
Solving Continuous Time Optimization Problems Using Dynamic Programming

William J. Polley
Department of Economics
Bradley University

1. The basic problem and its solution in the deterministic case

■ 1.1. General deterministic case

Dynamic programming in continuous time has seen many applications in economics and finance. A very readable exposition of the method appears in Chow (1997) and forms the basis for what is shown here. I have modified the notation slightly, and will only consider a one dimensional case. Note that this is not going to be an in depth exposition of all of the details. It is expected that the reader understand the basic optimization problem as set out in Chow (1997) or any other treatment of this topic. The emphasis here is on the method of solution and the use of *Mathematica* to obtain results.

The basic problem on which we concentrate our effort is the following:

$$V(x(t)) = \max_u E_t \int_t^{\infty} e^{-\rho(\tau-t)} r(x(\tau), u(\tau)) d\tau$$

subject to

$$dx = f(x, u) dt + S(x, u) dz$$

The value function is $V(x)$ (note that the time subscript will be omitted when understood). The subjective discount rate is ρ . The return function (or the instantaneous utility function) is r . The state variable is x (more generally, this could be a vector), and the control is u . The constraint defines a law of motion for the state variable. The function f is deterministic and a function of the state and the control while $S(x,u)dz$ is the stochastic term. Specifically, we model the stochastic component as a Wiener process. If we use a standard Wiener process, then $(S(x, u))^2 dt$ is the variance of $S(x,u)dz$. In general, the variance of the Wiener process (or covariance matrix if we are dealing with more than one dimension) can depend on both the state and the control at any instant. For now, we focus only on the deterministic part of the model.

subject to

$$dx = f(x, u) dt$$

To solve this problem, first write down the Bellman equation. If you are familiar with discrete time dynamic programming, you will notice that the form is identical except for the fact that instead of taking discrete steps, we take steps of size dt , where $dt \rightarrow 0$.

$$V(x(t)) = \max_u (r(x(t), u(t)) dt + e^{-\rho dt} V(x(t+dt)))$$

Recognizing that $e^{-\rho dt} \rightarrow (1-\rho dt)$ as $dt \rightarrow 0$ (recall the Taylor series expansion for e^x and $V(x(t+dt)) \rightarrow V(x(t)) + dV(x(t))$ as $dt \rightarrow 0$), we can write

$$V(x(t)) = \max_u \{ r(x(t), u(t)) dt + (1 - \rho dt) [V(x(t)) + dV(x(t))] + o(dt) \}$$

Rearranging terms and dividing by dt yields

$$\rho V(x(t)) = \max_u \left\{ r(x(t), u(t)) + \frac{1}{dt} dV(x(t)) + o(dt) \right\}$$

Ito's Lemma states that if $dx=f(x,u)dt+S(x,u)dz$, then

$$dV(x) = \left[f \frac{dV}{dx} + \frac{1}{2} \left(\frac{d^2 V}{dx^2} \Sigma \right) \right] dt + \frac{dV}{dx} S dz$$

This is the general form. If z is a standard Wiener process, then $\Sigma dt = dt$. Focusing on the deterministic case, we let $S(x,u)=0$ for all values of x and u .

$$\frac{dV(x)}{dt} = f(x, u) \frac{dV(x)}{dx}$$

Inserting this into the Bellman equation and letting $dt \rightarrow 0$, we obtain the following.

$$\rho V(x(t)) = \max_u \left\{ r(x(t), u(t)) + f(x, u) \frac{dV(x(t))}{dx} \right\}$$

The next task is to find the first order conditions, we can differentiate the right hand side of the Bellman equation with respect to u and set it equal to zero.

$$\frac{\partial r}{\partial u} + \frac{\partial f}{\partial u} \frac{dV}{dx} = 0 \tag{1}$$

If any boundary conditions need to be imposed, do so at this step. Solving the first order condition for the optimal u and calling it \hat{u} gives the decision rule for the control variable. Substituting this back into the Bellman equation gives us a first order differential equation which must be solved. Note that discerning the optimal value of the control will require knowledge of the state and the slope of the value function. The equation which must be solved is:

$$\rho V(x) = r(x, \hat{u}(x)) + f(x, \hat{u}(x)) \frac{dV}{dx} \tag{2}$$

Solving this differential equation completes the solution. However, even common (and simple) functional forms of the instantaneous utility function (natural log, square root, etc.) can yield differential equations which are highly non-linear and therefore difficult even for the computer to solve analytically. Fortunately, a

clever substitution makes the equation very easy to solve. For very simple problems, we can even get a closed form solution.

■ 1.2. An advantageous substitution to use in deriving closed form solutions to very simple problems

This substitution will allow us to make the differential equation simple enough for the computer to solve for some very simple examples. This works for examples 1 and 2 below. Let $\mu = \frac{dV}{dx}$.

$$\rho V(x) = [r(x, \hat{u}(x)) + f(x, \hat{u}(x)) \mu] \quad (3)$$

Differentiate (3) with respect to x.

$$\begin{aligned} \rho \mu &= \left[\frac{dr(x, \hat{u}(x))}{dx} + \frac{df(x, \hat{u}(x))}{dx} \mu + f(x, \hat{u}(x)) \frac{d\mu}{dx} \right] \\ \rho \mu &= \left[\frac{\partial r}{\partial x} + \frac{\partial r}{\partial \mu} \frac{d\mu}{dx} + \left[\frac{\partial f}{\partial x} + \frac{\partial f}{\partial \mu} \frac{d\mu}{dx} \right] \mu + f(x, \hat{u}(x)) \frac{d\mu}{dx} \right] \\ \left(\rho \mu - \frac{\partial r}{\partial x} - \frac{\partial f}{\partial x} \mu \right) dx &- \left(\frac{\partial r}{\partial \mu} + \frac{\partial f}{\partial \mu} \mu + f(x, \hat{u}) \right) d\mu = 0 \\ \frac{dx}{d\mu} &= \frac{\left(\frac{\partial r}{\partial \mu} + \frac{\partial f}{\partial \mu} \mu + f(x, \hat{u}) \right)}{\left(\rho \mu - \frac{\partial r}{\partial x} - \frac{\partial f}{\partial x} \mu \right)} \quad (4) \end{aligned}$$

Though we have not solved the equation, this is a form with which the computer can easily work, as long as f and r are very simple functions. In the event that even this form is too non-linear for the computer to solve easily, we might want to use another method. This is covered in section 2.

■ 1.3. Example 1: Log utility

Let's look at a very simple example. Let $r(x,u)=\ln(u)$ and $dx=-u dt$. This is a continuous time "cake eating" problem. The state variable, x, represents the amount of cake remaining. The control, u, represents the amount consumed. Note that (1) becomes:

$$\begin{aligned} \frac{1}{u} + (-1) \frac{dV}{dx} &= 0 \\ u &= \frac{1}{\mu} \end{aligned}$$

Substituting this result into (4) gives:

$$\begin{aligned} \frac{dx}{d\mu} &= \frac{-\frac{1}{\mu} - \frac{\mu}{\mu^2} - \frac{1}{\mu}}{(\rho \mu)} \\ &= -\frac{1}{\rho \mu^2} \quad (5) \end{aligned}$$

Mathematica solves this easily.

```
DSolve[x'[\mu] + (1 / (\rho (\mu ^ 2))) == 0, x[\mu], \mu]
```

```
{{x[\mu] -> \frac{1}{\mu \rho} + C[1]}}
```

Thus, $\mu = (\rho(x - c))^{-1}$ where c is a constant. Next, we put this result into (3). In *Mathematica*, we will call the value function "v1". In *Mathematica*, remember that "ln" is "Log".

```
v1[x_] = ((Log[1 / \mu] - 1) / \rho) /. {\mu -> (\rho (x - c))^{-1}}
\frac{-1 + Log[\rho [-c + x]]}{\rho}
```

Note that the constant, c , is simply an affine transformation of the value function, and we can safely ignore it. We then have:

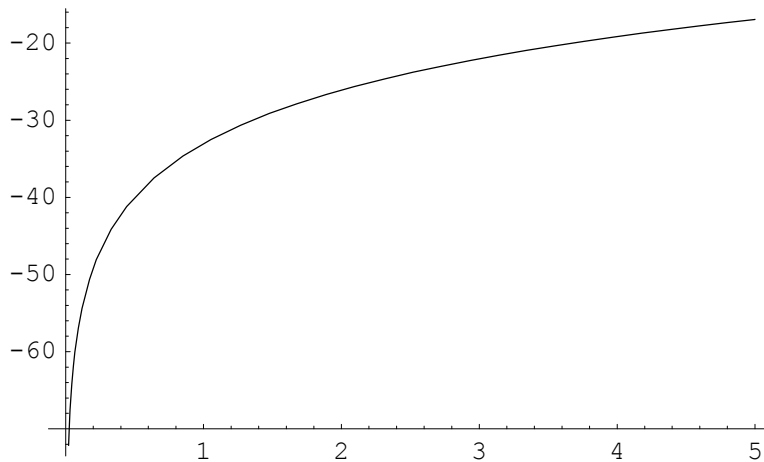
$$V(x) = \frac{-1 + \ln(x\rho)}{\rho} \quad (6)$$

We can plot the result in (6). First, we specify a value for ρ .

```
\rho = .1
```

```
0.1
```

```
Plot[\frac{-1 + Log[x \rho]}{\rho}, {x, .02, 5}]
```



```
- Graphics -
```

Note that since this example is so simple, we could have guessed that the value function would be logarithmic in the state variable. We can see how this works. Guess that $V(x) = A + B \ln(x)$, and compute the constants A and B which would cause the Bellman equation to hold.

$$u = \left(\frac{dV}{dx} \right)^{-1}$$

The Bellman equation (2) becomes:

$$\rho A + \rho B \ln(x) = \ln(u) - u \frac{dV}{dx}$$

$$\rho A + \rho B \ln(x) = \ln \left(\left(\frac{dV}{dx} \right)^{-1} \right) - 1$$

$$\rho A + \rho B \ln(x) = \ln \left(\frac{x}{B} \right) - 1$$

$$\rho A + \rho B \ln(x) = \ln(x) - \ln(B) - 1$$

$$B = 1 / \rho \text{ and } A = - \frac{\ln(B) + 1}{\rho} = \frac{\ln(\rho) - 1}{\rho}$$

$$\text{Therefore, } V(x) = \frac{\ln(\rho) - 1 + \ln(x)}{\rho}$$

This is exactly what we found above.

We now have all the information necessary to solve for the evolution of the state variable. In this case, $\frac{dx}{dt} = -u = -x\rho$. If we specify an initial condition, such as $x(0)=1$, then we can solve for $x(t)$.

```
Clear[ρ];
state[t_] = DSolve[{x[0] == 1, x'[t] + x[t] ρ == 0}, x[t], t]

{{x[t] -> e^{-t ρ}}}
```

And we are done.

■ 1.4. Example 2: Square root utility cake eating problem

Let $r(x,u)=2u^{1/2}$ and $dx = -u dt$. Note that (1) becomes:

$$\frac{1}{u^{1/2}} + (-1) \frac{dV}{dx} = 0$$

$$u = \left(\frac{1}{\mu} \right)^2$$

Substituting this result into (4) gives:

$$\begin{aligned} \frac{dx}{d\mu} &= \frac{-\frac{2}{\mu^2} + \frac{2\mu}{\mu^3} - \frac{1}{\mu^2}}{\rho\mu} \text{ Check this} \\ &= -\frac{\left(\frac{1}{\mu}\right)^2}{\rho\mu} \\ &= -\frac{1}{\rho\mu^3} \end{aligned} \tag{7}$$

```
DSolve[x'[μ] + (1 / (ρ (μ^3))) == 0, x[μ], μ]
```

```
{{x[μ] -> \frac{1}{2 μ^2 ρ} + C[1]}}
```

Ignoring the constant term, $\mu=(2 x\rho)^{-1/2}$.

```
v2[x_] = FullSimplify[((2 (1 /  $\mu$ ) - (1 /  $\mu$ )) /  $\rho$ ) /. { $\mu$  -> (2 x  $\rho$ )-1/2}]
```

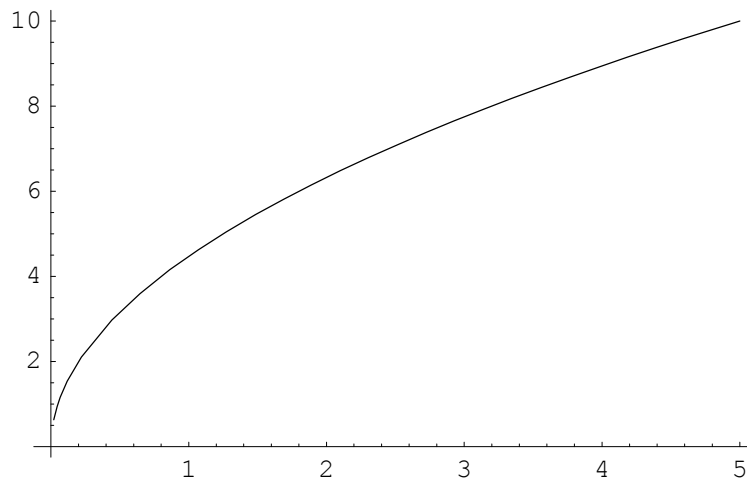
$$\frac{\sqrt{2} x}{\sqrt{x \rho}}$$

Again, let $\rho=0.1$.

```
 $\rho$  = .1
```

```
0.1
```

```
Plot[v2[x], {x, .02, 5}]
```



- Graphics -

Now we have, $\frac{dx}{dt} = -u = -2 x\rho$. If we specify $x(0)=1$, then we can solve for $x(t)$.

```
Clear[ $\rho$ ];
```

```
state[t_] = DSolve[{x[0] == 1, x'[t] + 2 x[t]  $\rho$  == 0}, x[t], t]
```

```
{{x[t] -> e-2 t  $\rho$ }}
```

As one would expect, the answer is very similar to that obtained in the first example. The focus here is on the method. The next example adds a new twist.

2. Capital accumulation

■ 2.1. Preliminaries

Now consider a constant elasticity of substitution (CES) return function, $r(x,u)=\frac{u^{1-\gamma}}{1-\gamma}$, and add capital accumulation by changing the law of motion of the state variable. The economic interpretation is that the state variable is the capital stock. The control variable is consumption. In familiar macroeconomic terms, $y=c+i$, where y is output, c is consumption, and i is gross investment. Suppose that $y(x)=\frac{\rho(x)^\alpha}{\alpha}$ where x is the capital stock. We shall assume that the capital stock depreciates at a constant rate, δ . From current output, some is devoted to consumption. The rest of the output is devoted to investment, part of which replaces the depreciated capital. Our law of motion, using the notation from the previous sections, becomes:

$$dx = \left(\frac{\rho x^\alpha}{\alpha} - u - \delta x \right) dt$$

We follow the methods used in the first section to compute one differential equation that will be used in the solution.

$$\begin{aligned} u^{-\gamma} + (-1) \frac{dV}{dx} &= 0 \\ u - \left(\frac{dV}{dx} \right)^{-\frac{1}{\gamma}} &= 0 \end{aligned} \tag{8}$$

The other differential equation is given by substituting the functional forms for f and r into (2).

$$\rho V(x) - \frac{u^{1-\gamma}}{1-\gamma} - \left(\frac{\rho x^\alpha}{\alpha} - u - \delta x \right) \frac{dV}{dx} = 0 \tag{9}$$

Equation (9) is much like what we would have had in the previous section making the substitution of μ . This differential equation is too complex (even with μ) for the computer to solve analytically. Judd (1998) has a chapter on projection methods which will be useful for this problem. The idea is to solve the system of differential equations defined by (8) and (9) by approximating the functions V and u with polynomials. We want to find polynomials \hat{V} and \hat{u} which approximately solve (8) and (9). The theorem of Weierstrass asserts that any continuous function can be approximated by a polynomial and that approximation is arbitrarily close as we let the degree of the polynomial increase.

There are many ways to approximate these functions with polynomials. The reader is referred to Judd (1998) for complete descriptions of several of them and comparisons among them. In this exercise, we will focus on one particular method because of its ease of use and its accuracy relative to other methods. This method is *Chebyshev collocation*. One of the most important steps is the computation of Chebyshev zeros. This is outlined in the next subsection. The steps involved in Chebyshev collocation are as follows:

1. Choose the degree of the approximating polynomials, and substitute $\hat{V}=\sum_{j=0}^n a_j x^j$ and $\hat{u}=\sum_{j=0}^n b_j x^j$ into equations (8) and (9).
2. Choose an interval on which you want to approximate the function. Compute the Chebyshev zeros on that interval. The number of Chebyshev zeros is equal to n .
3. Define the residual function $R_{i,j}(x_j, a, b)$ where $i=1,2$ (the number of equations) and $j=0\dots n$ to be the left hand sides of equations (8) and (9) with \hat{V} and \hat{u} in place of V and u , and where x_j represent the Chebyshev

zeros.

4. Solve the system of equations $R_{i,j}(x_j, a, b) = 0$ for all i and j . This is a system of $2n$ equations. The $2n$ variables are the a_j and b_j polynomial coefficients.

5. Verify that the approximation is acceptable by computing the residual throughout the interval. If the residual is small everywhere on the interval (10^{-6} is the criterion we will use in these examples), then we are done. If the residual exceeds that criterion at any point, then the approximation will not be acceptable at that point. We should either try again with a shorter interval, or increase the degree of the polynomials.

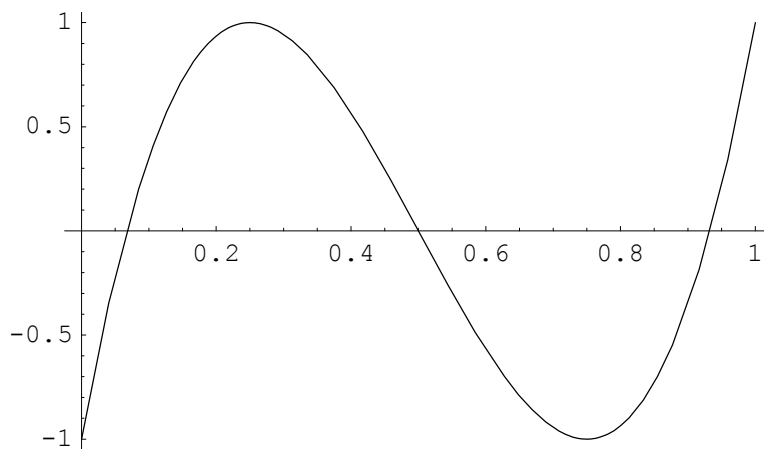
■ 2.2. Chebyshev zeros

Again, Judd (1998) provides a good reference, and the interested reader is referred there for more details. Here is how to put it to use. There is a theorem that says that for continuous functions, if we interpolate a function between the zeros of a Chebyshev polynomial over certain interval, then the polynomial will be a good approximation of the true function on that interval. The Chebyshev zeros are roots of the equation below.

$$\cos(n \cos^{-1}((2x - m_1 - m_2) / (m_1 - m_2))) = 0 \quad (10)$$

In (10), m_1 is the upper bound of the interval, m_2 is the lower bound, and n is the degree of the polynomials. For example, the left hand side of (10) for the interval $[0,1]$ and a 3rd degree polynomial can be plotted by *Mathematica*.

```
Plot[Cos[3 ArcCos[(2 x - 1 - 0) / (1 - 0)]], {x, 0, 1}]
```



- Graphics -

In general, this function oscillates between 0 and 1 over the interval, crossing the axis n times. A simple slice of *Mathematica* code can find the Chebyshev zeros to whatever decimal place desired. One could do this more efficiently, but this gives you a feel for what to do.

```

Clear[cheb];
chebyshev = Array[cheb, 3];
z[x_] = Cos[3 ArcCos[(2 x - 1 - 0) / (1 - 0)]];
j = 1;
step = .00005;
Timing[Do[
  cheb[j] = If[Sign[z[i]] ≠ Sign[z[i + step]], Round[10000 i] / 10000, 0];
  j = If[Sign[z[i]] ≠ Sign[z[i + step]], j + 1, j], {i, 0, 1, step}];]
Print[chebyshev];

{3.63 Second, Null}

```

$$\left\{ \frac{67}{1000}, \frac{1}{2}, \frac{933}{1000} \right\}$$

As you can see, this took less than 4 seconds on a 600MHz Pentium III. Run this on your computer and see how long it takes.

To the nearest 1/10,000 the Chebyshev zeros are [0.0670, 0.5000, 0.9330].

■ 2.3. A simple example

This example is taken from Judd (1998) so that we may verify his results. We let $\gamma = -2$, $\rho = 0.05$, $\alpha = 0.25$, and $\delta = 0$. The steady state of this optimal growth model is given by x^* which satisfies $y'(x^*) = \rho$. In this case, $x^* = 1$, so we will choose an interval centered on 1, namely [0.9, 1.1]. Note that steady state consumption in this problem will be 0.2.

Let $n=7$. The Chebyshev zeros are found by the *Mathematica* code.

```

Clear[cheb];
upper = 1.1;
lower = .9;
n = 7;
chebyshev = Array[cheb, n];
z[x_] = Cos[n ArcCos[(2 x - lower - upper) / (upper - lower)]];
j = 1;
step = .00005;
Timing[Do[
  cheb[j] = If[Sign[z[i]] ≠ Sign[z[i + step]], Round[10000 i] / 10000, 0];
  j = If[Sign[z[i]] ≠ Sign[z[i + step]], j + 1, j],
  {i, lower, upper, step}];]
Print[chebyshev];

{0.77 Second, Null}

```

$$\left\{ \frac{361}{400}, \frac{4609}{5000}, \frac{4783}{5000}, 1, \frac{5217}{5000}, \frac{5391}{5000}, \frac{5487}{5000} \right\}$$

With the Chebyshev zeros in hand, we compute the approximation. Here is the *Mathematica* code. Don't execute this segment of code until you have read about the start variable that I have used in the FindRoot

function and have executed those lines of code. I'll explain that variable at the end of the results. I'm putting the cart ahead of the horse just a bit, but with good reason. I want you to see the result and then take you back through the gory details of how to get there. It helps to have a clear picture of where you're going before you take that first step.

```

ρ = .05; α = .25;
ff[x_] = 0;
res = Array[ff, 14];
aa = Array[a, 7];
bb = Array[b, 7];
a[1] = a1; a[2] = a2; a[3] = a3; a[4] = a4; a[5] = a5; a[6] = a6;
a[7] = a7; a[8] = a8; a[9] = a9; b[1] = b1; b[2] = b2; b[3] = b3;
b[4] = b4; b[5] = b5; b[6] = b6; b[7] = b7; b[8] = b8; b[9] = b9;
yhprime[aa_, bb_] = Sum[aa[[j]] (j - 1) x^(j - 2), {j, 2, 7}];
yhat[aa_, bb_] = Sum[aa[[j]] x^(j - 1), {j, 1, 7}];
chat[aa_, bb_] = Sum[bb[[j]] x^(j - 1), {j, 1, 7}];
Do[
  res[[i]] = (ρ yhat[aa, bb] + 1 / chat[aa, bb] - (ρ (x^α) / α - chat[aa, bb])
    (yhprime[aa, bb])) /. {x → chebyshev[[i]]}, {i, 1, 7}];
Do[res[[i]] = (chat[aa, bb] - yhprime[aa, bb]^(-.5)) /.
  {x → chebyshev[[i - 7]]}, {i, 8, 14}];
system = Array[ff, 14];
Do[system[[i]] = (res[[i]] == 0), {i, 1, 14}];
sol = FindRoot[system, {a1, start[[1]]},
  {a2, start[[2]]}, {a3, start[[3]]}, {a4, start[[4]]},
  {a5, start[[5]]}, {a6, start[[6]]}, {a7, start[[7]]},
  {b1, start[[8]]}, {b2, start[[9]]}, {b3, start[[10]]},
  {b4, start[[11]]}, {b5, start[[12]]}, {b6, start[[13]]},
  {b7, start[[14]]}, AccuracyGoal → 14, MaxIterations → 2000]
v[x_] = yhat[aa, bb] /. sol
vprime[x_] = yhprime[aa, bb] /. sol
c[x_] = chat[aa, bb] /. sol
start = {a1, a2, a3, a4, a5, a6, a7, b1, b2, b3, b4, b5, b6, b7} /. sol;

{a1 → -153.908, a2 → 119.905, a3 → -134.172, a4 → 113.463, a5 → -62.1759,
a6 → 19.5757, a7 → -2.68738, b1 → 0.124727, b2 → -0.12962, b3 → 0.714715,
b4 → -1.05431, b5 → 0.825293, b6 → -0.337932, b7 → 0.057123}

-153.908 + 119.905 x - 134.172 x2 +
113.463 x3 - 62.1759 x4 + 19.5757 x5 - 2.68738 x6

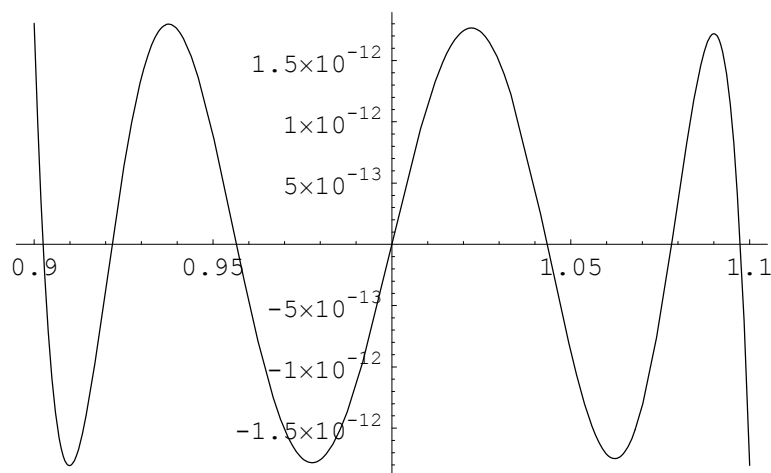
119.905 - 268.343 x + 340.388 x2 - 248.704 x3 + 97.8784 x4 - 16.1243 x5

0.124727 - 0.12962 x + 0.714715 x2 -
1.05431 x3 + 0.825293 x4 - 0.337932 x5 + 0.057123 x6

```

As you can see, the code found parameters a_i and b_i such that \hat{V} and \hat{u} are good approximations for the value function and the consumption function. Below those parameters are listed \hat{V} , \hat{V}' , and \hat{u} . I have renamed u to be c to remind us of its interpretation as a consumption function. The final step is to verify that the approximation is good throughout the interval. We want to verify that the errors in (8) and (9) as a percentage of steady state $r'(x)$ and steady state $\rho\hat{V}(x)$ respectively are small. To verify the first part, note that $r'(x)$ at the steady state is the square of the reciprocal of consumption at the steady state, 25 in this case. Substitute the parameters (sol) into the differential equation, divide by 25, and plot it over the interval.

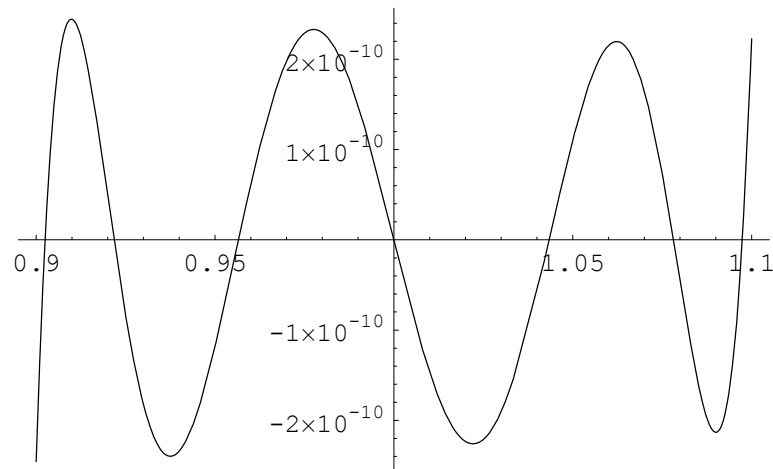
```
Plot[((chat[aa, bb] /. sol) (.04) - .04 (yhprime[aa, bb] /. sol) ^ (-.5)),
{x, .9, 1.1}]
```



- Graphics -

The approximation is quite good -- within 2×10^{-12} -- which is acceptable. Now we check equation (9).

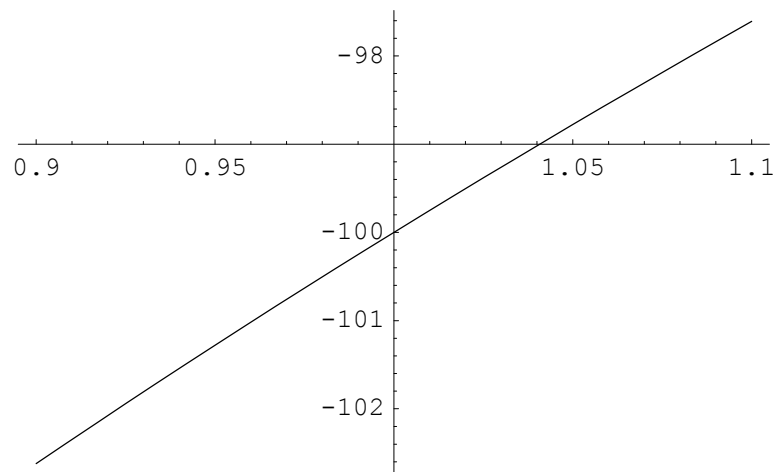
```
Plot[( $\rho$  (yhat[aa, bb] /. sol) / ( $\rho$  ((yhat[aa, bb] /. sol) /. x  $\rightarrow$  1)) +
  1 / ( $\rho$  ((yhat[aa, bb] /. sol) /. x  $\rightarrow$  1) (chat[aa, bb] /. sol)) -
  (.2 x.25 - chat[aa, bb] /. sol) (yhprime[aa, bb] /. sol) /
  ( $\rho$  (yhat[aa, bb] /. sol) /. x  $\rightarrow$  1)), {x, .9, 1.1}]
```



- Graphics -

It is within 3×10^{-10} which is also in the acceptable range. Because the AccuracyGoal parameter was set to 14, Newton's method requires 14 significant digits of accuracy. That is the accuracy at the Chebyshev zeros. Now, let's plot the functions. First the value function:

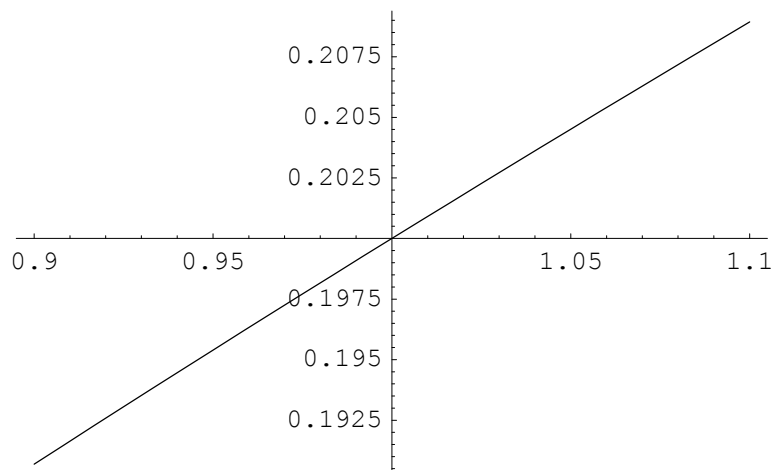
```
Plot[v[x], {x, .9, 1.1}]
```



- Graphics -

The consumption function:

```
Plot[c[x], {x, .9, 1.1}]
```



- Graphics -

Things appear to be in order. Now for a few words about the start variable. The essential part of this method is solving a system of nonlinear equations for the polynomial coefficients. This is far from a trivial task. Fortunately *Mathematica* has a very powerful function called `FindRoot` which can solve for the roots of polynomial equations. `FindRoot` uses Newton's method. Of course, Newton's method can be difficult to apply because you may end up converging to complex roots. The initial guess determines the root to which you will converge. When the degree of the equations is small, this is not a problem. The traditional example of using Newton's method to solve $x^3+1=0$ for all three of its complex roots is easy to do because the basins of attraction are fairly simple.

```
FindRoot[x^3 + 1 == 0, {x, -.9}]
```

```
{x → -1.}
```

```
FindRoot[x^3 + 1 == 0, {x, .5 + I}]
```

```
{x → 0.5 + 0.866025 i}
```

```
FindRoot[x^3 + 1 == 0, {x, .5 - I}]
```

```
{x → 0.5 - 0.866025 i}
```

Of course if the initial guess is near the boundary between basins of attraction, the path to convergence may be long, and it may converge to a solution that you don't expect. See Gleick (1987).

But the point is clear. For polynomials of degree 3, Newton's method is quite workable. For polynomials of degree 6, it may be entirely unclear which of 6 complex roots to which you will converge based on your initial guess -- or if you will converge at all. The initial guess is as much art as science, but there are ways to make it a little more scientific. The way that I solved this problem was to begin with a much smaller interval. First I considered $[0.99, 1.01]$ and started with an initial guess (which I put into the start variable):

```
start = {-130, 100, -70, 30, -15, 1, -1, 1, -1, .5, -.3, .1, -.1, .1}
{-130, 100, -70, 30, -15, 1, -1, 1, -1, 0.5, -0.3, 0.1, -0.1, 0.1}
```

$$\hat{V}(x) = -130 + 100x - 70x^2 + 30x^3 - 15x^4 + x^5 - x^6$$

$$\hat{U}(x) = 1 - x + .5x^2 - .3x^3 + .1x^4 - .1x^5 + .1x^6$$

This worked. Several other guesses that I tried first gave complex roots or failed to converge, yielding solutions that made no sense economically. The idea is to make an educated guess about the coefficients and try several possibilities. For example, I had a strong suspicion that the signs of the coefficients would alternate and that the coefficients would get small as the degree of the term increased. Take whatever information you know about the function or that you could logically guess about it and try it. Keep in mind that if the start value causes any of the expressions to be complex, the program will try to find a complex root. If the start value yields real numbers in all of the expressions in the system of equations, it will search for a real root. Take care to choose start values that do not cause your equations to have imaginary numbers (e.g. a start value that makes $\text{yprime}[aa,bb] < 0$ at any of the Chebyshev zeros is not a good start value).

Actually, the numerical solution that the program produced was complex, but the imaginary term was reported to be on the order of 10^{-20} . I dropped the imaginary part and used the real part as the initial guess on the next try.

Then, I expanded the interval and used the solution from the smaller interval as the initial guess on the new interval. I gave "start" the following value:

```
start = {-153.84894966286586`, 119.59654226770418`,
-133.5206075136264`, 112.75548370268214`, -61.76695660790063`,
19.460878800926565`, -2.6763909869199836`, 0.12507426811107736`,
-0.1303352383404009`, 0.7140344503611584`, -1.0513983249653749`,
0.8225881326155545`, -0.33703229152433856`, 0.05706900374229135`}
```

Then I computed Chebyshev zeros on the interval [0.95, 1.05] and ran the program again. I took the solution to this problem as the initial guess for the interval [0.9, 1.1]. The "start" value I used was:

```
start = {-153.84147891969795`, 119.54052455896426`,
-133.352498499471`, 112.49383232777218`, -61.54254070701384`,
19.359829138438588`, -2.6576678989922433`, 0.12488347000463412`,
-0.12900912494853448`, 0.7100416352010236`, -1.0448711458897002`,
0.8165645946236293`, -0.33408127083554`, 0.056471841790396125`}
```

Convergence was almost immediate because, as you can see, the guess was pretty close. When dealing with smooth functions, take note that the coefficients of the approximating polynomial will not change very much when changing the interval by a small amount. Exploit this fact when computing your approximations.

■ 2.4. But how good is it really?

Let's briefly compare the result that we obtained in the previous example to another method of obtaining the consumption function. Then we will apply this method to the "cake eating" problem of section 1. For that problem, we know the true solution and can test the accuracy of our method.

■ 2.4.1. Dynamic programming versus a direct solution of the consumption function

The purpose of this comparison is not to claim that either method is better than the other, but to show that they produce the same result. The modeler is the only one who can determine which method really is better for the problem in question. Different methods have advantages and disadvantages for different problems.

Consider the problem

$$V(x(t)) = \max_u E_t \int_t^{\infty} e^{-\rho(\tau-t)} r(x(\tau), u(\tau)) d\tau$$

subject to

$$dx = (f(x) - u) dt$$

This can be solved by solving one differential equation.

$$u'(x) (f(x) - u(x)) - \frac{r'(x)}{r''(x)} (\rho - f'(x)) = 0 \quad (11)$$

See Judd (1998) or do this as an exercise. Hint: this does not use dynamic programming methods.

We modify the code from the previous example slightly to accommodate this new single equation. We will again use a 6th degree polynomial. Before running this example, I recomputed the previous example with an interval of [0.25, 1.75] and used the consumption function obtained there as the initial guess in this problem. The results will be compared. Note: here is the notation for the functions plotted:

c = the function from the previous exercise

c2 = the function that was obtained from expanding the interval from the previous exercise to [0.25, 1.75] (not shown: left as an exercise)

c3 = the function computed from solving (11).

This was the start value that was used.

```
start = {-160.567221896244`, 164.87112698437852`,
  -253.2304203784646`, 270.5840857885814`, -171.5528341248298`,
  57.9011116877548`, -7.996892378634976`, 0.1090278528550018`,
  -0.20884567387179243`, 1.4133772991055695`, -2.619379069352586`,
  2.361738946259268`, -1.0329515909561`, 0.17515729005070577`}
```

Only the last 7 elements are relevant.

Compute the Chebyshev zeros.

```

Clear[cheb];
upper = 1.75;
lower = .25;
n = 7;
chebyshev = Array[cheb, n];
z[x_] = Cos[n ArcCos[(2 x - lower - upper) / (upper - lower)]];
j = 1;
step = .000005;
Timing[
  Do[cheb[j] = If[Sign[z[i]] ≠ Sign[z[i + step]], Round[100000 i] / 100000,
    0]; j = If[Sign[z[i]] ≠ Sign[z[i + step]], j + 1, j],
    {i, lower, upper, step}];]
Print[chebyshev];

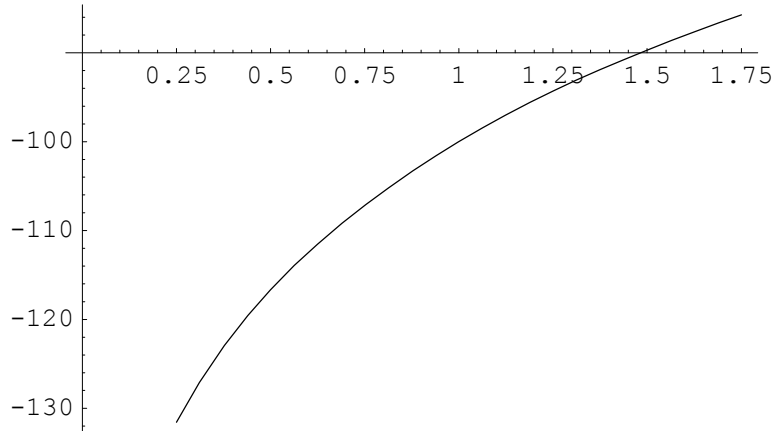
{39.44 Second, Null}

{ $\frac{168}{625}$ ,  $\frac{20681}{50000}$ ,  $\frac{33729}{50000}$ , 1,  $\frac{132541}{100000}$ ,  $\frac{158637}{100000}$ ,  $\frac{1082}{625}$ }

```

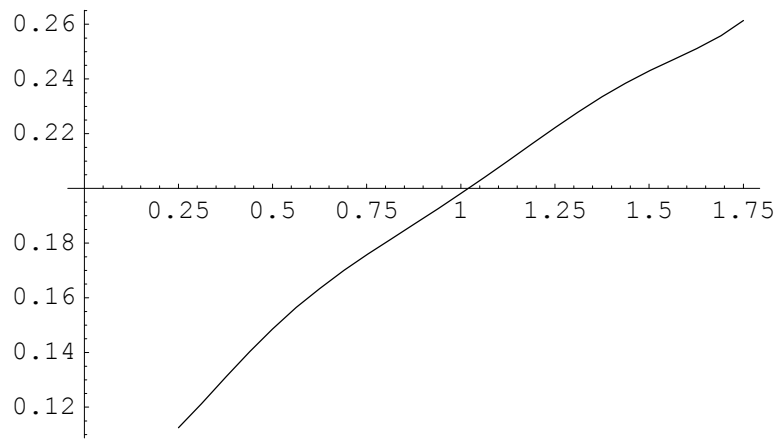
In case you're interested, here are the value function and consumption function for the dynamic program over the interval [0.25, 1.75].

```
Plot[v2[x], {x, .25, 1.75}]
```



- Graphics -

```
Plot[c2[x], {x, .25, 1.75}]
```



- Graphics -

Here is the code for the alternative method.

```

ρ = .05; α = .25;
ff[x_] = 0;
res = Array[ff, 14];
aa = Array[a, 7];
bb = Array[b, 7];
a[1] = a1; a[2] = a2; a[3] = a3; a[4] = a4; a[5] = a5; a[6] = a6;
a[7] = a7; a[8] = a8; a[9] = a9; b[1] = b1; b[2] = b2; b[3] = b3;
b[4] = b4; b[5] = b5; b[6] = b6; b[7] = b7; b[8] = b8; b[9] = b9;
chprime[aa_, bb_] = Sum[bb[[j]] (j - 1) x^(j - 2), {j, 2, 7}];
chat[aa_, bb_] = Sum[bb[[j]] x^(j - 1), {j, 1, 7}];
Do[res[[i]] =
  (chprime[aa, bb] (ρ (x^α) / α - chat[aa, bb]) + (.5 chat[aa, bb])
    (ρ) (1 - x^(α - 1))) /. {x → chebyshev[[i]]}, {i, 1, 7}];
system = Array[ff, 7];
Do[system[[i]] = (res[[i]] == 0), {i, 1, 7}];
Timing[sol = FindRoot[system, {b1, start[[8]]},
  {b2, start[[9]]}, {b3, start[[10]]}, {b4, start[[11]]},
  {b5, start[[12]]}, {b6, start[[13]]}, {b7, start[[14]]},
  AccuracyGoal → 13, MaxIterations → 2000];]
Print[sol];
c3[x_] = chat[aa, bb] /. sol
start = {a1, a2, a3, a4, a5, a6, a7, b1, b2, b3, b4, b5, b6, b7} /. sol

{0.06 Second, Null}

{b1 → 0.0563213, b2 → 0.28807, b3 → -0.338608,
 b4 → 0.348586, b5 → -0.216729, b6 → 0.0722617, b7 → -0.00990247}

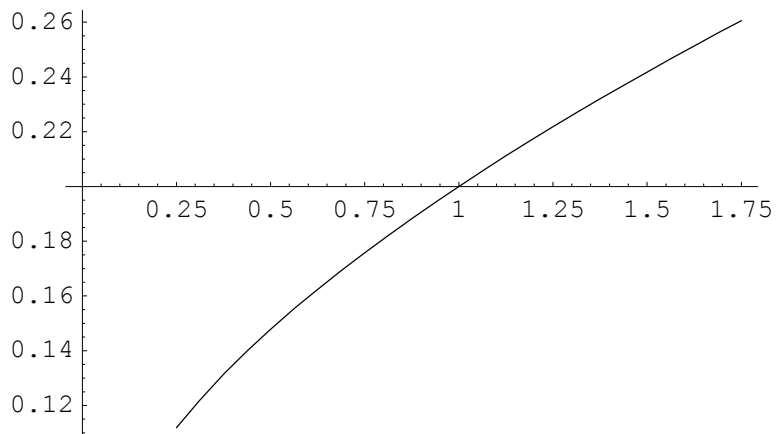
0.0563213 + 0.28807 x - 0.338608 x2 +
 0.348586 x3 - 0.216729 x4 + 0.0722617 x5 - 0.00990247 x6

{a1, a2, a3, a4, a5, a6, a7, 0.0563213, 0.28807,
 -0.338608, 0.348586, -0.216729, 0.0722617, -0.00990247}

```

Since our guess was quite good, convergence was almost immediate. We plot the consumption function that results from that approximation....

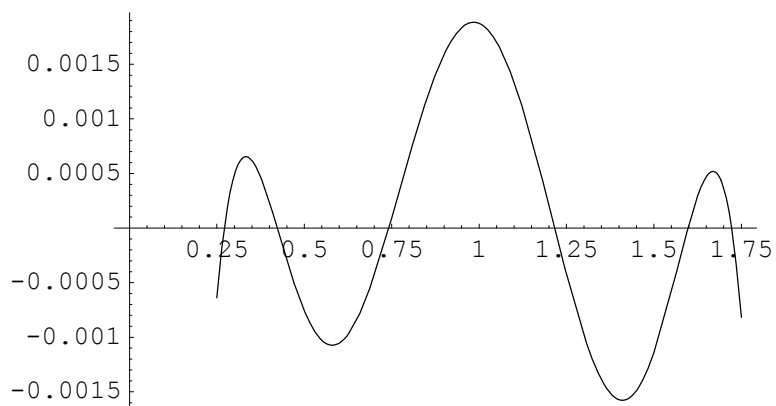
```
Plot[c3[x], {x, .25, 1.75}]
```



- Graphics -

...and compare it to the dynamic programming method. They are within 0.002 of each other throughout the interval. That's actually quite good for only having a six degree polynomial approximation. Increase the degree to improve the accuracy of *either* method.

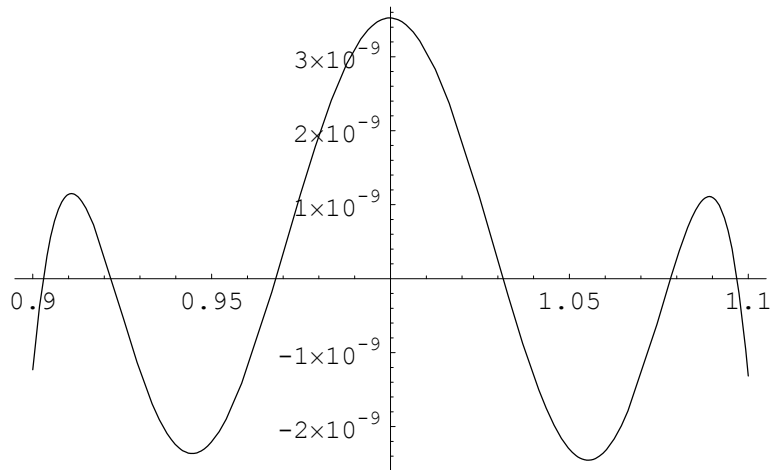
```
Plot[c3[x] - c2[x], {x, .25, 1.75}]
```



- Graphics -

You might be curious to see how the methods compare when we compare them on a smaller interval. Let $c4[x]$ be the consumption function of the alternate method. We compare that to the $c[x]$ found above.

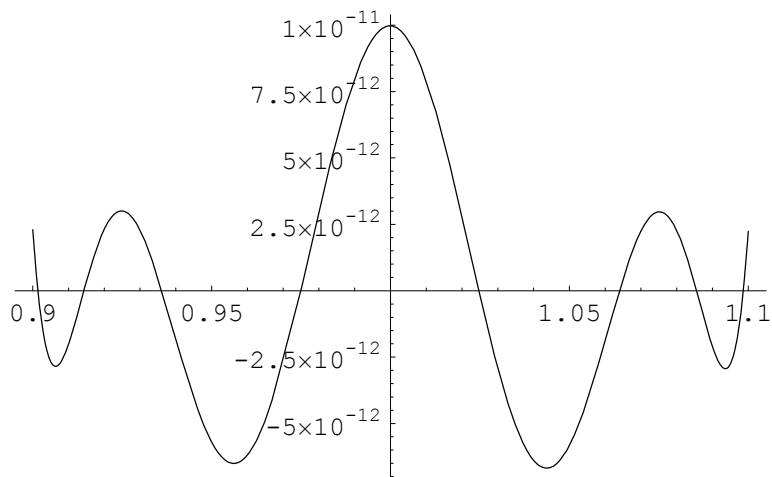
```
Plot[c4[x] - c[x], {x, .9, 1.1}]
```



- Graphics -

Obviously, the approximation is much better over the smaller interval.

The careful reader will notice that in the code there were statements defining $a[8]$, $a[9]$, $b[8]$, and $b[9]$ to facilitate just making a few modifications to try this exercise with polynomials of degree 8. Not much is gained by repeating the code or redrawing the functions. However, we are interested in how the difference between the two methods decreases when using polynomials of degree 8 in both methods.



- Graphics -

The difference is significantly smaller.

■ 2.4.2. Cake eating revisited

Now, we will compare the result obtained through the approximation method to the *true* value function and consumption function which we know. Of course, one can only do this for a very limited set of problems. The purpose of the exercise is to show just how good the method can be. Refer to section 1.3. for the basic problem. One differential equation is:

$$\frac{1}{u} + (-1) \frac{dV}{dx} = 0$$

$$u = \left(\frac{dV}{dx} \right)^{-1}$$

We eschew the substitution made in section 1.3. (it is not needed to solve the problem numerically), and the Bellman equation becomes:

$$\rho V(x) - \ln(u) + u \frac{dV}{dx} = 0$$

The code is modified for this problem. Let $\rho=0.1$ and $n=7$. There is no steady state (even at $x=0$, $\ln(0)$ is undefined), so we choose some arbitrary interval, $[0.45, 0.55]$ and first compute the Chebyshev zeros.

```
Clear[cheb];
upper = .55;
lower = .45;
n = 7;
chebyshev = Array[cheb, n];
z[x_] = Cos[n ArcCos[(2 x - lower - upper) / (upper - lower)]];
j = 1;
step = .00005;
Timing[Do[
  cheb[j] = If[Sign[z[i]] != Sign[z[i + step]], Round[10000 i] / 10000, 0];
  j = If[Sign[z[i]] != Sign[z[i + step]], j + 1, j],
  {i, lower, upper, step}];]
Print[chebyshev];

{0.38 Second, Null}
```

```
{ 282, 4609, 4783, 1, 5217, 539, 5487 }
{ 625, 10000, 10000, 2, 10000, 1000, 10000 }
```

There are a couple ways to proceed with a start value. We could start with a reasonable guess that the solution for \hat{V} will be closely related to the Taylor series expansion of $\ln(x)$ around the point 0.5. *Mathematica* can find this for us.

```
Collect[Series[Log[x], {x, .5, 6}], x]

-3.14315 + 12. x - 30. x2 + 53.3333 x3 - 60. x4 + 38.4 x5 - 10.6667 x6
```

Of course, if we "cheat" and look at the true solution, we might want to start with a guess that takes the Taylor series of the true solution around the point 0.5.

```
Collect[Series[-1 / rho + Log[x] / rho + Log[rho] / rho, {x, .5, 6}], x]
-64.4573 + 120. x - 300. x^2 + 533.333 x^3 - 600. x^4 + 384. x^5 - 106.667 x^6
```

We could also just start with a vector of ones. I do not recommend starting with a vector of ones in most cases. It does not always work, and in large problems it might converge slowly. This, however, is a pretty simple problem, so it does work. If you don't know anything about the solution at all, it is worth trying a vector of ones and hope for the best. I include all three guesses below. You can verify that they all lead to the same solution. For the first two, note that I used the Taylor series to get the guess for \hat{V} , but put in a vector of ones for the consumption function. It is hard to come up with a good guess for the consumption function because the true function is linear. Yet, evaluated at x , it must yield the value of the reciprocal of our function "vprimecake" evaluated at x . Hence, the polynomial approximation will not be linear, and it is not immediately apparent what would make a good guess.

Note how close the second guess is to the actual function computed. The polynomial computed is *almost* a Taylor series approximation!

```
start = {-3.1415, 12, -30, 53.333, -60, 38.4, -10.667, 1, 1, 1, 1, 1, 1, 1}
start = {-64, 120, -300, 533, -600, 384, -106, 1, 1, 1, 1, 1, 1, 1}
start = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
```

```

ρ = .1;
ff[x_] = 0;
res = Array[ff, 14];
aa = Array[a, 7];
bb = Array[b, 7];
a[1] = a1; a[2] = a2; a[3] = a3; a[4] = a4; a[5] = a5; a[6] = a6;
a[7] = a7; a[8] = a8; a[9] = a9; b[1] = b1; b[2] = b2; b[3] = b3;
b[4] = b4; b[5] = b5; b[6] = b6; b[7] = b7; b[8] = b8; b[9] = b9;
yhprime[aa_, bb_] = Sum[aa[[j]] (j - 1) x^(j - 2), {j, 2, 7}];
yhat[aa_, bb_] = Sum[aa[[j]] x^(j - 1), {j, 1, 7}];
chat[aa_, bb_] = Sum[bb[[j]] x^(j - 1), {j, 1, 7}];
Do[res[[i]] =
  (ρ yhat[aa, bb] - Log[chat[aa, bb]] + (chat[aa, bb]) yhprime[aa, bb]) /.
  {x → chebyshev[[i]]}, {i, 1, 7}];
Do[res[[i]] = (chat[aa, bb] - yhprime[aa, bb]^(-1)) /.
  {x → chebyshev[[i - 7]]}, {i, 8, 14}];
system = Array[ff, 14];
Do[system[[i]] = (res[[i]] == 0), {i, 1, 14}];
system[[4]] = ((yhprime[aa, bb] /. {x → chebyshev[[4]}}) ==
  ((1 / (ρ x)) /. x → chebyshev[[4]]));
sol = FindRoot[system, {a1, start[[1]]}, {a2, start[[2]]},
  {a3, start[[3]]}, {a4, start[[4]]}, {a5, start[[5]]},
  {a6, start[[6]]}, {a7, start[[7]]}, {b1, start[[8]]},
  {b2, start[[9]]}, {b3, start[[10]]}, {b4, start[[11]]},
  {b5, start[[12]]}, {b6, start[[13]]}, {b7, start[[14]]},
  AccuracyGoal → 14, MaxIterations → 30000]
vcake[x_] = yhat[aa, bb] /. sol
vprimecake[x_] = yhprime[aa, bb] /. sol
ccake[x_] = chat[aa, bb] /. sol
start = {a1, a2, a3, a4, a5, a6, a7, b1, b2, b3, b4, b5, b6, b7} /. sol;

{a1 → -64.5241, a2 → 120.733, a3 → -303.321, a4 → 541.261, a5 → -610.487,
  a6 → 391.259, a7 → -108.708, b1 → 0.0500501, b2 → -0.504271,
  b3 → 3.03625, b4 → -8.12705, b5 → 12.222, b6 → -9.79127, b7 → 3.26452}

-64.5241 + 120.733 x - 303.321 x2 +
  541.261 x3 - 610.487 x4 + 391.259 x5 - 108.708 x6

120.733 - 606.642 x + 1623.78 x2 - 2441.95 x3 + 1956.3 x4 - 652.251 x5

0.0500501 - 0.504271 x + 3.03625 x2 -
  8.12705 x3 + 12.222 x4 - 9.79127 x5 + 3.26452 x6

```

Basically, the program is the same as the one used in section 2.3. You will notice one additional line of code:

```
system[[4]] = ((yhprime[aa, bb] /. {x → chebyshev[[4]]}) ==
  ((1 / (ρ x)) /. x → chebyshev[[4]]));
```

This additional line of code is necessary because of a fundamental difference in the two problems. In section 2.3, there is a steady state which pins down the value function. We can actually evaluate \hat{V} at the steady state. We don't really make use of that in the solution algorithm, but it turns out that the solution to the system of equations is unique. This is not the case in the cake eating problem. Recall in section 1.3 that after solving for x , we let the constant term be zero. All we wanted was a specification of the value function up to an affine transformation. So we need to pin down the value function, and we do so by forcing the derivative of the value function to evaluate to a given constant at some point. The point we choose, is the 4th of the 7 Chebyshev zeros (the middle one -- 0.5 in this case) and the constant we impose is what we obtain by evaluating the derivative of the true value function at 0.5. That is what this line of code does. Use it instead of one of the usual equations associated with the point 0.5 (e.g. `system[[4]]`). Note that we do not have to know the true value function to obtain a solution. We could impose any constant we want and obtain a valid solution because the value function is only determined up to an affine transformation in this problem. (As an exercise, put in another constant and see what results.)

It is important to recognize the need to pin down the value function in this way. The program will not converge without it.

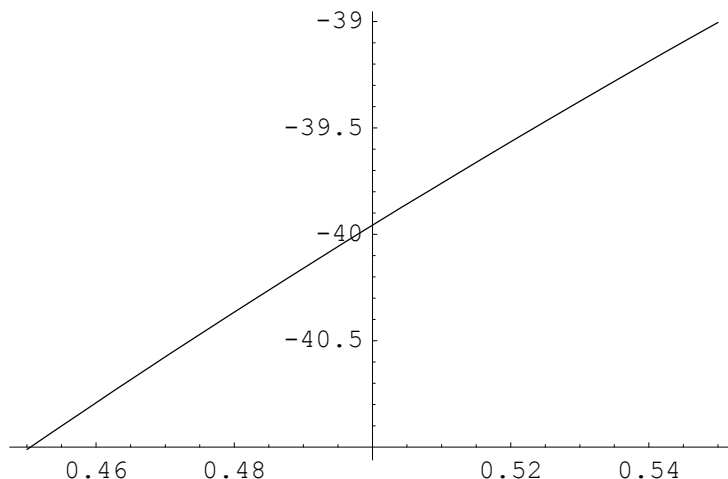
Recall the true value function from section 1.3.

$$V(x) = \frac{-1 + \ln(x\rho)}{\rho}$$

The derivative $V'(x)$ is equal to $(x\rho)^{-1}$.

As before, we plot the true value, but this time only over the interval in question.

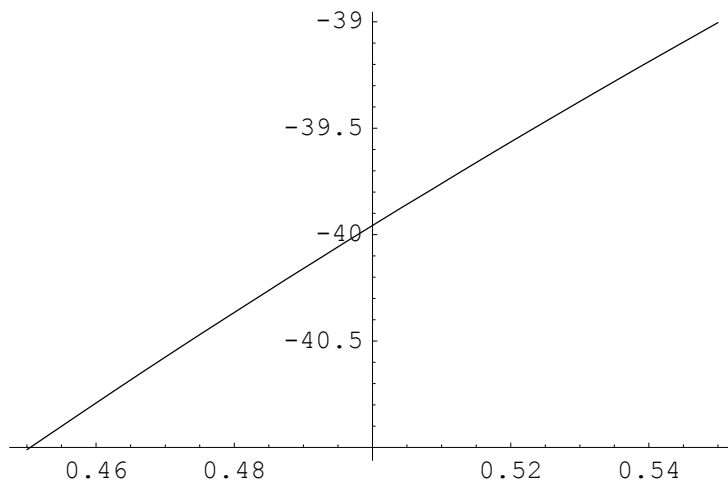
```
Plot[ $\frac{-1 + \text{Log}[x \rho]}{\rho}$ , {x, .45, .55}]
```



- Graphics -

Now plot the value function computed by the program.

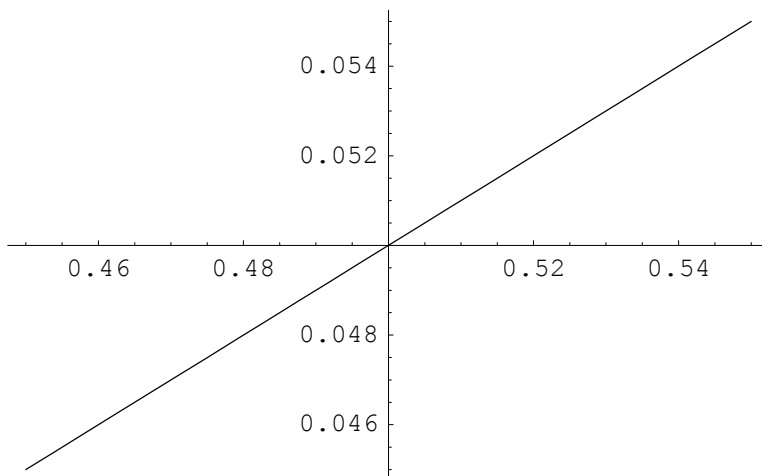
```
Plot[vcake[x], {x, .45, .55}]
```



- Graphics -

The consumption function:

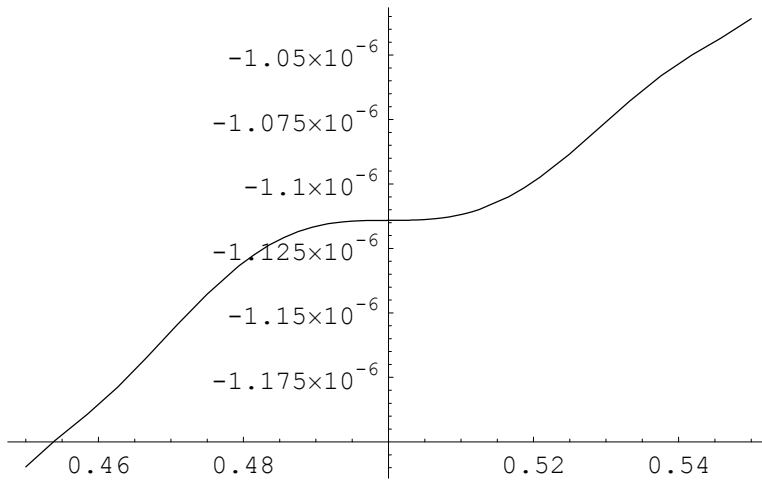
```
Plot[ccake[x], {x, .45, .55}]
```



- Graphics -

Now plot the difference between the two.

```
Plot[ $\frac{\text{Log}[\rho] - 1 + \text{Log}[\mathbf{x}]}{\rho} - \text{vcake}[\mathbf{x}], \{\mathbf{x}, .45, .55\}, \text{PlotRange} \rightarrow \text{All}]$ 
```



- Graphics -

If our criterion was 10^{-6} , we exceed it by a little bit. Our point having been made, the interested reader is left to see how much of an improvement could be made by using higher order polynomials.

Just to see how the affine transformation works, try the following line of code:

```
system[[4]] = (yhprime[aa, bb] /. {x → chebyshev[[4]}) == 10);
```

This is equivalent to a leftward shift of 0.5.

```

ρ = .1;
ff[x_] = 0;
res = Array[ff, 14];
aa = Array[a, 7];
bb = Array[b, 7];
a[1] = a1; a[2] = a2; a[3] = a3; a[4] = a4; a[5] = a5; a[6] = a6;
a[7] = a7; a[8] = a8; a[9] = a9; b[1] = b1; b[2] = b2; b[3] = b3;
b[4] = b4; b[5] = b5; b[6] = b6; b[7] = b7; b[8] = b8; b[9] = b9;
yhprime[aa_, bb_] = Sum[aa[[j]] (j - 1) x^(j - 2), {j, 2, 7}];
yhat[aa_, bb_] = Sum[aa[[j]] x^(j - 1), {j, 1, 7}];
chat[aa_, bb_] = Sum[bb[[j]] x^(j - 1), {j, 1, 7}];
Do[res[[i]] =
  (ρ yhat[aa, bb] - Log[chat[aa, bb]] + (chat[aa, bb]) yhprime[aa, bb]) /.
  {x → chebyshev[[i]]}, {i, 1, 7}];
Do[res[[i]] = (chat[aa, bb] - yhprime[aa, bb]^(-1)) /.
  {x → chebyshev[[i - 7]]}, {i, 8, 14}];
system = Array[ff, 14];
Do[system[[i]] = (res[[i]] == 0), {i, 1, 14}];
system[[4]] = ((yhprime[aa, bb] /. {x → chebyshev[[4]}}) == 10);
sol = FindRoot[system, {a1, start[[1]]},
  {a2, start[[2]]}, {a3, start[[3]]}, {a4, start[[4]]},
  {a5, start[[5]]}, {a6, start[[6]]}, {a7, start[[7]]},
  {b1, start[[8]]}, {b2, start[[9]]}, {b3, start[[10]]},
  {b4, start[[11]]}, {b5, start[[12]]}, {b6, start[[13]]},
  {b7, start[[14]]}, AccuracyGoal → 14, MaxIterations → 30000]
vcake[x_] = yhat[aa, bb] /. sol
vprimecake[x_] = yhprime[aa, bb] /. sol
ccake[x_] = chat[aa, bb] /. sol
start = {a1, a2, a3, a4, a5, a6, a7, b1, b2, b3, b4, b5, b6, b7} /. sol;

{a1 → -39.9377, a2 → 19.6917, a3 → -17.831,
 a4 → 17.542, a5 → -13.8022, a6 → 7.03297, a7 → -1.67473,
 b1 → 0.0515417, b2 → 0.0813878, b3 → 0.0935123,
 b4 → -0.250281, b5 → 0.376356, b6 → -0.301482, b7 → 0.100509}

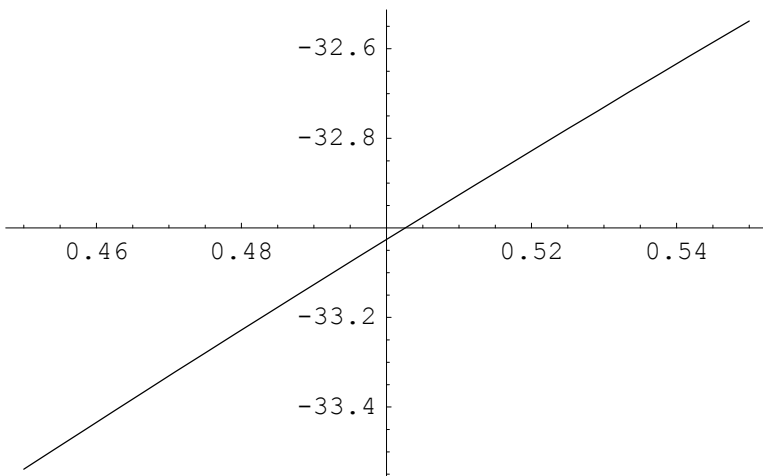
-39.9377 + 19.6917 x - 17.831 x2 +
 17.542 x3 - 13.8022 x4 + 7.03297 x5 - 1.67473 x6

19.6917 - 35.6619 x + 52.6261 x2 - 55.2086 x3 + 35.1648 x4 - 10.0484 x5

0.0515417 + 0.0813878 x + 0.0935123 x2 -
 0.250281 x3 + 0.376356 x4 - 0.301482 x5 + 0.100509 x6

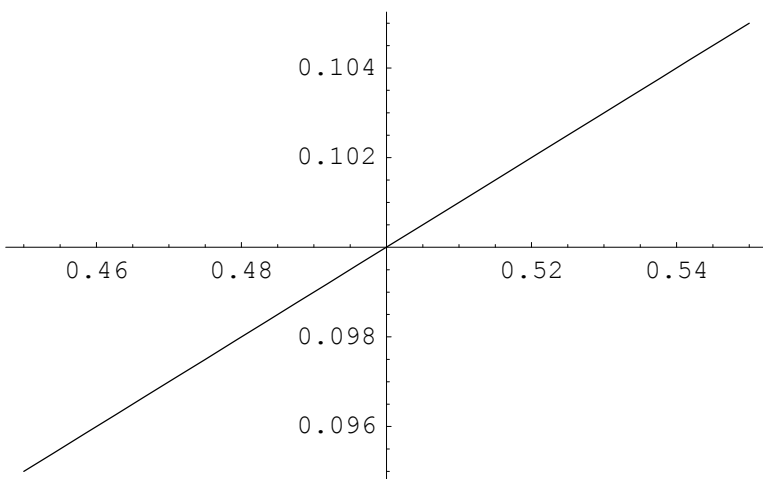
```

```
Plot[vcake[x], {x, .45, .55}]
```



- Graphics -

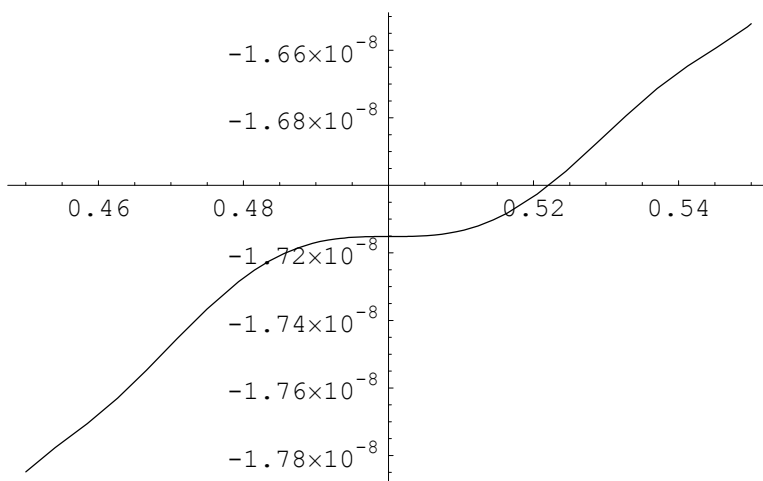
```
Plot[ccake[x], {x, .45, .55}]
```



- Graphics -

The solution is very close to the true value function shifted left by 0.5.

```
Plot[ $\frac{-1 + \text{Log}[\rho (x + .5)]}{\rho} - \text{vcake}[x], \{x, .45, .55\}, \text{PlotRange} \rightarrow \text{All}]$ 
```



- Graphics -

Even if we did not know the true value function, we would still arrive at a solution. One still must be careful, however. Depending on the interval chosen we might get a solution that doesn't make economic sense. For example, consider running the following program which solves the problem over the interval $[-0.005, 0.105]$. If we imposed the condition that the value function is $\ln(x)$ and not $\ln(x-c)$, the function should not be defined on this interval. But this line of code puts in a shift.

```
system[[4]] = ((yhprime[aa, bb] /. {x → chebyshev[[4] ]}) == 10) ;
```

This shift is a leftward shift of 0.95. The value function will be defined from $(-0.95, \infty)$, and you can see the shift. (Look at the last line and the final graph.)

```

Clear[cheb];
upper = .105;
lower = -.005;
n = 7;
chebyshev = Array[cheb, n];
z[x_] = Cos[n ArcCos[(2 x - lower - upper) / (upper - lower)]];
j = 1;
step = .00005;
Timing[Do[
  cheb[j] = If[Sign[z[i]] ≠ Sign[z[i + step]], Round[10000 i] / 10000, 0];
  j = If[Sign[z[i]] ≠ Sign[z[i + step]], j + 1, j],
  {i, lower, upper, step}];]
Print[chebyshev];
start = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
ρ = .1;
ff[x_] = 0;
res = Array[ff, 14];
aa = Array[a, 7];
bb = Array[b, 7];
a[1] = a1; a[2] = a2; a[3] = a3; a[4] = a4; a[5] = a5; a[6] = a6;
a[7] = a7; a[8] = a8; a[9] = a9; b[1] = b1; b[2] = b2; b[3] = b3;
b[4] = b4; b[5] = b5; b[6] = b6; b[7] = b7; b[8] = b8; b[9] = b9;
yhprime[aa_, bb_] = Sum[aa[[j]] (j - 1) x^(j - 2), {j, 2, 7}];
yhat[aa_, bb_] = Sum[aa[[j]] x^(j - 1), {j, 1, 7}];
chat[aa_, bb_] = Sum[bb[[j]] x^(j - 1), {j, 1, 7}];
Do[res[[i]] =
  (ρ yhat[aa, bb] - Log[chat[aa, bb]] + (chat[aa, bb]) yhprime[aa, bb]) /.
  {x → chebyshev[[i]]}, {i, 1, 7}];
Do[res[[i]] = (chat[aa, bb] - yhprime[aa, bb]^(-1)) /.
  {x → chebyshev[[i - 7]]}, {i, 8, 14}];
system = Array[ff, 14];
Do[system[[i]] = (res[[i]] == 0), {i, 1, 14}];
system[[4]] = ((yhprime[aa, bb] /. {x → chebyshev[[4]]) == 10);
sol = FindRoot[system, {a1, start[[1]]},
  {a2, start[[2]]}, {a3, start[[3]]}, {a4, start[[4]]},
  {a5, start[[5]]}, {a6, start[[6]]}, {a7, start[[7]]},
  {b1, start[[8]]}, {b2, start[[9]]}, {b3, start[[10]]},
  {b4, start[[11]]}, {b5, start[[12]]}, {b6, start[[13]]},
  {b7, start[[14]]}, AccuracyGoal → 14, MaxIterations → 30000]
vcake[x_] = yhat[aa, bb] /. sol
vprimecake[x_] = yhprime[aa, bb] /. sol
ccake[x_] = chat[aa, bb] /. sol
start = {a1, a2, a3, a4, a5, a6, a7, b1, b2, b3, b4, b5, b6, b7} /. sol;
Plot[vcake[x], {x, -.005, .105}]
Plot[ccake[x], {x, -.005, .105}]
Plot[ $\frac{-1 + \text{Log}[\rho (x + .95)]}{\rho}$  - vcake[x], {x, -.005, .105}, PlotRange → All]

```

{0.44 Second, Null}

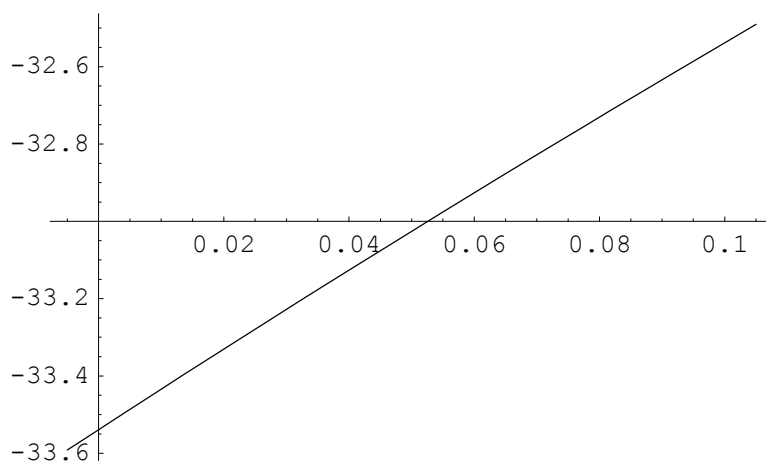
$\left\{-\frac{9}{2500}, \frac{7}{1000}, \frac{261}{10000}, \frac{1}{20}, \frac{369}{5000}, \frac{93}{1000}, \frac{259}{2500}\right\}$

{a1 → -33.5388, a2 → 10.5263, a3 → -5.54017, a4 → 3.88776, a5 → -3.06549,
a6 → 2.51348, a7 → -1.67619, b1 → 0.095, b2 → 0.1, b3 → 2.2317×10^{-6} ,
b4 → -0.000144243, b5 → 0.0032302, b6 → -0.0301401, b7 → 0.100602}

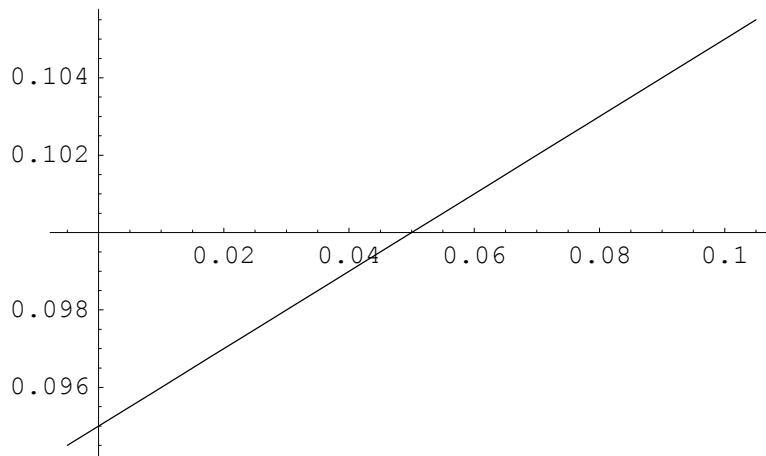
$-33.5388 + 10.5263 x - 5.54017 x^2 +$
 $3.88776 x^3 - 3.06549 x^4 + 2.51348 x^5 - 1.67619 x^6$

$10.5263 - 11.0803 x + 11.6633 x^2 - 12.262 x^3 + 12.5674 x^4 - 10.0572 x^5$

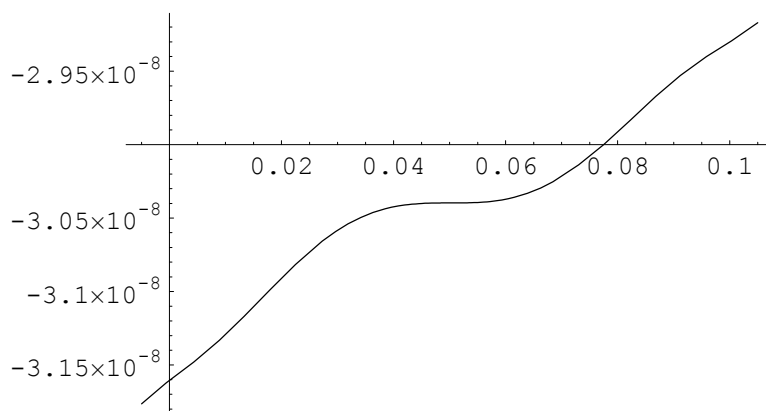
$0.095 + 0.1 x + 2.2317 \times 10^{-6} x^2 -$
 $0.000144243 x^3 + 0.0032302 x^4 - 0.0301401 x^5 + 0.100602 x^6$



- Graphics -



- Graphics -



- Graphics -

The moral of the story is to keep an eye out for this sort of difficulty when computing functions. If your results do not make sense, it might be due to something like this. Make sure you haven't inadvertently made a transformation of one of the variables.

3. References

- Chow, Gregory C., 1997, *Dynamic Economics: Optimization by the Lagrange Method*, Oxford: Oxford UP.
Judd, Kenneth L., 1998, *Numerical Methods in Economics*, Cambridge: MIT Press.
Gleick, James, 1987, *Chaos: Making a New Science*, New York: Penguin Books.