Macroeconomic Theory - Real Business Cycle Model (RBC)

Darapheak Tin

Research School of Economics,
Australian National University

Sample Teaching Slides I

Table of Contents

RBC (BGP) - Steady State

Setup (1/5)

Aggregate production function $Y_t = A_t K_t^{\alpha} H_t^{1-\alpha}$ where $H_t \equiv h_t \cdot L_t$.

The per capita output is therefore:

$$y_t = A_t\, k_t^{\alpha} h_t^{1-\alpha}, \qquad 0 < \alpha < 1, \quad h_t \in (0,1)$$

Let $A_t = \bar{A}_t \cdot e^{z_t}$ with a growth trend $\bar{A}_{t+1}/\bar{A}_t = 1 + g_A$ and a stationary AR(1) shock $z_t = z_{t-1} + \varepsilon_t,\ \varepsilon_t \sim \mathcal{N}(0, \sigma_\varepsilon^2)$. Population grows at rate n.

On a balanced growth path (BGP), prices are time invariant, meaning $\frac{r_{t+1}}{r_t} = 1$, while h_t is stationary and $z_t = 0$. This implies:

$$\frac{r_{t+1}}{r_t} = \underbrace{\frac{A_{t+1}}{A_t}}_{=(1+g_A)} \underbrace{\left(\frac{k_{t+1}}{k_t}\right)^{\alpha-1}}_{=(1+g)^{\alpha-1}} \underbrace{\left(\frac{h_{t+1}}{h_t}\right)^{1-\alpha}}_{=1} = 1$$

Hence the common BGP growth of k_t, y_t is

$$1 + g \ = \ (1+g_A)^{\frac{1}{1-\alpha}} \qquad \Rightarrow \qquad g \ = \ (1+g_A)^{\frac{1}{1-\alpha}} - 1.$$

Setup (2/5)

From the resource constraint of our representative household:

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

Divide both sides by k_t:

$$\frac{c_t}{k_t} + \underbrace{(1+g) - (1-\delta)}_{\text{constant}} = \underbrace{\frac{y_t}{k_t}}_{\text{constant}}$$

Hence, c_t must grow at the same rate g.

Growth of Aggregates. Aggregate variables scale with population n:

$$\text{Growth of } (K, Y, C) = (1+g)(1+n)$$

Setup (3/5)

Define stationary (trend–adjusted) variables:

$$\tilde{x}_t \equiv \frac{X_t}{\left[(1+g)^t(1+n)^t\right]}, \qquad X \in \{K, Y, C\}$$

In other words, $\tilde{x}_t$ represents a "per effective worker" variable.

For instance, ARC in stationary units is:

$$(1+g)(1+n)\tilde{k}_{t+1} = \tilde{y}_t - \tilde{c}_t + (1-\delta)\tilde{k}_t.$$

But what exactly is $\tilde{y}$?

$$\tilde{y} = \frac{A_t K_t^{\alpha} H_t^{\alpha}}{(1+g)^t(1+n)^t} = \frac{A_t k_t^{\alpha} h_t^{1-\alpha}}{(1+g)^t} = \frac{A_t}{(1+g)^{(1-\alpha)t}} \cdot \left(\frac{k_t}{(1+g)^t}\right)^{\alpha} h_t^{1-\alpha}$$

where $A_t = \underbrace{A_0(1+g_A)^t}_{\equiv \bar{A}_t} e^{z_t} = A_0(1+g)^{(1-\alpha)t} e^{z_t}$.

Let $A_0 \equiv A_{SS}$, this results in:

$$\tilde{y} = A_{SS} e^{z_t} \tilde{k}^{\alpha} h_t^{1-\alpha}$$

Setup (4/5)

With household utility function:

$$u(c_t, l_t) = \frac{\left(c_t^{\gamma} l_t^{1-\gamma}\right)^{1-\sigma}}{1-\sigma} \qquad \text{where } l_t = 1 - h_t$$

and the aggregate resource constraint (per capita term):

$$c_t + (1+n)k_{t+1} = y_t - (1-\delta)k_t$$

We express the dynamic equilibrium equations on BGP as follows:

$$\text{CL:} \quad \frac{1-\gamma}{\gamma} \frac{\tilde{c}_t}{1-h_t} = (1-\alpha)\frac{\tilde{y}_t}{h_t} \tag{1}$$

$$\begin{aligned} \text{EE:} \quad & \tilde{c}_t^{\gamma(1-\sigma)-1}(1-h_t)^{(1-\gamma)(1-\sigma)} \\ & = \beta\, E_t\Big[\big((1+g)\tilde{c}_{t+1}\big)^{\gamma(1-\sigma)-1}(1-h_{t+1})^{(1-\gamma)(1-\sigma)}(r_{t+1}+1-\delta)\Big] \end{aligned} \tag{2}$$

$$\text{ARC:} \quad (1+g)(1+n)\,\tilde{k}_{t+1} = \tilde{y}_t - \tilde{c}_t + (1-\delta)\tilde{k}_t \tag{3}$$

Setup (5/5)

At the (growth-adjusted) steady state with $z_t = 0$ and constants $(\tilde{c}, \tilde{h}, \tilde{k})$:

$$\textbf{CL:} \qquad \frac{1-\gamma}{\gamma}\,\frac{\tilde{c}}{1-h} = (1-\alpha)\,A_{ss}\,\tilde{k}^{\alpha}\,h^{-\alpha}.$$

$$\textbf{EE:} \qquad (1+g)^{1-\gamma(1-\sigma)} = \beta\,(r+1-\delta)$$

$$\textbf{ARC:} \qquad (1+g)(1+n)\,\tilde{k} = A_{ss}\tilde{k}^{\alpha}h^{1-\alpha} - \tilde{c} + (1-\delta)\tilde{k}.$$

Prices (firm FOCs):

$$r = \alpha\,\frac{y}{k} = \alpha\,A_{ss}\tilde{k}^{\alpha-1}h^{1-\alpha}, \qquad w = (1-\alpha)\,\frac{y}{h} = (1-\alpha)\,A_{ss}\tilde{k}^{\alpha}h^{-\alpha}$$

Calibration

Parameter		Value
Discount factor	β	0.93432960048692
Relative risk aversion (1/EIS)	σ	1
Consumption weight	γ	0.468148849
Population growth	n	0
Capital share	α	0.35
Depreciation	δ	0.048080529
TFP level (SS)	A_{ss}	1
Shock persistence	ρ	0.79684266
Shock s.d.	σ_ε	0.011877488
TFP growth	g_A	0
BGP growth (y,k,c)	g	$(1+g_A)^{1/(1-\alpha)} - 1 = 0$

TFP shock parameters are estimates from `aa_RBC_Facts.m`:

- Compute A using Y, K, H data and $\alpha = 0.35$
- Regression: $\log(A_t) = b_1 + b_2 \cdot t + z_t$
- $\hat{z}_t = log(A_t) - \hat{b}_1 - \hat{b}_2 \cdot t$
- Then run regression $\hat{z}_t = \rho \hat{z}_{t-1} + \varepsilon_t$ to obtain $\hat{\rho}$ and $\hat{\varepsilon}_t$
- $\sigma_\varepsilon = \sqrt{\mathbb{V}(\hat{\varepsilon}_t)}$

Steady State - func_SS_eq.m

```
function y = func_SS_eq(x, gamma,sigma,beta,n,A,delta,alpha,g_BGP)
% INPUTS:
%   x     = [c, h, k]';
%   parameters: gamma, sigma...
% OUTPUTS:
%   y = residual vectors [CL; EE; ARC] at the steady state.
% ------------------------------------------------------------

c = x(1);               % consumption
h = x(2);               % labor (work time)
k = x(3);               % capital

y_out = A * k^alpha * h^(1-alpha); % output
w = (1-alpha) * y_out / h;         % wage
r = alpha * y_out / k;             % rental rate

eq1 = ((1-gamma)/gamma) * c / (1-h) - w; % CL
eq2 = (1+g_BGP)^(1 - gamma*(1 - sigma)) - beta * (r + 1 - delta); % EE
eq3 = c + k * (n + g_BGP + n*g_BGP + delta) - y_out; % ARC

y = [eq1; eq2; eq3];

end
```

Steady State - aa_RBC_main.m (1/2)

```
clear; close all;
addpath('./A_tools');

% -------- Calibrated parameters --------
% Preferences
beta  = 0.93432960048692;  % Discount factor
sigma = 1;                 % CES curvature (IES = 1/sigma); sigma=1 log
gamma = 0.468148849;       % Weight on consumption
n     = 0;                 % Population growth

% Technology and shocks
A_ss   = 1;                % TFP level in SS
delta = 0.048080529;       % Depreciation
alpha = 0.35;              % Capital share
rho    = 0.79684266;       % AR(1) persistence
sigma_e= 0.011877488;      % Shock s.d.
g_A    = 0;                % Trend TFP growth
g_BGP = (1+g_A)^(1/(1-alpha)) - 1; % BGP growth of k, y, c
```

Steady State - aa_RBC_main.m (2/2)

```
% -------- Solve steady state --------
% Optimiation options
tol = 1e-12; % Tolerance
options = optimoptions(@fsolve, 'Display', 'none', ...
    'FunctionTolerance', tol, 'OptimalityTolerance', tol, ...
    'StepTolerance', tol);

% Packing up initial guesses and parameters
x0 = [0.4, 0.5, 1]';      % Initial guess: [c, h, k]'
param_ss = {gamma, sigma, beta, n, A_ss, delta, alpha, g_BGP}; % List

% Numerical solution with fsolve
[x, fval_ss] = fsolve(@(x) func_SS_eq(x, param_ss{:}), x0, options);

% Unpack steady-state variables
c_ss = x(1);              % Consumption
h_ss = x(2);              % Hours (worktime)
k_ss = x(3);              % Capital

% Implied quantities and prices
y_ss = A_ss * k_ss^alpha * h_ss^(1-alpha); % Output
w_ss = (1 - alpha) * y_ss / h_ss; % Wage
r_ss = alpha * y_ss / k_ss;      % Rental rate
x_ss = y_ss - c_ss;              % Investment (level)
i_ss = (y_ss - c_ss) / y_ss;     % Investment rate (= saving rate)
```

Steady State - Output

```
% Output table
VarNames1 = {'c_ss','h_ss','k_ss','y_ss','x_ss','i_ss'};
VarNames2 = {'w_ss','r_ss'};
% Build tables and display
SS_Table1 = array2table([c_ss, h_ss, k_ss, y_ss, x_ss, i_ss], ...
    'VariableNames', VarNames1, 'RowNames', {'SteadyState'});
SS_Table2 = array2table([w_ss, r_ss], ...
    'VariableNames', VarNames2, 'RowNames', {'SteadyState'});
disp('--- Steady State: Quantities ----------------------------');
disp(SS_Table1);
disp('--- Steady State: Prices --------------------------------');
disp(SS_Table2);
```

Quantities		**Prices**	
Variable	**Value**	**Variable**	**Value**
c_{ss}	0.61533	w_{ss}	1.16530
h_{ss}	0.40011	r_{ss}	0.11837
k_{ss}	2.12100		
y_{ss}	0.71731		
x_{ss}	0.10198		
i_{ss}	0.14217		

RBC (BGP) - Policy function

A naive approach to DSGE

Goal: Simulate IRFs / stochastic paths when TFP is random.

The challenge:

- c_t, h_t must satisfy CL, EE, and ARC.
- The EE includes an $\mathbb{E}_t[\cdot]$ over next period's shocks z_{t+1} and *future* choices (c_{t+1}, h_{t+1}).
- **Naive approach**: solves a system of nonlinear equations for every possible z realization in every period $\Rightarrow$ slow and numerically fragile.

Solution?

Policy function approximation (1/2)

The idea (Euler projection & policy approximation):

- `func_h_approx.m`: Postulate a smooth rule $h(A, k; \mathbf{b})$

$$
\begin{aligned}
s(A, k;\ b) = {} & b_1 + b_2 \log A + b_3 \log k + b_4 (\log A)^2 + b_5 (\log k)^2 \\
& + b_6 (\log A)(\log k)
\end{aligned}
$$

  where

$$
h(A, k;\ b) \ = \ \frac{1}{1 + \exp\big(- s(A, k;\ b)\big)} \qquad (0 < h < 1)
$$

- `func_EE.m`: Fit $\mathbf{b}$ to *minimize* the Euler residuals (e.g., sum of squared residuals) on a grid of (A, k).
- Compute $\mathbb{E}_t[\cdot]$ deterministically with *Gaussian quadrature* nodes/weights $\Rightarrow$ smooth, fast objective.

Policy function approximation (2/2)

How this pays off in simulation:

1. Given (A_t, k_t), evaluate $h_t = h(A_t, k_t; b^\star)$
2. $A_t = A_{ss}e^{z_t}$ where z_t is drawn from $\mathcal{N} \sim (0, \sigma_\varepsilon^2)$
3. Use CL to get $c_t = \frac{\gamma}{1-\gamma} w_t(1 - h_t)$
4. Update $k_{t+1} = \dfrac{y_t + (1-\delta)k_t - c_t}{(1+g)(1+n)}$

Results: No root-finding each period; paths/IRFs become a computationally inexpensive forward recursion.

Proposed Policy Function – func_h_approx.m

```
function h = func_h_approx(A,k,b)
    % func_h_approx.m : Approximate policy function for labor h(A
        ,k;b)
    % Uses a quadratic in logs, then a logistic map to keep h in
        (0,1).

    % INPUTS:
    %   A, k : TFP and capital (scalars or matrices)
    %   b    : 1x6 vector of coefficients
    % OUTPUT:
    %   h    : policy function for labor

    s = b(1) ...
      + b(2)*log(A) + b(3)*log(k) ...
      + b(4)*log(A).^2 + b(5)*log(k).^2 ...
      + b(6)*log(A).*log(k);

    h = 1 ./ (1 + exp(-s)); % logistic map to enforce 0<h<1
end
```

EE Residuals - func_EE.m (1/2)

```
function RSS = func_EE(b, A, k, gamma, sigma, beta, A0, g_BGP, rho,
     alpha, delta, n, xq, wq)
   % func_EE.m calculates the Residual Sum of Squares from the
   % Euler equation. This function is minimized to estimate the policy-
       function coefficients b.
   % INPUTS:
   %   b = [b1,...,b6]   : parameters of policy function for h (labor)
   %   State variables   : A, k
   %   List of parameters : gamma, sigma...
   %   xq, wq             : quadrature nodes/weights for shocks

   % Period t: endogenous variables
   z = log(A ./ A0);                       % A_t = A0*exp(z_t)
   h = func_h_approx(A, k, b);             % labor h_t
   y = A .* k.^alpha .* h.^(1 - alpha);    % output y_t
   w = (1 - alpha) . *y ./ h;              % wage w_t
   c = (gamma/(1-gamma)) .* w .* (1 - h); % consumption c_t

   % LHS of Euler (marginal utility today)
   LHS = c.^(gamma*(1-sigma)-1) .* (1 - h).^((1-gamma)*(1-sigma));
```

EE Residuals - func_EE.m (2/2)

```
    % Period t+1: endogenous variables
    k1 = (y + (1 - delta).*k - c) ./ ((1 + g_BGP)*(1 + n)); % k_{t+1}
    E  = zeros(size(A)); % initializing conditional expectation

    % Integrate over Normal shocks via quadrature to get E_t
    for i = 1:length(xq)
        z1 = rho .* z + xq(i);                  % z_{t+1} | z_{t}
        A1 = A0 .* exp(z1);                     % A_{t+1}
        h1 = func_h_approx(A1, k1, b);          % h_{t+1}
        y1 = A1 .* k1.^alpha .* h1.^(1 - alpha);% y_{t+1}
        r1 = alpha .* y1 ./ k1;                 % R_{t+1}
        w1 = (1 - alpha).*y1 ./ h1;             % w_{t+1}
        c1 = (gamma/(1-gamma)) .* w1 .* (1 - h1); % c_{t+1} (CL in t+1)
        % Inside expectation: grow-adjusted next-period MU * gross return
        E_inside = (c1*(1 + g_BGP)).^(gamma*(1 - sigma) - 1) ...
                .* (1 - h1).^((1 - gamma)*(1 - sigma)) ...
                .* (r1 + 1 - delta);
        % Accumulate with quad. weights (prob. of xq(i))
        E = E + E_inside .* wq(i);
    end

    RHS = beta .* E;
    EE  = LHS - RHS;                    % pointwise Euler residuals
    RSS = sum(EE.^2,'all');             % objective: residual sum of squares
end
```

EE Residuals - What `func_EE.m` is doing

We choose b so the Euler equation holds *globally* on a grid of states (A, k) under Normal shocks ε_t.

`func_EE.m`: Given b, state (A, k), discretized shock values (ε_i) and corresponding probabilities (w_i), pref. and tech. parameters,

1. Calculate h using the policy $h = h(A, k, b)$
2. Get y, w, r and, from $c = \frac{\gamma}{1-\gamma} w(1-h) \Rightarrow$ **LHS** of EE
3. Calculate $k' = \dfrac{y + (1-\delta)k - c}{(1+g)(1+n)}$
4. For each discretized shock value (quadrature node) ε_i:
   - Get $z_i' = \rho z + \varepsilon_i$ where $z = \log(\frac{A}{A_0})$ (based on $A = A_0 e^z$)
   - Build $A_i' = A_0 e^{z_i'}$, then $h_i' = h(A_i', k'; b)$, and y_i', r_i', w_i', c_i'.
   - Compute $E_t(., z_i'|z) =$ growth-adjusted next $\text{MU}_i \times$ gross return_i.
5. Take the weighted sum (expectation): $E_t(.|z) = \sum_{i=1}^{M} E_t(., z_i'|z) w_i$
6. Multiply $E_t(.|z)$ by β to get **RHS** of EE.
7. Residual: **EE** $=$ **LHS** $-$ **RHS**. Square it and sum across grid $\Rightarrow$ **RSS**(b).

Minimise RSS(b) with `fminunc` $\Rightarrow$ fitted policy $h(A, k; b^\star)$.

EE Residuals - Gaussian Quadrature

On the RHS of our EE, we have a conditional expectation term:

$$\mathbb{E}_t\left[G(\varepsilon_{t+1})|z_t\right] \quad \text{with} \quad \varepsilon_{t+1} \sim \mathcal{N}(0, \sigma_\varepsilon^2)$$

(ε_{t+1} is used to compute $z_{t+1} = \rho z_t + \varepsilon_{t+1}$)

Gaussian Quadrature (via `qnwnorm`) discretizes the normally distributed shock ε by (smartly) selecting nodes and weights $\{x_i, w_i\}_{i=1}^{n_Q}$ such that:

$$\mathbb{E}[G(\varepsilon)] \approx \sum_{i=1}^{n} w_i \, G(x_i)$$

Note

For further detail, see supplementary material in the Appendix (click here).

Policy function approximation - Recap

For large state-space models with stochastic components, there are multiple benefits to the policy function approximation:

- **Speed & stability:** Avoids nonlinear solves at each state $\times$ shock $\times$ time;
- **Accuracy of $E_t[\cdot]$:** Gaussian quadrature replaces random draws with deterministic nodes/weights for $\mathcal{N}(0, \sigma^2)$. This gives a smooth, noise-free objective, achieving accuracy with few nodes.
- **Global fit*:** A single smooth rule $h(A, k;\, b)$ approximately enforces the Euler condition across the state space (not just locally).
- **Feasibility:** Logistic mapping ensures $0 < h < 1$ by construction;
- **Fast and cheap simulation:** Once $b^\star$ is found, IRFs/time series follow by forward iteration of the policy and laws of motion.

Note

(*) "Global" means relative to the **chosen basis and grid**. Accuracy improves with richer bases and denser (finer) state $\times$ shock grids.
(**) "Basis" refers to *a set of simple functions you linearly combine to approximate the true policy function h*. You can enrich the basis by including more terms, higher order, interactions in the $h(A, k, b)$ you postulate. See supplementary material for discussion.

Solving for policy function - aa_RBC_main.m

```
% State space (A, k)
n_A = 11; n_k = 11;                % grid points of A and k
frac_A = 0.1; frac_k = 0.1;        % +/- ranges around SS
A_grid = linspace(A_ss*(1-frac_A), A_ss*(1+frac_A), n_A);
k_grid = linspace(k_ss*(1-frac_k), k_ss*(1+frac_k), n_k);
[A,k] = ndgrid(A_grid, k_grid); % n_A x n_k matrices

% Gaussian quadrature (for E_t[.])
n_q    = 5;                  % Number of nodes for numerical integration
[xq,wq] = qnwnorm(n_q, 0, sigma_e^2); % Quadrature nodes & weights

% Policy functional form (labor)
b0 = [log(h_ss./(1-h_ss)), zeros(1,5)]; %Init guess (h=h_ss across A,k)

% Pack parameters for the Euler residual
param_EE = {A,k, gamma,sigma,beta,A_ss,g_BGP,rho,alpha,delta,n, xq, wq};

% Minimization options
tol = 1e-12;
options = optimoptions(@fminunc, ...
    'Algorithm', 'quasi-newton', ... % 'quasi-newton' or 'trust-region'
    'Display','final', 'OptimalityTolerance',tol, 'StepTolerance',tol);

% Solve for b: minimize sum of squared Euler residuals
[b, RSS] = fminunc('func_EE', b0, options, param_EE{:});
```

RBC (BGP) - Simulation

Simulation - aa_RBC_main.m (1/3)

```
% -------- Experiments: the user needs to choose k_0 --------
% [1] 'IRF' : impulse response (one-time shock at t=1)
% [2] 'TS' : stochastic simulation (shocks every period)
fig_type = 'IRF'; % 'TS'

k0    = k_ss;       % initial capital
T_fig = 101;        % periods shown in figures (0..T_fig-1)
T     = 500;        % total simulated periods (0..T-1)

% Generate the shock process
% iid e_t ~ N(0, sigma_e^2) <-- used only in TS mode
if strcmp(fig_type,'TS')
    rng(1);                          % reproducible shocks
    e_t = sigma_e*randn(T,1);        % innovations
end

% Preallocation
z_t = zeros(T,1);
A_t = zeros(T,1);
h_t = zeros(T,1);
k_t = zeros(T,1); k_t(1) = k0;
y_t = zeros(T,1);
c_t = zeros(T,1);
r_t = zeros(T,1);
w_t = zeros(T,1);
```

Simulation - aa_RBC_main.m (2/3)

```
% -------- Generate time paths of endogenous variables --------
% t=0 at steady state;
% t=1 shock arrives
for i = 1:T
    if strcmp(fig_type,'IRF')    % one-time shock at t = 1
        if i==1
            z_t(i) = sigma_e;     % +1 std TFP shock
        else
            z_t(i) = rho*z_t(i-1); % AR(1) decay
        end
    elseif strcmp(fig_type,'TS') % shocks arrives every period from t>=1
        if i==1
            z_t(i) = e_t(i);
        else
            z_t(i) = rho*z_t(i-1) + e_t(i);
        end
    end

    % TFP, policy, prices, quantities
    A_t(i) = A_ss*exp(z_t(i));
    h_t(i) = func_h_approx(A_t(i), k_t(i), b);
    y_t(i) = A_t(i)*k_t(i)^alpha * h_t(i)^(1-alpha);
    r_t(i) = alpha*y_t(i)/k_t(i);
    w_t(i) = (1-alpha)*y_t(i)/h_t(i);
    c_t(i) = (gamma/(1-gamma))*w_t(i)*(1-h_t(i));
```

Simulation - aa_RBC_main.m (3/3)

```
    % next-period capital (stationary accumulation)
    if i < T
        k_t(i+1) = (y_t(i)+(1-delta)*k_t(i)-c_t(i)) / ((1+g_BGP)*(1+n));
    end
end

x_t = y_t - c_t;  % Investment
i_t = (y_t - c_t) ./ y_t; % Investment rate (= saving rate)

% Deviations from steady state (percent)
A_dev = (A_t - A_ss)/A_ss;
c_dev = (c_t - c_ss)/c_ss;
h_dev = (h_t - h_ss)/h_ss;
k_dev = (k_t - k_ss)/k_ss;
y_dev = (y_t - y_ss)/y_ss;
w_dev = (w_t - w_ss)/w_ss;
r_dev = (r_t - r_ss)/r_ss;
x_dev = (x_t - x_ss)/x_ss;
i_dev = (i_t - i_ss)/i_ss;
```

Simulatoin - IRFs vs. Time Series

Impulse response functions (IRFs) answer: "What happens after a one-off 1σ TFP shock?"

- Deterministic path post-shock (given $b^\star$).
- Useful for propagation and sign/persistence checks.

Stochastic simulations of time series (TS) answer: "What does the economy look like under ongoing uncertainty?"

- Draw shocks every period.
- Can calculate moments (e.g., volatilities, relative vol, correlations) from model-generated series $\Rightarrow$ a key diagnostic of model fit.
- With curvature, long-run means differ from deterministic SS (risk effects).

Plotting IRFs/Time Series (level) - aa_RBC_main.m (1/2)

```
% Figure names
if strcmp(fig_type,'IRF')
    figname1 = 'IRF_levels'; figname2 = 'IRF_dev';
elseif strcmp(fig_type,'TS')
    figname1 = 'TS_levels'; figname2 = 'TS_dev';
end

T_plot = min(T, T_fig);
t  = 0:T_plot-1;         % periods shown on x-axis
ix = 1:T_plot;           % indices into series

%  ---------- Transition Dynamics (level) ----------
figure('Name', figname1);
set(gcf,'Units','inches','Position',[0 0 10 6.67]); % 3:2 aspect

% 1) TFP
subplot(3,3,1);
plot(t, A_t(ix), '-b','LineWidth',1.5); hold on;
yline(A_ss, 'k--','LineWidth',1);
title('TFP $A_t$', 'FontWeight','normal','Interpreter','latex');
xlabel('Periods','FontSize',8); ylabel('Level');
grid on; xlim([t(1) t(end)]);
center_axis(A_t(ix), A_ss);
```

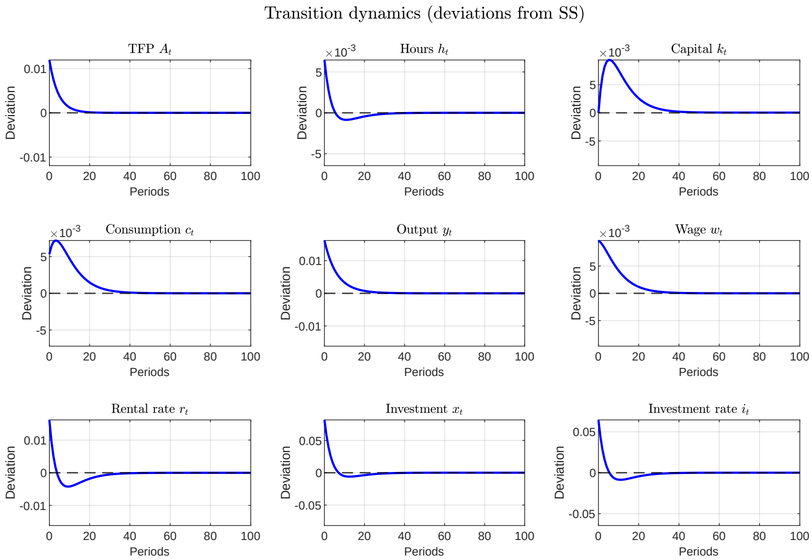
Plotting IRFs/Time Series (level) - aa_RBC_main.m (2/2)

```
...
...
...
% 9) Investment rate
subplot(3,3,9);
plot(t, i_t(ix), '-b','LineWidth',1.5); hold on;
yline(i_ss, 'k--','LineWidth',1);
title('Investment rate $i_t$', 'FontWeight','normal','Interpreter','
    latex');
xlabel('Periods','FontSize',8); ylabel('Level');
grid on; xlim([t(1) t(end)]);
center_axis(i_t(ix), i_ss);

subplot_custom(1.9,1,0.7) %Customizing the plots

% Global title and export
sgtitle('Transition dynamics (levels)', 'FontSize',12, ...
       'FontWeight','bold', 'Interpreter','latex');

% Print-safe sizing for LaTeX inclusion
set(gcf,'PaperUnits','inches');
pos = get(gcf,'Position'); set(gcf,'PaperPosition',[0 0 pos(3) pos(4)]);
set(gcf,'PaperSize',[pos(3) pos(4)]);
print('-dpdf', figname1);
print('-depsc', figname1);
```

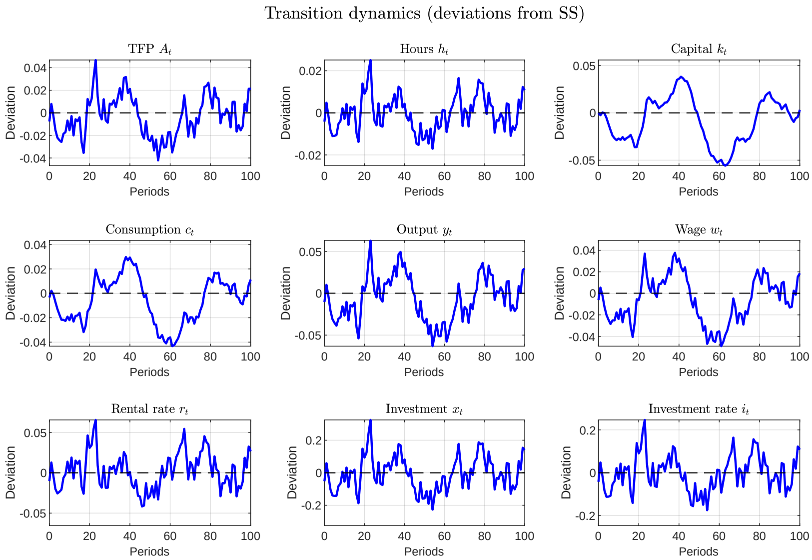
Plotting IRFs/Time Series (dev.) - aa_RBC_main.m

```
% ---------- Transition Dynamics (Deviations from SS) ----------
figure('Name',figname2)
set(gcf,'Units','inches','Position',[0 0 10 6.67]); % 3:2 aspect

% 1) TFP deviation
subplot(3,3,1);
plot(t, A_dev(ix), '-b','LineWidth',1.5); hold on;
yline(0, 'k--','LineWidth',1);
title('TFP $A_t$', 'FontWeight','normal','Interpreter','latex');
xlabel('Periods','FontSize',8); ylabel('Deviation');
grid on; xlim([t(1) t(end)]); center_axis(A_dev(ix), 0);
...
...
...
% 9) Investment rate deviation
subplot(3,3,9);
plot(t, i_dev(ix), '-b','LineWidth',1.5); hold on;
yline(0, 'k--','LineWidth',1);
title('Inv. rate $i_t$', 'FontWeight','normal','Interpreter','latex');
xlabel('Periods','FontSize',8); ylabel('Deviation');
grid on; xlim([t(1) t(end)]); center_axis(i_dev(ix), 0);
...
...
sgtitle('Transition dynamics (deviations from SS)', ...
       'FontSize',12, 'FontWeight','bold','Interpreter','latex');
...
```

IRFs (dev.) - One-off shock



Stochastic Simulations of Time Series (dev.)



Stochastic Simulations - Moments (aa_RBC_main.m)

```
if strcmp(fig_type,'TS')
    Results = [y_dev, c_dev, x_dev, h_dev, w_dev];
    Names = {'Y','C','I','H','w'}; %Variables names

    % Model Moments
    STD_model_absolute = std(Results)';      % 5x1
    STD_model_relative = STD_model_absolute / std(y_dev);
    Correlation_model = corrcoef(Results);
    CorrWithY = Correlation_model(:,1); % correlation with y_dev

    %Putting the above statistics in a table
    TAB_model = table(STD_model_absolute, STD_model_relative, ...
        CorrWithY, ...
        'RowNames', Names, ...
        'VariableNames', {'STD_model_absolute','STD_model_relative','
             CorrWithY'});

    disp('--- Stochastic Simulation: Moments (vs Y) -----------');
    disp(TAB_model);
    writetable(TAB_model, 'TAB_model.csv', 'WriteRowNames', true);
end
```

Stochastic Simulations - Model vs. Data moments

	Std. dev. (abs.)		Std. dev. rel. to Y		Corr. with Y	
Variable	**Model**	**Data**	**Model**	**Data**	**Model**	**Data**
Y	0.03018	0.04043	1.00000	1.00000	1.00000	1.00000
C	0.02151	0.03456	0.71258	0.85475	0.87804	0.89628
I	0.11632	0.07883	3.85390	1.94994	0.84553	0.59861
H	0.00916	0.02899	0.30350	0.71699	0.73944	0.61009
w	0.02421	0.06963	0.80198	1.72241	0.96692	0.93348

What our calibrated RBC model can capture:

- Investment is most volatile, while consumption is relatively smooth.
- All series are procyclical (positive corr. with Y), wages very tightly so.

Note

Model moments are from the simulated RBC economy; data moments are from linearly detrended log series in `aa_RBC_Facts.m`.

Equity Premium - Derivation (1/2)

Euler Equation:

$$u_c(c_t, 1-h_t) = \beta \mathbb{E}_t[u_c(c_{t+1}, 1-h_{t+1})\, R_{t+1}]$$

$$\mathbb{E}_t\left[\underbrace{\beta\,\frac{u_c(c_{t+1}, 1-h_{t+1})}{u_c(c_t, 1-h_t)}}_{\equiv m_{t+1}}\, R_{t+1}\right] = 1$$

Risk-free return: If R_{t+1} is known at t, then let $R^f_{t+1} :=$ risk-free R_{t+1}

$$1 = \mathbb{E}_t\left[m_{t+1} R^f_{t+1}\right] = R^f_{t+1}\, \mathbb{E}_t[m_{t+1}] \;\Rightarrow\; R^f_{t+1} = \frac{1}{\mathbb{E}_t[m_{t+1}]}\,.$$

Equity Premium - Derivation (2/2)

Equity premium: If R_{t+1} is not known at t (distribution is still known)

$$\begin{aligned} \mathbb{E}_t[m_{t+1}R_{t+1}] &= 1 \\ \Rightarrow \mathbb{E}_t[m_{t+1}]\,\mathbb{E}_t[R_{t+1}] + \mathrm{Cov}_t(m_{t+1}, R_{t+1}) &= 1 \\ \Rightarrow \mathbb{E}_t[R_{t+1}] + \frac{\mathrm{Cov}_t(m_{t+1}, R_{t+1})}{\mathbb{E}_t[m_{t+1}]} &= \underbrace{\frac{1}{\mathbb{E}_t[m_{t+1}]}}_{R^f_{t+1}} \end{aligned}$$

Hence,

$$\underbrace{\mathbb{E}_t[R_{t+1}] - R^f_{t+1}}_{\text{Risk Premium}} = -\frac{\mathrm{Cov}_t(m_{t+1}, R_{t+1})}{\mathbb{E}_t[m_{t+1}]}$$

Intuition: $\mathrm{Cov}(m_{t+1}, R_{t+1}) < 0$ implies risky assets $\Rightarrow$ need a premium. This is because R_{t+1} is high when m_{t+1} is low (*good times*), and low when when m_{t+1} is high (*bad times*).

Note:* "Bad times" refer to periods with lower consumption than previous periods. Since $c_{t+1} < c_t$, this results in higher m_{t+1} than otherwise.

Stoch. Simulations - Equity Premium (aa_RBC_main.m)

```
% Risk table
if strcmp(fig_type,'TS')
   %Equity premium calculations
   MUc_t = gamma * c_t.^(gamma*(1-sigma) - 1) .* (1 - h_t).^((1-gamma)
        *(1-sigma)); %Marginal utility from consumption
   M  = beta * MUc_t(2:end) ./ MUc_t(1:end-1); %Stochastic discount
        factor
   Rk = r_t(2:end) + 1 - delta; %gross return on capital

   cov_MR = cov(M, Rk); %2x2 covariance matrix
   cov_MR = cov_MR(1,2);
   EP     = - cov_MR / mean(M); % Equity premium

   % SDF m_{t+1} = beta * MUc_{t+1} / MUc_t;
   % gross capital return R^k_{t+1} = r_{t+1} + 1 - delta
   Risk_Table = table(EP, mean(Rk), mean(M), std(Rk), ...
       'VariableNames', {'EquityPremium','MeanGrossReturn','MeanSDF','
            StdGrossReturn'}, ...
       'RowNames', {'Sample'});

   disp('--- Pricing Statistics ----------------------------------');
   disp(Risk_Table);
   writetable(Risk_Table, 'TAB_risk.csv', 'WriteRowNames', true);
end
```

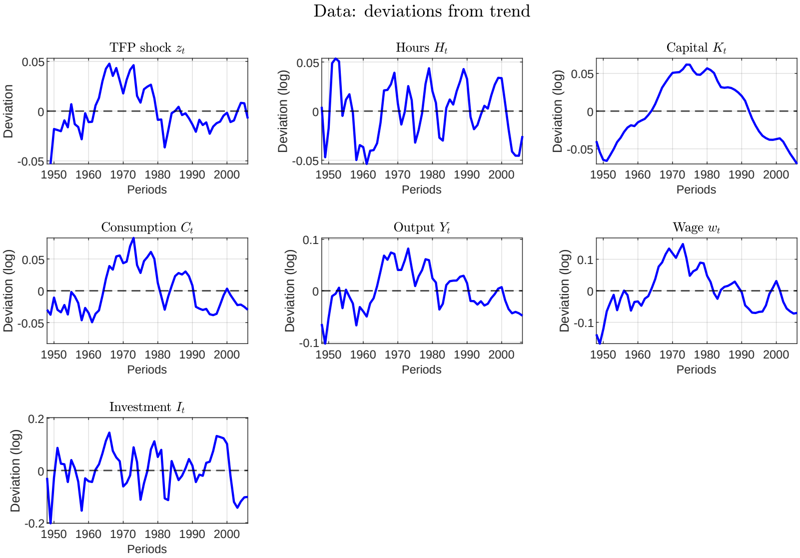
Stochastic Simulations - Equity Premium

	Equity Premium	Mean Gross Return	Mean SDF	Std. Gross Return
Sample	1.4865×10^{-5}	1.0702	0.93438	0.0028772

- In the baseline RBC with σ=1, small shocks, and "MPK-based" return $\Rightarrow$ smooth m_{t+1} and R^k_{t+1} $\Rightarrow$ tiny risk premium.
- To obtain a larger risk premium, we might need more realistic/volatile asset returns (price+dividends), higher risk aversion/curvature, habits, long-run risk, or rare disasters.

RBC (BGP) - RBC vs. Observed Dynamics

Observed dynamics from data



Can the RBC dynamics match the observed dynamics?

Objective: Validate the propagation mechanism of the model (policy rule, CL/ARC/FOCs) from shock identification.

- **Apples-to-apples:** Use the same TFP innovation path as in the data. Differences then reflect the model's internal dynamics, not different shocks.
- **External check:** With the estimated policy $h(A, k; b^{\star})$, the model produces time paths for (h, k, c, y, w, I). Do they comove and persist like the data?

Data-driven RBC simulation using true z_t

Data gives $\{z_t^d\}$ (log TFP deviations). We build the true TFP level series:

$$A_t^d = A_{ss} \exp(z_t^d).$$

After aligning k_0 with k_0^d from data, at each t, we calculate:

1. $h_t = h(A_t^d, k_t; b^\star)$
2. $y_t = A_t^d k_t^\alpha h_t^{1-\alpha}, \quad r_t = \alpha y_t / k_t, \quad w_t = (1-\alpha) y_t / h_t.$
3. $c_t = \dfrac{\gamma}{1-\gamma} w_t (1-h_t).$
4. Accumulation (growth-adjusted):

$$k_{t+1} = \frac{y_t + (1-\delta)k_t - c_t}{(1+g_{\mathrm{BGP}})(1+n)}.$$

5. Convert outputs to log deviations

We then compare the model-generated dynamics with the data dynamics.

Note

The code to estimate $\{z_t^d\}$ is provided in `./Facts/aa_RBC_Facts.m`.

RBC vs. Data Dynamics - aa_RBC_main.m (1/3)

```
% 1) Load detrended data (log deviations) and true z_t
Data_in = readtable('./Facts/Data_residuals.csv');
t_data = Data_in.t(:);
z_data = Data_in.z(:);         % true z_t (log TFP deviation)
H_dev_d = Data_in.H_dev(:);
K_dev_d = Data_in.K_dev(:);
C_dev_d = Data_in.C_dev(:);
Y_dev_d = Data_in.Y_dev(:);
w_dev_d = Data_in.w_dev(:);
I_dev_d = Data_in.I_dev(:);
T = length(z_data);

% 2)  Simulate model with imposed z_t
% Construct TFP levels;
A_t = A_ss .* exp(z_data); % convert log deviations to levels with exp
     (.)

% Choose initial capital alignment
align_k0_with_data = true; % true: match data's initial K deviation
if align_k0_with_data
    k0 = k_ss * exp(K_dev_d(1))
else
    k0 = k_ss
end
```

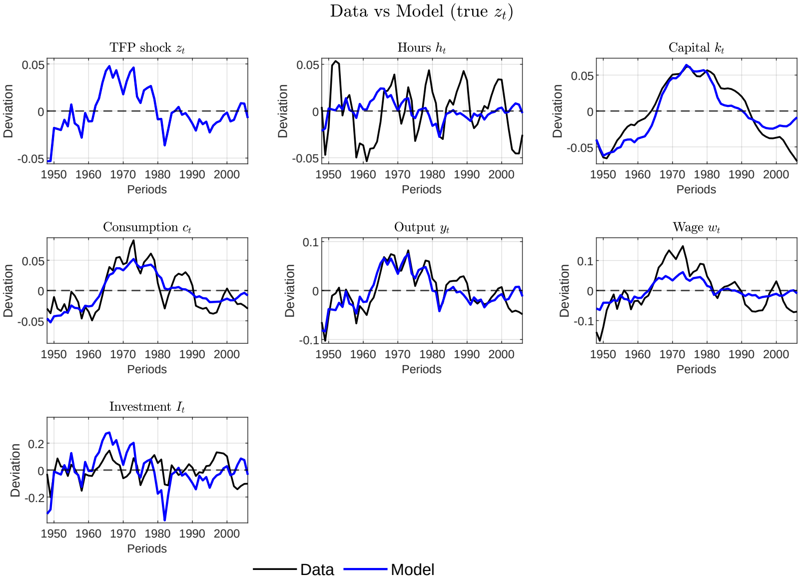
RBC vs. Data Dynamics - aa_RBC_main.m (2/3)

```
% Preallocate
k_t = zeros(T,1); k_t(1) = k0;
h_t = zeros(T,1); y_t = zeros(T,1);
w_t = zeros(T,1); r_t = zeros(T,1);
c_t = zeros(T,1); x_t = zeros(T,1);

for i = 1:T
    % Policy and prices/quantities at t
    h_t(i) = func_h_approx(A_t(i), k_t(i), b);      % policy h(A,k;b*)
    y_t(i) = A_t(i) * k_t(i)^alpha * h_t(i)^(1-alpha); % output
    r_t(i) = alpha * y_t(i) / k_t(i);               % rental rate
    w_t(i) = (1-alpha) * y_t(i) / h_t(i);           % wage
    c_t(i) = (gamma/(1-gamma)) * w_t(i) * (1 - h_t(i)); % CL/LL (log)
    x_t(i) = y_t(i) - c_t(i);                       % investment
    if i < T
        k_t(i+1) = (y_t(i) + (1 - delta)*k_t(i) - c_t(i)) ...
                   / ((1 + g_BGP)*(1 + n));
    end
end

log_dev = @(x, xs) log(x) - log(xs); % Convert outputs to log deviations
z_dev_m = z_data;
H_dev_m = log_dev(h_t, h_ss); K_dev_m = log_dev(k_t, k_ss);
C_dev_m = log_dev(c_t, c_ss); Y_dev_m = log_dev(y_t, y_ss);
w_dev_m = log_dev(w_t, w_ss); I_dev_m = log_dev(x_t, x_ss);
```

RBC vs. Data Dynamics - aa_RBC_main.m (3/3)

```
% Data vs Model
figname_cmp = 'Data_vs_Model_deviations';
T_plot = length(t_data); t = t_data(:)'; ix = 1:T_plot;

figure('Name', figname_cmp);
set(gcf,'Units','inches','Position',[0 0 10 6.67]); % 3:2 aspect
...
...
...
% 3) k_t
subplot(3,3,3); plot(t(ix), K_dev_d(ix), '-k'); hold on;
plot(t(ix), K_dev_m(ix), '-b','LineWidth',1.5); yline(0,'k--',1);
title('Capital $k_t$','FontWeight','normal','Interpreter','latex');
xlabel('Periods','FontSize',8); ylabel('Deviation'); grid on;
xlim([t(ix(1)) t(ix(end))]);
...
...
...
set(gcf,'PaperUnits','inches'); pos = get(gcf,'Position');
set(gcf,'PaperPosition',[0 0 pos(3) pos(4)]); set(gcf,'PaperSize',[pos
     (3) pos(4)]);
print('-dpdf', figname_cmp);
```

RBC vs. Data Dynamics - Not too bad!!!



RBC vs. Data - What works, what doesn't

Strengths

- With imposed z_t, the model tracks y and c comovement and timing.
- Investment is more volatile than output (right sign/order most of the time).
- Wages are procyclical and smoother than Y.

Mismatches (esp. hours)

- Hours h_t: amplitude and phase are completely off (**classic RBC hours puzzle**).
- Wages are smoother than data
- Investment is more volatile and misses data pattern after 2000s

RBC vs. Data - Why the misses?

Measurement & detrending

- Hours: per capita vs per worker; utilization; home production.
- Trend removal: linear vs HP can shift amplitudes/phases (for data).
- Measurement errors in data residuals.

Model (economic)

- Preferences: higher Frisch elasticity (i.e., how hours respond to wages, holding wealth effect constant).
- Technology/margins: investment adjustment costs.
- Shocks beyond TFP (preference; government spending).
- Market structure: wage/price rigidities (moving toward NK features).

Solution/approximation

- Model uses the previously fitted $h(A, k; b^{\star})$; if the basis is too restrictive, dynamics may be biased.
- Check sensitivity to $(\gamma, \sigma, \beta, \delta)$; re-target second moments.

RBC with Taxes (Dynare)

Setup (1/4)

Household problem

$$\max_{\{c_t,\ell_t,k_{t+1}\}} \; E_0 \sum_{t=0}^{\infty} \beta^t \, u(c_t, \ell_t)$$

s.t.

$$(1+\tau_t^c)c_t + k_{t+1} = (1-\tau_t^n)w_t(1-\ell_t) + \Big[1 + (1-\tau_t^k)(q_t - \delta)\Big]k_t + T_t$$

Lagrangian:

$$\mathcal{L} = E_0 \sum_{t\geq 0} \beta^t \Big\{ u(c_t,\ell_t) + \lambda_t \Big[(1-\tau_t^n)w_t(1-\ell_t) + [1+(1-\tau_t^k)(q_t-\delta)]k_t + T_t - (1+\tau_t^c)c_t - k_{t+1} \Big] \Big\}$$

Setup (2/4)

First-order conditions:

$$\frac{\partial \mathcal{L}}{\partial c_t}: \quad u_c(c_t, \ell_t) - \lambda_t(1 + \tau_t^c) = 0$$

$$\frac{\partial \mathcal{L}}{\partial \ell_t}: \quad u_\ell(c_t, \ell_t) - \lambda_t\big(-(1 - \tau_t^n) w_t\big) = 0$$

$$\frac{\partial \mathcal{L}}{\partial k_{t+1}}: \quad -\lambda_t + \beta E_t\Big[\lambda_{t+1}\Big(1 + (1 - \tau_{t+1}^k)(q_{t+1} - \delta)\Big)\Big] = 0$$

Optimal intra-temporal trade-off condition (CL):

$$\frac{u_\ell(c_t, \ell_t)}{u_c(c_t, \ell_t)} = \frac{(1 - \tau_t^n) w_t}{1 + \tau_t^c}$$

Euler equation (EE):

$$\frac{u_c(c_t, \ell_t)}{1 + \tau_t^c} = \beta E_t\left[\frac{u_c(c_{t+1}, \ell_{t+1})}{1 + \tau_{t+1}^c}\Big(1 + (1 - \tau_{t+1}^k)(q_{t+1} - \delta)\Big)\right]$$

Setup (3/4)

Firm:

$$\max_{k_t,n_t} A_t k_t^{\alpha} n_t^{1-\alpha} - q_t k_t - w_t n_t \;\Rightarrow\; q_t = \alpha A_t k_t^{\alpha-1} n_t^{1-\alpha}, \quad w_t = (1-\alpha) A_t k_t^{\alpha} n_t^{-\alpha}.$$

Government:

$$T_t = \tau_t^c c_t + \tau_t^n w_t n_t + \tau_t^k (q_t - \delta) k_t$$

Aggregate resource constraint and law of motion:

$$y_t = c_t + i_t, \qquad i_t = k_{t+1} - (1-\delta) k_t.$$

Read me

This tutorial is very long already, so I made this section brief. Only key equations and the main Dynare `model_tauC_1_v2.mod` file are provided. The rest of the code can be found in `run_all_taxes.m` (main script for execution) and `model_tauC_1_v1.mod`.

Setup (4/4)

Given the household utility function $u(c_t, l_t) = \gamma \log(c_t) + (1-\gamma)\log(\ell_t)$, we arrive at the following system of equations:

$$
\begin{aligned}
\frac{c_t}{1-n_t} &= \frac{(1-\tau_t^n)}{(1+\tau_t^c)} \cdot \frac{\gamma}{(1-\gamma)} w_t \\
\frac{c_{t+1}}{c_t} &= \beta E_t\left[\frac{1+\tau_t^c}{1+\tau_{t+1}^c}\left(1+(1-\tau_{t+1}^k)(q_{t+1}-\delta)\right)\right] \\
y_t &= A_t k_t^{\alpha} n_t^{1-\alpha} \\
k_{t+1} &= i_t + (1-\delta)k_t \\
i_t &= y_t - c_t \\
w_t &= (1-\alpha)A_t k_t^{\alpha} n_t^{-\alpha} \\
q_t &= \alpha A_t k_t^{\alpha-1} n_t^{1-\alpha} \\
T_t &= \tau_t^c c_t + \tau_t^n w_t n_t + \tau_t^k (q_t - \delta)k_t \\
\log(A_{t+1}) &= \rho \log(A_t) + \varepsilon_{t+1}
\end{aligned}
$$

Note:* If you declare only (k, c, n, A) in Dynare, then you can close the system with the CL, EE, ARC, and the AR(1) equation for A; other objects such as y, i, w, q can be computed ex post in MATLAB. However, if you want Dynare to report time paths for y, i, w, q (i.e., store them in `oo_.endo_simul`), you must also declare them as endogenous and include their defining identities/FOCs in the `model` block.

Dynare file - model_tauC_1_v2.mod (1/4)

```
// Macro controls
@#ifndef TAX_CHOICE
  @#define TAX_CHOICE = 1        // 1=tauc, 2=tauk, 3=taun
@#endif

@#ifndef DELTA_TAX
  @#define DELTA_TAX = 0.01      // absolute change to the chosen tax
@#endif

@#ifndef DELAY_LENGTH
  @#define DELAY_LENGTH = 8      // number of periods to delay
@#endif

@#ifndef SIM_HORIZON
  @#define SIM_HORIZON = 60      // perfect-foresight simulation length
@#endif

// 1) Endogenous & exogenous
var Y, C, I, T, K, N, R, W, A;        // endogenous
varexo e tauc taun tauk;              // exogenous (taxes & TFP shock)

// 2) Parameters & calibration
parameters alpha beta delta gamma rho;
alpha = 0.35;  beta  = 0.97;  delta = 0.06;
gamma = 0.40;  rho   = 0.95;
```

Dynare file - model_tauC_1_v2.mod (2/4)

```
// 3) Model equations
model;
    (1+tauc)*C = (gamma/(1-gamma))*(1-N)*(1-taun)*(1-alpha)*Y/N;
    1 = beta * (((1+tauc) * C) / ((1+tauc(+1)) * C(+1))) ...
            * ((1-tauk) * (R(+1)-delta)+1);
    Y = A * (K(-1)^alpha) * (N^(1 - alpha));
    K = I + (1 - delta) * K(-1);
    I = Y - C;
    W = (1 - alpha) * A * (K(-1)^alpha) * (N^(-alpha));
    R = alpha * A * (K(-1)^(alpha - 1)) * (N^(1 - alpha));
    T = tauc * C + taun * W * N + tauk * (R - delta) * K(-1);
    log(A) = rho * log(A(-1)) + e;
end;

// 4) Initial steady state guess
initval;
  A = 1; e = 0;
  tauc = 0.10; tauk = 0.22; taun = 0.35;
  Y = 1; C = 0.8; N = 0.3;
  K = 3.5; I = 0.2;
end;

steady;
resid;
check;
```

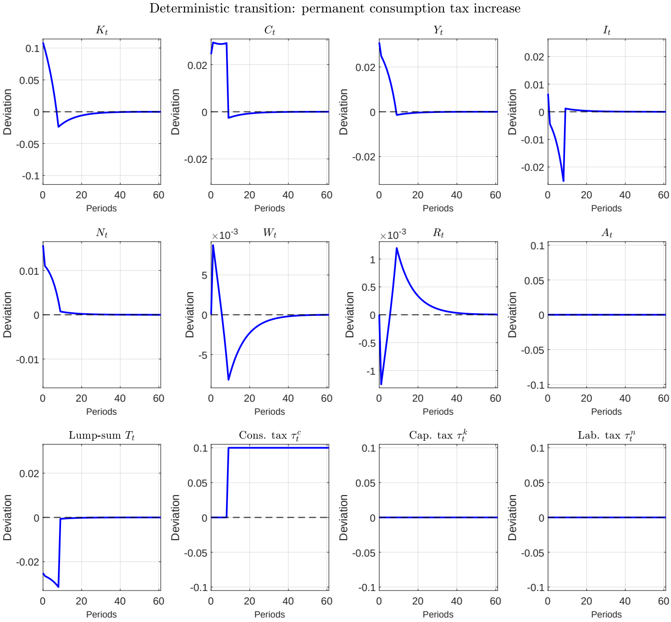
Dynare file - `model_tauC_1_v2.mod` (3/4)

```
// 5) New permanent steady state
endval;
  A = 1;
  e = 0;
  // switch the chosen tax to its NEW_TAX level; others stay at baseline
  @#if TAX_CHOICE==1
    tauc = 0.10 + @{DELTA_TAX};
    tauk = 0.22;
    taun = 0.35;
  @#elseif TAX_CHOICE==2
    tauc = 0.10;
    tauk = 0.22 + @{DELTA_TAX};
    taun = 0.35;
  @#else
    tauc = 0.10;
    tauk = 0.22;
    taun = 0.35 + @{DELTA_TAX};
  @#endif
  Y = 1; C = 0.8; N = 0.3;
  K = 3.5; I = 0.2;
end;

steady;
resid;
check;
```

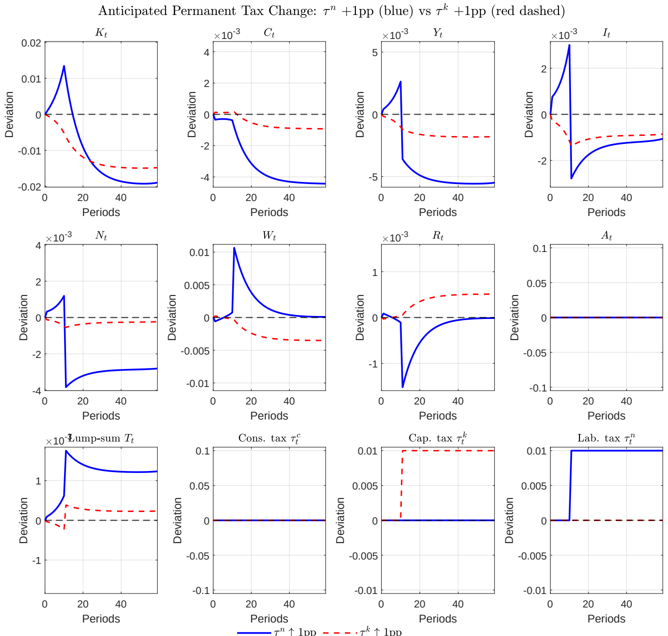
Dynare file - model_tauC_1_v2.mod (4/4)

```
// 6) Shocks
 @#if TAX_CHOICE==1
  shocks;
      var tauc;
      periods 1:@{DELAY_LENGTH};
      values 0.10;
  end;
 @#elseif TAX_CHOICE==2
  shocks;
      var tauk;
      periods 1:@{DELAY_LENGTH};
      values 0.22;
  end;
 @#else
  shocks;
      var taun;
      periods 1:@{DELAY_LENGTH};
      values 0.35;
  end;
 @#endif

// 7) Deterministic transition
simul(periods = @{SIM_HORIZON});
```

Anticipated permanent $\uparrow \tau_c$ - `run_all_taxes.m`



Anticipated τ_c vs. τ_k changes - `run_all_taxes.m`



Supplementary Material

EE Residuals - Gaussian Quadrature (1/2)

On the RHS of our EE, we have a conditional expectation term:

$$\mathbb{E}_t\left[G(\varepsilon_{t+1})|z_t\right] \quad \text{with} \quad \varepsilon_{t+1} \sim \mathcal{N}(0, \sigma_\varepsilon^2)$$

(ε_{t+1} is used to compute $z_{t+1} = \rho z_t + \varepsilon_{t+1}$)

Given z_t and suppress t subscript, its intergral form is:

$$\mathbb{E}_t\big[G(\varepsilon)\big] = \int_{-\infty}^{\infty} G(\varepsilon) \underbrace{\frac{1}{\sqrt{2\pi}\sigma_\varepsilon} e^{-\varepsilon^2/(2\sigma_\varepsilon^2)}}_{f(\varepsilon) \equiv \text{prob. density of } \varepsilon} d\varepsilon.$$

With the change of variable $\varepsilon = \sigma_\varepsilon \sqrt{2}\, u$,

$$\mathbb{E}_t\big[G(\varepsilon)\big] = \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} G(\sigma_\varepsilon \sqrt{2}\, u)\, e^{-u^2}\, du$$

Gaussian Quadrature (via `qnwnorm`) discretizes the normally distributed shock ε by (smartly) selecting nodes and weights $\{x_i, w_i\}_{i=1}^{n_Q}$ such that:

$$\mathbb{E}[G(\varepsilon)] \approx \sum_{i=1}^{n} w_i\, G(x_i)$$

where $x_i = \sigma_\varepsilon \sqrt{2}\, u_i, \quad w_i = \omega_i / \sqrt{\pi}$

EE Residuals - Gaussian Quadrature (2/2)

Brute-force method, which evaluates expectation many times for every *state* $\times$ *shock* $\times$ *optimizer iteration*, is computationally expensive. Gaussian quadrature has the following advantages:

- **Pre-compute & reuse:** Quadrature nodes/weights are calculated once and reused at every grid point and optimizer step.
- **Deterministic & smooth:** No sampling noise $\Rightarrow$ the objective is smooth; gradient-based solvers (e.g. `fminunc`) behave reliably.
- **High accuracy with few nodes:** Gaussian quadrature is exact for polynomials up to degree $2n_q - 1$ and very accurate for smooth G; Few nodes ($n_q \in [5, 9]$) typically suffices (compared to hundreds/thousands for Monte Carlo at similar accuracy).
- **Reproducible & stable:** No Monte Carlo noise, no seed dependence; optimization is not perturbed by randomness.

Further resources: **MathTheBeautiful - Gaussian Quadrature** series: Part 1, Part 2, Part 3.

Back to Main Section

Policy approximation with a basis (1/3)

Goal. Approximate an unknown policy $h(A, k)$ with a parametric family built from simple *basis functions* of the state.

Approximation space. Pick functions $\{\phi_j(A, k)\}_{j=1}^m$ (the *basis*) and parameters $b = (b_1, \ldots, b_m)$, and set

$$s(A, k; b) = \sum_{j=1}^{m} b_j \, \phi_j(A, k), \qquad h(A, k; b) = \frac{1}{1 + e^{-\, s(A,k;b)}} \in (0, 1).$$

We then choose b to minimize Euler residuals on a grid.

Why a basis?

- Builds a *smooth, global* rule in the chosen function space.
- Lets us trade off *flexibility* (more basis terms) vs *stability*.
- Logistic map ensures $0 < h < 1$ (feasibility for labour).

Back to Main Section

Policy approximation with a basis (2/3)

Our baseline: Second-order polynomial in logs:

$$s(A, k; b) = b_1 + b_2 \log A + b_3 \log k + b_4 (\log A)^2 + b_5 (\log k)^2 + b_6 (\log A)(\log k).$$

Policy (logistic mapping):

$$h(A, k; b) = \frac{1}{1 + \exp\big(-s(A, k; b)\big)} \in (0, 1).$$

Pros: simple, fast, captures curvature and $A \times k$ interaction.
Limitations: may be too rigid, leading to accuracy loss far from the SS.

Policy approximation with a basis (3/3)

We can enrich our baseline basis by extend the log-polynomial with cubic and mixed terms. One such example is:

$$\begin{aligned} s(A, k; b) = {} & b_1 + b_2 \log A + b_3 \log k + b_4 (\log A)^2 + b_5 (\log k)^2 + b_6 (\log A)(\log k) \\ & + b_7 (\log A)^3 + b_8 (\log k)^3 + b_9 (\log A)^2 \log k + b_{10} \log A \, (\log k)^2 . \end{aligned}$$

Tips.

- Start from a simple baseline; add terms gradually; monitor Euler residuals (sup/avg) and other feasibility constraints (e.g., $k' \geq 0$).
- You can also improve estimation by *centering and scaling logs*: $\ell_A = \log(A/A_{ss})$, $\ell_k = \log(k/k_{ss})$ before forming powers.
- There are other candidates for basis such as *Chebyshev polynomials* that can greatly improve numerical stability and global accuracy. This is beyond the scope of this tutorial, and I leave it to you to explore this topic further if interested.