BEPP 931 - Solution to Problem Set 7

Projection Method and Finite Difference Method

Shasha Wang and Cung Truong Hoang

April 3, 2020

Question 1 - Projection - An ordinary Differential Equation Example

The Projection method is a general method using approximation methods and finite-dimensional methods to solve problems with unknown functions.

For **technical details**, I used Matlab symbolic toolbox to take derivatives for approximation, and use "*solve*" to solve for the resulted first order conditions. In addition, I wrote a routine that can take a system of **any** degree of freedom. In this question we are already given 1 "free" condition that $y(0) = 1$, but my routine can take more than one "free" conditions, i.e., one more free conditions, one fewer coefficients to determine via projection.

For **question description**, consider the differential equation

$$y' - y = 0, \quad y(0) = 1$$

which has the solution $y = e^x$. We will use projection methods to solve it over the interval $0 \leq x \leq 3$.

To approximate the ODE, we assume an order 3 polynomial with 4 coefficients to determine. But since we are given $y(0) = 1$, we can use this condition to pin down the constant term of the polynomial and then determine the other three coefficients using different methods. Below we report both the resulted coefficients of different methods and the L^2 error of different methods.

Here we make a just-identified system. Since we have only three unknowns, let's make three nodes over the interval $0 \leq x \leq 3$, i.e., $n = 3$.

Define the operator $\mathcal{N} : C^1 \to C^0$ by

$$\mathcal{N}y = \frac{dy}{dx} - y.$$

The solution to the IVP satisfies $\mathcal{N}y = 0$ and $y(0) = 1$.

Approximate the solution using the family of functions

$$\widehat{y}(x; a) = 1 + \sum_{j=1}^{n} a_j x^j.$$

Note that $\widehat{y}(0; a) = 1$.

Define the residual function pointwise by

$$R(x; a) = (\mathcal{N}\widehat{y})(x) = -1 + \sum_{j=1}^{n} a_j \left(j x^{j-1} - x^j \right).$$

Choose a such that $R(x;a)$ is "nearly" zero for all $x \in [0,3]$, the region of interest.

Below are some brief introduction to the eight methods we adopt.

Least-squares Method

Least-squares method is to minimize the sum of squared errors

$$\min_a \int_0^3 R(x;a)^2 dx,$$

Quadratic objective $\Rightarrow n$ linear first-order conditions with n unknowns,

i.e.,

$$\min_a < R(x;a), R(x;a) > \Rightarrow < R(x;a), \frac{\partial}{\partial a_i} R(x;a) >= 0, \quad i = 1, \ldots, n\Gamma$$

Galerkin

If $R(\cdot;a)$ is the zero function, then

$$\int_0^3 R(x;a)x^j dx = 0, \quad j = 1, \ldots, n$$

n linear equations in n unknowns,

i.e.,

$$P_i(a) =< R(x;a), \phi_i(x) >= 0, \quad i = 1, \ldots, n$$

where $\phi_i(x)$ is the basis function of projection.

Method of Moments

If $R(\cdot;a)$ is the zero function, then

$$\int_0^3 R(x;a)x^{j-1} dx = 0, \quad j = 1, \ldots, n$$

n linear equations in n unknowns,

i.e.,

$$P_i(a) =< R(x;a), x^{i-1} >= 0, \quad i = 1, \ldots, n$$

We can see that with ordinary polynomials as base functions, the method of moments and Galerkin method are similar, with one-degree difference.

Subdomain

If $R(\cdot; a)$ is the zero function, then

$$\int_0^3 R(x; a) I_{D_i}(x) dx = 0, \quad j = 1, \ldots, n$$

n linear equations in n unknowns,

i.e.,

$$P_i(a) =< R(x; a), I_{D_i}(x) >= 0, \quad i = 1, \ldots, n$$

where $\{D_i\}_{i=1}^n$ is a sequence of sets covering D and I is the indicator function.

Collocation

$$P_i(a) = R(x_i; a) =< R(x; a), \delta(x - x_i) >= 0, \quad i = 1, \ldots, n$$

where $\{x_i\}_{i=1}^n$ is a sequence of points in the desired region and δ is the Dirac delta function[1].

Here we discuss two special cases: Uniform (bad) and orthogonal (good) collocation.

Collocation using Chebyshev Nodes

Chebyshev collocation: Use zeros of n th order Chebyshev polynomial adapted to the interval $[0, 3]$:

$$x_j = \left(1 - \cos\left(\frac{2j-1}{2n}\pi\right)\right)\frac{3}{2}$$

n linear equations in n unknowns.

Collocation using Uniform Nodes

Uniform collocation: If $R(\cdot; a)$ is the zero function, then

$$R(x_j; a) = 0, \quad j = 1, \ldots, n$$

Use uniform grid on the interval $[0, 3]$:

$$x_j = (j-1)\frac{3}{n-1}$$

n linear equations in n unknowns.

[1] In mathematics, the Dirac delta function is a generalized function or distribution introduced by the physicist Paul Dirac. It is used to model the density of an idealized point mass or point charge as a function equal to zero everywhere except for zero and whose integral over the entire real line is equal to one.

Taylor Power Series

The power series method uses information about y and its derivatives at $t = 0$ to compute the coefficients.

Since $y' - y = 0$ at $t = 0$, we know that $y'(0) = 1$. But $\hat{y}' = a_1 + 2a_2x + 3a_3x^2$ and $\hat{y}'(0) = a_1$. Hence $a_1 = 1$. We can also fix a_2 and a_3 in this way. Assuming a smooth solution, we can differentiate the differential equation to find $y'' - y' = 0$. since this holds at $t = 0, y''(0) = y'(0) = 1$. However, $\hat{y}'' = 2a_2 + 6a_3x$ and $\hat{y}''(0) = 2a_2$. Hence $a_2 = 1/2$. Similarly $y''' - y'' = 0, \hat{y}'''(0) = 1 = 6a_3$, implying that $a_3 = 1/6$. The final approximation is

$$\hat{y} \cong 1 + x + \frac{x^2}{2} + \frac{x^3}{6}$$

which indeed are the first four terms of the Taylor series expansion of ϵ.

By construction we can see this method will only produce an approximation that is very good near 0.

Optimal L^2

This method implements the least squares fit of a polynomial satisfying the boundary condition to the function e^x. For same of comparison, we adopt a cubic polynomial here.

For **technical details**, I used *lsqcurvefit* to implement the curve fitting.

Table 1 shows coefficients solved by different methods, and Table 2 shows errors of different methods using different order of polynomials.

	a2	a3	a4
Least squares	1.2903	-0.8065	0.6586
Galerkin	10.0000	-6.2500	2.9167
Method of moments	2.2857	-1.4286	0.9524
Subdomain	2.5000	-1.5000	1.0000
Chebyshev collocation	1.6923	-1.2308	0.8205
Uniform collocation	1.0000	-1.0000	0.6667
Power series	1.0000	0.5000	0.1667
Optimal L^2	1.7552	-0.8393	0.7792

Table 1. Solutions for Coefficients

n	LS	Galerkin	MM	Subdomain	Cheby Collo	Unif Collo	Power Series	Optimal L^2
3	1.043878e+01	5.579178e+02	2.773685e-01	2.071693e+00	4.860034e+00	2.794510e+01	1.420991e+01	2.734607e-02
4	2.255107e-02	1.570592e+00	1.310533e-03	1.915799e-02	8.364983e-02	1.665798e+00	3.355045e+00	5.695894e-04
5	2.386958e-05	3.245450e-02	1.647008e-05	4.807460e-04	6.257320e-04	2.240014e-02	6.043421e-01	8.409812e-06
6	1.744456e-07	3.896694e-04	1.780743e-07	4.090577e-06	3.575188e-06	3.833211e-04	8.521116e-02	9.244284e-08
7	1.468069e-09	3.697134e-06	1.504377e-09	7.032359e-08	2.045394e-08	4.744195e-06	9.620447e-03	7.850826e-10
8	9.948686e-12	2.757619e-08	1.013637e-11	4.365642e-10	9.809466e-11	5.572598e-08	8.869798e-04	5.302377e-12
9	5.486436e-14	1.662029e-10	5.567567e-14	5.368878e-12	4.323541e-13	5.017980e-10	6.792094e-05	2.914719e-14
10	2.509040e-16	8.252644e-13	2.538896e-16	2.486763e-14	1.605306e-15	4.225201e-12	4.383361e-06	1.329018e-16

Table 2. Error of Projection Methods

From the results, we see that

1. Coefficients vary significantly from method to method, at least for the cubic polynomial as the basis function of projection.
2. Optimal L^2 does the best in terms of accuracy, with Least-square, Method of Moments, and Collocation with Chebyshev nodes slightly worse but good enough. Power series does the worst expectedly, since it only uses the point where the function value is given.
3. Collocation are expected to yield high accuracy, but with a uniform grid the collocation method does not do as well as with Chebyshev nodes.

4. What is most weird is that Galerkin does VERY bad with small n's, and with large n's it's doing good enough, but I don't see it perform as well as other methods such as Least-squares or Method of Moments. I DON'T KNOW WHY.

Question 2 - Signaling

This problem's ODE is about solving a job market signaling problem.

A worker's ability $n \in [n_m, n_M]$ is unobservable. The worker's education y is observable.

A worker of ability n pays a cost of $C(y,n) = \frac{y}{n}$ to acquire y years of education.

A worker of ability n produces output $S(y,n) = ny^\alpha$, where $\alpha > 0$. The worker's output is unobservable.

The wage schedule is $w(y)$

The worker chooses y optimally, i.e.,

$$w'(y) = C_y(y,n)$$

.

Let $n(y)$ denote the solution to the worker's problem.

The offered wage equals the expected productivity given the observed education. In the equilibrium of a competitive market, expectations are correct, i.e.,

$$w(y) = S(y, n(y))$$

.

Differentiating the equilibrium condition and substituting the optimality condition yields

$$n'(y) = \frac{C_y(y,n(y)) - S_y(y,n(y))}{S_n(y,n(y))} = \frac{n(y)^{-1} - n(y)\alpha y^{\alpha-1}}{y^\alpha}$$

.

A worker of ability n_m makes the socially efficient choice y_m, where

$$S_y\left(y_m, n_m\right) = C_y\left(y_m, n_m\right) \Rightarrow y_m = \left(\alpha n_m^2\right)^{\frac{1}{1-\alpha}}$$

.

The solution to the IVP is

$$n(y) = y^{-\alpha}\sqrt{\frac{2\left(y^{1+\alpha} + D\right)}{1+\alpha}}$$

where $D = \frac{1+\alpha}{2}\left(n_m y_m^\alpha\right)^2 - y_m^{1+\alpha}$ Let $\alpha = 0.25$ and $n_m = 0.1$. This implies $y_m = 3.3930(-4)$.

Finite-Difference Method

For **technical details** , I use damping to do the fixed point iteration for the implicit euler method.

For **results**, the maximal error over $[y_m, y_m + 10]$ is

h	Euler Explicit	rk1	rk4	Euler Fixed Point	Euler Newton	Euler Fzero	trapezoid
0.1	1.1595	0.44152	0.031478	0.11512	0.43441	2.9996	0.20539
0.01	0.13014	0.014568	0.023024	0.025942	0.13014	2.9996	0.059129
0.001	0.019254	0.0024507	7.3866e-05	0.0026098	0.019254	0.0026098	0.0065751
0.0001	0.00093414	4.5474e-05	1.0514e-07	0.00069234	0.0039418	0.00069234	9.4379e-05

We can see that RK4 performs the best, and trapezoid just the second best.

The efficiency needs to be taken into accounts here. Below shows the running time in seconds.

h	Euler Explicit	rk1	rk4	Euler Fixed Point	Euler Newton	Euler Fzero	trapezoid
0.1	0.027364	0.010793	0.013495	0.14289	0.1409	0.19792	0.052848
0.01	0.061231	0.098192	0.083835	0.24927	0.44847	0.86752	0.23327
0.001	0.16069	0.2745	0.46096	1.7852	3.3949	3.4503	1.3251
0.0001	0.984712	2.34582	4.14894	15.1485	33.7973	38.7255	15.0914

Combining accuracy and efficiency, rk4 is the best among all these methods for solving IVP.

Collocation

Chebyshev collocation: Approximate n over the interval $[y_m, y_m + 10]$ using the family of functions

$$\widehat{n}(y;a) = \sum_{i=0}^{k} a_i T_i \left(2\frac{y - y_m}{10} - 1 \right)$$

where T_i is the i th order Chebyshev polynomial.

Define the residual function

$$R(y;a) = \hat{n}'(y;a) - \frac{\hat{n}(y;a)^{-1} - \hat{n}(y;a)\alpha y^{\alpha-1}}{y^{\alpha}}$$

.

Let $\{y_j\}_{j=1}^{k}$ denote the zeros of the k th order Chebyshev polynomial adapted to the interval $[y_m, y_m + 10]$:

$$y_j = \left(1 - \cos\left(\frac{2j-1}{2k}\pi \right) \right) 5 + y_m$$

.

Projection conditions:

$$R\left(y_j;a\right) = 0, \quad j = 1, \ldots, k$$
$$\widehat{n}\left(y_m;a\right) - n_m = 0$$

$\Rightarrow k + 1$ nonlinear equations in $k + 1$ unknowns.

The maximal error over $[y_m, y_m + 10]$ is

k	Chebyshev	Cubic Spline Uniform	Cubic Spline Uniform-in-logs
5	4.261354e+00	9.492322e-01	7.787492e-01
10	3.228814e+00	6.614608e-01	3.834341e-01
15	6.014615e+00	5.428027e-01	2.484413e-01
20	6.018083e+00	4.704417e-01	1.845889e-01
25	3.657677e+00	4.232275e-01	1.421167e-01
50	5.899419e-03	3.025984e-01	9.461856e-02
100	2.872882e-04	1.462254e-01	5.959567e-02

Table 3. Signaling - Collocation

Question 3 - Learning by Doing

In this project, I implement both projection and finite-difference method to compute both the time paths and the policy function.

For **conclusions**, projection is both faster and more accurate than finite difference, with **Chebyshev collocation being the most accurate and stable**. Linear and cubic spline collocations are sensitive to initial guesses and only infeasible due to lack of invertibility of coefficients.

For **technical details**,

1. I used symbolic math toolbox to do analytical computation of derivatives and implement L'Hopital's Rule for the IVP of policy function. I also use *rk4* for shooting with an initial value and *bvpsolve* for collocation of BVP in compecon toolbox.
2. For collocation, initial guesses do matter a lot, especially when there are two mathematical solutions yet only one economically acceptable solution to the problem. If the initial guess is not good enough, the solver will most likely go wrong.

For **question description**, the model is about Learning-by-Doing, with Q - experience/knowledge stock as the state variable and and q - effort as the control variable.

The individual wants to maximize his lifetime utility. Intuitively, being knowledgeable/experienced yields higher utility, and "just the right amount of" efforts yields higher utility - being too lazy would drag down one's spirit and hurt utility and too assiduous would be exhausting. Furthermore, there should be an interaction term between knowledge and effort - more knowledge would make effort less taxing and even more enjoyable.

The individual solves

$$\max_q \int_0^\infty e^{-\rho t}\pi(q,Q)dt$$

subject to

$$\dot{Q} = q - \frac{Q}{100}, \quad Q(0) = 0$$

where $\pi(q,Q) = q - \frac{q^2}{2} - (Q+1)^{-0.2}q$ and $\rho = 0.04$. That is, q denotes quantity and Q cumulative quantity or the stock of "experience." The current value Hamiltonian is

$$H(Q,q,\lambda,t) = q - \frac{q^2}{2} - (Q+1)^{-0.2}q + \lambda\left(q - \frac{Q}{100}\right)$$

The optimality condition

$$0 = \frac{\partial H}{\partial q} = 1 - q - (Q+1)^{-0.2} + \lambda$$

implies

$$0 = -\dot{q} + 0.2(Q+1)^{-1.2}\left(q - \frac{Q}{100}\right) + \dot{\lambda}.$$

Eliminating $\dot{\lambda}$ and λ from the costate equation

$$\dot{\lambda} = \rho\lambda - \frac{\partial H}{\partial Q} = \rho\lambda - \left(0.2(Q+1)^{-1.2}q - \lambda\frac{1}{100}\right)$$

yields

$$\dot{q} = -0.2(Q+1)^{-1.2}\frac{Q}{100} + \left(-1 + q + (Q+1)^{-0.2}\right)\left(\rho + \frac{1}{100}\right)$$

Thus we have a two-point BVP:

$$\begin{gathered}\dot{q} = -0.2(Q+1)^{-1.2}\tfrac{Q}{100} + \left(-1 + q + (Q+1)^{-0.2}\right)\left(\rho + \tfrac{1}{100}\right), \\ \dot{Q} = q - \tfrac{Q}{100}, \quad Q(0) = 0, \quad \lim_{t\to\infty} e^{-\rho t}|\lambda(t)Q(t)| < \infty.\end{gathered}$$

The steady states are

$$Q^* = 0, q^* = 0 \text{ and } Q^* = 57.4128, q^* = 0.5741.$$

The first steady state is inconsistent with optimality, so we use the second to solve the BVP.

Phase Diagram

To better understand the dynamic of the system, I plot a phase diagram. From Figure 1 we learn that it is very hard and subtle to pin down a stable path for the system, because although knowledge is less likely to explode/deplode, both laziness and its lack thereof has inertia - the dynamic goes that laziness would generate more laziness and diligence would generate more diligence.

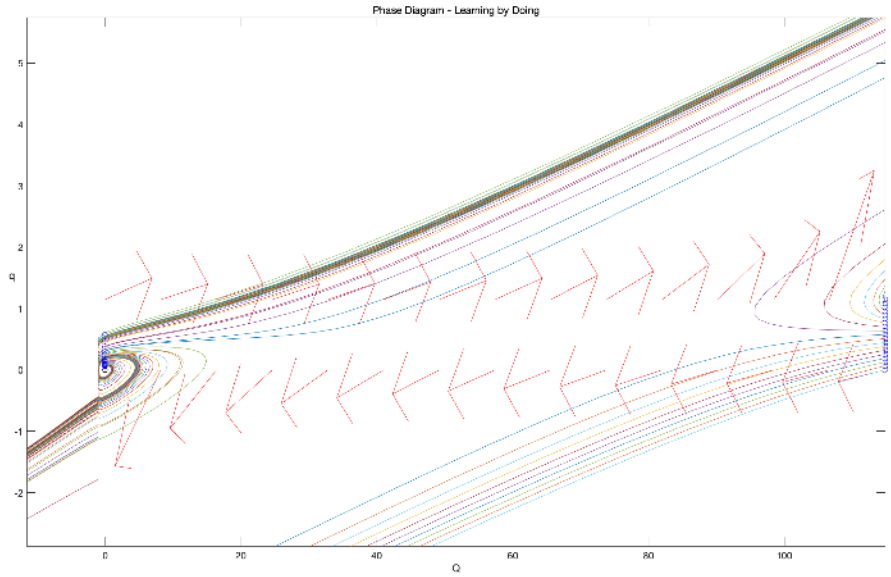

Figure 1. Caption

Typically we solve the system by two methods - projection and finite difference. What is worth nothing about the algorithm to take for the latter method is that we can no longer follow the algorithms we usually use where we step forward until either one of the two laws of motions changes sign[2] , because the counteractive effect between the state and the control variables are not built in the model as the model of optimal growth or that of lifecycle consumption.

[2]Typically we use either of the two algorithms for the finite difference method:

1. shooting for $Q(T) = Q^*$, where T is the first time such that $\dot{Q}(T) <= 0$ or $\dot{q}(T) <= 0$;
2. reverse shooting for $Q_0 = 0$, where T is the first time such that $\dot{Q}(T) >= 0$ or $\dot{q}(T) >= 0$;

On the contrary, the two variables have reinforcing effect upon each other. Hence I revised the algorithm so that I not stop till the law of motion changed sign, but till the state variable goes past its steady state value and then I compare the resulted value of the choice variable with its steady state value. If the steady state value is larger, increase the original value and if smaller, make the initial value also smaller.

Time Paths

I derive the time paths of both the choice and the state variable using the following two methods.

Projection

Chebyshev collocation: Approximate q and Q over the interval $[0, T]$ using the family of functions

$$\widehat{q}(t;a) = \sum_{i=0}^{k} a_i T_i \left(2\frac{t}{T} - 1\right)$$
$$\widehat{Q}(t;b) = \sum_{i=0}^{k} b_i T_i \left(2\frac{t}{T} - 1\right)$$

where T_i is the i th order Chebyshev polynomial. Define the residual functions

$$R^q(t;a,b) = -\hat{q}'(t;a) - 0.2(\widehat{Q}(t;b)+1)^{-1.2}\frac{\hat{Q}(t;b)}{100} + \left(-1 + \hat{q}(t;a) + (\widehat{Q}(t;b)+1)^{-0.2}\right)\left(\rho + \frac{1}{100}\right)$$
$$R^Q(t;a,b) = -\widehat{Q}'(t;b) + \widehat{q}(t;a) + \frac{\widehat{Q}(t;b)}{100}$$

Chebyshev collocation (cont'd): Let $\{t_j\}_{j=1}^{k}$ denote the zeros of the k th order Chebyshev polynomial adapted to the interval $[0, T]$:

$$t_j = \left(1 - \cos\left(\frac{2j-1}{2k}\pi\right)\right)\frac{T}{2}$$

Projection conditions:

$$R^q(t_j;a;b) = 0, \quad j = 1,\ldots,k$$
$$R^Q(t_j;a;b) = 0, \quad j = 1,\ldots,k$$
$$\hat{q}(T;a) - q^* = 0$$
$$\hat{Q}(0;b) - Q_0 = 0$$

$\Rightarrow 2(k+1)$ nonlinear equations in $2(k+1)$ unknowns. The maximal residuals over $[0, 1000]$ are shown in Table 4.

k	Chebyshev		Cubic Spline		Linear	
	q	Q	q	Q	q	Q
5	1.709002e-02	4.568565e-05	7.281373e-03	1.113196e-01	1.090836e-02	1.882710e-01
10	7.567905e-03	2.513259e-03	5.305814e-03	1.272518e-01	1.013506e-02	1.823334e-01
15	3.626435e-03	6.860680e-04	4.046434e-03	9.741119e-02	9.199427e-03	1.745083e-01
20	1.796707e-03	2.419364e-04	3.409080e-03	7.682380e-02	8.436968e-03	1.636664e-01
25	9.019391e-04	9.693474e-05	2.771553e-03	6.168683e-02	7.790317e-03	1.520744e-01
50	3.327294e-05	1.939065e-06	1.448052e-03	2.897151e-02	5.414646e-03	1.016899e-01
100	6.043169e-08	1.945524e-09	8.845388e-04	9.699723e-03	3.016132e-03	5.769812e-02

Table 4. Max Error of q and Q

Figure 2 shows the projection paths.

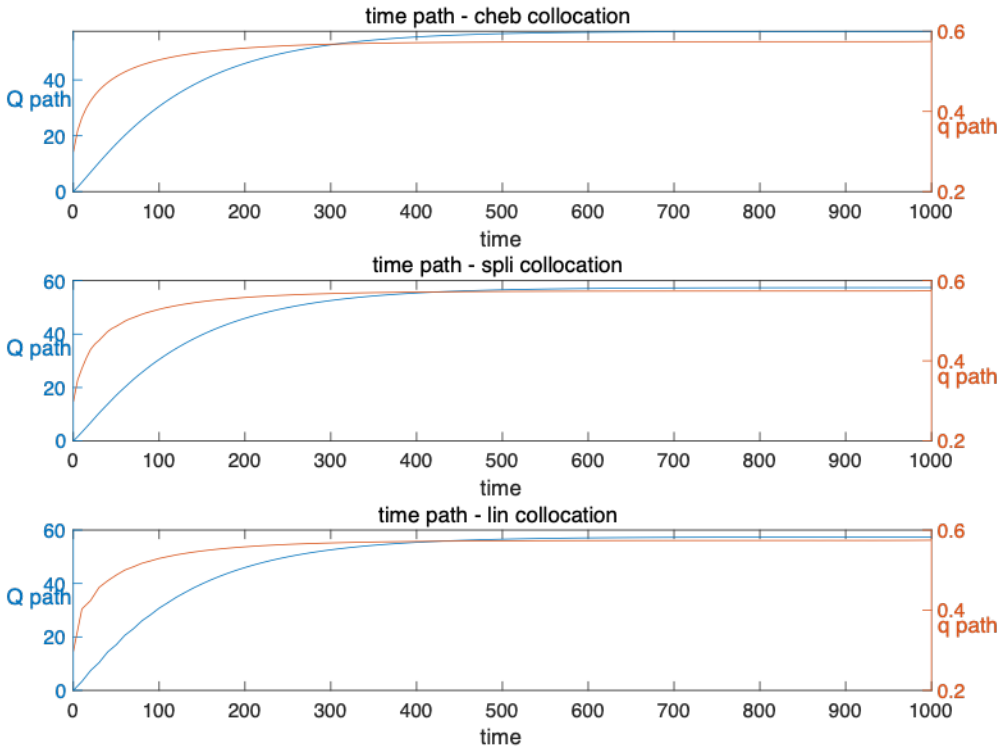

Figure 2. Time Paths - by different collocation with $k = 100$

Finite Difference

Finite-difference methods: Choose a large time horizon T. Use reverse shooting, i.e., fix $Q(T) = Q^*$ and vary $q(T)$ to shoot for $Q(0) = 0$. The homework asks to assess sensitivity to choice of T, but since I proceed the path until the state variable Q passes the initial value or steady state value, this question is irrelevant. And I find the algorithm is sensitive to step-size.

More specifically, the algorithm is of two-layer, with the inner layer solving the IVP and the outer layer solving the nonlinear equation (choose the right guess to make the end value exact right).

Figure 3 are the two figures showing the paths of ordinary shooting and inverse shooting, from which we see the inverse shooting is more stable. I can hardly get the results for forward shooting, and the iteration is cut short after Q goes above the steady state level.

Policy Function

Define the policy function $\psi(Q)$ such that $q(t) = \psi(Q(t))$. Thus $\dot{q}(t) = \psi'(Q(t))\dot{Q}(t)$ and $\psi(Q)$ satisfies the ODE

$$\psi'(Q) = \frac{-0.2(Q+1)^{-1.2}\frac{Q}{100} + \left(-1 + \psi(Q) + (Q+1)^{-0.2}\right)\left(\rho + \frac{1}{100}\right)}{\psi(Q) - \frac{Q}{100}}.$$

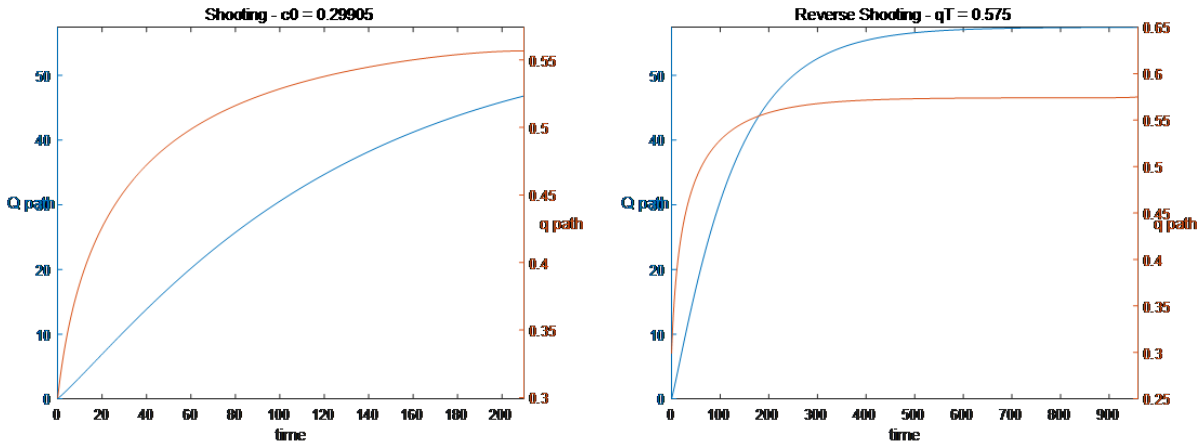

Figure 3. Time Paths

Projection

Chebyshev collocation: Approximate ψ over the interval $[0, 100]$ using the family of functions

$$\widehat{\psi}(Q; a) = \sum_{i=0}^{k} a_i T_i \left(2\frac{Q}{100} - 1 \right)$$

where T_i is the i th order Chebyshev polynomial. Define the residual function

$$R(Q; a) = \hat{\psi}'(Q; a) - \frac{-0.2(Q+1)^{-1.2}\frac{Q}{100} + \left(-1 + \widehat{\psi}(Q; a) + (Q+1)^{-0.2}\right)\left(\rho + \frac{1}{100}\right)}{\widehat{\psi}(Q; a) - \frac{Q}{100}}$$

Let $\{Q_j\}_{j=1}^{k}$ denote the zeros of the k th order Chebyshev polynomial adapted to the interval $[0, 100]$:

$$Q_j = \left(1 - \cos\left(\frac{2j-1}{2k}\pi\right)\right) 50$$

Projection conditions:

$$R\left(Q_j; a\right) = 0, \quad j = 1, \ldots, k$$
$$\widehat{\psi}\left(Q^*; a\right) - q^* = 0$$

$\Rightarrow k + 1$ nonlinear equations in $k + 1$ unknowns.

The maximal residual over $[0, 100]$ is shown in Table 5.

k	Chebyshev	Cubic Spline	Linear Spline	CS w. Uniform Nodes	CS w. Uniform-in-logs Nodes
5	3.133578e-02	6.278649e-02	2.393275e+01	5.821972e+00	1.804341e+01
10	1.068065e-02	1.257693e-02	3.723084e-02	2.379785e-02	2.260635e+00
15	3.566416e-03	7.462735e-03	2.305088e-02	1.445241e-02	2.001314e+00
20	1.225170e-03	5.620596e-03	1.754692e-02	1.030867e-02	2.134524e+01
25	4.287964e-04	4.989749e-03	1.432918e-02	7.848943e-03	4.321694e+00
50	2.559900e-06	2.606764e-03	8.327746e-03	2.946948e-03	2.746521e+03
100	5.931014e-08	8.544895e-04	5.195615e-03	8.114919e-04	2.960451e+00

Table 5. Error of Policy Function Projection

Figure 4 shows the policy function approximation by collocation.

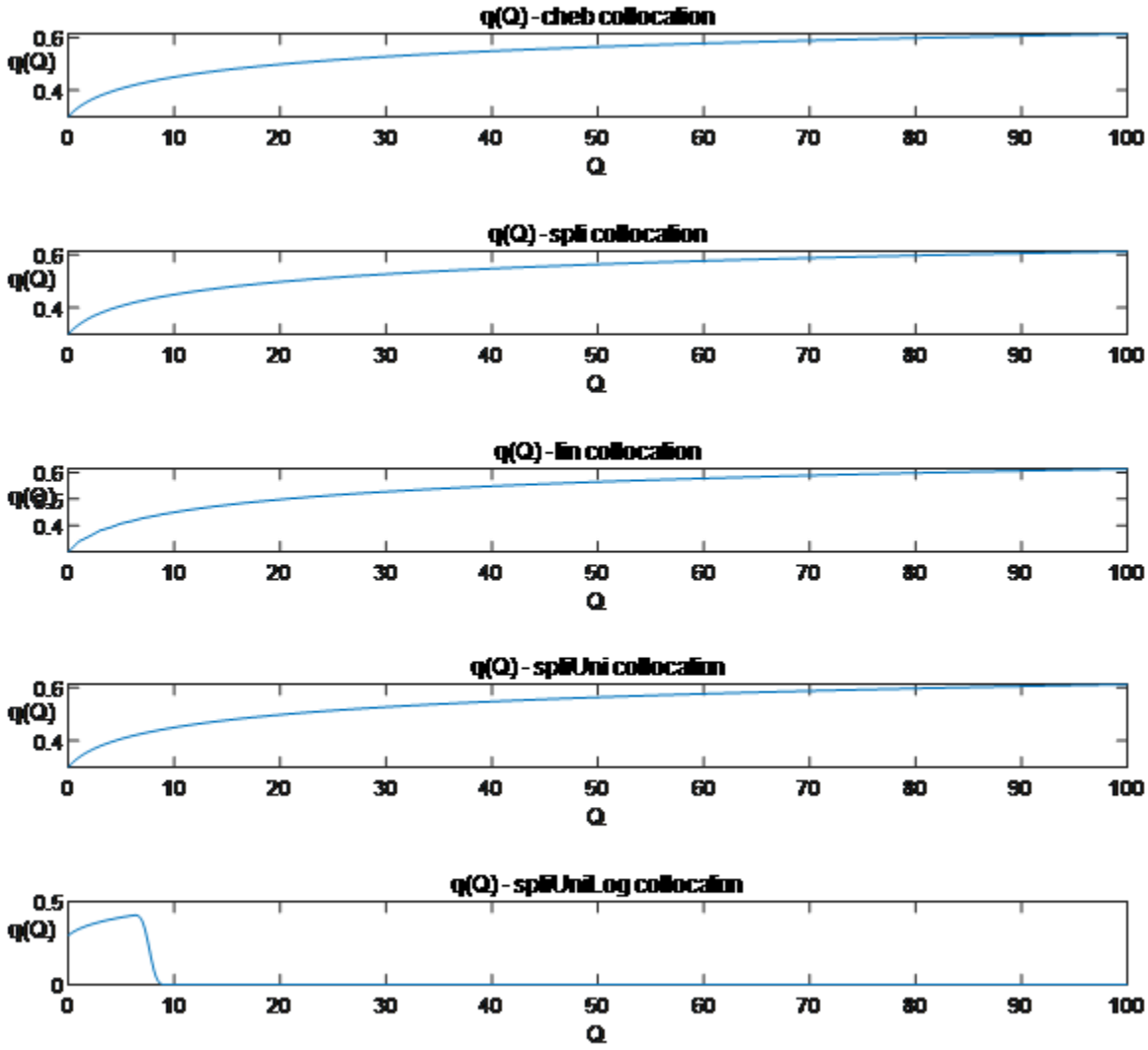

Figure 4. Policy - Collocation

Figure 5 shows the error of policy function by collocation.

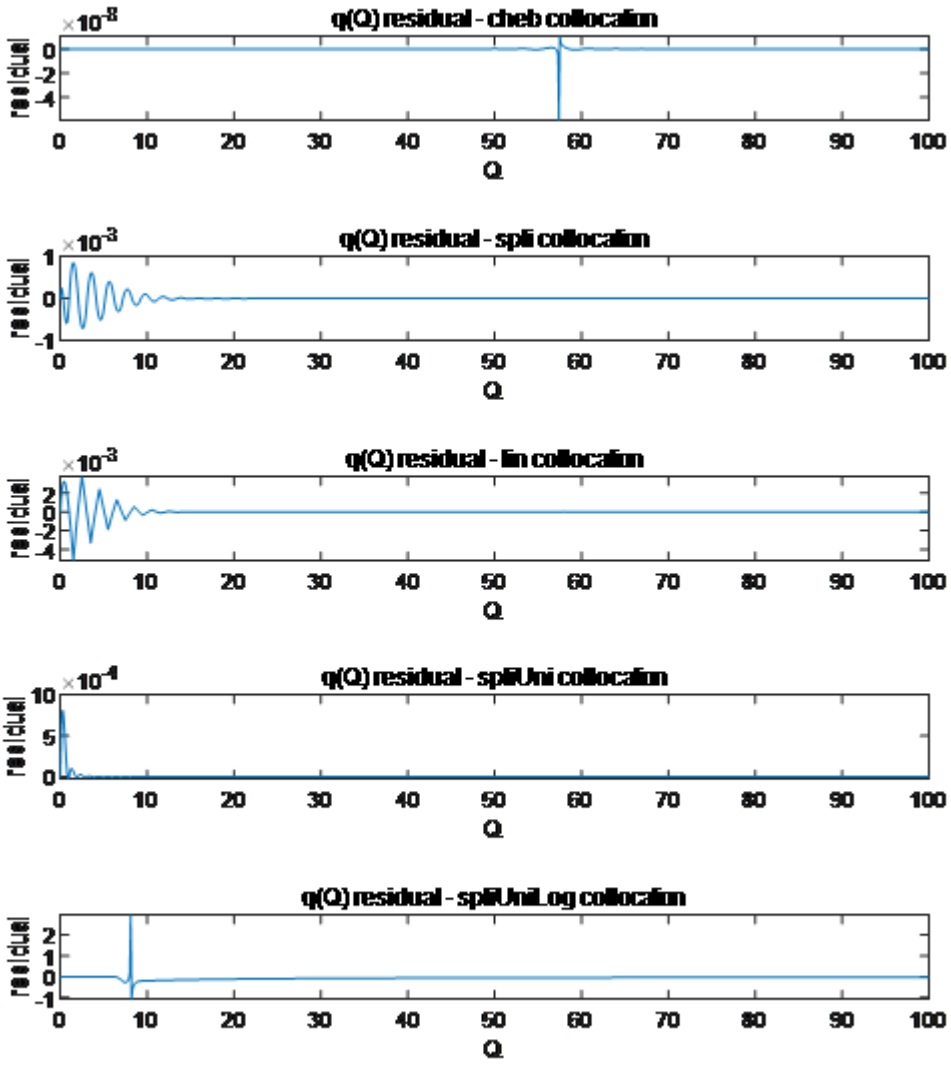

Figure 5. Policy - Collocation

Finite Difference

As mentioned in the conclusion part, I rescaled the q or Q to put them in the same magnitude such that the policy function turns out more concave. Below I present the results of the two scales (i.e., original, and $\tilde{q} = 100 * q$) for sake of comparison.

Finite-difference methods: Use L'Hôpital's rule and a symbolic math package to obtain

$$\psi'(Q^*) \in \{0.0588, 0.0012\}$$

Use Euler or Runge-Kutta methods with a step size of h to approximate ψ over the interval $[0, 100]$ − starting from $(Q^* - h, q^* - h\psi'(Q^*))$ − starting from $(Q^* + h, q^* + h\psi'(Q^*))$ Note that $\psi'(Q^*) = 0.0588$ implies $\psi(Q) < 0$ around $Q = 0$ and thus cannot be a solution, which can be shown in Figure 6.

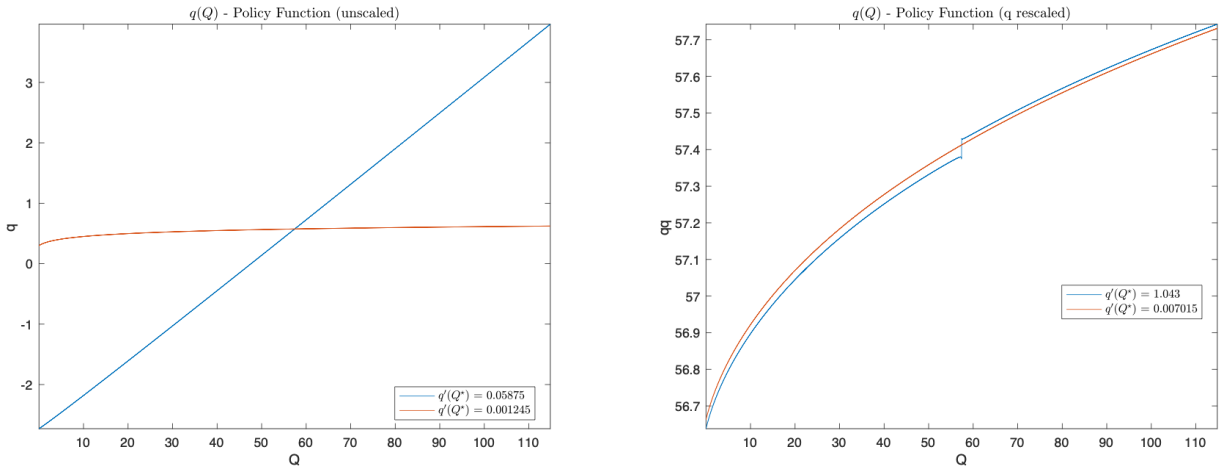

Figure 6. Policy - Shooting Algorithm

It is obvious that the blue lines are unacceptable as policy functions.

What is worth noting is that when I rescaled Q instead - $\tilde{Q} = \frac{Q}{100}$, there is only one solution to $\psi'(Q^*)$, from which we derive the policy function as in Figure 7.

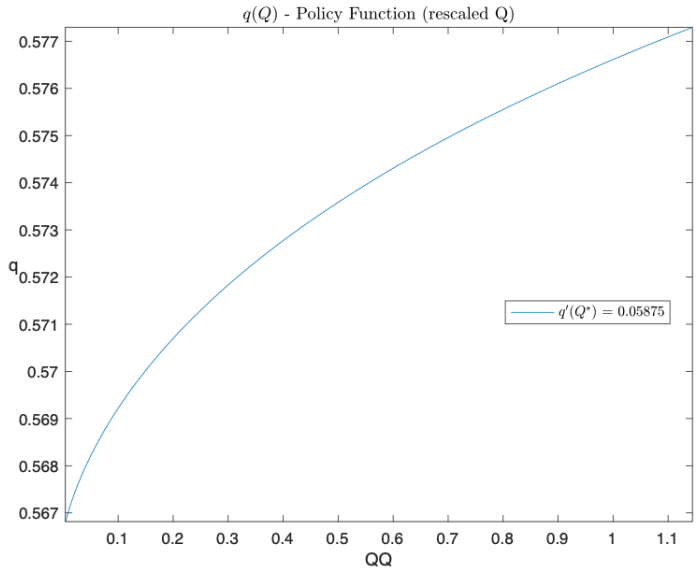

Figure 7. Policy (Rescaled Q) - Shooting Algorithm

Question 4 - Life-cycle Consumption

We first solve the continuous life-cycle model with forward shooting. We solve the two-point BVP given by $\dot{A}$ and $\dot{c}$ along with starting value $A(0) = 0$ with Runge-Kutta-4.We shoot for $\hat{A}(T) = A(T) = 0$ by varying the starting value $c(0)$. We initialize with $c(0) = 0.5$ and increase whenever $\hat{A}(T) > \epsilon$ and decrease our starting guess whenever $\hat{A}(T) < -\epsilon$. This algorithm converges in 19 iterations and we plot the resulting paths for $A(t)$ and $c(t)$ in Figure 8. We obtain an increasing consumption path over time whereas asset holdings are negative in the first periods but then increase to positive and reverting back to 0 in the terminal period.

Next, we use projection methods to solve the same problem. We approximate c and A with Chebyshev polynomial of

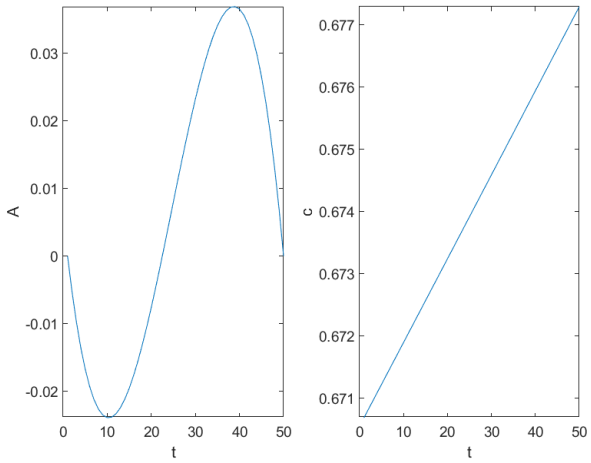

Figure 8. Asset and Consumption Path - Forward Shooting

varying degree. Table 6 shows the maximum residuals we obtain over the interval $[0, 50]$. We compute these residuals for a much finer grid than nodes we used for the Chebyshev collocation while evaluating the residuals at the optimal coefficients which solve the system of equations. In comparison to the suggested solution, our maximum residuals are much lower by an order of up to 10^{-7} starting from a polynomial of order 15 and higher.

k	R^c	R^A
5	2.28e-07	0.0086
10	4.00e-08	3.01e-07
15	2.265e-17	6.86e-13
20	8.33e-17	1.84e-14
25	1.08e-16	7.33e-15
50	1.53e-16	3.77e-14
100	3.75e-16	2.39e-13

Table 6. Max Error of R^c and R^A

We solve the discrete life-cycle model in a similar fashion using projection methods and Chebyshev approximation for S_t. We vary the order of the Chebyshev polynomial and present the maximum residuals in Table 7.

k	R^S
5	7.84e-07
10	9.860e-11
15	3.06e-09
20	2.23e-06
25	8.42e-08

Table 7. Max Error of R^S

Question 5 - Optimal Growth

We begin solving for the policy function $\psi(k)$ with a shooting algorithm. We split the grid for capital in 2 parts: below and above the steady state k^*. When starting just below k^*, we shoot backwards to k_0 and we shoot forward when we start just above k^*. We approximate $\psi(k)$ with Runge-Kutta 4. Figure 9 shows the resulting policy function from the shooting algorithm. In Table 8 we replicate Table 10.2. Our errors are slightly smaller compared to Table 10.2 by a factor of 10^{-1}.

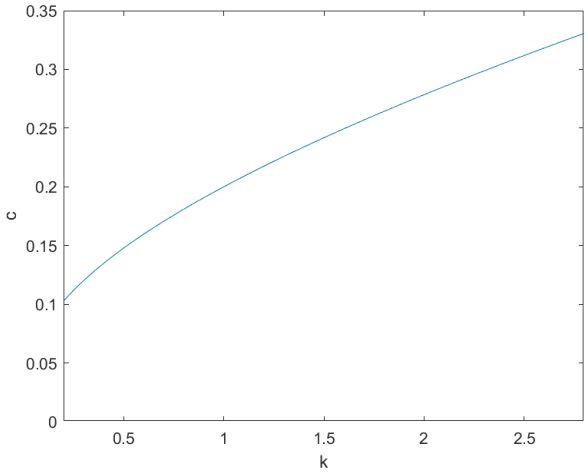

Figure 9. Policy Function: Shooting Algorithm

		Errors		
k	c	h=0.1	h=0.01	h=0.001
0.2	0.1027	3.5e-05	3.2e-09	3.2e-13
0.5	0.1478	3.8e-06	5.2e-10	6.0e-14
0.9	0.1907	2.0e-04	6.7e-09	3.3e-14
1.1	0.2089	1.8e-04	1.1e-08	2.5e-13
1.5	0.2418	8.3e-06	4.3e-10	4.87e-15
2	0.2784	2.7e-06	1.4e-10	6.1e-16
2.5	0.3118	1.6e-06	8.0e-11	3.9e-15
2.8	0.3307	1.2e-06	6.4e-11	3.7e-15

Table 8. Optimal Growth with Reverse Shooting

We also use projection methods to solve for the policy function. We choose to approximate the policy function with a Chebyshev polynomial of order 50. We notice that the resulting policy function is extremely sensitive to the initial guess for the Chebyshev coefficients that we feed into Matlab's `fsolve`. The initial guess that worked for us was derived from our guess of the policy function $\psi^0 = c^*(\frac{k_j}{k^*})$. Figure 10 shows the policy function that resulted from this projection method. Figure 11 shows the oscillating residual function which has a dip at k^*.

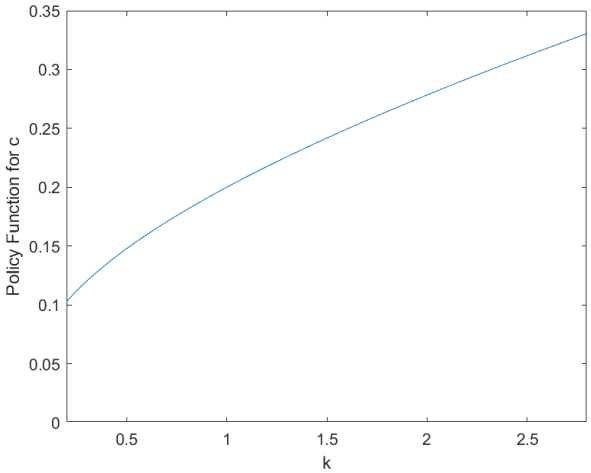

Figure 10. Policy Function: Projection Method

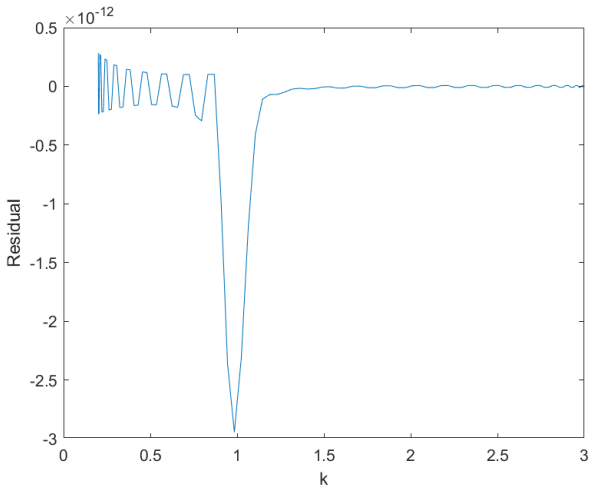

Figure 11. Residual Function: Projection Method

Question 6 - Optimal Growth and Value Function

We first implement a straightforward value function iteration. We initialize with $V^0 = \frac{u(c_0(k))}{1-\beta}$ where $c_0(k) = F(k) - k$. With a capital grid of 500 points, this algorithm converges in 89 iterations and 0.4 seconds to yield the value function as plotted in Figure 12 and the policy function for consumption in Figure 13. The kinks in the policy function stem from the discretization of the capital grid. As we refine the coarseness of the grid, we obtain a smoother policy function.

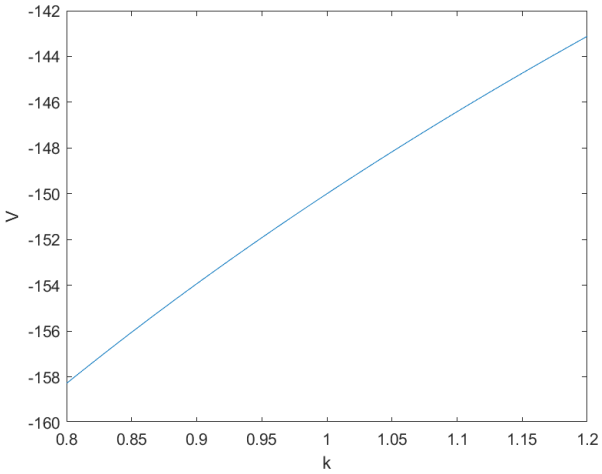

Figure 12. Value Function - Value Function Iteration

Next, we improve on this by adopting an alternating sweep Gauss Seidel approach where we traverse the capital grid in increasing order for every even iteration and in decreasing order for every odd iteration. Within each iteration we find the new guess for the value function using the most updated input. In other words, for every capital stock we have traversed we find the updated value function and feed this into the updating stage for the next capital stock. This approach converges in 4 iterations and takes 0.09 seconds. We obtain the same plot as in Figure 12.

Thirdly, we use parametric approximation together with value function iteration. We choose to approximate the value function with a 10th-order Chebyshev polynomial using 20 nodes. We find this algorithm to take much longer, 510 iterations and 26 seconds until convergence. For the outer loop we impose a convergence criterion of $\epsilon = 10^{-8}$ and within the loop we call Matlab's optimization routine `fminbnd` to solve for the Chebyshev coefficients setting

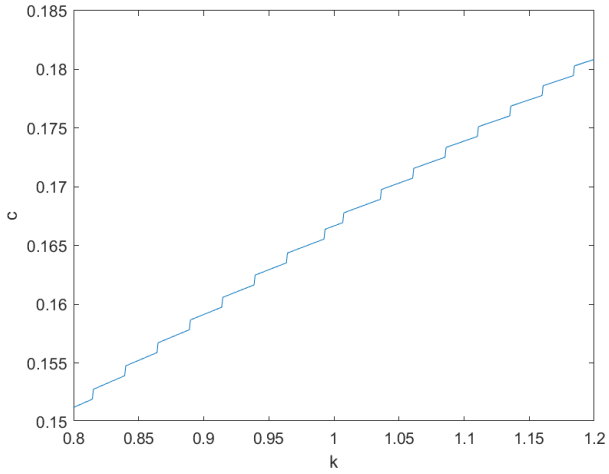

Figure 13. Policy Function - Value Function Iteration

the tolerance to $\epsilon = 10^{-6}$. However, we obtain the same value function as with the other approaches.

Lastly, we performed a policy function iteration which requires 31 iterations in 0.3 seconds to converge, yielding similar results to the value function approach. We summarize our findings in Table 9 below.

	Value Func	Gauss-Seidel	Chebyshev	Policy Func
iterations	89	4	510	31
time in sec	0.4	0.09	26	0.3

Table 9. Speed Comparison