Lecture 4
Solution Methods

Assaf Patir

7.11.2013

So far we have seen two ideas for solving the functional equation: guess-and-verify and value function iteration. The first approach only works if you're lucky, and the second one always works, but is often difficult to implement. In this lecture we discuss a few more approaches to approximate solutions and discuss their implementation.

1 Policy Function Iteration

The idea behind the policy function iteration is that instead of iterating over the value function v we iterate over the policy function g. Here is how it works: start with some initial guess for the policy function, i.e. $g_0 : X \to X$ such that $g_0(x) \in \Gamma(x)$. Then at every step

1. Calculate

$$v_j(x) = \sum_{t=0}^{\infty} \beta^t F(x_t, g_j(x_t))$$

where $x_t = g^t(x)$. $v_j(x)$ is the value that the agent would get if starting at x she would use the policy function g_j forever.

2. Generate a new policy function by solving the two-period problem

$$g_{j+1}(x) = \arg\max_{y \in \Gamma(x)} [F(x, y) + \beta v_j(y)]. \tag{1}$$

3. $j \to j + 1$ and repeat.

This algorithm is similar to the value function iteration, but often converges faster. For a discussion of the conditions under which it is better, see L-S page 1013.

2 Implementation

For both the value function iteration and this last algorithm the problem remains that the maximization equation usually can't be solved analytically, which means that implementing these algorithm can be a more difficult task than it appears. Here I briefly discuss a few approaches to implementation.

State space discretization: In this straightforward approach we simply replace the state space X with a finite version of it. For example, if $X = [0, L] \subset \mathbb{R}$, we might choose to replace it with $\hat{X} = \{0, L/N, 2L/N, \ldots, L\}$ for some large N. The maximization problem (such as the one in (1)) is then a simple matter of choosing the largest entry in an $(N + 1)$-tuplet. Thus, it is very easy to implement numerically.

While this approach is simple and will often work well, it suffers from the so called curse of dimensionality. While choosing $N = 10^3$ might be sufficient when $X \subseteq \mathbb{R}$, if $X \subseteq \mathbb{R}^k$ the same mesh size requires N^k points. The state space of a modestly sized general equilibrium model can easily include ten variable, so the number of points quickly grows beyond what a computer can handle.

Projection: The idea here is to define both g and v as a linear combination of some (possible different) set of base functions:

$$g_j(x) = \sum_{a=1}^{N} g_j^a \, \phi_a(x) \qquad\qquad v_j(x) = \sum_{b=1}^{K} v_j^b \, \psi_b(x),$$

where $\{\phi_a(x)\}_{a=1}^N$ and $\{\psi_b(x)\}_{b=1}^K$ are a set of known functions (e.g. the Chebychev polynomials are used in many applications). Then, at every step we find a set of coefficients g_j^a and v_j^b to best approximate the functions calculated according to some metric. In other words, we calculate v and g and then find the projection of the function we calculated to the linear space defined by the span of the base functions.

There are two potential advantages to this approach: first, the number of base functions needed (N and K) to achieve a reasonable approximation is usually quite small; and, second, a smart choice of functions can simplify many of the maximization problems and drastically increase computation speed. One disadvantage of this approach is that because we are not choosing $g(x)$ directly, implementing the constraints in this approach can be very tricky and sometimes just impossible.

Shape-preserving splines: Another way of reducing the number of points needed for describing the functions v and g is to approximate them using splines.

Splines are piecewise polynomial functions, i.e. if we are considering the interval $[0, L]$, the spline $s(x)$ can be written

$$s(x) = \begin{cases} a_0^1 + a_1^1 x + \cdots + a_p^1 x^p & x \in [0, x_1] \\ \cdots \\ a_0^j + a_1^j x + \cdots + a_p^j x^p & x \in [x_{j-1}, x_j] \\ \cdots \\ a_0^n + a_1^n x + \cdots + a_p^n x^p & x \in [x_{n-1}, x_n = L]. \end{cases}$$

The spline is thus defined by the $n + 1$ nodes $(x_0, \ldots, x_n)$ and by the $n \times (p + 1)$ coefficients a_i^j. The coefficients are also constrained to make the spline continuously differentiable a few times at the nodes. The advantage of splines is that there are algorithms that choose the nodes optimally so to make the total number of nodes needed small. The idea then, similarly to the projection method, is to calculate g and v at each step and then approximate them with a spline before moving to the next step.

The advantages of splines over the discrete-state-space method is that they handle interpolation more accurately, and reduce the computation time. Furthermore, unlike projection methods, constraints are more readily implemented and one specific set of splines called shape-preserving also ensure that the value function stays concave at every step.

3 Shooting Algorithm

The shooting algorithm is a useful tool for calculating paths in a dynamic programming problem when we know that the path we are interested in converges to a point. Unlike the previous methods, this is directly based on the Euler equation. To demonstrate this method, consider the neoclassical growth model, for which the equations are:

$$\frac{u'(c_t)}{u'(c_{t+1})} = \beta f'(k_{t+1}) \tag{2}$$

$$c_t + k_{t+1} = f(k_t). \tag{3}$$

We have found the steady state k^* and have proved that all optimal paths must converge to it. This leads to the following algorithm:

1. Guess $c_0 \in [0, f(k_0)]$.
2. For $t = 0, 1, \ldots, T$

(a) Calculate k_{t+1} using (3).

(b) Calculate c_{t+1} using (2).

3. If $k_T > k^*$, increase c_0. If $k_T < k^*$, decrease c_0. Start again.

The algorithm is repeated until k_T is close enough to k^*. T should be chosen to be sufficiently large.

4 Perturbation

By far the most frequently used method of analysis in macroeconomics is that of perturbation. Typically, only first-order perturbation is used, in which case it is also called linearization. This method requires that we know the policy function at some point, and usually that point will be a fixed point $g(x^*) = x^*$. We then try to approximate g by constructing a Taylor expansion around x^*.

To illustrate this, consider some model for which the solution is given implicitly by the equation

$$R(x, g(x)) = 0. \tag{4}$$

Since we know that this holds for all x, we can differentiate and evaluate at $x = x^*$:

$$R_1(x^*, g(x^*)) + R_2(x^*, g(x^*))g'(x^*) = 0 \Rightarrow g'(x^*) = -\frac{R_1(x^*, g(x^*))}{R_2(x^*, g(x^*))}$$

By differentiating (4) twice and evaluating at $x = x^*$

$$R^*_{11} + 2R^*_{12}g'(x^*) + R^*_{22}g'(x^*)^2 + R^*_2 g''(x^*) = 0,$$

where $R^*_{11} = R_{11}(x^*, g(x^*))$ and similarly for the other partial derivatives. Using the result for $g'(x^*)$, we find

$$g''(x^*) = -\frac{R^*_{11}}{R^*_2} + \frac{2R^*_{12}R^*_1}{(R^*_2)^2} - \frac{R^*_{22}(R^*_1)^2}{(R^*_2)^3}$$

Obviously, we can continue and go on to higher order terms. The result allows us to then write

$$g(x) \approx g(x^*) + g'(x^*)(x - x^*) + \frac{1}{2}g''(x^*)(x - x^*)^2 + \cdots$$

Notice that the convergence is determined by the first order term, but if one wants to evaluate the effect of uncertainty, then one must go at least to third order.

Let us demonstrate this for the neoclassical growth model. Equations (2) and (3) can be combined into

$$R(k, g(k), g(g(k))) = \frac{u'(f(k) - g(k))}{u'(f(g(k)) - g(g(k)))} - \beta f'(g(k)) = 0.$$

Denote the steady state by k^*. Differentiating once and evaluating at $k = k^*$:

$$\frac{u''(f(k) - g(k))[f'(k) - g'(k)]}{u'(f(g(k)) - g(g(k)))} -$$
$$- \frac{u'(f(k) - g(k))u''(f(g(k)) - g(g(k)))[f'(g(k)) - g'(g(k))]g'(k)}{[u'(f(g(k)) - g(g(k)))]^2} -$$
$$- \beta f''(g(k))g'(k) = 0.$$

Using $g(k^*) = k^*)$ and $f'(k^*) = \beta^{-1}$:

$$\frac{u''(f(k^*) - k^*)}{u'(f(k^*) - k^*)}[\beta^{-1} - g'(k^*)][1 - g'(k^*)] - \beta f''(k^*)g'(k^*) = 0,$$

which can be rewritten

$$g'(k^*)^2 - \phi g'(k^*) + \beta^{-1} = 0, \tag{5}$$

where

$$\phi = 1 + \frac{1}{\beta} + \beta \frac{u'(c^*)}{u''(c^*)} f''(k^*).$$

Equation (5) has two positive solutions, and one can show that one is greater than one and the other is smaller than one. Since we know that there is convergence, we choose the smaller one. This analysis can be extended as before to find the higher order derivatives at k^*, so we can get better approximations of g. However, there is much more to be said about the first order approximation, so in what follows we focus on generalizing our results.

5 Linear Difference Equation

5.1 Mathematical Background

Before going back to dynamic programming, we develop some theory for linear difference equations. Consider the following problem

$$z_{t+1} = Az_t, \quad t = 0, 1, \ldots$$

where $x_t \in \mathbb{R}^n$, A is an n-by-n matrix of constants, and z_0 is given.

Recall from linear algebra that any matrix can be put in the form $A = B^{-1}\Lambda B$, where $\det B \neq 0$ and Λ is the Jordan form of A. For the sake of simplicity, we are going to assume that Λ is actually diagonal, but note that this analysis goes through in the more general when Λ only takes the Jordan block diagonal form.[1] Thus,

$$\Lambda = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix},$$

and the λ_i's potentially be equal to each other. If we define $w_t = Bz_t$, then $z_{t+1} = Az_t$ implies

$$\begin{aligned} z_{t+1} &= Az_t = B^{-1}\Lambda B z_t \\ Bz_{t+1} &= \Lambda B z_t \\ w_{t+1} &= \Lambda w_t \\ w_t &= \Lambda w_{t-1} = \Lambda^2 w_{t-2} = \cdots = \Lambda^t w_0, \end{aligned}$$

and since

$$\Lambda^t = \begin{pmatrix} \lambda_1^t & & & \\ & \lambda_2^t & & \\ & & \ddots & \\ & & & \lambda_n^t \end{pmatrix},$$

we can also write

$$w_t^{(i)} = \lambda_i^t w_0^{(i)}.$$

For every $i = 1, \ldots, n$, if $\lambda_i > 1$ the sequence $w_t^{(i)}$ will diverge unless $w_0^{(i)} = 0$. When $\lambda_i < 1$ the sequence $w_t^{(i)}$ will converge to zero regardless of $w_0^{(i)} = 0$. Thus, overall the sequence of vectors w_t converges if and only if for every $\lambda_i > 1$, the corresponding $w_0^{(i)}$ vanishes.

Since $z_t = B^{-1}w_t$ and B^{-1} is a full rank matrix of constants, the sequence z_t will converge whenever w_t converges and vice-versa. Thus, if we arrange the matrix Λ such that the first m eigenvalues $\lambda_1, \ldots, \lambda_m$ are the ones that are smaller than unity, convergence occurs if the last $n - m$ entries of (Bz_0) are null.

[1] If you don't know what the Jordan form is, for now it will be enough that you know that for the 'typical' matrix A, a base can be chosen in $\mathbb{R}^n$ such that A is diagonal in that base. We then write A as $B^{-1}\Lambda B$, where Λ is diagonal and B is the change-of-base matrix.

5.2 Linearized Dynamic Planning

We are interested in solving the problem

$$\max_{\{x_t\}} \sum_{t=0}^{\infty} \beta^t F(x_t, x_{t+1}),$$

where $x_t = (x_t^1, \ldots, x_t^l) \in \mathbb{R}^l$. For simplicity, we consider a problem without constraints. The Euler equations are

$$\frac{\partial}{\partial y^i} F(x,y)\Big|_{x=x_t, y=x_{t+1}} + \beta \frac{\partial}{\partial x^i} F(x,y)\Big|_{x=x_{t+1}, y=x_{t+2}} = 0.$$

Notice that there are l equations. Assume that we have found a steady-state solution $x_t = x^*$. We wish to linearize the Euler equations around that solution, that is, to find the linear approximation when all variables are close to x^*. The resulting equation is

$$P' \cdot \hat{x}_t + Q \cdot \hat{x}_{t+1} + \beta P \cdot \hat{x}_{t+2} = 0, \tag{6}$$

where $\hat{x}_t = x_t - x^*$, and the l-by-l matrices P,Q and R are given by

$$P_{ij} = \frac{\partial}{\partial x^i} \frac{\partial}{\partial y^j} F(x,y)\Big|_{x=y=x^*},$$
$$Q_{ij} = \frac{\partial}{\partial y^i} \frac{\partial}{\partial y^j} F(x,y)\Big|_{x=y=x^*} + \beta \frac{\partial}{\partial x^j} \frac{\partial}{\partial x^i} F(x,y)\Big|_{x=y=x^*}.$$

To proceed, we define the $2l$-dimensional vectors

$$z_t = \begin{pmatrix} \hat{x}_t \\ \hat{x}_{t+1} \end{pmatrix}.$$

The equation (6) can be rewritten as a $2l$-by-$2l$ matrix equation

$$\begin{pmatrix} P' & Q \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} \hat{x}_t \\ \hat{x}_{t+1} \end{pmatrix} + \begin{pmatrix} 0 & \beta P \\ -I & 0 \end{pmatrix} \begin{pmatrix} \hat{x}_{t+1} \\ \hat{x}_{t+2} \end{pmatrix} = 0.$$

Assuming that $\det P \neq 0$, we have $z_{t+1} = A z_t$, where

$$A = \begin{pmatrix} 0 & I \\ -\beta^{-1} P^{-1} & 0 \end{pmatrix} \begin{pmatrix} P' & Q \\ 0 & I \end{pmatrix} = \begin{pmatrix} 0 & I \\ -\beta^{-1} P^{-1} P' & -\beta^{-1} P^{-1} Q \end{pmatrix}.$$

We have rewritten the linearized dynamic planning problem in the form we analyzed in the previous section. The next step is to calculate the eigenvalues of A. The following lemma establishes that the eigenvalues of A come in pairs:

Lemma 1. *Let A be as defined above, and let λ be an eigenvalue of A. Then $(\beta\lambda)^{-1}$ is also an eigenvalue of A.*[2]

Proof. Let λ be a nonzero eigenvalue of A, i.e. λ solves $\det(A - \lambda I_{2l}) = 0$. Using the identity

$$\det \begin{pmatrix} X & Y \\ Z & W \end{pmatrix} = \det(X) \det(W - ZX^{-1}Y),$$

we find

$$\begin{aligned} \det(A - \lambda I_{2l}) = \det \begin{pmatrix} -\lambda I_l & I_l \\ -\beta^{-1}P^{-1}P' & -\beta^{-1}P^{-1}Q - \lambda I_l \end{pmatrix} = \\ = (-\lambda)^l \det(-\beta^{-1}P^{-1}Q - \lambda I_l - \beta^{-1}\lambda^{-1}P^{-1}P') = 0, \quad (7) \end{aligned}$$

which implies that the determinant on the last line vanishes. Similarly

$$\det(A - (\beta\lambda)^{-1} I_{2l}) = (-\beta\lambda)^{-l} \det(-\beta^{-1}P^{-1}Q - (\beta\lambda)^{-1} I_l - \lambda P^{-1}P').$$

However, since for any matrix $\det X = \det(X')$ and using the fact that $Q' = Q$

$$\begin{aligned} &\det(-\beta^{-1}P^{-1}Q - (\beta\lambda)^{-1} I_l - \lambda P^{-1}P') = \\ &= \det(-\beta^{-1}QP'^{-1} - (\beta\lambda)^{-1} I_l - \lambda P P'^{-1}) = \\ &= \det(P[-\beta^{-1}P^{-1}Q - (\beta\lambda)^{-1}P^{-1}P' - \lambda I_l]P'^{-1}) = \\ &\qquad = \det(P) \det(-\beta^{-1}P^{-1}Q - (\beta\lambda)^{-1}P^{-1}P' - \lambda I_l) \det(P)^{-1} = 0. \end{aligned}$$

Therefore $\det(A - (\beta\lambda)^{-1} I_{2l}) = 0$, which proves that $(\beta\lambda)^{-1}$ is also an eigenvalue of A. □

Since $\lambda \cdot (\beta\lambda)^{-1} = \beta^{-1} > 1$, within each of the pairs of the eigenvalues of A at least one is greater than unity in absolute value. Of course, it is also possible that both eigenvalues within the pair are larger than unity, therefore the number of eigenvalues that are smaller than unity is smaller than l. If we decompose $A = B^{-1}\Lambda B$ as in the previous subsection, and arrange Λ such that the first $m \leq l$ eigenvalues are the ones smaller than unity, then, as we have seen before, convergence occurs if the last $2l - m$ entries of (Bz_0) are null. Thus, we have $2l - m \geq l$ conditions that we need to satisfy.

Recall that $z_0 = (x_0, x_1)'$, and that x_1 are free for the agent to chose, while x_0 are predetermined. Therefore, there are exactly l degrees of freedom available, and we can distinguish between two cases:

[2]Note that since A is a nonsingular matrix ($\det A = \det(-\beta^{-1}P^{-1}P') = (-\beta)^{-l} \neq 0$), all of its eigenvalue are nonzero.

- If $m = l$, we have exactly the required number of degrees of freedom to satisfy the criterion for convergence, and therefore there is exactly one sequence that solves the linearized problem (6).
- If $m < l$, we do not have enough degrees of freedom to satisfy the equations, which implies that we can not assure convergence for all initial conditions. Therefore, the steady state is not stable.

5.3 Example: A two-sector growth model

In the previous subsection we have considered a dynamic programming problem without constraints. In the neoclassical growth model we have put the model in this form by using the resource constraints to substitute out c_t. However, in more complicated models one may not want to do that, and the point here is that the logic of the previous subsection works nonetheless. We shall demonstrate this with an example, that follows Uzawa (1965) and Lucas (1988). We will return to this model later when we discuss growth theory.

Consider an economy in which labor can be used for two tasks: producing physical goods and accumulating human capital. The physical goods can be consumed or used as physical capital and are produced using physical capital k_t and human capital x_t according to $y_t = k_t^\alpha(\phi_t x_t)^{1-\alpha}$, where $\phi_t \in [0,1]$ is the fraction of human capital employed in producing these goods. The remainder $1-\phi_t$ is used for accumulating human capital according to

$$x_{t+1} = x_t + A(1-\phi_t)x_t.$$

The utility-flow function is $u(c) = c^{1-\gamma}/(1-\gamma)$. The maximization problem is

$$\begin{aligned}
&\max_{\{c_t,\phi_t\}} \sum_{t=0}^{\infty} \beta^t u(c_t) \\
&\text{s.t.} \quad x_{t+1} = x_t + A(1-\phi_t)x_t, \\
&\qquad c_t + k_{t+1} = (1-\delta)k_t + k_t^\alpha(\phi_t x_t)^{1-\alpha}.
\end{aligned}$$

We derive the FOCs in the usual way. After eliminating the lagrange multipliers, we get the four equations that define the system:

$$\frac{1}{\beta}\Big(\frac{c_{t+1}}{c_t}\Big)^\gamma = 1 - \delta + \alpha\Big(\frac{\phi_{t+1}x_{t+1}}{k_{t+1}}\Big)^{1-\alpha}, \tag{8a}$$

$$\Big(\frac{c_{t+1}}{c_t}\Big)^\gamma\Big(\frac{k_t x_{t+1}\phi_{t+1}}{k_{t+1}x_t\phi_t}\Big)^\alpha = \beta(1+A), \tag{8b}$$

$$c_t + k_{t+1} = (1-\delta)k_t + k_t^\alpha(\phi_t x_t)^{1-\alpha}, \tag{8c}$$

$$x_{t+1} = x_t + A(1-\phi_t)x_t. \tag{8d}$$

We guess a solution $k_t = \mu^t k^*$, $x_t = \mu^t x^*$, $c_t = \mu^t c^*$ and $\phi_t = \phi^*$. This is a solution if

$$\frac{1}{\beta}\mu^\gamma = 1 - \delta + \alpha\Big(\frac{\phi^* x^*}{k^*}\Big)^{1-\alpha}, \tag{9a}$$

$$\mu^\gamma = \beta(1 + A), \tag{9b}$$

$$\frac{c^*}{k^*} + \mu = 1 - \delta + \Big(\frac{\phi^* x^*}{k^*}\Big)^{1-\alpha}, \tag{9c}$$

$$\mu = 1 + A(1 - \phi^*). \tag{9d}$$

Notice that (9b) determines μ, and then (9d) ϕ^*. After that (9a) determine x^*/k^* and (9c) determines c^*/k^*. k^* itself is not determined, because as long as the ratios are constant, changing the value of k^* is equivalent to redefining $t = 0$.

In the perturbation approach we consider small deviations from the above balanced growth path. We define

$$\hat{k}_t = \frac{k_t - \mu^t k^*}{\mu^t k^*}, \qquad \hat{x}_t = \frac{x_t - \mu^t x^*}{\mu^t x^*}, \qquad \hat{c}_t = \frac{c_t - \mu^t c^*}{\mu^t c^*}, \qquad \hat{\phi}_t = \frac{\phi_t - \phi^*}{\phi^*}.$$

When linearizing it is useful to note

$$k_t^\alpha = (\mu^t k^*)^\alpha (1 + \hat{k}_t)^\alpha \approx (\mu^t k^*)^\alpha (1 + \alpha \hat{k}_t). \tag{10}$$

The linearized equations are

$$\begin{aligned}
&\frac{1}{\beta}\mu^\gamma \gamma(\hat{c}_{t+1} - \hat{c}_t) - \alpha\Big(\frac{\phi^* x^*}{k^*}\Big)^{1-\alpha}(1-\alpha)(\hat{\phi}_{t+1} + \hat{x}_{t+1} - \hat{k}_{t+1}) = 0,\\
&\gamma(\hat{c}_{t+1} - \hat{c}_t) - \alpha(\hat{k}_{t+1} - \hat{k}_t - \hat{x}_{t+1} + \hat{x}_t + \hat{\phi}_{t+1} - \hat{\phi}_t) = 0\\
&\frac{c^*}{k^*}\hat{c}_t + \mu\hat{k}_{t+1} = (1-\delta)\hat{k}_t + \Big(\frac{\phi^* x^*}{k^*}\Big)^{1-\alpha}[\alpha\hat{k}_t + (1-\alpha)(\hat{x}_t + \hat{\phi}_t)] = 0,\\
&\mu(\hat{x}_{t+1} - \hat{x}_t) = -A\phi^*\hat{\phi}_t.
\end{aligned}$$

Using equations (9) we can write this in matrix notation $Pz_{t+1} + Qz_t = 0$, where

$$z_t = (\hat{k}_t, \hat{x}_t, \hat{c}_t, \hat{\phi}_t)',$$

$$P = \begin{pmatrix}
(A+\delta)(1-\alpha) & -(A+\delta)(1-\alpha) & \gamma(1+A) & -(A+\delta)(1-\alpha)\\
-\alpha & \alpha & \gamma & \alpha\\
-\mu & 0 & 0 & 0\\
0 & \mu & 0 & 0
\end{pmatrix},$$

$$Q = \begin{pmatrix}
0 & 0 & -\gamma(1+A) & 0\\
\alpha & -\alpha & -\gamma & -\alpha\\
1+A & \alpha^{-1}(1-\alpha)(A+\delta) & -c^*/k^* & \alpha^{-1}(1-\alpha)(A+\delta)\\
0 & -\mu & 0 & 1+A-\mu
\end{pmatrix}.$$

We therefore have $z_{t+1} = -P^{-1}Qz_t$, so we need to find that eigenvalues of $A = -P^{-1}Q$. This is hard to do by hand, but a computer easily finds that A has one eigenvalue in the unit circle $|\lambda_1| < 1$, one that is exactly equal to one $\lambda_2 = 1$, and two eigenvalues that are outside of it $|\lambda_{3,4}| > 1$. Denote the eigenvectors corresponding to the eigenvalues by $z^{(i)}$, and define the matrix B by placing the eigenvectors in rows and taking the inverse:

$$B = \begin{pmatrix} | & | & | & | \\ z^{(1)} & z^{(2)} & z^{(3)} & z^{(4)} \\ | & | & | & | \end{pmatrix}^{-1},$$

By definition $AB^{-1} = B^{-1}\Lambda$, so $A = B^{-1}\Lambda B$. We now know that we need that the last two components of Bz_0 vanish, which determines c_0 and ϕ_0. To make this very explicit, we demand

$$0 = \begin{pmatrix} B_{3,1} & B_{3,2} & B_{3,3} & B_{3,4} \\ B_{4,1} & B_{4,2} & B_{4,3} & B_{4,4} \end{pmatrix} \begin{pmatrix} \hat{k}_0 \\ \hat{x}_0 \\ \hat{c}_0 \\ \hat{\phi}_0 \end{pmatrix},$$

which leads us to our final answer:

$$\begin{pmatrix} \hat{c}_0 \\ \hat{\phi}_0 \end{pmatrix} = -\begin{pmatrix} B_{3,3} & B_{3,4} \\ B_{4,3} & B_{4,4} \end{pmatrix}^{-1} \begin{pmatrix} B_{3,1} & B_{3,2} \\ B_{4,1} & B_{4,2} \end{pmatrix} \begin{pmatrix} \hat{k}_0 \\ \hat{x}_0 \end{pmatrix}.$$

Notice that this is a policy function.

As a side comment, in this model, the unit eigenvector is actually exact, and it will remain exactly one even with higher orders corrections taken into account. The reason has to do with a specific symmetry that this model has. Remember that when doing perturbations what we are doing is starting from a known solution and considering an economy that starts close to that solution. In this example the known solution is $(k_t, x_t, c_t, \phi_t) = (\mu^t k^*, \mu^t x^*, \mu^t c^*, \phi^*)$, and we are considering (k_0, x_0) close to the initial values (k^*, x^*). However, if it happens to be the case that $(k_0, x_0) = (\tau k^*, \tau x^*)$ for some $\tau > 0$, i.e. the ratio k_0/x_0 is the same as in the steady state k^*/x^*, then there is nothing to solve. Choosing $(k_t, x_t, c_t, \phi_t) = (\mu^t k_0, \mu^t x_0, \mu^t c^* k_0/k^*, \phi^*)$ will be a solution, and this is simply equivalent to an economy that follows the same growth path, but starts from a different point in time.

A Linearizing the Euler Equation (8a)

This appendix is dedicated to working out linearizing (8a) in full detail. Here is the equation again:

$$0 = -\frac{1}{\beta}\Big(\frac{c_{t+1}}{c_t}\Big)^{\gamma} + 1 - \delta + \alpha\Big(\frac{\phi_{t+1}x_{t+1}}{k_{t+1}}\Big)^{1-\alpha}.$$

Using (10)

$$\begin{aligned}
&c_{t+1}^{\gamma} \approx (\mu^{t+1}c^*)^{\gamma}(1+\gamma\hat{c}_{t+1}),\\
&c_t^{\gamma} \approx (\mu^{t}c^*)^{\gamma}(1+\gamma\hat{c}_{t})\\
&\phi_{t+1}^{1-\alpha} \approx \phi^{*1-\alpha}(1+(1-\alpha)\hat{\phi}_{t+1}),\\
&x_{t+1}^{1-\alpha} \approx (\mu^{t+1}x^*)^{1-\alpha}(1+(1-\alpha)\hat{x}_{t+1}),\\
&k_{t+1}^{1-\alpha} \approx (\mu^{t+1}k^*)^{1-\alpha}(1+(1-\alpha)\hat{k}_{t+1}).
\end{aligned}$$

Therefore,

$$\frac{c_{t+1}^{\gamma}}{c_t^{\gamma}} \approx \frac{(\mu^{t+1}c^*)^{\gamma} + (\mu^{t+1}c^*)^{\gamma}\gamma\hat{c}_{t+1}}{(\mu^{t}c^*)^{\gamma} + (\mu^{t}c^*)^{\gamma}\gamma\hat{c}_{t}} = \mu^{\gamma}\frac{1+\gamma\hat{c}_{t+1}}{1+\gamma\hat{c}_t} \approx \mu^{\gamma}(1+\gamma\hat{c}_{t+1}-\gamma\hat{c}_t).$$

and

$$\begin{aligned}
&\Big(\frac{\phi_{t+1}x_{t+1}}{k_{t+1}}\Big)^{1-\alpha} \approx \frac{\phi^{*1-\alpha}(1+(1-\alpha)\hat{\phi}_{t+1})(\mu^{t+1}x^*)^{1-\alpha}(1+(1-\alpha)\hat{x}_{t+1})}{(\mu^{t+1}k^*)^{1-\alpha}(1+(1-\alpha)\hat{k}_{t+1})} =\\
&= \left(\frac{\phi^*x^*}{k^*}\right)^{1-\alpha}\frac{(1+(1-\alpha)\hat{\phi}_{t+1})(1+(1-\alpha)\hat{x}_{t+1})}{1+(1-\alpha)\hat{k}_{t+1}} \approx\\
&\qquad\qquad \approx \left(\frac{\phi^*x^*}{k^*}\right)^{1-\alpha}[1+(1-\alpha)(\hat{\phi}_{t+1}+\hat{x}_{t+1}-\hat{k}_{t+1})]
\end{aligned}$$

If we substitute these expressions into (8a), we get

$$\begin{aligned}
&0 = -\frac{1}{\beta}[\mu^{\gamma}(1+\gamma\hat{c}_{t+1}-\gamma\hat{c}_t)]+1-\delta+\alpha\left(\frac{\phi^*x^*}{k^*}\right)^{1-\alpha}[1+(1-\alpha)(\hat{\phi}_{t+1}+\hat{x}_{t+1}-\hat{k}_{t+1})] =\\
&= \left\{-\frac{1}{\beta}\mu^{\gamma}+1-\delta+\alpha\left(\frac{\phi^*x^*}{k^*}\right)^{1-\alpha}\right\}\\
&\qquad\qquad -\frac{1}{\beta}\mu^{\gamma}\gamma(\hat{c}_{t+1}-\hat{c}_t)+\alpha\left(\frac{\phi^*x^*}{k^*}\right)^{1-\alpha}(1-\alpha)(\hat{\phi}_{t+1}+\hat{x}_{t+1}-\hat{k}_{t+1}).
\end{aligned}$$

The term in curly braces vanishes because of (9a). This is the whole idea behind expanding around something that is a known solution: the zeroth order vanishes, and we are left with only the first order. This is how we get the first linearized equation.