

$$c_t + i_t = y_t$$

$$k_{t+1} = (1 - \delta)k_t + i_t$$

$$z_t = (1 - \rho)\bar{z} + \rho z_{t-1} + \sigma_z \varepsilon_t, \quad \varepsilon_t \sim \mathbb{N}(0, 1)$$

Variables(8): n, w, c, y, r^k, k, i, z

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:

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

- 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:

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

FUNCTIONAL APPROXIMATION

- True RE solution only exists in very special cases
- Find an approximating function that maps the state space to the optimal decision rule for consumption:

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

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

ALGORITHM OUTLINE

1. Use the log-linear solution to obtain $\mathcal{P}_n^0$ and set $q = 1$

On node $i = 1, 2, \dots$

2. Solve for the updated state (k', z') , given (k_i, z_i)
3. Use linear interpolation to compute $\mathcal{P}_n^q(k', z')$
4. Solve for the time $t + 1$ variables that enter expectations
5. Evaluate expectations using numerical integration:

$$E[\Phi(z')] = \beta E[c(k', z')^{-\sigma} (\alpha z' k'^{\alpha-1} n'^{1-\alpha} + 1 - \delta)]$$

6. Use a solver to find $\mathcal{P}_n^q(k_i, z_i)$ that satisfies the system

Then...

7. Calculate the distance between updates: $\mathcal{P}_n^q - \mathcal{P}_n^{q-1}$
8. If $|\text{dist}| < \text{tol}$, then stop. If not, then set $q = q + 1$ and repeat steps 2-7 using $\mathcal{P}_n^q = \mathcal{P}_n^{q-1}$ as the new initial conjecture

INITIAL CONJECTURE

- Use the linear solution from `gensys` as a guess for $\mathcal{P}_n$
- Map the model to the following form:

$$G_0 \hat{Y}_t = G_1 \hat{Y}_{t-1} + \Psi \varepsilon_t + \Pi \eta_t + C$$

where $\hat{Y}$ is a vector of variables, ε is a vector of shocks, C is a vector of constants, and η is a vector of forecast errors

- Linear solution takes the form:

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

- Convert the state space to deviations from steady state
- Compute an initial conjecture for all nodes ($i = 1, \dots, N$):

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

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

RBC MODEL: LINEAR SYSTEM

Log-Linear system of equations:

$$\hat{w}_t = \eta \hat{n}_t + \sigma \hat{c}_t$$

$$\sigma(\hat{c}_t - E_t \hat{c}_{t+1}) + (1 - \beta(1 - \delta)) E_t \hat{r}_{t+1}^k = 0$$

$$\hat{w}_t = \hat{y}_t - \hat{n}_t$$

$$\hat{r}_t^k = \hat{y}_t - \hat{k}_t$$

$$\hat{y}_t = \hat{z}_t + \alpha \hat{k}_t + (1 - \alpha) \hat{n}_t$$

$$\bar{c} \hat{c}_t + \bar{n} \hat{n}_t = \bar{y} \hat{y}_t$$

$$\hat{k}_t = (1 - \delta) \hat{k}_{t-1} + \delta \hat{i}_{t-1}$$

$$\hat{z}_t = \rho \hat{z}_{t-1} + \sigma_z \varepsilon_t$$

Variables(8): $\hat{n}, \hat{w}, \hat{c}, \hat{y}, \hat{r}^k, \hat{k}, \hat{i}, \hat{z}$

PIECEWISE LINEAR INTERPOLATION

- Goal: Find the policy function value $\mathcal{P}_n(k', z')$
- Interpolate in the k direction:

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

- Then interpolate in the z direction:

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

- Combine these two equations:

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

PIECEWISE LINEAR INTERPOLATION

- We have policy function values on the nearest nodes:

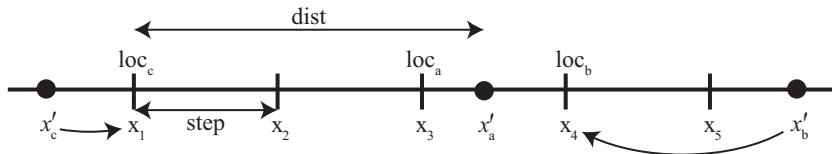
$$[\mathcal{P}_n(k_i, z_j), \mathcal{P}_n(k_i, z_{j+1}), \mathcal{P}_n(k_{i+1}, z_j), \mathcal{P}_n(k_{i+1}, z_{j+1})]$$

once we determine the grid indices, i, j

- Locate the grid point to left of x' , $x \in \{k, z\}$

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

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



PIECEWISE LINEAR INTERPOLATION

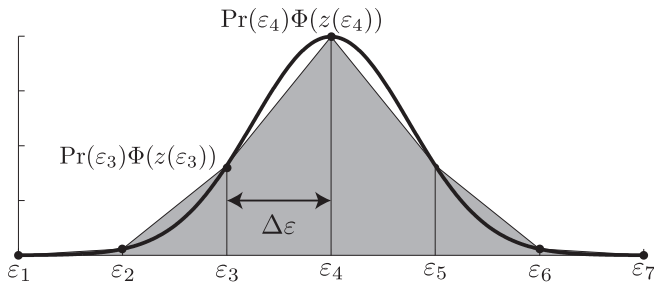
- Use a nested loop to calculate the interpolated value:

```
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 states

INTEGRATION: TRAPEZOID RULE

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



INTEGRATION: GAUSS-HERMITE

- Expectation 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, $v = (\varepsilon - \mu)/(\sqrt{2}\sigma)$:

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

- v_i are the roots of $H_n(x)$ and ω_i are weights given by

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

H_n is the physicist's Hermite polynomial of order n

INTEGRATION: ROUWENHORST

- Used to approximate an exogenous $AR(1)$ process
- Kopecky and Suen (2010) show the Rouwenhorst method outperforms other approximations of an $AR(1)$ process
- Approximation is an n -state Markov switching process
- 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

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

- The bounds and transition matrix are chosen to match conditional and unconditional mean and variance of z

INTEGRATION: ROUWENHORST

- 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 is computed recursively:
 - For $n = 2$, let $q = (\rho + 1)/2$:

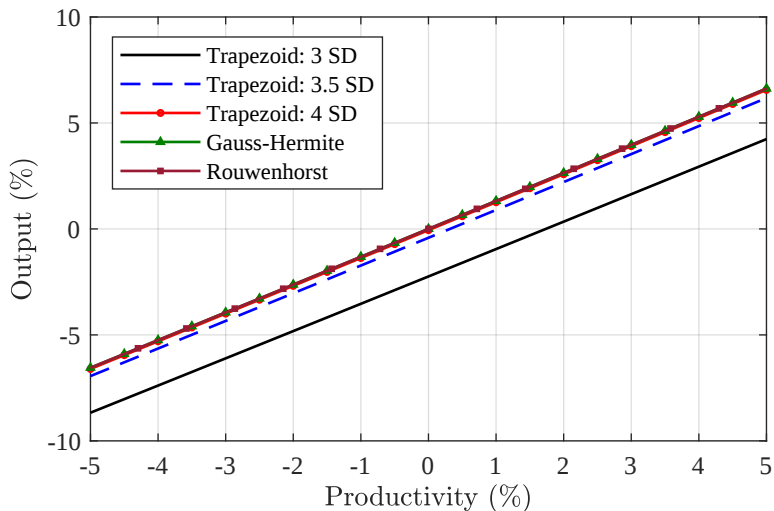
$$P_2 = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} = \begin{bmatrix} q & 1-q \\ 1-q & q \end{bmatrix}$$

- For $n > 2$:

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

Divide rows 2 through $n-1$ by 2 so they sum to 1

POLICY FUNCTION COMPARISON



ADVANTAGES OF ROUWENHORST

- Improves accuracy:
 - ▶ Matches 5 statistics of an $AR(1)$ process: 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 the other methods
 - ▶ 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

GAUSS-HERMITE: INTERPOLATION

Suppose there are 4 states: 2 endogenous and 2 exogenous:

```
% Calling function
for i2 = 1,ne2
    for i1 = 1,nel
        pfMat(i1,i2) = interp(grid, stateup(i1,i2), pfs)
    end
end

% Interpolation function
for m4 = 0,1
    for m3 = 0,1
        for m2 = 0,1
            for m1 = 0,1
                wtemp = w1(m1+1)*w2(m2+1)*w3(m3+1)*w4(m4+1)
                sum = sum + wtemp*pf(loc1+m1, loc2+m2, loc3+m3, loc4+m4)
            end
        end
    end
end
```

ROUWENHORST: INTERPOLATION

Suppose there are 4 states: 2 endogenous and 2 exogenous:

```
% Calling function
pfMat = interp(grid, endogenous, state, pfs)

% Interpolation function
for m2 = 0,1
    for m1 = 0,1
        sum = sum + w1(m1+1)*w2(m2+1)*pf(:, :, loc1+m1, loc2+m2)
    end
end
```

No longer need to interpolate on the exogenous dimensions,
which significantly reduces the number of computations

ROUWENHORST: INTEGRATION

- Conditional expectation of an n -state Markov process:

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

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

$$E[\Phi(z', \nu')|s_z = i_z, s_\nu = i_\nu] = \sum_{j_\nu=1}^{n_\nu} \sum_{j_z=1}^{n_z} p_{i_\nu j_\nu} p_{i_z j_z} \Phi(z_{j_z}, \nu_{j_\nu})$$

where $i_z, j_z \in \{1, 2, \dots, n_z\}$ and $i_\nu, j_\nu \in \{1, 2, \dots, n_\nu\}$

IMPLEMENTATION: TWO SHOCKS

- Script: Compute shock realizations:

```
zpMat = G.z_nodes(:,ones(O.epsnu_pts,1));
nupMat = permute(G.nu_nodes(:,ones(O.epsz_pts,1)), [2,1]);
```

- Script: Compute transition matrix:

```
z_weightVec = G.z_weight(G.z_gr(inode) == G.z_grid,:)' ;
nu_weightVec = G.nu_weight(G.nu_gr(inode) == G.nu_grid,:)' ;
z_weightMat = z_weightVec(:,ones(O.epznu_pts,1));
nu_weightMat = permute(nu_weightVec(:,ones(O.epsz_pts,1)), [2,1]);
weightMat = z_weightMat.*nu_weightMat
argzero = csolve('eqm', start, [], crit, itmax, state, ...
                 structures, zpMat, nupMat, weightMat);
```

- Eqm: Compute all realizations and integrate:

```
Econs = sum(weightMat.*PhiMat,'all');
```

EXAMPLE 1: REAL FRICTIONS

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

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

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

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

$$1 = q_t \left(1 - \frac{\nu}{2} \left(\frac{i_t}{i_{t-1}} - 1 \right)^2 - \nu \frac{i_t}{i_{t-1}} \left(\frac{i_t}{i_{t-1}} - 1 \right) \right) + \nu E_t \left[x_{t+1} q_{t+1} \left(\frac{i_{t+1}}{i_t} \right)^2 \left(\frac{i_{t+1}}{i_t} - 1 \right) \right]$$

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

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

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

$$c_t + i_t = y_t$$

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

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

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

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

EXAMPLE 2: EPSTEIN-ZIN PREFERENCES

Introduce Epstein-Zin preferences into the baseline model

$$\chi w_t(1 - n_t) = (1 - \chi)c_t$$

$$1 = E_t[x_{t+1}(r_{t+1}^k + 1 - \delta)]$$

$$x_t = \beta(u_t/u_{t-1})^{1-1/\psi}(c_{t-1}/c_t)(J_t/a_{t-1})^{1/\psi-\gamma}$$

$$u_t = c_t^\chi(1 - n_t)^{1-\chi}$$

$$a_t = \begin{cases} \exp(E_t[\ln J_{t+1}]) & \text{if } \gamma = 1 \\ (E_t[J_{t+1}^{1-\gamma}])^{1/(1-\gamma)} & \text{if } \gamma \neq 1 \end{cases}$$

$$J_t = \begin{cases} u_t^{1-\beta} a_t^\beta & \text{if } \psi = 1 \\ \left((1 - \beta)u_t^{1-1/\psi} + \beta a_t^{1-1/\psi} \right)^{\frac{1}{1-1/\psi}} & \text{if } \psi \neq 1 \end{cases}$$

State variables: $\{k, z\}$ (unchanged); Policy functions: $\{n, J\}$

EXAMPLE 3: STOCHASTIC VOLATILITY

- Productivity is $AR(1)$ with stochastic volatility

$$\ln z_t = \rho_z \ln z_{t-1} + \sigma_{z,t-1} \varepsilon_{z,t}$$

$$\ln \sigma_{z,t} = (1 - \rho_{sv}) \ln \bar{\sigma}_a + \rho_{sv} \ln \sigma_{z,t-1} + \sigma_{sv} \varepsilon_{sv,t}$$

- Cannot discretize $\ln z_t$ with Rouwenhorst
- Instead, discretize...
 - ▶ $\varepsilon_{z,t}$ with Rouwenhorst
 - ▶ $\ln \sigma_{z,t}$ with Rouwenhorst
 - ▶ $\ln z_t$ with evenly-spaced nodes after choosing bounds

SETTING UP THE GRID

```
% Rouwenhorst for productivity volatility shock
[G.epssigz_nodes,G.epssigz_weight] = ...
    rouwenhorst(log(P.sigz),P.rhosv,P.sigsv,O.epssigz_pts);
% Rouwenhorst for productivity shock
[G.epsz_nodes,G.epsz_weight] = ...
    rouwenhorst(0,0,1,O.epsz_pts);

% Grid points for productivity shock volatility
G.lnsigz_grid = G.epssigz_nodes';
% Grid points for productivity level
G.lnz_grid = linspace(O.lnzmin,O.lnzmax,O.lnz_pts);
% Grid points for capital
G.k_grid = linspace(O.kmin,O.kmax,O.k_pts);

% Grid arrays
[G.lnsigz_gr,G.lnz_gr,G.k_gr] = ...
    ndgrid(G.lnsigz_grid,G.lnz_grid,G.k_grid);
```

INTERPOLATION WITH SV

- Shock realizations

```
lnsigzpMat = repmat(G.epssigz_nodes,[1,0.epsz_pts]);
epszMat = permute(repmat(G.epsz_nodes,...
    [1,0.epssigz_pts]),[2,1]);
```

- Inside time-iteration loop, but before the call to `csolve`:

```
lnzpMat = (1-P.rhoz)*log(P.zbar)+P.rhoz*G.lnz_gr(i)
    + exp(G.lnsigz_gr(i))*epszMat;
```

- Interpolation in `eqm.m`:

```
npMat = zeros(0.epssigz_pts,0.epsz_pts);
kpMat = kp*ones(0.epssigz_pts,0.epsz_pts);
npMat(:) = Fallterp31(...
    G.lnsigz_grid,G.lnz_grid,G.k_grid,...
    lnsigzpMat(:),lnzpMat(:),kpMat(:),...
    pf.n);
```

ENDOGENOUS UNCERTAINTY

- In a nonlinear model, the expected volatility of forecast errors (i.e., uncertainty) endogenously varies over time
- Definition follows “Measuring Uncertainty” (AER, 2015)

$$\mathcal{U}_t = \sqrt{E_t[(\ln y_{t+h} - E_t[\ln y_{t+h}])^2]}$$

where h is the forecast horizon

- After solving the model, create a policy function for $\mathcal{U}_t$
 - ▶ Loop across the nodes due to the expectation
 - ▶ Can simulate like any other policy function
 - ▶ Similar procedure can be used for asset prices

EXAMPLE 4: MONETARY POLICY AND ZLB

- Phillips curve:

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

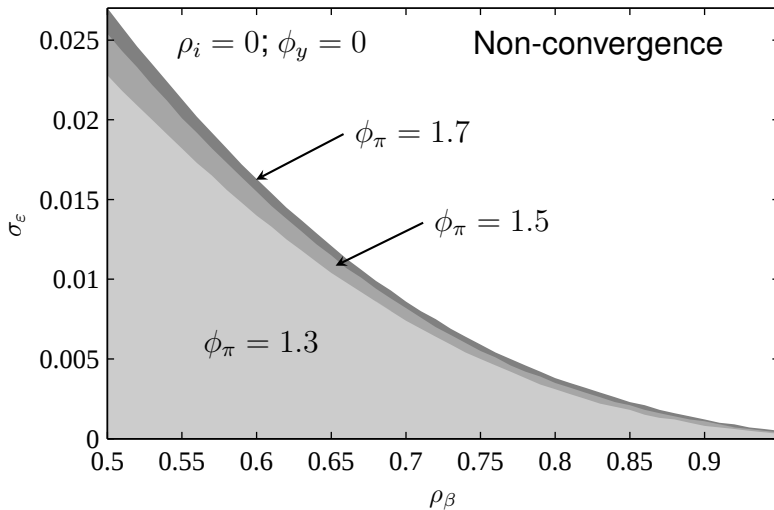
- Monetary policy rule:

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

$$i_t^* = (i_{t-1}^*)^{\rho_i} (\bar{l}(\pi_t/\bar{\pi})^{\phi_\pi} (y_t/y_{t-1})^{\phi_y})^{1-\rho_i} \exp(\sigma_\nu \nu_t)$$

- Adds π_t as a policy function, y_{t-1} and i_{t-1}^* as endogenous state variables, and ν_t as an exogenous state variable
- Convergence region imposes a tradeoff between the expected frequency and average duration of ZLB events (See “The Zero Lower Bound: Frequency, Duration, and Numerical Convergence,” BEJM, 2015)

CONVERGENCE (SHADED) REGIONS



EXAMPLE 5: POLICY SWITCHING

- “Monetary and Fiscal Policy Switching”
(Chung, Davig and Leeper, JMCB, 2007)
- Monetary and fiscal policy rules:

$$r_t = \bar{r}(\pi_t/\bar{\pi})^{\phi(s_t)} \exp(\varepsilon_{r,t})$$

$$\tau_t = \bar{\tau}(b_{t-1}/\bar{b})^{\gamma(s_t)} \exp(\varepsilon_{\tau,t})$$

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

- Transition probabilities:

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

CONTINUOUS+DISCRETE SHOCKS

- Discretize the continuous shocks using Rouwenhorst
- Create a grid for the policy regime

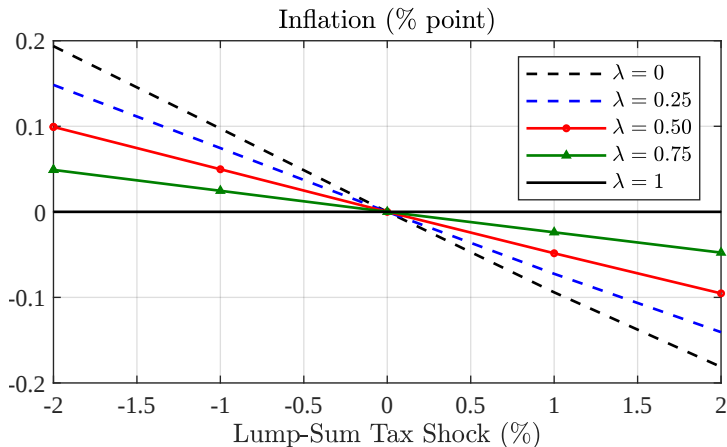
```
G.s_nodes = [0,1]';
G.s_weight = [P.p11 1-P.p11;
              1-P.p22 P.p22];
```

- Set the integration weights by creating arrays

```
% Integration weights
epsfp_weightVec = G.epsfp_weight(1,:)' ;
epsmp_weightVec = G.epsmp_weight(1,:)' ;
s_weightVec = G.s_weight(G.s_gr(i) == G.s_grid,:)' ;
epsfp_weightArr3 = ...
    epsfp_weightVec(:,ones(O.mp_pts,1),ones(O.s_pts,1));
epsmp_weightArr3 = permute(...
    epsmp_weightVec(:,ones(O.fp_pts,1),ones(O.s_pts,1)), [2,1,3]);
s_weightArr3 = permute(...
    s_weightVec(:,ones(O.epsfp_pts,1),ones(O.epsmp_pts,1)), [2,3,1]);
weightArr3 = epsfp_weightArr3.*epsmp_weightArr3.*s_weightArr3;
```

EXPECTATIONAL EFFECTS: LUMP-SUM TAX

Average duration of time spent in Regime 1 in the ergodic distribution: $\lambda = (1 - p_{22}) / (2 - p_{11} - p_{22})$



EXAMPLE 6: DMP AND NET FIRM ENTRY

- “Cyclical Net Entry and Exit” (EER, 2021)
- Active firm value function:

$$\begin{cases} J_{A,t}^F \in (0, \psi_n), & \text{if } \Delta Z = 0 \\ J_{A,t}^F = \psi_n, & \text{if } \Delta Z \geq 0 \\ J_{A,t}^F = 0, & \text{if } \Delta Z \leq 0 \end{cases}$$

- Introduce auxiliary variable $\mu_{A,t}$ to impose these conditions

$$J_{A,t}^F = \min \{ \max \{ 0, \mu_{A,t} \}, \psi_n \}, \quad \Delta Z_t = \mu_{A,t} - J_{A,t}^F$$

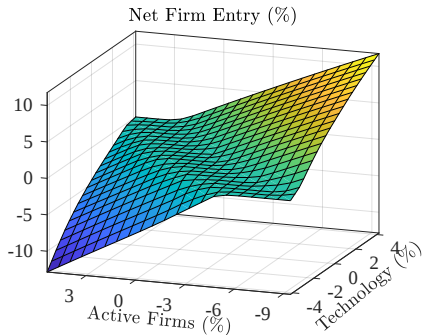
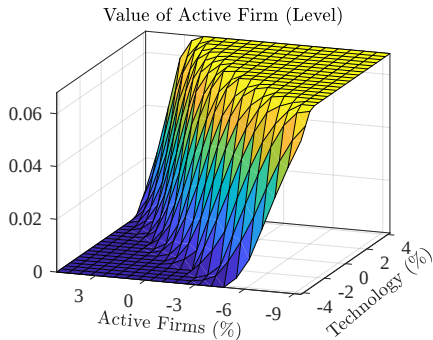
- Introduce second auxiliary variable $\mu_{V,t}$ to ensure $V_t \geq 0$

$$V_t = \max \{ 0, \mu_{V,t} \}^2, \quad \lambda_{V,t} = \max \{ 0, -\mu_{V,t} \}^2$$

$$\frac{\kappa - \lambda_{V,t}}{q_t} = w_t - w_t^n + E_t [x_{t+1} (1 - s_{t+1}) \frac{\kappa - \lambda_{V,t+1}}{q_{t+1}}]$$

POLICY FUNCTIONS FOR FIRM DYNAMICS

Significant nonlinearities in the model: Net firm entry is positive when the active firm value hits the entry cost



EXAMPLE 7: LABOR FORCE PARTICIPATION

- Same auxiliary variable procedure is needed to solve models with endogenous labor force participation
- Transform the complementary slackness condition

$$\lambda_{\ell,t}(1 - \ell_t) = 0, \quad \lambda_{\ell,t} \geq 0$$

into the following auxiliary variable specification:

$$\begin{aligned}\ell_t &= n_t + u_t \\ \ell_t &= 1 - \max\{0, \mu_{u,t}\}^2 \\ \lambda_{\ell_t} &= \max\{0, -\mu_{u,t}\}^2\end{aligned}$$

- Procedure follows Garcia and Zangwill (1981)

EXAMPLE 8: OIL PRODUCTION RISK

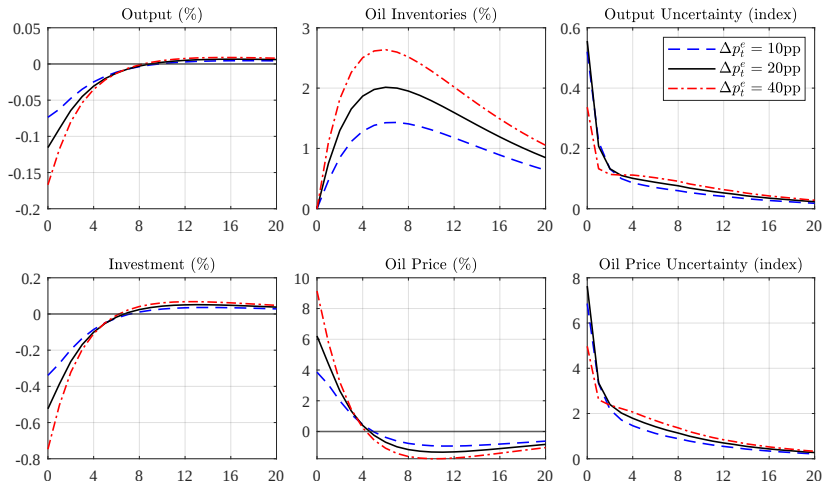
- “Geopolitical Oil Price Risk and Economic Fluctuations” (Dallas Fed WP #2403, 2025)
- Oil production:

$$o_t^s = \underbrace{a_t^o}_{\text{Permanent Component}} \times \underbrace{e_t}_{\text{Transitory Component}}$$

- Transitory component captures geopolitical disruptions:

$$\begin{aligned} \ln e_t &= \ln \bar{e} - \zeta_e (v_t^e - \bar{\pi}_1^e), \\ \Pr(v_{t+1}^e = 1 | v_t^e = 0) &= p_t^e, \quad \Pr(v_{t+1}^e = 1 | v_t^e = 1) = \bar{q}^e \\ \ln p_t^e &= \min\{0, (1 - \rho_p^e) \ln \bar{p}^e + \rho_p^e \ln p_{t-1}^e + \sigma_p^e \varepsilon_{p,t}^e\} \end{aligned}$$

OIL DISASTER PROBABILITY SHOCK



Notes: Responses in deviations from the baseline. Assumes no disasters are realized.

GENERALIZED IMPULSE RESPONSES

- Used to study model dynamics away from steady state
- The GIRF of x_{t+h} over horizon h is defined as:

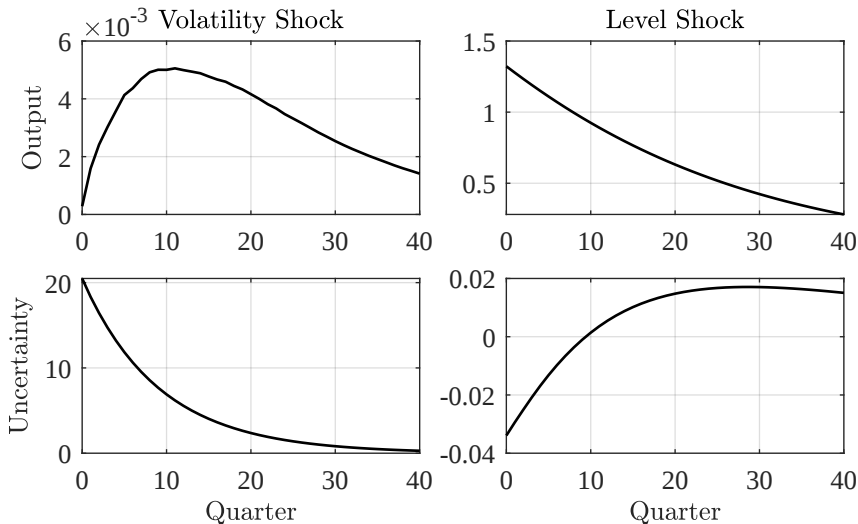
$$\mathcal{G}(x_{t+h}|\varepsilon_{t+1} = \xi, \mathbf{z}_t) = E_t[x_{t+h}|\varepsilon_{t+1} = \xi, \mathbf{z}_t] - E_t[x_{t+h}|\mathbf{z}_t]$$

where the conditional expectations are computed based on the mean path from repeated simulations of the model

- Initialize simulations at $\mathbf{z}_t$, which equals the average of periods where $z \in \mathbf{z}$ is within a certain distance of z^*

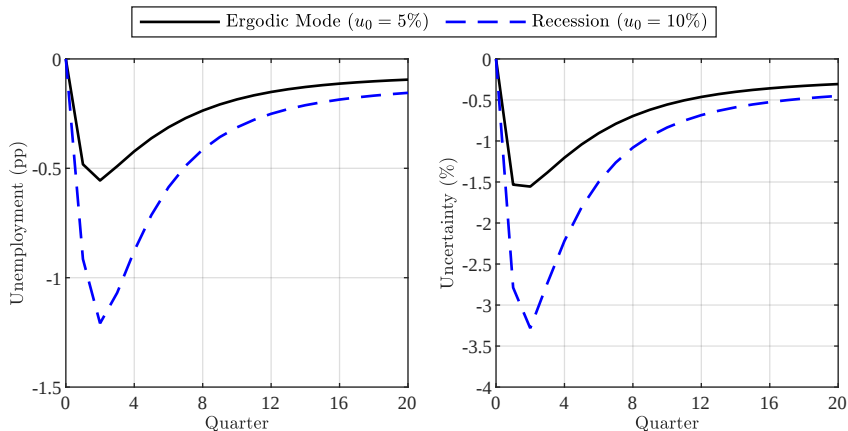
```
z_idx = exp(ergdraws(:,V.z)) > z0-0.005 & ...
        exp(ergdraws(:,V.z)) < z0+0.005;
initvec = mean(ergdraws(z_idx,:),1)';
```

RBC MODEL: UNCERTAINTY



DMP MODEL: STATE-DEPENDENCE

Responses to an increase in the level of productivity



VARIANCE DECOMPOSITION

Consider two total variance decompositions:

1. Let $\{\varepsilon_{-j}\}_{t+1}^{t+h}$ denote a realization of all shocks except j in periods $t+1$ to $t+h$, then law of total variance implies

$$V_t[y_{t+h}] = \underbrace{E_t[V_t[y_{t+h}|\{\varepsilon_{-j}\}_{t+1}^{t+h}]]}_{\text{Total Effect of } \varepsilon_j} + \underbrace{V_t[E_t[y_{t+h}|\{\varepsilon_{-j}\}_{t+1}^{t+h}]]}_{\text{Residual Variance of } \varepsilon_{-j}}$$

2. Conditioning on the j th shock, $\{\varepsilon_j\}_{t+1}^{t+h}$, implies

$$V_t[y_{t+h}] = \underbrace{E_t[V_t[y_{t+h}|\{\varepsilon_j\}_{t+1}^{t+h}]]}_{\text{Total Effect of } \varepsilon_{-j}} + \underbrace{V_t[E_t[y_{t+h}|\{\varepsilon_j\}_{t+1}^{t+h}]]}_{\text{Direct Effect of } \varepsilon_j}$$

Method parses out the direct and interaction effects of shock j

DECOMPOSITION IMPLEMENTATION

Suppose the model has a TFP level and volatility shock

```
for isim = 1:nsimsTVD
    % Simulate: Fixed level shocks
    shocks = cat(3,shocksTVD, repmat(shocksTVD(:,isim),[1,nsimsTVD]));
    ysim_Flev = simulation(pf,P,S,G,V,shocks,ergmean);
    % Simulate: Fixed volatility shocks
    shocks = cat(3,repmat(shocksTVD(:,isim),[1,nsimsTVD]),shocksTVD);
    ysim_Fvol = simulation(pf,P,S,G,V,shocks,ergmean);
    % Integrate: Across volatility shocks
    EysimVol(:, :, isim) = mean(ysim_Flev(2:end, :, :), 3, 'omitnan');
    VysimVol(:, :, isim) = var(ysim_Flev(2:end, :, :), [], 3, 'omitnan');
    % Integrate: Across level shocks
    EysimLev(:, :, isim) = mean(ysim_Fvol(2:end, :, :), 3, 'omitnan');
    VysimLev(:, :, isim) = var(ysim_Fvol(2:end, :, :), [], 3, 'omitnan');
end
% Integrate: Across level shock
TEVol = mean(VysimVol, 3)';
DELev = var(EysimVol, [], 3)';
% Integrate: Across volatility shock
TELev = mean(VysimLev, 3)';
DEVol = var(EysimLev, [], 3)';
```

ESTIMATION EXAMPLE 1: SMM

- Draw shocks, $\mathcal{E}_{T,N_s}$, simulate starting from a random draw from the ergodic distribution, and compute moments
- Run R simulations and compute the average moments
- Estimated parameter values, $\hat{\Theta}$, minimize:

$$J(\Theta, \mathcal{E}) = [\hat{\Psi}_T^D - \bar{\Psi}_{R,T}^M(\Theta, \mathcal{E})]' \hat{\Sigma}_T^D [\hat{\Psi}_T^D - \bar{\Psi}_{R,T}^M(\Theta, \mathcal{E})]$$

$\hat{\Psi}_T^D$: empirical targets based on GMM

$\hat{\Sigma}_T^D$: diagonal inverse of the covariance matrix

- Use `fminsearch` to find the optimal parameter values

```
[Phi,J] = fminsearch('objfunc',draw0,options,...
                    PsiD,OmegaD,O,P,V,...
                    shocks)
```

ESTIMATION EXAMPLE 2: PARTICLE FILTER

1. Initialize the filter by drawing from the ergodic distribution
2. For all particles $p \in \{1, \dots, N_p\}$ apply the following steps:
 - 2.1 Draw $\mathbf{e}_{t,p} \sim \mathcal{N}(\bar{\mathbf{e}}_t, I)$, 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 $t \in \{1, \dots, T\}$. $\log \ell^{model} = \sum_{t=1}^T \log \ell_t^{model}$

PARTICLE ADAPTATION

1. “Macroeconomic Dynamics Near the ZLB: A Tale of Two Countries” (ReStud, 2018)
2. Given $\mathbf{z}_{t-1}$ and a guess for $\bar{\mathbf{e}}_t$, obtain $\mathbf{z}_t$ and $\mathbf{w}_{t,p}$
3. Calculate $\hat{\mathbf{x}}_t^{model}$, such as $[\log(g_t \tilde{y}_t^{gdp} / \tilde{y}_{t-1}^{gdp}), \log(\pi_t), \log(i_t)]$
4. 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 is the diagonal of the ME covariance matrix

5. Use `fminsearch` to find $\bar{\mathbf{e}}_t$ that maximizes

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

INTRODUCTION TO MEX

- Advantages of MATLAB:
 - ▶ Many built-in functions
 - ▶ 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): Lets programmers write sections of their code using a compiled language (e.g., Fortran) and call it as a function in MATLAB
- Intermediate step toward full Fortran implementation
- Requires users must write a “Gateway” function that allows MATLAB to communicate with compiled code

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 that holds the inputs
 - ▶ plhs: Pointer to an array that will hold the outputs
 - ▶ nrhs: number of right-hand inputs
 - ▶ nlhs: number of left-hand outputs
- 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 n_pr
2 mwSize nk,nz
3 real*8, allocatable, dimension(:, :) :: n
4 nk = mxGetN(prhs(1)) !Capital grid
5 nz = mxGetN(prhs(2)) !Productivity grid
6 allocate(n(nk,nz))
7 n_pr = mxGetPr(prhs(3))
8 call mxCopyPtrToReal8(n_pr,n,nk*nz)

```

- Create `pf` (line 3), load the dimensions of `pf` from inputs (lines 4 and 5), and allocate the memory (line 6)
- Grab the address of the `pf` and store in `n_pr` (line 7), and then copy the information to Fortran variable `n` (line 8)

CREATE OUTPUT MATRIX

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

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

- Create a $1 \times ne$ output vector of type double (lines 3-4) and assign the address (line 5)
- Call the 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)
- MATLAB uses the Parallel Computing Toolbox (PCT)
- The PCT requires the following alterations to the code:
 - ▶ Replace `for` loops with `parfor` loops so MATLAB distributes each step in the loop across the 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 global variables and instead use parameter lists and variable arrays as direct inputs into functions

PARALLEL PROCESSING IN FORTRAN

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

```
!$omp parallel default(shared) private(g,s,arg)
!$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,...,arg)
        pfn_up(i1,i2) = arg(1,1)
    end do
end do
!$omp end do
!$omp end parallel
```

MESSAGE PASSING INTERFACE (MPI)

- Initialize and finalize MPI only once:

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

- Processes do not communicate unless ordered:

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

- Need a way to merge results across cores:

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

- Cores may not finish at the same time:

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

PARALLEL PROCESSING WITH MPI

On a given iteration, apply the following:

```
do inode = myid + 1, Gnodes, nprocs
  g(1,1) = pf_n(inode)
  s(1,1) = Gk_gr(inode)
  s(2,1) = Gz_gr(inode)
  call csolve(g,s,...,arg)
  pfn_up(inode) = arg(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)
```

SOFTWARE AND SCHEDULER COMMANDS

- Use SSH Client to connect
 - ▶ (Windows) PuTTY (<http://www.putty.org>)
 - ▶ (MacOS) Built-in terminal
- Use SFTP client to transfer files
 - ▶ (Windows) WinSCP (<http://winscp.net>)
 - ▶ (MacOS) FileZilla (<http://filezilla-project.org>)
- Scheduler commands for SLURM
(<http://slurm.schedmd.com>):
 - ▶ `squeue`: displays all submitted jobs
 - ▶ `sinfo`: display cluster usage by queue type
 - ▶ `scancel`: cancels a running job
 - ▶ `sbatch runscript`: submits job to queue
 - ▶ `source`: read and execute the content of a file

EXAMPLE: RUN SCRIPT

- Maximum wall time is typically 48 hours
- Cluster may have partitions with different node types
- All program output is written to an OUT file

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --out=test/OUT
#SBATCH --partition=normal-32g
#SBATCH --time=1:00:00
#SBATCH --nodes=1
```

```
prun ./test.out
```

EXAMPLE: JOB ARRAY

- Job arrays useful for code that only differs in the seed
- Add to the run script:

```
#SBATCH --array=[tasksids]
prun ./test.out >&
      test/out-' $SLURM_ARRAY_TASK_ID'
```

- Add to the main program:

```
call get_environment_variable
      ("SLURM_ARRAY_TASK_ID",taskidstr)
read(taskidstr,*) taskid
seed = taskid + 123456
call drandinitialize(3,1,seed,1,rs,lrs,info)
```

COMMAND LINE INPUTS

- Pass an input from the command line to the program
- On the run line, add the input(s) after the program name

```
prun ./results.out arg1 arg2
```

- In the main program, load the input(s) (as a string)

```
character(len=:), &  
    allocatable :: arg1,arg2  
call loadinput(1,arg1)  
call loadinput(2,arg2)
```

where `loadinput` is a subroutine (see next slide)

- If needed, convert to a number

```
real(dp) :: num1,num2  
read(arg1,*) num1  
read(arg2,*) num2
```

LOAD INPUT SUBROUTINE

```
subroutine loadinput(iarg,argout)
```

```
! Input
```

```
integer :: iarg
```

```
! Output
```

```
character(len=:), allocatable :: argout
```

```
! Internal
```

```
integer :: arglen
```

```
call get_command_argument(iarg,length=arglen)
```

```
allocate(character(arglen) :: argout)
```

```
call get_command_argument(iarg,value=argout)
```

```
end subroutine loadinput
```

EXAMPLE: BATCH RUN SCRIPT

```
#!/bin/bash
args='0.5 1.0 5.0'
for arg in $args
do
    out='arg' $arg '-out'
    echo $out
    jobname='arg' $arg
    echo $jobname
    /bin/cp test.stub test.txt
    echo "#SBATCH --partition=normal-32g" >> test.txt
    echo "#SBATCH --time=1:00:00" >> test.txt
    echo "#SBATCH --nodes=1" >> test.txt
    echo "prun ./test.out ${arg} >& ${out}" >> test.txt
    sbatch --output=/dev/null
        --job-name=$jobname test.txt
    sleep 1
done
```

ADDITIONAL RESOURCES

- For more information on the solution method see
 - ▶ “Accuracy, Speed and Robustness of Policy Function Iteration” (*Computational Economics*, 2014)
- For more info on nonlinear estimation see
 - ▶ “The Zero Lower Bound and Endogenous Uncertainty” (*Economic Journal*, 2017)
 - ▶ “The Zero Lower Bound and Estimation Accuracy” (*Journal of Monetary Economics*, 2020)
- All of the code is available at:
<http://alexrichterecon.com>
- Contact: alex.richter@dal.frb.org