Numerical Methods | **Problem Set 2**
Professor: Cezar Santos | **March 17, 2023**
TA: Rafael Vetromille | **Student:** Luan Borelli

Preliminary comments. The solution codes for this problem set are separated into three files, one for each method. Although different methods, all three codes have a similar structure for constructing and solving the system of equations commonly involved in projection methods. This structure can be summarized in three functions:

`basis(·)`: The basis function, $\psi_i(k)$, to be used to approximate the consumption policy function. The format of this function will depend on the method under consideration. E.g., in the case of Exercise 1 (Chebyshev collocation), the basis functions will be given by Chebyshev polynomials, while in the case of Exercise 2 (Finite Element Methods), they will be given by piecewise-linear bases.

`c_hat(`k`, `z`, `γ`)`: The "approximate" consumption policy function, given by

$$\hat{c}(k, z; \gamma) = \sum_{i=1}^{n} \gamma_{i,z} \psi_i(k), \text{ where } \gamma = (\gamma_{1,z}, \cdots, \gamma_{n,z}), \quad \forall z \in Z_{\text{grid}}. \tag{1}$$

`R(`k`, `z`, `γ`)`: The residuals function, given by

$$R(k, z, \gamma) = \hat{c}(k, z, \gamma)^{-\mu} - \beta \mathbb{E}_{z'}[\hat{c}(k'(k, z), z', \gamma)^{-\mu}(\alpha z' k'(k, z)^{\alpha-1} + 1 - \delta)], \tag{2}$$

where $k'(k, z) = zk^{\alpha} + (1 - \delta)k - \hat{c}(k, z, \gamma)$ and $\hat{c}(\cdot)$ is defined as in (1).

`system(`γ`)`: A function representing the system of equations to be solved,[1] whose format will depend on the method under consideration.

Notice that, in practice, the γ can either be a column vector or a matrix, depending on how the above functions are structured and programmed. In the case of a column vector, we have one system for each state, so in total we solve #S problems, one for each state *separately* (i.e., we call the solver #S times).[2] When, on the other hand, we consider γ as a matrix, we are able to solve a single (unique) big system for all states "at once" (i.e., we call the solver only once). These are just two alternative but numerically equivalent ways of structuring the solutions. That said, I consider γ as a matrix in all the exercises of this problem set.

---

[1] Using Julia's `NLSolve` package.

[2] #S here denotes the number of states of the problem. In our case, seven.

1. *For this problem set, you must use projection methods. For this, solve the model using a global projection method. In particular, use Chebyshev polynomials and the collocation method to solve the model. Provide evidence for your solution: figures for the value/policy functions, running times, Euler errors, etc.*

Solution. For the Chebyshev polynomials I define the function

$$\texttt{chebyshev\_polynomial}(x, d) = \cos(d \cdot \arccos(x)), \tag{3}$$

where d stands for the polynomial degree.

For the Chebyshev polynomials' *roots*, I define two functions:

$$\texttt{chebyshev\_root}(i, d) = -\cos\left(\frac{2i-1}{2d}\pi\right) \quad \text{and} \quad \texttt{chebyshev\_roots}(d).$$

The former calculates the i-th root of a Chebyshev polynomial of degree d, while the latter returns a vector containing all the d distinct roots of a Chebyshev polynomial of degree d.

Chebyshev polynomials are defined in $[-1, 1]$. Thus we need to further define the following two functions: `normalized_k(`k`)` and `unnormalized_k(`k`)`. The former translates values from K_{grid} into the $[-1, 1]$ interval, while the latter translates values from $[-1, 1]$ into the K_{grid} domain:[3]

$$\texttt{normalized\_k}(k) = 2\frac{k-a}{b-a} - 1 \quad \text{and} \quad \texttt{unnormalized\_k}(k) = \frac{b-a}{2}(k+1) + a. \tag{4}$$

For the Chebyshev collocation method we use Chebyshev polynomials as basis functions — so the `basis`$(\cdot)$ function referred in the preliminary comments is the `chebyshev_polynomial`$(\cdot)$ function here — and the roots of Chebyshev polynomials as the collocation points. For this, we need to choose a maximum degree for the Chebyshev polynomials (i.e., the number of basis functions to be used, which will also determine the size of the system of equations to be solved). I choose 5. The system to be solved then becomes:

$$R(r_1, z, \gamma) = 0 \tag{5}$$
$$R(r_2, z, \gamma) = 0 \tag{6}$$
$$\vdots \tag{7}$$
$$R(r_5, z, \gamma) = 0 \tag{8}$$

where $\gamma = (\gamma_{1,\text{z}}, \cdots, \gamma_{n,\text{z}}) \quad \forall z \in Z_{\text{grid}}$, R is defined as in (2) and $r_1, \ldots, r_5$ are the *unnormalized* Chebyshev roots of a degree-5 Chebyshev polynomial — i.e., the Chebyshev roots projected into the K_{grid} domain.

[3] For this exercise, I consider the same grid for k considered in previous problem sets. That is, the interval $K_{\text{grid}} \equiv [0.75 \cdot k_{\text{ss}}, 1.25 \cdot k_{\text{ss}}]$ discretized considering 500 grid points.

One additional consideration about this exercise: it turns out that the collocation method is quite sensitive with respect to the initial guess considered. And, depending on the guess, the model may not converge properly to the appropriate solution. The problem is that the greater the chosen maximum degree d, the greater will be the vector of coefficients of basis functions. Therefore, the greater will be the dimension of the initial guess that we must "elaborate" and the more difficult it will be to "predict" a reasonable guess.

In order to avoid this problem, I develop an initial guess "improvement trick": the idea is to start with a low maximum degree d (say, one), then solve the system using an arbitrary guess, use the solution (concatenating an additional zero at the end of the vector so to match dimensions) as an initial guess to re-solve the model, now with $d+1$ (say, two) as the maximum degree, and repeat this process until reaching the final desired maximum degree (say, five). With this successive improvement of guesses, proper convergence of the solution becomes more likely.

That said, the code took approximately 0.60 seconds to run. After solving the model, from the consumption policy function I recover the capital policy function. Then, in a similar fashion to what was done in the Problem Set 2 for the Endogenous Grid Method case, I also recover the value function by iterating using the just-obtained policy functions.

Graphical results are reported in Figure 1. Statistics on the Euler Equation Errors are reported in Table 1.

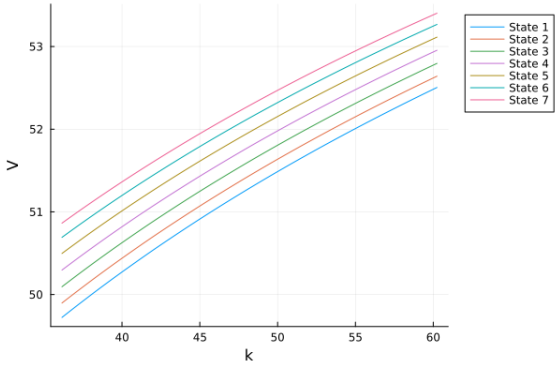
Table 1: Euler Equation Errors (EEEs) statistics — Chebyshev collocation.

Euler Equation Errors' Statistics	
Mean	-8.01
Median	-7.88
Maximum	-7.49
Minimum	-10.59

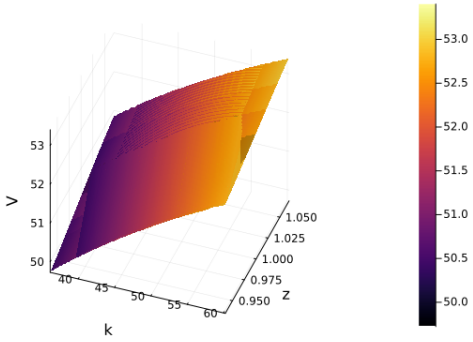
Focusing on the mean, we can interpret that, on average, a \$1 mistake is made for every $\$10^{8.01} \approx \$102,329,299$ spent. It is important to notice *how* accurate this is. For this purpose, notice that in Problem Set 2, the *best* approximate solution we found was the one associated with the Endogenous Grid Method; which, as we observed, yielded a mean EEE of approximately -4.06. That is, in the *best* case of Problem Set 2 — which was already pretty good — we had a \$1 mistake being made for every $\$10^{4.06} \approx \$11,481$ spent. Here we have a \$1 mistake being made for every *hundred million* dollars. This is huge!

□

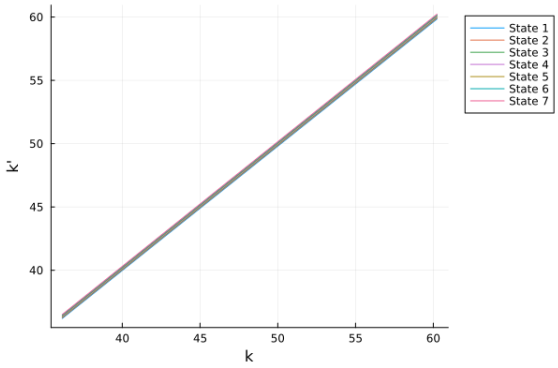
Figure 1: Chebyshev collocation results — Value function, policy functions and EEEs.



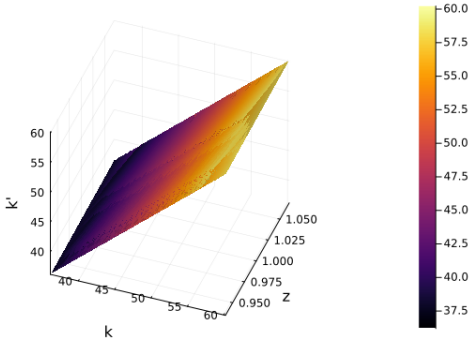
(a) Value function — 2D plot.



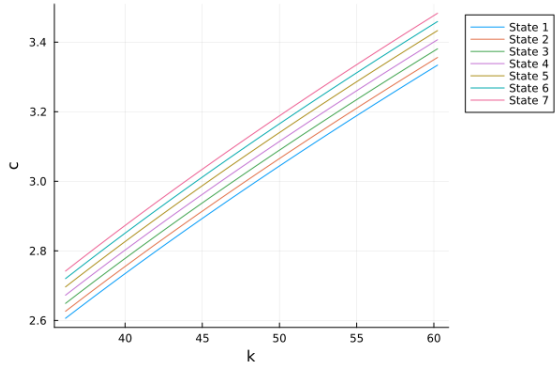
(b) Value function — 3D plot.



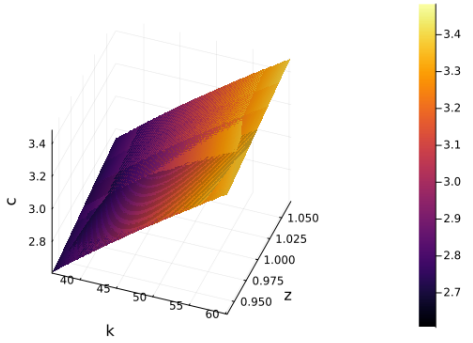
(c) Capital policy function — 2D plot.



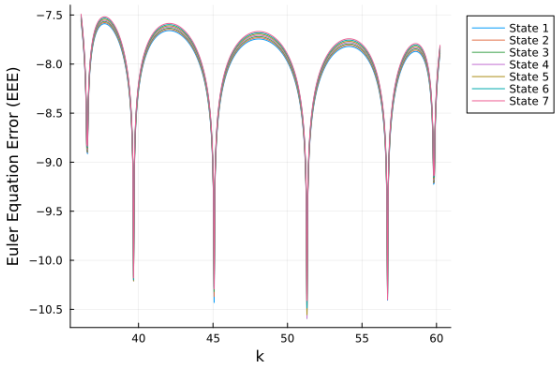
(d) Capital policy function — 3D plot.



(e) Consumption policy function — 2D plot.



(f) Consumption policy function — 3D plot.



(g) Euler Equation Errors.

2. *Now, use again a projection method: the finite elements method. Divide the space in several elements. To solve this, use both the collocation and the Galerkin methods. Again, provide evidence!*

Solution. ***(i)*** First, we consider the Finite Element Method (FEM) with collocation. For this case, we consider the following (piecewise linear) basis functions:

$$\psi_i(k) = \begin{cases} \frac{k-k_{i-1}}{k_i-k_{i-1}} \text{ if } k \in [k_{i-1}, k_i] \\ \frac{k_{i+1}-k}{k_{i+1}-k_i} \text{ if } k \in [k_i, k_{i+1}] \\ 0 \qquad \text{o/w} \end{cases} \tag{9}$$

The functions for `c_hat`$(\cdot)$ and $R(\cdot)$ keep the same, as specified in (1) and (2), respectively.

We just need a few collocation points to have a good approximate solution for the consumption policy function. I consider 15 collocation points, so we end up with a square system of 15 equations in 15 parameters:

$$R(k_1, z, \gamma) = 0 \tag{10}$$
$$R(k_2, z, \gamma) = 0 \tag{11}$$
$$\vdots \tag{12}$$
$$R(k_{15}, z, \gamma) = 0 \tag{13}$$

where $k_1, \cdots, k_{15}$ are the collocation points, obtained from a linearly spaced grid between $0.75 \cdot k_{ss}$ and $1.25 \cdot k_{ss}$, as usual.

Again, the convergence turns out to be very sensitive to the chosen initial guess. Thus, we need a smart guess. After some testing, I choose an initial guess for the γ's consisting of 15 linearly spaced points between 1 and 5, for each of the 7 possible states. It is possible to show that the parameters obtained as the solution for this system, for each (k, z), are exactly the consumption policy function itself, evaluated at each (k, z) — i.e., $c(k_i, z) = \gamma_{i,z}$ for all $i = 1, \ldots, 15$ and $z \in Z_{\text{grid}}$.[4]

The code took approximately 9.49 seconds to run. After solving, observe that the initial grid we considered for solving the model had only 15 points, so the resulting policy functions have only 15 points as well. In view of this, a natural question is: what if we want to evaluate the policy function in a "richer" grid? Easy! We can simply generate a new grid of the desired size, and then recalculate the policy function using the coefficients obtained by the solution of the original system — which considered only 15 grid points — on this new increased grid. That said, the results presented below are projected onto a grid of size 500.

[4]If you are unsure about this result, you can retrieve the policy function again as done in Exercise 1 and you will see that the values obtained for the consumption will be exactly the same values of the parameters resulting from the solution of the system.

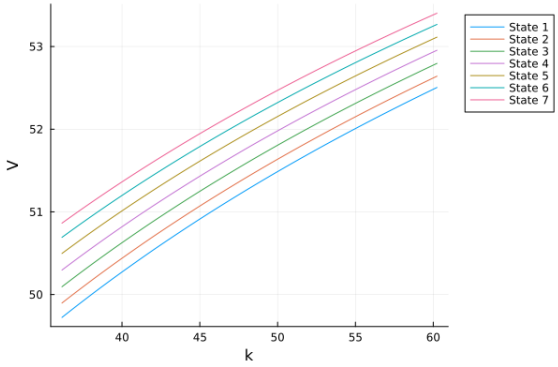
Graphical results are reported in Figure 2. Statistics on the Euler Equation Errors are reported in Table 2.

Table 2: Euler Equation Errors (EEEs) statistics — FEM + Collocation.

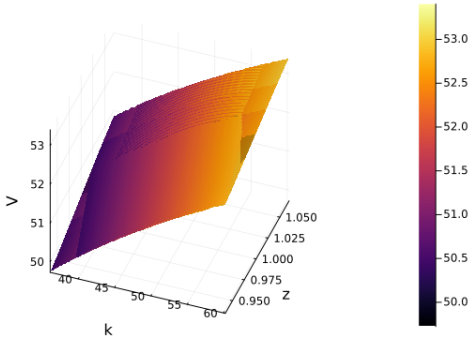
Euler Equation Errors' Statistics	
Mean	-5.22
Median	-5.14
Maximum	-4.15
Minimum	-11.03

Focusing on the mean, we can interpret that, on average, a \$1 mistake is made for every $\$10^{5.22} \approx \$165,958$ spent. As we can see, this result is not as accurate as the Chebyshev collocation method, but it is still *very* accurate — still substantially more accurate than the Endogenous Grid Method, which was the best method of Problem Set 2.

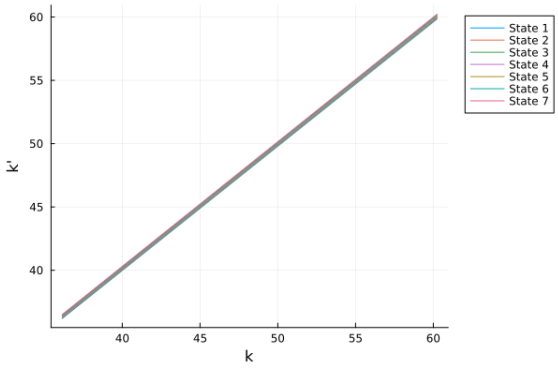
Figure 2: FEM + Collocation results — Value function, policy functions and EEEs.



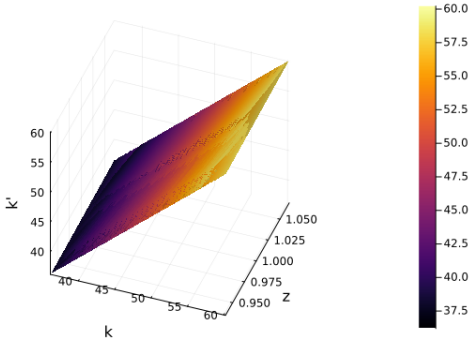
(a) Value function — 2D plot.



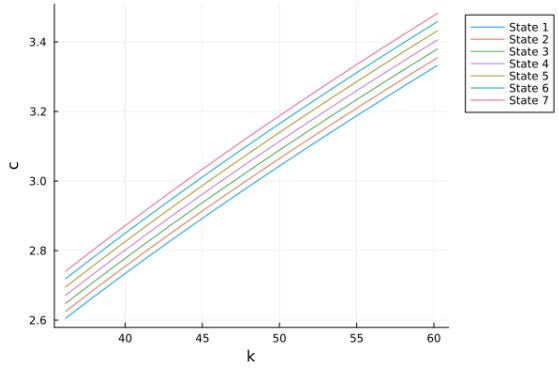
(b) Value function — 3D plot.



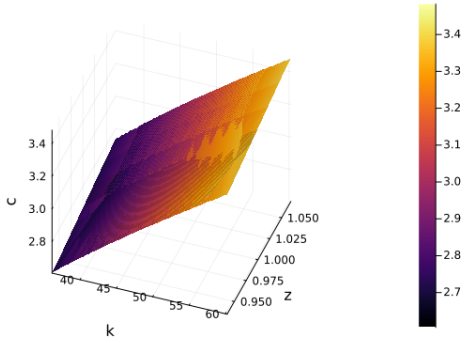
(c) Capital policy function — 2D plot.



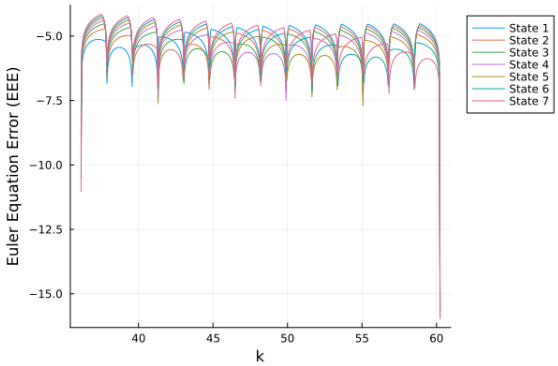
(d) Capital policy function — 3D plot.



(e) Consumption policy function — 2D plot.



(f) Consumption policy function — 3D plot.



(g) Euler Equation Errors.

(ii) Now we consider the Finite Element Method (FEM) with Galerkin Method. For this, we need to define a numerical integration function. I use the package `FaustGaussQuadrature` to obtain the quadrature nodes and weights. I consider the Gauss-Legendre quadrature, but other quadratures could be considered as well (e.g., Gauss-Chebyshev). I then define a function `integral_gl`(f, a, b, n), which numerically integrates a given function f between the points a and b, using n Gauss-Legendre quadrature points:

$$\texttt{integral\_gl}(f, a, b, n) = \int_b^b f(x)\ dx \approx \frac{b-a}{2} \sum_{i=1}^{n} w_i f\left(\frac{b-a}{2}\xi_i + \frac{a+b}{2}\right),$$

where w_i are the weights and ξ_i the nodes obtained using the `FastGaussQuadrature` package.

The functions `basis`$(\cdot)$, `c_hat` and `R` are defined exactly as in the previous case (FEM + Collocation). The only difference here lies in the way we define the system to be solved. In the Galerkin method the system becomes

$$\int_{k_1}^{k_n} R(k, z, \gamma)\psi_1(k)\ dk = 0 \tag{14}$$

$$\int_{k_1}^{k_n} R(k, z, \gamma)\psi_2(k)\ dk = 0 \tag{15}$$

$$\vdots \tag{16}$$

$$\int_{k_1}^{k_n} R(k, z, \gamma)\psi_n(k)\ dk = 0 \tag{17}$$

The key aspect of FEM is that the basis functions are local. In this case, the Galerkin method reduces to

$$\int_{[k_{i-1},k_i)\cup[k_i,k_{i+1}]} R(k, z, \gamma)\psi_i(k)\ dk = 0 \quad \forall i, \tag{18}$$

ignoring the special cases at $i = 1$ and $i = n$. Then, substituting ψ_i and rearranging,

$$\int_{[k_{i-1},k_i)} R(k, z, \gamma)\frac{k - k_{i-1}}{k_i - k_{i-1}}\ dk + \int_{[k_i,k_{i+1}]} R(k, z, \gamma)\frac{k_{i+1} - k}{k_{i+1} - k_i}\ dk = 0 \quad \forall i. \tag{19}$$

Define

$$\bar{E}_i(\gamma) = \int_{[k_i,k_{i+1}]} R(k, z, \gamma)\frac{k_{i+1} - k}{k_{i+1} - k_i}\ dk \tag{20}$$

and

$$\underline{E}_i(\gamma) = \int_{[k_{i-1},k_i)} R(k, z, \gamma)\frac{k - k_{i-1}}{k_i - k_{i-1}}\ dk. \tag{21}$$

The system then becomes simply

$$\bar{E}_1(\gamma) = 0 \tag{22}$$
$$\underline{E}_2(\gamma) + \bar{E}_2(\gamma) = 0 \tag{23}$$
$$\vdots \tag{24}$$
$$\underline{E}_{n-1}(\gamma) + \bar{E}_{n-1}(\gamma) = 0 \tag{25}$$
$$\underline{E}_n(\gamma) = 0. \tag{26}$$

Again, I consider $n = 15$ grid points for k. One advantage of FEM with Galerkin method is that evaluating the integral accurately requires only a *few* quadrature points, because the support is small — just $[k_{i-1}, k_{i+1}]$, rather than a large $[k_1, k_n]$. Therefore, I choose only 15 quadrature points — but I could choose even fewer, without substantial losses of accuracy.

Under this setup, the code took approximately 192 seconds to run. This time performance could, however, be improved without much loss of precision by simply reducing the number of grid points and/or quadrature points.[5]

Graphical results are reported in Figure 3. Statistics on the Euler Equation Errors are reported in Table 3.

Table 3: Euler Equation Errors (EEEs) statistics — FEM + Galerkin.

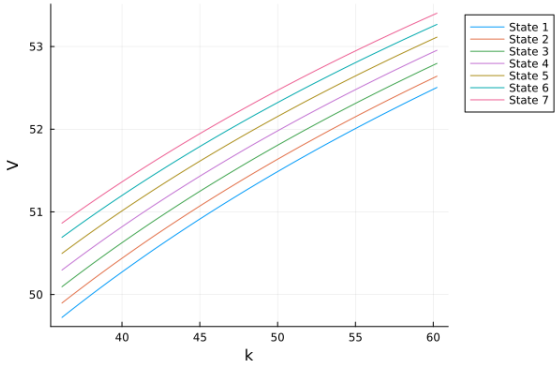
Euler Equation Errors' Statistics	
Mean	-5.57
Median	-5.47
Maximum	-4.38
Minimum	-9.83

Again focusing on the mean, we can interpret that, on average, a \$1 mistake is made for every $\$10^{5.57} \approx \$371,535$ spent. As we can see, this result is more accurate than the results obtained by FEM + Galerkin, but still not as accurate as the results obtained by Chebyshev collocation.

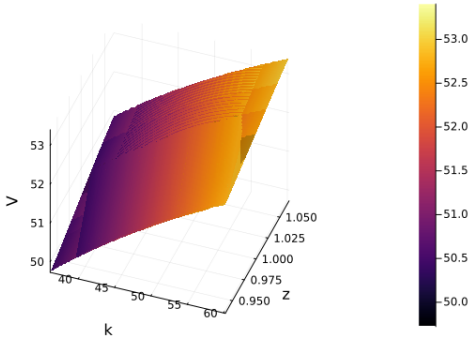
□

[5]For example: with 15 collocation points and 5 quadrature points, the code took approximately 60 seconds to run. With 3 quadrature points, approximately 40 seconds. With two, approximately 28 seconds. And yet the political functions and value remained sufficiently accurate.

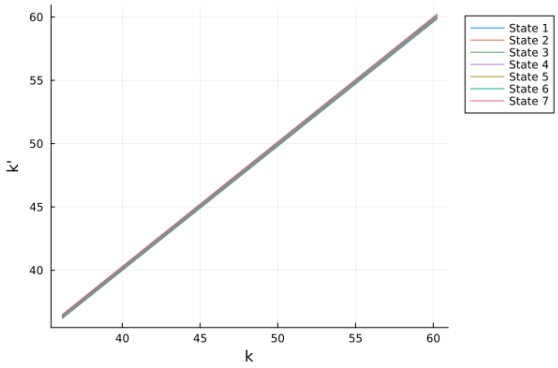
Figure 3: FEM + Galerkin results — Value function, policy functions and EEEs.



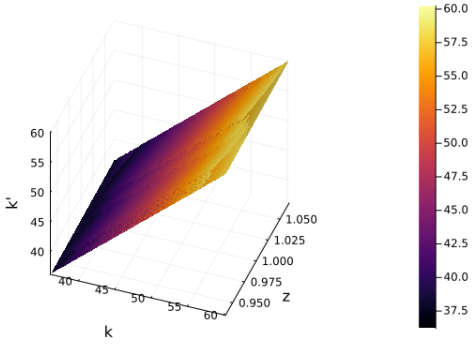
(a) Value function — 2D plot.



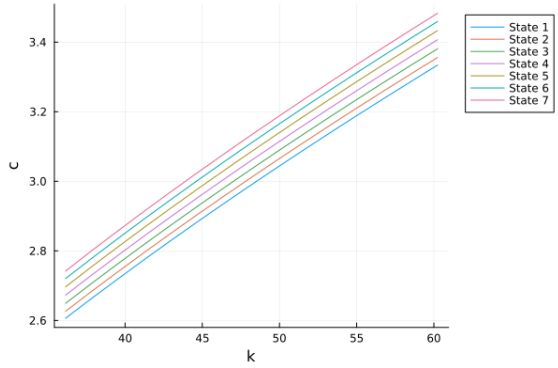
(b) Value function — 3D plot.



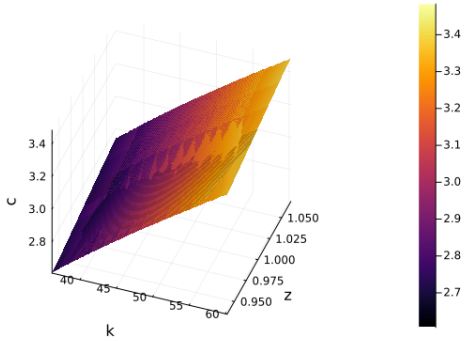
(c) Capital policy function — 2D plot.



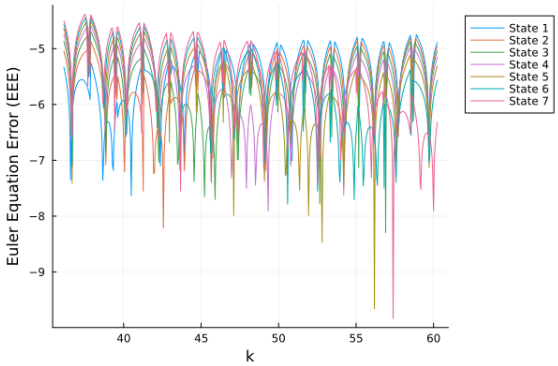
(d) Capital policy function — 3D plot.



(e) Consumption policy function — 2D plot.



(f) Consumption policy function — 3D plot.



(g) Euler Equation Errors.