Porblem Set 1

1 Integration

I implement the four numerical methods based on the formulas in the lecture notes. For each method, I divided the interval $[0, T]$ into n subintervals, varying n to compare the results. In the Monte Carlo method, I sample a single point within each subinterval and evaluate the integrand at that point. The benchmark ("true value") for comparison is the result obtained using Simpson's rule with $n = 10^8$.

Figures 1 and 2 illustrate the comparisons in computation time and relative errors. Regarding computation time, the Midpoint rule, Trapezoidal rule, and Monte Carlo methods show similar performance, while Simpson's rule is faster for smaller n values ($n = 10$) but becomes slower as n becomes large.

In terms of relative errors, Simpson's rule produces the most accurate result, and this accuracy is particularly noticeable when n gets larger. The Midpoint and Trapezoidal rules are close in accuracy, while the Monte Carlo method performs least accurately overall.

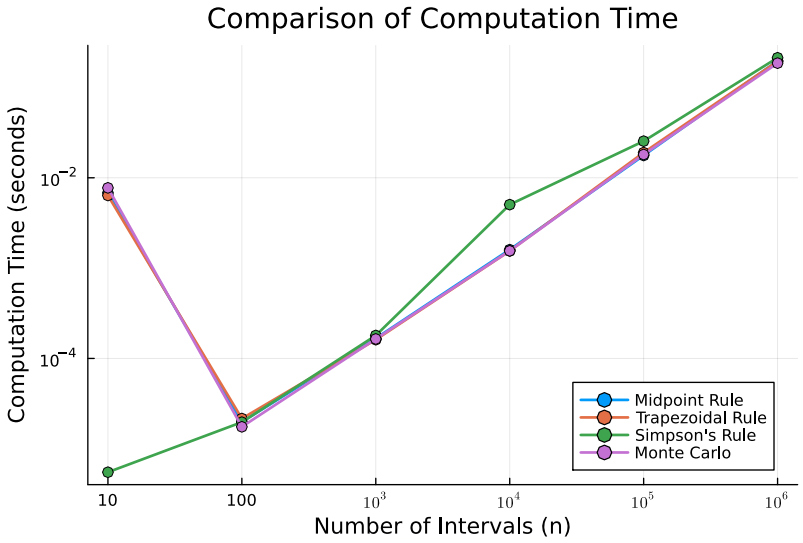

Figure 1: Computation time

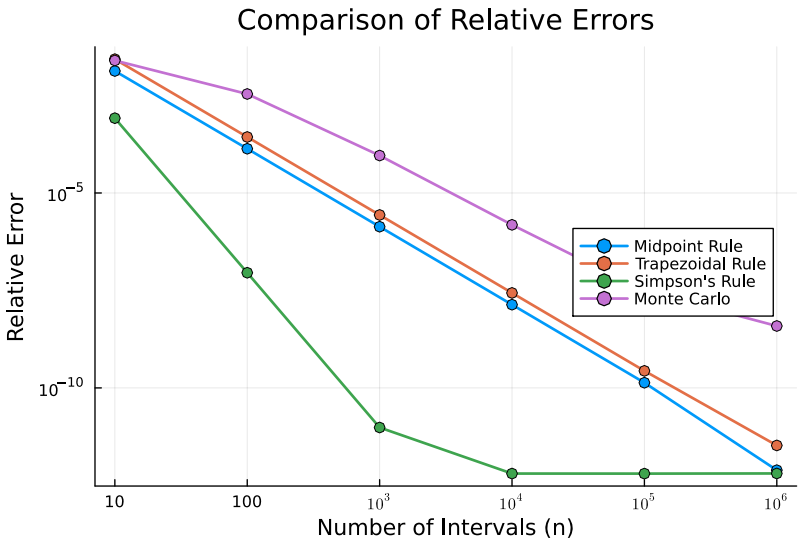

Figure 2: Relative errors

2 Optimization: basic problem

The objective function is the well-known Rosenbrock function. I first compute its gradient and Hessian matrix for later use:

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} -400x(y-x^2) + 2x - 2 \\ 200(y-x^2) \end{bmatrix}$$

$$H(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} -400(y-x^2) + 800x^2 + 2 & -400x \\ -400x & 200 \end{bmatrix}$$

I implement the four optimization methods based on the formulas in the lecture notes. I set the tolerance to 10^{-5} and maximum number of iterations to 10000. The minimizer of this function is $(1,1)$, and I use $(0,0)$ as the initial guess. For both the steepest descent and conjugate gradient methods, I use line search to determine the optimal step size α.

Table 1 presents the absolute bias of the optimization results for each method. The Newton-Raphson method performs best, although it requires computing the Hessian matrix. The BFGS method, which approximates the Hessian using gradient information, ranks second in accuracy. The steepest descent and conjugate gradient methods, relying solely on gradient information, are comparatively less accurate.

Table 2 compares the computation time and iteration count for each of the four methods. As expected, the Newton-Raphson and BFGS methods require fewer iterations and less

Table 1: Absolute Bias

Method	x	y
Newton-Raphson	0	0
BFGS	6.7×10^{-9}	1.39×10^{-8}
Steepest Descent	2.5×10^{-5}	4.9×10^{-5}
Conjugate Gradient	9.7×10^{-6}	1.93×10^{-5}

computation time to reach the minimizer. The steepest descent method, however, reaches the maximum iteration limit without achieving the required tolerance, consistent with its known limitations when applied to the Rosenbrock function. The conjugate gradient method significantly improves on steepest descent, reaching the minimum in only 109 iterations.

Table 2: Computation Time

Method	Time	Iterations
Newton-Raphson	2.5×10^{-9}	3
BFGS	5.2×10^{-5}	49
Steepest Descent	1.3×10^{-2}	10000
Conjugate Gradient	2.1×10^{-4}	109

3 Computing Pareto efficient allocations

The social planner's problem is

$$\begin{aligned} \max_{x_j^i} \quad & \sum_{j=1}^{n} \lambda_j \sum_{i=1}^{m} \alpha_j^i \frac{\left(x_j^i\right)^{1+w_j^i}}{1+w_j^i}, \\ \text{s.t.} \quad & \sum_{j=1}^{n} x_j^i = \sum_{j=1}^{n} e_j^i, \quad \forall i = 1, \ldots, m \end{aligned}$$

where λ_j are individual weights, α_j^i and wj^i are the preference parameters specific to agent j and good i.

Case 1: $m = n = 3$, homogeneous preference parameters and social weights

In this case, I set all α_j^i to 1, w_j^i to -0.5, and λ_j to be 1. For the endowment, I draw e_j^i uniformly from $\{10, 11, \ldots, 20\}$, as shown in Table 3. I use Nelder–Mead method to solve the optimization. The optimal allocation and individual utility levels are reported in Table

4. We can see that each individual's consumption of good i is almost equal, close to (but not exactly as) what theory predicts.

Table 3: Endowment (Cases 1,2)

	Good 1	Good 2	Good 3
Agent 1	13.00	19.00	20.00
Agent 2	16.00	13.00	18.00
Agent 3	12.00	14.00	15.00

Table 4: Allocation and Utility Level (Case 1)

	Good 1	Good 2	Good 3	Utility
Agent 1	13.13	15.44	17.03	23.36
Agent 2	13.93	15.28	19.42	24.09
Agent 3	13.94	15.28	16.55	23.42

Case 2: $m = n = 3$, heterogeneous preference parameters and social weights

In this case, I draw α_j^i uniformly from $\{0.5, 0.6, \ldots, 1.5\}$, w_j^i from $\{-0.6, -0.55, \ldots, -0.2\}$, and λ_j from $\{0.5, 0.6, \ldots, 1.5\}$. λ_j are then reweighted so that the mean weight is 1. The endowments remain the same as in Case 1. Table 5 shows the optimal allocation and individual utility levels. Compared to Table 4, the allocation of each good exhibits greater heterogeneity, reflecting the individuals' diverse preferences for the goods.

Table 5: Allocation and Utility Level (Case 2)

	Good 1	Good 2	Good 3	Utility
Agent 1	29.71	4.67	20.75	28.71
Agent 2	7.71	26.68	18.80	29.49
Agent 3	3.58	14.44	13.43	25.79

Case 3: $m = n = 10$, homogeneous preference parameters and social weights

As in Case 1, I set all α_j^i to 1, w_j^i to -0.5, and λ_j to be 1. Table 6 shows the utility of agents when m and n equal 10. The utility levels of individuals are similar but not identical, and they are less similar compared to Case 1. This suggests that when m and n get larger, the method performs worse.

Case 4: $m = n = 10$, heterogeneous preference parameters and social weights I randomly draw α_j^i, w_j^i, λ_j, and λ_j as in Case 2. Compared with 6, the utility of each individual

Table 6: Utility Level (Case 3)

Agent	Utility	Agent	Utility
1	78.97	6	80.45
2	79.40	7	74.35
3	76.19	8	76.99
4	76.53	9	80.57
5	79.26	10	78.54

shows greater heterogeneity. This increased variation results not only from the heterogeneous weights assigned by the social planner but also from differences in individual utility parameters.

Table 7: Utility Level (Case 4)

Agent	Utility	Agent	Utility
1	88.25	6	117.01
2	84.03	7	81.08
3	87.15	8	84.55
4	93.71	9	94.72
5	80.58	10	85.97

4 Computing Equilibrium allocations

Agent j's problem is: given prices p^i, $i = 1, \ldots, m$,

$$\begin{aligned} \max_{x_j^i} \quad & \sum_{i=1}^{m} \alpha_j^i \frac{\left(x_j^i\right)^{1+w_j^i}}{1+w_j^i}, \\ \text{s.t.} \quad & \sum_{i=1}^{m} p^i x_j^i = \sum_{i=1}^{m} p^i e_j^i \end{aligned}$$

To solve the agent's problem, we write the Lagrangian for agent j with multiplier μ_j for the budget constraint:

$$\mathcal{L}_j = \sum_{i=1}^{m} \alpha_j^i \frac{(x_j^i)^{1+\omega_j^i}}{1+\omega_j^i} - \mu_j \left(\sum_{i=1}^{m} p^i x_j^i - \sum_{i=1}^{m} p^i e_j^i \right).$$

The first-order condition with respect to each x_j^i is:

$$\frac{\partial \mathcal{L}_j}{\partial x_j^i} = \alpha_j^i (x_j^i)^{\omega_j^i} - \mu_j p^i = 0.$$

Rearranging, we get:

$$x_j^i = \left(\frac{\mu_j p^i}{\alpha_j^i} \right)^{\frac{1}{\omega_j^i}}.$$

We can then represent the budget constraint of agent j as:

$$\sum_{i=1}^{m} p^i \left(\frac{\mu_j p^i}{\alpha_j^i} \right)^{\frac{1}{\omega_j^i}} = \sum_{i=1}^{m} p^i e_j^i.$$

In equilibrium, the total demand for each good must equal the total endowment for that good. Therefore, for each good i:

$$\sum_{j=1}^{n} x_j^i = \sum_{j=1}^{n} e_j^i.$$

Substituting the expression for x_j^i from the FOCs, we get:

$$\sum_{j=1}^{n} \left(\frac{\mu_j p^i}{\alpha_j^i} \right)^{\frac{1}{\omega_j^i}} = \sum_{j=1}^{n} e_j^i.$$

Along with n budget constraints, this forms a system of $m + n$ nonlinear equations in the prices $p = (p^1, p^2, \ldots, p^m)$ and multipliers $\mu_1, \ldots, \mu_n$, which we can solve numerically. In fact, one equation is redundant, so we have to normalize $p^1 = 1$.

In computation, I set $n = m = 10$ and draw utility parameters as described in Case 4 of the social planner's problem. I use NLsolve package in Julia to solve the equations. The equilibrium prices are shown in Table 8.

Table 8: Equilibrium Prices

Good	Price	Good	Price
1	1.00	6	2.39
2	1.17	7	1.73
3	1.90	8	1.65
4	1.74	9	1.43
5	2.00	10	1.36

5 Value Function Iteration

5.1 Social Planner

The social planner's problem is

$$V_{sp}(I_-, K, z) = \max_{C,G,L,I,K'} \ln C + 0.2 \ln G - \frac{1}{2}L^2 + \beta \sum_{z'} \pi^z(z'|z) V_{sp}(I, K', z'),$$

$$\text{s.t.} \quad C + I + G = e^z K^\alpha L^{1-\alpha}$$

$$K' = (1-\delta)K + \left[1 - \psi\left(\frac{I}{I_-} - 1\right)^2\right] I$$

where $\beta = 0.97$, $\alpha = 0.33$, $\delta = 0.1$, and $\psi = 0.05$.

5.2 Steady State

We normalize the steady state capital $K = 1$. Then the steady state variables (labor L, output Y, consumption C, investment I, government spending G, wage w, and rental rate r) can be solved from the following equations.

$$I = (1-\delta)K$$

$$w = (1-\alpha)K^\alpha L^{-\alpha}$$

$$r = \alpha K^{\alpha-1} L^{1-\alpha}$$

$$Y = K^\alpha L^{1-\alpha}$$

$$G = \tau_{ss} w L$$

$$C + I + G = Y$$

$$L = \frac{(1-\tau)w}{C}$$

The computed steady state is summarized in Table 9.

5.3 VFI with a fixed grid

Recursive competitive equilibrium

Table 9: Steady State

Variable	Value	Variable	Value
K	1.00	I	0.10
L	0.84	G	0.15
Y	0.89	w	0.71
C	0.64	r	0.29

The RCE are policy functions $g_C(\tau, z, i_-, k, I_-, K)$, $g_L(\tau, z, i_-, k, I_-, K)$, $g_{K'}(\tau, z, i_-, k, I_-, K)$, $g_I(\tau, z, i_-, k, I_-, K)$, pricing functions $w(\tau, z, I_-, K)$, $r(\tau, z, I_-, K)$, $G(\tau, z, I_-, K)$, and law of motion $H_{K'}(\tau, z, I_-, K)$, $H_I(\tau, z, I_-, K)$, such that

(1) Household's problem: Given pricing function w, r, and G, policy functions g_C, g_L, $g_{K'}$, and g_I solve

$$V(\tau, z, i_-, k, I_-, K) = \max_{C,L,K',I} \ln C + 0.2 \ln G - \frac{1}{2}L^2 + \beta \sum_{z'} \sum_{\tau'} \pi^z(z'|z)\pi^\tau(\tau'|\tau)V(\tau', z', i_-, k, I, K')$$

$$\begin{aligned} s.t. \quad & C + I = (1-\tau)w(\tau, z, I_-, K)L + r(\tau, z, I_-, K)K \\ & G = G(\tau, z, I_-, K) \\ & K' = H_{K'}(\tau, z, I_-, K) \\ & I = H_I(\tau, z, I_-, K) \end{aligned}$$

(2) Wage and return to capital equal marginal productivities:

$$\begin{aligned} w(\tau, z, I_-, K) &= e^z \cdot (1-\alpha)K^\alpha g_L(\tau, z, I_-, K)^{-\alpha} \\ r(\tau, z, I_-, K) &= e^z \cdot \alpha K^{\alpha-1} g_L(\tau, z, I_-, K)^{1-\alpha} \end{aligned}$$

(3) Government budget constraint:

$$G(\tau, z, I_-, K) = \tau w(\tau, z, I_-, K) g_L(\tau, z, I_-, K, I_-, K)$$

(4) Resource constraint:

$$g_c(\tau, z, I_-, K, I_-, K) + g_I(\tau, z, I_-, K, I_-, K) + G(\tau, z, I_-, K) = e^z K^\alpha L(\tau, z, I_-, K)^{1-\alpha}$$

(5) Consistency conditions:

$$H_I(\tau, z, I_-, K) = g_I(\tau, z, I_-, K, I_-, K)$$
$$H_{K'}(\tau, z, I_-, K) = g_{K'}(\tau, z, I_-, K, I_-, K)$$
$$H_{K'}(\tau, z, I_-, K) = \delta K + \left[1 - \psi\left(\frac{H_I(\tau, z, I_-, K)}{I_-} - 1\right)^2\right] H_I(\tau, z, I_-, K)$$

The ideal way to solve the VFI

Based on this definition of RCE, there are six state variables: aggregate states τ,z, I_-, K, and individual states i_-, k. To solve the equilibrium, we start with a guess of pricing functions $w^0(\tau, z, I_-, K)$, $r^0(\tau, z, I_-, K)$, $G^0(\tau, z, I_-, K)$, and laws of motion $H_I^0(\tau, z, I_-, K)$ and $H_{K'}^0(\tau, z, I_-, K)$. Given the guesses, we can solve the household's problem by VFI given the guesses of pricing functions. Based on the obtained policy functions $g_L^0(\tau, z, I_-, K)$, $g_I^0(\tau, z, I_-, K)$, and $g_{K'}^0(\tau, z, I_-, K)$, we can update the pricing functions as $w^1(\tau, z, I_-, K)$, $r^1(\tau, z, I_-, K)$, $G^1(\tau, z, I_-, K)$, and laws of motion as $H_I^1(\tau, z, I_-, K)$ and $H_{K'}^1(\tau, z, I_-, K)$. We repeat the above process until the pricing functions and laws of motion converge.

My (failed) approach

The ideal approach requires us to use six state variables and guess five four-dimensional functions, which is impossible to compute when the grid size is large. To simplify the problem, my naive assumption was that we can get rid of individual states if we write the representative household problem as follow (which is wrong): given $w(\tau, z, I_-, K)$ and $r(\tau, z, I_-, K)$, policy functions g_C, g_L, g_I, and $g_{K'}$ solve

$$V(\tau, z, I_-, K) = \max_{C,L,K',I} (1-\beta)(\ln C + 0.2 \ln G - \frac{1}{2}L^2) + \beta \sum_{z'} \sum_{\tau'} \pi^z(z'|z)\pi^\tau(\tau'|\tau) V(\tau', z', I, K')$$
$$s.t. \quad C + I = (1-\tau)w(\tau, z, I_-, K)L + r(\tau, z, I_-, K)K$$
$$K' = \delta K + \left[1 - \psi\left(\frac{I}{I_-} - 1\right)^2\right] I$$
$$G = \tau w(\tau, z, I_-, K)L$$

Here I rescale the flow utility function by $1 - \beta$, as suggested in the lecture.

As before, I start with a guess $w^0(\tau, z, I_-, K)$ and $r^0(\tau, z, I_-, K)$, and then solve the household problem given $w^0(\tau, z, I_-, K)$ and $r^0(\tau, z, I_-, K)$. Based on the obtained labor policy function $g_L^0(\tau, z, I_-, K)$, we can update the pricing functions as $w^1(\tau, z, I_-, K)$ and $r^1(\tau, z, I_-, K)$. We repeat the above process until the pricing functions converge.

When solving the household problem, one can first solve the static labor decision problem:

$$L = \frac{(1-\tau)w(\tau, z, I_-, K)}{(1-\tau)w(\tau, z, I_-, K)L + r(\tau, z, I_-, K)K - I} + 0.2\frac{1}{L}.$$

We can thus represent the optimal L as a function of investment decision I and current states, denoted by $L^*(I, \tau, z, I_-, K)$. We can also substitute C and K' as a function of I based on the constraints, then the number of maximizers in the rep household's problem is reduced to 1 (only investment decision). Formally, we are solving

$$\begin{aligned} V(\tau, z, I_-, K) = \max_I (1-\beta)\{&\ln[(1-\tau)w(\tau, z, I_-, K)L^*(I, \tau, z, I_-, K) + r(\tau, z, I_-, K)K - I] \\ &+ 0.2\ln[\tau w(\tau, z, I_-, K)L^*(I, \tau, z, I_-, K)] - \frac{1}{2}L^*(I, \tau, z, I_-, K)^2\} \\ &+ \beta \sum_{z'}\sum_{\tau'} \pi^z(z'|z)\pi^\tau(\tau'|\tau) V\left(\tau', z', I, \delta K + \left[1 - \psi\left(\frac{I}{I_-} - 1\right)^2\right] I\right). \end{aligned}$$

Several details when doing the VFI

Discretization: τ and z follow the discrete Markov chain described in the problem. Grid points of I_- are equally spaced in $[0.5I_{ss}, 1.5I_{ss}]$ (50 points), while grid points of K are equally spaced in $[0.7I_{ss}, 1.3I_{ss}]$ (250 points).

Interpolation: when we are doing VFI, we are essentially updating the value function at $3 \times 5 \times 50 \times 250$ grid points. After each update, I do linear interpolation on matrix $[V(\tau, z, :, :)]$ for each τ and z and use it in the next iteration.

Maximization: I use the Nelder-Mead method in Julia to search for the optimal I and restrict it to lie within $[0.5I_{ss}, 1.5I_{ss}]$.

Initial guess: I use the steady state life-time utility $V(\tau, z, I_-, K) = \ln C_{ss} + 0.2\ln G_{ss} - 0.5L_{ss}^2$ as an initial guess. For the pricing functions, I use $w(\tau, z, I_-, K) = w_{ss}$ and $r(\tau, z, I_-, K) = r_{ss}$ as an initial guesses.

Impulse response functions: To get the IRF, I first find the corresponding AR(1) process by moment-matching: suppose the AR(1) process for z is $z_t = \mu_z + \rho_z z_{t-1} + \sigma_z \epsilon_t$, where

$\epsilon_t \sim N(0,1)$. Then we can back out μ_z by matching the first moment, σ_z by matching the second moment, and ρ_z by matching the correlation $Corr(z_t, z_{t-1})$. Similar for τ. Then we can simulate the economy using the obtained policy functions and z(τ)-path: $\{z_{ss}+\sigma_z, z_{ss}+\rho\sigma_z, z_{ss}+\rho^2\sigma_z, \ldots, z_{ss}+\rho^{T-1}\sigma_z\}$. The (scaled) IRF of variable Y is then calculated by $\{(Y(t)-Y_{ss})/Y_{ss}, t=1,\ldots,T-1\}$.

Results

As shown in Figure 3, it took 739.4 seconds to solve the equilibrium: 4 iterations to find the pricing functions with tolerance 10^{-3}, and 191 iterations to find the value functions with tolerance 10^{-6}.

```
Value Function Iteration 189
Value function error: 1.0552017977172312e-6
Value Function Iteration 190
Value function error: 1.0220801184290806e-6
Value Function Iteration 191
Value function error: 9.900132492690616e-7
Value function converged.
Pricing function errors: w_error = 0.0003963707392519167, r_error = 0.0001621456246821973
Pricing functions converged.
Computation completed.
Time taken for 6.3: 739.415952 seconds
```

Figure 3: Computation time: VFI with fixed grids

The IRFs to a std of tax shock and TFP shock are shown in Figure 4 and Figure 5, separately. We see that the economy converges to a new steady state. This is a totally wrong result. The underlying reason is that when we drop the individual states, the rep household can actually pick K' and I to "manipulate" next periods' prices $w(\tau', z', I, K')$ and $r(\tau', z', I, K')$. This contradicts the definition of decentralized economy equilibrium. So the prices will end up somewhere else instead of the original steady state.

5.4 Switching between Policy and Value Function Iteration

I implement the method in the following way. Denote the number of iteration to be ITER. If mod(ITER, 10) == 1, I maximize over I to solve the value function iteration:

$$\begin{aligned} V^{(\text{ITER})}(\tau, z, I_-, K) = \max_I\ & (1-\beta)\{\ln[(1-\tau)w(\tau,z,I_-,K)L^*(I,\tau,z,I_-,K) + r(\tau,z,I_-,K)K - I] \\ & + 0.2\ln[\tau w(\tau,z,I_-,K)L^*(I,\tau,z,I_-,K)] - \frac{1}{2}L^*(I,\tau,z,I_-,K)^2\} \\ & + \beta\sum_{z'}\sum_{\tau'}\pi^z(z'|z)\pi^\tau(\tau'|\tau)V^{(\text{ITER}-1)}\left(\tau', z', I, \delta K + \left[1-\psi\left(\frac{I}{I_-}-1\right)^2\right]I\right). \end{aligned}$$

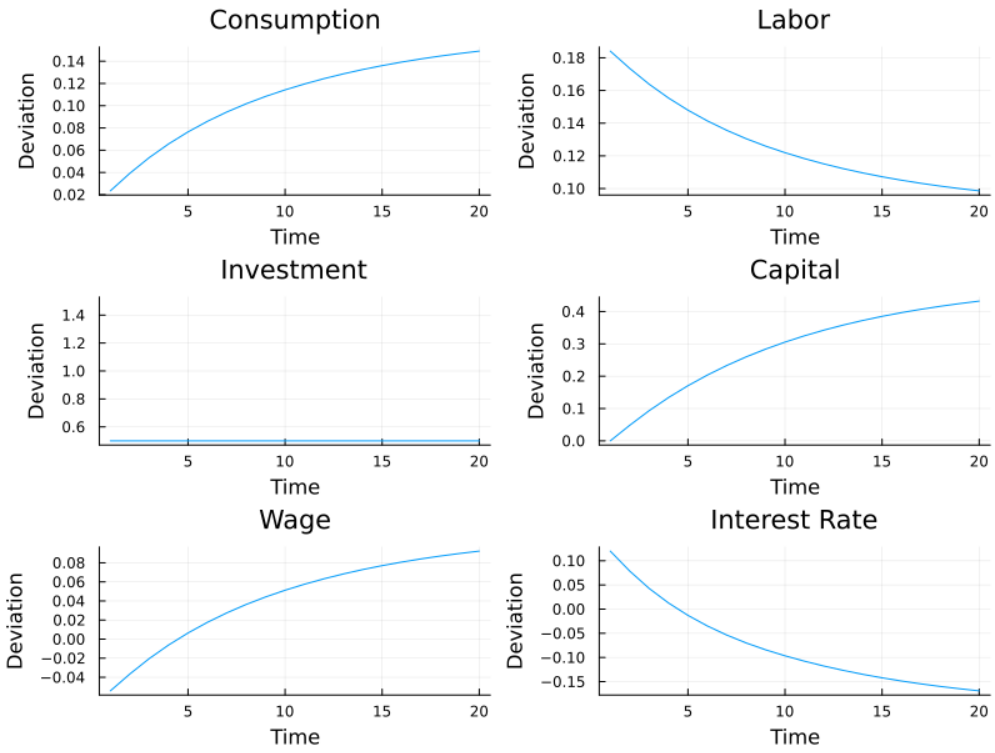

Figure 4: IRFs to tax shock

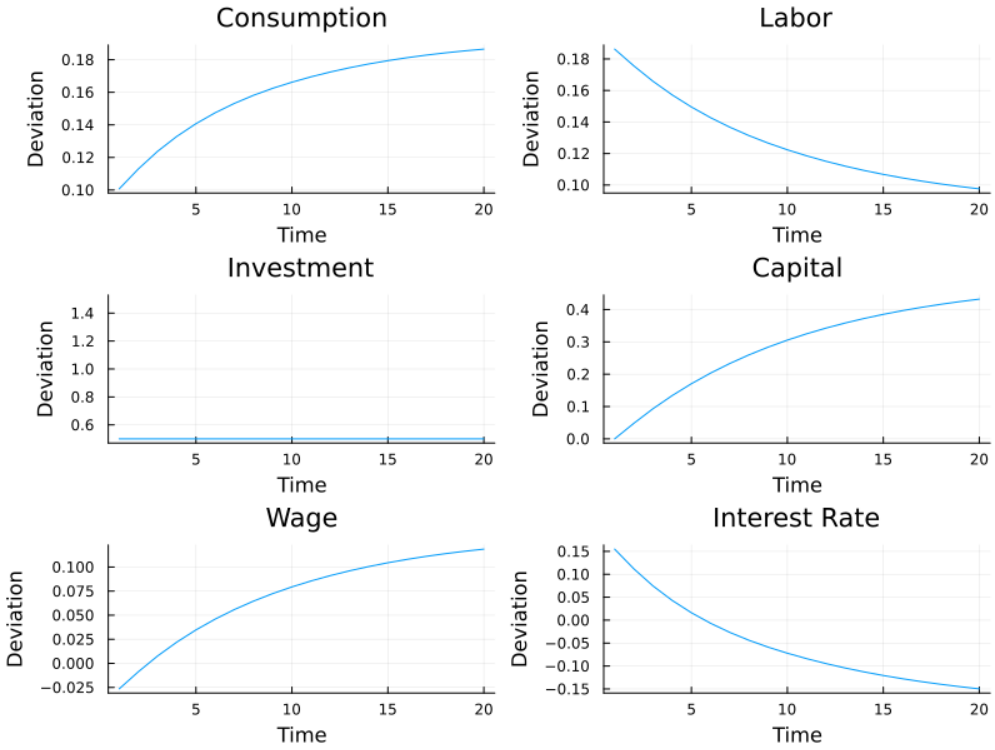

Figure 5: IRFs to TFP shock

Otherwise, I skip the maximization step and only update the value function:

$$
\begin{aligned}
V^{(\mathrm{ITER})}(\tau, z, I_-, K) = (1-\beta)\{&\ln[(1-\tau)w(\tau, z, I_-, K)L^*(I, \tau, z, I_-, K) + r(\tau, z, I_-, K)K - I] \\
&+ 0.2\ln[\tau w(\tau, z, I_-, K)L^*(I, \tau, z, I_-, K)] - \frac{1}{2}L^*(I, \tau, z, I_-, K)^2\} \\
&+ \beta \sum_{z'} \sum_{\tau'} \pi^z(z'|z)\pi^\tau(\tau'|\tau) V^{(\mathrm{ITER}-1)}\left(\tau', z', I, \delta K + \left[1 - \psi\left(\frac{I}{I_-} - 1\right)^2\right] I\right).
\end{aligned}
$$

As shown in Figure 6, it took 348.3 seconds to solve the equilibrium: 4 iterations to find the pricing functions with tolerance 10^{-3}, and 227 iterations to find the value functions with tolerance 10^{-6}. In this case, the convergence is achieved with shorter time and more iterations.

```
Value Function Iteration 226
Value function error: 1.0120019625636 8e-6
Value Function Iteration 227
Value function error: 9.816413841789995e-7
Value function converged.
Pricing function errors: w_error = 0.0003963707392519167, r_error = 0.0001621456246821973
Pricing functions converged.
Computation completed.
Time taken for 6.6: 348.3116612 seconds
```

Figure 6: Computation time: Switching between PFI and VFI

5.5 Multigrid

As described in the question, I first do VFI using 100 capital grid points. Denote the solved value function as $V_{(100)}(\tau, z, I_-, K)$. Then I use the interpolated $V_{(100)}(\tau, z, I_-, K)$ as a initial guess for $V_{(500)}(\tau, z, I_-, K)$, and then solve the value function with 500 grid points by VFI. Similar for solving the value function with 5000 grid points.

As shown in Figures 7 - 9, It took 380.0 seconds to solve the first VFI, 125.1 to solve the second, and 1914.5 to solve the final one.

```
Main Iteration 5
Value Function Iteration 1
Value function error: 6.600272906209792e-5
Value function converged.
Pricing function errors: w_error = 0.0003963707392519167, r_error = 0.0001621456246821973
Pricing functions converged.
Computation completed.
Time taken for 6.7 step 1: 379.957646 seconds
```

Figure 7: Computation time: Multigrid, VFI 100 grids

```
Value Function Iteration 19
Value function error: 1.0723333760287446e-5
Value Function Iteration 20
Value function error: 9.670817769924511e-6
Value function converged.
Pricing function errors: w_error = 0.0003963707392519167, r_error = 0.0001621456246821973
Pricing functions converged.
Computation completed.
Time taken for 6.7 step 2: 125.100584 seconds
```

Figure 8: Computation time: Multigrid, VFI 500 grids

```
Value Function Iteration 21
Value function error: 9.147241288864905e-7
Value function converged.
Pricing function errors: w_error = 0.0003963707392519167, r_error = 0.0001621456246821973
Pricing functions converged.
Computation completed.
Time taken for 6.7 step 3: 1409.4054684 seconds
Time taken for 6.7 in total: 1914.4636983999999 seconds
"6.8 Stochastic grid\n"
```

Figure 9: Computation time: Multigrid, VFI 5000 grids

5.6 Stochastic Grid

To implement the stochastic grid idea proposed by Rust (1997), I randomly draw the capital grid points from a truncated normal distribution: $N(K_{ss}, 0.01K_{ss})$ truncated at $0.75K_{ss}$ and $1.25K_{ss}$, for each iteration. Unfortunately, I couldn't make the value function converge within tolerance 10^{-6}.