4 Perturbation Solution Methods for Economic Growth Models

Kenneth L. Judd and Sy-Ming Guu

4.1 Introduction

Economic growth is one of the most important macroeconomic phenomena. With economic growth comes the possibility of improving the living standards of all in a society. Economic growth has been studied by all generations of economists. Economists have used optimal control theory and dynamic programming to formalize the study of economic growth, yielding many important insights. Unfortunately, most of these methods are generally qualitative and do not yield the kind of precise quantitative solutions necessary for econometric analysis and policy analysis.

There are many ways to compute numerical solutions to economic growth models; see Taylor and Uhlig (1990) and Judd (1991) for a discussion of standard numerical analytic methods applied to a simple stochastic growth model. In this paper, we will focus on solutions arising from perturbation methods. Perturbation methods are distinct from standard numerical analytic procedures in that they use algebraic manipulations of equilibrium equations to derive information about a solution at a point, producing a local solution. In contrast, numerical analytic procedures take a more global approach, using information about the solution at several points and linking that information to form an approximate solution. The advantage of perturbation solutions is that they produce solutions which are, in some sense, the best possible near some point, and can be quickly computed. The supposed weakness of perturbation methods is that their quality falls as one moves away from the starting point. We will show how to compute perturbation solutions in simple growth models, and demonstrate that they often produce approximate solutions of high quality globally.

This chapter will examine methods based on the analyses of Bensoussan, Judd, and Judd and Guu; the reader should see those papers for the formal mathethematical results which underly the formal procedures described below. Section 4.2 will discuss both Taylor series and Padé approximation methods.

Section 4.3 will outline a simple optimal growth model in continuous time, show how to produce perturbation solutions, and discuss a way to evaluate their quality. Section 4.4 extends the continuous-time analysis to uncertainty and section 4.5 applies the same method to a discrete-time optimal growth model. While perturbation methods can be used for a much wider variety of models, these examples provide a good introduction to the techniques.

4.2 Approximations

Perturbation methods of approximation begin by computing a function and its derivatives at a point. We should therefore begin with a discussion of approximation methods based on such information. If we have a function, $f(x)$, and we want to study it near the point x_0, we compute f and several of its derivatives at x_0, and then use this information to construct good approximations for f in the neighborhood of x_0. We only assume that these derivatives exist. We will discuss two basic methods using this point-based information: Taylor series and Padé approximants.

4.2.1 Taylor Series Approximations

The most basic local approximation method is based on Taylor's Theorem:

Taylor's Theorem: If $f \in C^{n+1}[a, b]$ and $x, x_0 \in [a, b]$, then

$$f(x) = f(x_0) + (x - x_0) f'(x_0) + \frac{(x - x_0)^2}{2} f''(x_0) + \cdots + \frac{(x - x_0)^n}{n!} f^{(n)}(x_0) + R_{n+1}(x)$$

where

$$R_{n+1}(x) = \frac{1}{n!} \int_{x_0}^{x} (x - t)^n f^{(n+1)}(t)\, dt = \frac{(x - x_0)^{(n+1)}}{(n+1)!} f^{(n+1)}(\xi),$$

for some ξ between x and x_0.

A Taylor series approximation of $f(x)$ based at x_0 is the degree n polynomial in Taylor's Theorem. It uses only derivative information at x_0 and the error of this approximation is proportional to the $n+1$'th derivative of f and

$$\frac{(x - x_0)^{(n+1)}}{(n+1)!}.$$

It is therefore valid to a high order near x_0, and asymptotically exact if f is an analytic function. Generally, this approximation is good only near x_0 and decays rapidly away from x_0. The hope is that we are only interested in values

of x sufficiently close to x_0 or that we are "lucky" and the approximation is good away from x_0.

4.2.2 Padé Approximations

Taylor series approximations construct a polynomial to approximate f. An alternative way to use the same information is to construct a rational function, that is, a ratio of two polynomials, which agrees with f and its first n derivatives at x_0. The basic rational approximation method based at a point is called *Padé Approximation*.

The (m, n) Padé approximant of f at x_0 is a rational function

$$r(x) = \frac{p(x)}{q(x)}$$

where $p(x)$ and $q(x)$ are polynomials, the degree of p is m, the degree of q is n, and

$$\frac{d^k}{dx^k}(p - f\,q)\,(x_0), \quad k = 0, \cdots, m+n$$

This definition says that if $f(x)$ is approximated by $p(x)/q(x)$, then the first n derivatives of $q(x)f(x) - p(x)$ will be zero. Such derivative conditions impose linear constraints on the coefficients of p and q. The only restriction on p and q is that the degrees of p and q add up to at most n. This implies that there are $n+2$ unknown coefficients of p and q. The n derivatives of f at x_0 and the value of f at x_0 provide $n+1$ conditions. Since we are interested only in p/q, we can set the leading coefficient of q equal to 1 without loss of generality. Hence, the ratio p/q can be uniquely specified. (See Cuyt and Wuytack (1986) for a more rigorous demonstration of the critical properties of Padé approximation.) Usually we take p and q to be of equal degree, or make p one degree greater.

Theory and experience say that Padé approximants are better global approximants than Taylor series approximations, that is, the error grows less rapidly as we move away from x_0. For this reason, computers typically use Padé approximants to compute trigonometric, exponential, and other functions. We will use both Padé and Taylor approximants in our calculations below.

4.3 Optimal Growth Models

The solution of optimal growth models is important for dynamic modelling in economics. In this section, we will show how one can compute high-order polynomial and rational approximations to optimal policy functions in simple growth models.

The basic focus in economic growth models is how a country allocates resources and income between current consumption and investment. By forego-

ing current consumption, a country increases its wealth, its future output and income, and its future potential consumption. The more a country allocates to investment, the more rapidly income and potential consumption grows. However, that growth comes at the sacrifice of consumption today. Intuitively, the choice between consumption and investment depends on how important current consumption is today relative to the return on the investment. Economic growth models allow us to formalize these ideas.

We will first examine a simple continuous-time growth problem. We will assume that $f(k)$ is a concave production function, $u(c)$ is a concave utility function of consumption, c, and that the capital stock, k, obeys the law of motion

$$\frac{dk}{dt} = f(k) - c.$$

This law says that the rate of increase in capital equals current output minus current consumption. We will allow consumption to exceed output, permitting decreases in k. This is a very simple model, assuming that there is only one good produced in the economy and that it is used for both consumption and investment purposes. However, it will allow us to simply display perturbation methods, which, as shown in Judd (1991), can also be used to study models with several goods and several kinds of capital stocks.

The basic growth problem is choosing a consumption path, $c(t)$, which is feasible—that is, it keeps the capital stock nonnegative at all times, and maximizes the discounted sum of utility. This is represented by the control problem

$$\begin{aligned} V(k_0) &= \max_c \int_0^\infty e^{-\rho t} u(c)\, dt \\ \frac{dk}{dt} &= f(k) - c \\ k(0) &= k_0 \end{aligned}$$

$V(k_0)$ will denote the total value when one follows the optimal dynamic consumption plan when k_0 is the initial capital stock.

Dynamic programming methods study the value function by constructing a Bellman equation. The Bellman equation for this dynamic programming problem is

$$\rho V(k) = u\big(C(k)\big) + V'(k)\left(f(k) - C(k)\right) \tag{1}$$

The first-order condition for the indicated maximization is

$$0 = u'\left(C(k)\right) - V'(k) \tag{2}$$

This equation implies that consumption can be expressed as a function of capital, $c = C(k)$. Note that calendar time plays no role here, as is expected since the horizon is infinite and neither the utility function nor the production function depends on time in any crucial way.

When we substitute the first-order condition into the Bellman equation, we get a pair of equations in the two unknown functions:

$$0 = u(C(k)) + V'(k)(f(k) - C(k)) - \rho V(k)$$
$$u'C(k) = V'(k)$$

These equations describe our problem. We will simplify the problem by eliminating V and V' and reducing the problem to one equation in one unknown function, $C(k)$. If we differentiate the Bellman equation with respect to k we get

$$0 = u'(C(k))C'(k) + V''(k)(f(k) - C(k)) + V'(k)(f'(k) - C'(k)) - \rho V'(k)$$

If we differentiate the first-order condition with respect to k, we get

$$u''(C(k))C'(k) = V''(k)$$

which allows us to eliminate V'' and arrive at a single equation for C:

$$0 = C'(k)(f(k) - C(k)) - (u'(C(k))/u''(C(k)))(\rho - f'(k)) \tag{3}$$

It is this equation which will be the basis for our perturbation solution for various choices of $f(k)$ and $u(c)$.

In particular, we will assume a Cobb-Douglas production function with capital share .25, and an isoelastic utility function with intertemporal elasticity of substitution equal to .1. Our first *Mathematica* instructions will clear various variables and specify tastes and technology:

```
In[1]:= Clear[f,u,kss,gamma,rho,alpha,ctay,ccoef,
        cpade,numcoef,dencoef]
        rho = .05;
        alpha = .25;
        f[k_] = (rho/alpha) k^alpha;
        gamma = -10;
        u[c_] = c^(1+gamma)/(1+gamma);
```

Our objective is to find the first n derivatives of $C(k)$ at kss. This can be accomplished by defining the policy function in terms of the parameters `ccoef[i]`:

```
In[2]:= n = 5;
        ctay[k_] := Sum[ccoef[i] (k-kss)i/i!, {i,0,n}]
```

where kss will serve as the starting point of our approximation, and will be determined below.

We next define the critical function of equation (3), which for our choice of utility function reduces to:

```
In[3]:= bellman[k_] = - rho*ctay[k] -
                      gamma*ctay[k]*Derivative[1][ctay][k]+
                      gamma*f[k]*Derivative[1][ctay][k]+
                      ctay[k]*Derivative[1][f][k];
```

Equation (3) says that bellman[k] is always zero at all capital stocks k. It is this fact which we will now exploit, since this implies that all derivatives are

also zero at all k, a condition which in turn imposes identifying conditions on the derivatives of C.

4.3.1 Steady State Determination

Our basic equation, (3), is a first-order differential equation. Such differential equations need an initial condition to define a unique solution, that is, we need to find a k where we know the solution to C. We shall determine an initial condition by computing the steady state. The steady state capital stock is that k where there is no savings, that is, where $f(k) = C(k)$. From equation (3), saving is zero if and only if

$$f'(k) = \rho$$

The solution to this condition, denoted kss, is the steady state capital stock. We have defined the production function so that

```
In[1]:= kss = 1;
```

At the steady state, net invesment is zero, and consumption equals output, implying the conditions

```
In[2]:= ctay[kss] = f[kss];
        ccoef[0] = f[kss]

Out[2]:= 0.2
```

The steady state capital stock will serve as the point on which we base our perturbation procedures. Economically, this is a special point. If the country is at kss, then it will stay there forever. This implies that if $k_0 = kss$, then we know the entire future path of consumption and capital, and that we know both $C(k_0)$ and $V(k_0)$.

From this beginning, we can compute the necessary derivatives of $C(k)$ near kss and construct our approximation.

4.3.2 Computing C'(kss):

Computing the first derivative of the consumption policy function is a linearization procedure, commonly used in analyzing these models. However, the usual procedure revolves around linearizing a system of differential equations and is limited by this ODE perspective. We will pursue a more general approach, not limited to problems with an ODE formulation.

The key fact for our procedure is that `bellman[k] = 0` at all k, which in turn implies that all derivatives of `bellman[k]` are also zero at all k. Therefore, we first differentiate the Bellman equation with respect to k,

```
In[1]:= xbellman[k_] = D[bellman[k],k];
```

We then evaluate the resulting expression at kss,

```
In[2]:= xbellman[kss]
Out[2]:=
                                                  2
        -0.0075 - 0.5 ccoef[1] + 10 ccoef[1]
```

Since `xbellman[kss] = 0`, we can solve this quadratic equation for the possible values of $C'(kss) =$ `ccoef[1]`,

```
In[3]:= root = Solve[xbellman[kss]==0,ccoef[1]]

Out[3]:= {{ccoef[1] -> 0.062081}, {ccoef[1] -> -0.012081}}
```

This equation has two roots of opposite sign, but we can select one. In particular, only the positive root is consistent with V being concave at kss. Therefore,

```
In[4]:= ccoef[1] = root[[1]][[1]][[2]]

Out[4]:= 0.062081
```

This derivative corresponds to the usual linearization procedure widely used to study these models. Analysis usually stops with this linearization. However, we will go on to compute higher-order derivatives.

4.3.3 Computing Higher Derivatives:

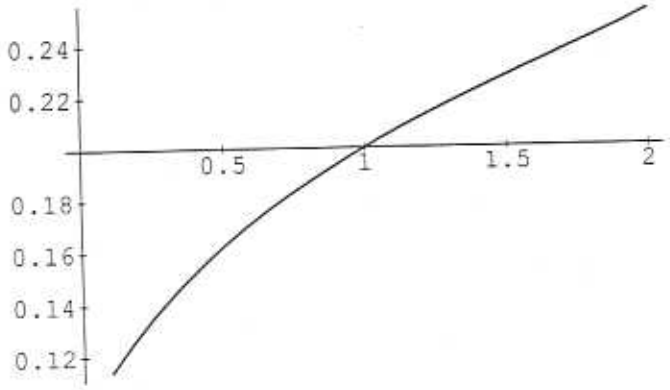
In order to compute higher derivatives of C at kss, we need only to take more derivatives of `bellman[k]` evaluate them at kss, and solve out for the appropriate derivative of C, as in the following:

```
In[1]:= For[j=2,j<n+1,j++,
        xbellman[k_] = D[xbellman[k],k];
        root=Solve[xbellman[kss]==0,ccoef[j]];
        ccoef[j] = root[[1]][[1]][[2]]
        ]
```

This procedure will, in succession, compute the first $n-1$ derivatives of C. Note that we constantly redefine `xbellman[k]`; this is done to save space since these expansions can grow rapidly. Each new derivative of xbellman produces a higher derivative of `bellman[k]`, and introduces a new, and unknown, derivative of C. When `xbellman[kss]` is computed, that new derivative of C at kss is the only unknown. Furthermore, as shown in Judd (1991), the new derivative of C at kss appears linearly. More precisely, at stage j, the first $j-1$ derivatives of C are known, `xbellman[kss]` involves only the first j derivatives of C, and the unknown derivative, the j'th, appears linearly in the equation `xbellman[kss] == 0`. Therefore, we have reduced the problem of computing the higher-order derivatives of C to a sequence of linear problems.

With the first several derivatives of C at kss computed, we have computed the Taylor series approximation. We can plot the consumption function:

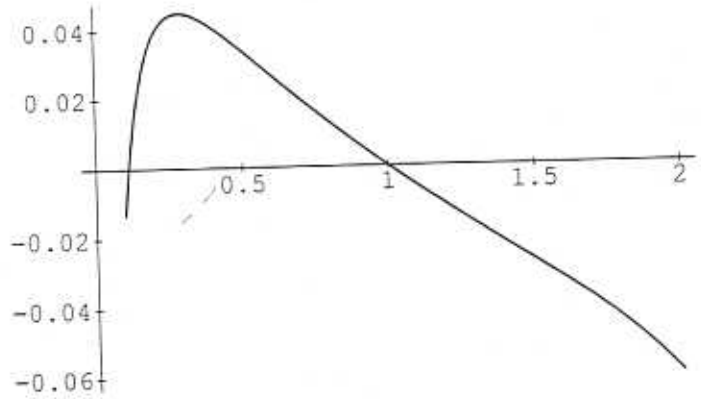
```
In[2]:= Plot[ctay[x],{x,.1,2}]
```




```
Out[2]:= -Graphics-
```

and the saving ratio function,

```
In[3]:= Plot[(f[x]-ctay[x])/f[x],{x,.1,2.}]
```




```
Out[3]:= -Graphics-
```

From the saving rate function, we can see that the approximation is not good at small capital stocks since we know from theory that savings is positive at all capital stocks below the steady state, which equals 1 here. However, near the steady state, the policy function appears to be sensible, implying stable dynamics near the steady state. Unfortunately, this tells us little about how good the approximation is.

4.3.4 Computing the Padé Approximation

An alternative use of these derivatives is to compute a Padé approximation. This is accomplished by finding a pair of polynomials such that their ratio has

the same derivative properties at kss as C. The degree of the numerator will be mpade,

```
In[1]:= mpade = Floor[n/2];
```

npade will be the degree of the denominator,

```
In[2]:= npade = n - mpade;
```

and we choose them so that the numerator is the same degree or one degree greater than the denominator.

We next define the numerator and denominator, and compute the coefficients of the Padé approximation. We first define a function centered at k=1:

```
In[3]:= c1[k_] = ctay[k+1]
Out[3]=
                                                2            3
        0.2 + 0.062081 k - 0.0184069 k  + 0.0103787 k  -

                                4            5
          0.00702861 k  + 0.00522644 k
```

Next we define the numerator and denominator polynomials:

```
In[4]:= numpol[x_] = Sum[numcoef[i] x^(i-1),{i,npade+1}];
        denpol[x_] = Sum[dencoef[i] x^(i-1),{i,mpade+1}];
```

We expand the function, c1, to be approximated at $x = 0$:

```
In[5]:= taylor[x_] = Series[c1[x],{x,0,npade+mpade}]
Out[5]=
                                              2            3
        0.2 + 0.062081 x - 0.0184069 x  + 0.0103787 x  -

                              4            5        6
          0.00702861 x + 0.00522644 x  + O[x]
```

We next express the error in the rational approximation:

```
In[6]:= diff = numpol[x] - taylor[x] denpol[x];
```

It is obvious that we can multiply both numpol[x] and denpol[x] by a common constant and have an equivalent rational function. We need to eliminate this degree of indeterminacy. Without loss of generality, we can normalize the representation by

```
In[7]:= dencoef[1] = 1;
```

We now solve for the polynomial coefficients by invoking the linear derivative conditions which define a Padé approximation:

```
In[8]:= s1 = Solve[LogicalExpand[diff==0]];
          Do[numcoef[i]=numcoef[i]/.s1[[1]],{i,npade+1}]
          Do[dencoef[i]=dencoef[i]/.s1[[1]],{i,mpade+1}]
          pade[x_]=numpol[x]/denpol[x]
```

```
Out[8]=                                    2             3
        0.2 + 0.276831 x + 0.0929694 x  + 0.00449471 x
        ------------------------------------------------
                                    2
                1 + 1.07375 x + 0.223585 x
```

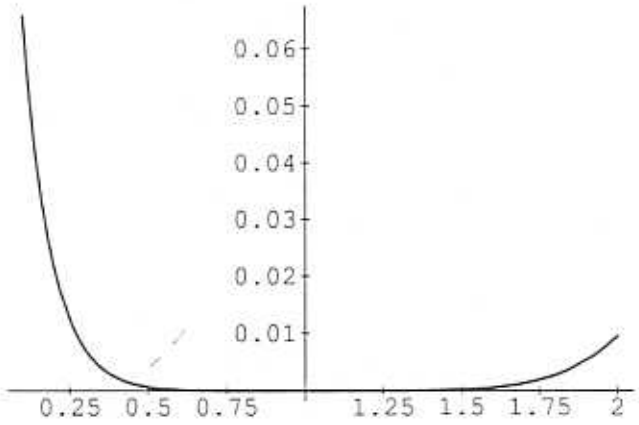
This direct procedure is not the most efficient way to compute the coefficients of a Padé expansion. See Cuyt and Wuytack (1986) for a discussion of superior procedures. We choose this procedure for its directness and simplicity.

To get a policy function for consumption, we shift the Padé approximation so that the resulting function is centered on the steady state:

```
In[9]:= cpade[k_] = pade[k-1];
```

We next plot the difference between the Taylor and Padé approximations:

```
In[10]:= Plot[(ctay[x]-cpade[x])/f[x],{x,.1,2.}, PlotRange->All]
```




```
Out[10]:= -Graphics-
```

We see that the two approximations are very similar except for the outer values of capital. At this point one wonders which is better. We next consider a procedure for making a judgment on that issue.

4.3.5 Evaluating the Quality of the Approximations

We next examine the quality of the approximations over a wide range of capital stocks. To do this we substitute the Taylor and Padé approximations into equation (3), yielding functions of k which measure the error in equation (3) generated by these approximations. Since this error is not unit-free, we then normalized the error function, commonly referred to as the *residual*, by `rho` and the steady-state consumption. We define `residtay[k]` to be the residual function of the Taylor approximation,

```
In[1]:= residtay[k_] := Abs[((f'[k]-rho)*ctay[k]+
                      gamma*(f[k]-ctay[k])*ctay'[k])/(rho*ctay[kss])]
```

and `residpade[k]` is the residual function for the Padé approximation,

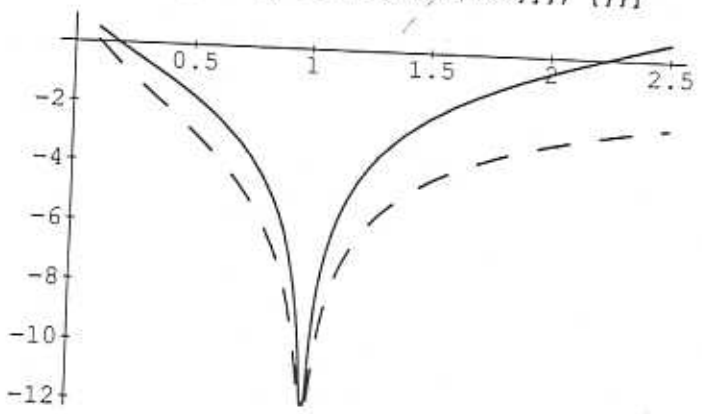
```
In[2]:= residpade[k_] := Abs[((f'[k]-rho)*cpade[k]+
          gamma*(f[k]-cpade[k])*cpade'[k])/(rho*cpade[kss])]
```

Both of these residuals are small near the steady state; for example,

```
In[3]:= {residtay[.98], residtay[1.02],residpade[.98],
        residpade[1.02]}
Out[3]=                -11                 -11                -13
       {3.62898 10    , 3.46191 10    , 6.80142 10    ,

                  -13
        6.01198 10    }
```

We will examine the relative quality of the two approximations by examining which has the smaller residual function. To do this we plot the residuals on a logarithmic scale. To avoid underflow, we will plot the base 10 logarithm of the residual's magnitude, plus, to prevent overflow, a small number.

```
In[4]:= eps = 0.000000000001;
        Plot[{Log[residpade[x]+eps]/Log[10.],
              Log[residtay[x]+eps]/Log[10.]},{x,.1,2.5},
        PlotStyle->{{Dashing[{.05,.05}]}, {}}]
```




```
Out[4]:= -Graphics-
```

These plots show that both expansions are excellent near the steady state, but that the Padé approximation, represented by the solid line, is substantially better away from the steady state. In fact the Padé approximation remains an excellent approximation, with an error of less than one part in a thousand, even for capital stocks more than double the steady state, whereas the Taylor approximation has unacceptable errors at those capital stocks.

Considerations from complex analysis indicate the limitations of the Taylor series expansion. Because of the singularity in the production function at $k = 0$, one expects that the consumption policy function is also singular at $k = 0$; this would be the case if consumption is roughly proportional to output. From complex analysis, this would imply that the Taylor series approximation centered

at $k = 1$ cannot be valid outside of $[0, 2]$. Therefore, the poor performance of the Taylor series for $k > 2$ is not surprising. The good performance of the Padé approximation is not limited by the singularity at $k = 0$, and performs very well even for $k = 2.5$.

In this section we have shown how to compute a high order approximation of $C(k)$ for a simple continuous-time growth model. In the sections below we show how to extend this to discrete time, uncertainty, and labor supply.

4.3.6 A Perturbation Package

We will now collect the basic functions above into a set of functions which will allow us to compactly compute the important functions. We will define a function which will automatically compute the Taylor series expansion of the consumption policy function around the steady state:

```
In[1]:= ctayfunc[alpha_,gamma_,rho_,n_] :=
        Block[{f,u,kss,ccoef,c,root,bellman,xbellman},
              f[k_] = (rho/alpha) k^alpha;
              u[c_] = c^(1+gamma)/(1+gamma);
              c[k_] := Sum[ccoef[i] (k-kss)^i/i!, {i,0,n}];
              bellman[k_] = - rho*c[k] -
                        gamma*c[k]*Derivative[1][c][k]+
                        gamma*f[k]*Derivative[1][c][k]+
                        c[k]*Derivative[1][f][k];
              kss = 1;
              c[kss] = f[kss];
              ccoef[0] = f[kss];
              xbellman[k_] = D[bellman[k],k];
              root = Solve[xbellman[kss]==0,ccoef[1]];
              rt1=root[[1]][[1]][[2]];
              rt2=root[[2]][[1]][[2]];
              If[rt1>0,rt=rt1,rt=rt2];
              ccoef[1] = rt;
              For[j=2,j<n+1,j++,
                  xbellman[k_] = D[xbellman[k],k];
                  root=Solve[xbellman[kss]==0,ccoef[j]];
                  ccoef[j] = root[[1]][[1]][[2]]
                 ];
              Sum[ccoef[i] (x-kss)^i/i!, {i,0,n}]
             ]
```

To use `ctayfunc`, we invoke it as part of a function definition, as in the following application which replicates the example above:

```
In[2]:= ctay[x_] = ctayfunc[.25,-10.,.05,5]
```

$$\text{Out}[2]= 0.2 + 0.062081\,(-1+x) - 0.0184069\,(-1+x)^2 + 0.0103787\,(-1+x)^3 - 0.00702861\,(-1+x)^4 + 0.00522644\,(-1+x)^5$$

The Padé expansion of a function C around a point, xpt, using the first n derivatives of C can be computed by `cpadefunc`:

```
In[3]:= cpadefunc[c_,xpt_,n_] :=
        Block[{mpade,npade,numpol,numcoef,denpol,
               dencoef,taylor,pade,c1},
              mpade = Floor[n/2];
              npade = n - mpade;
              c1[x_] = c[x+xpt];
              numpol[x_] = Sum[numcoef[i] x^(i-1),
                               {i,npade+1}];
              denpol[x_] = Sum[dencoef[i] x^(i-1),
                               {i,mpade+1}];
              taylor[x_] = Series[c1[x],{x,0,npade+mpade}];
              diff = numpol[x] - taylor[x] denpol[x];
              dencoef[1] = 1.;
              s1 = Solve[LogicalExpand[diff==0]];
              Do[numcoef[i]=numcoef[i]/.s1[[1]],{i,npade+1}];
              Do[dencoef[i]=dencoef[i]/.s1[[1]],{i,mpade+1}];
              pade[x_]=numpol[x]/denpol[x];
              pade[x-xpt]
              ]
```

We check `cpadefunc` by asking it to repeat the example above:

```
In[4]:= cpadefunc[ctay,1,5]

Out[4]=
                                                  2                  3
0.2 + 0.276831 (-1 + x) + 0.0929694 (-1 + x)  + 0.00449471 (-1 + x)
-------------------------------------------------------------------
                                    2
1. + 1.07375 (-1 + x) + 0.223585 (-1 + x)
```

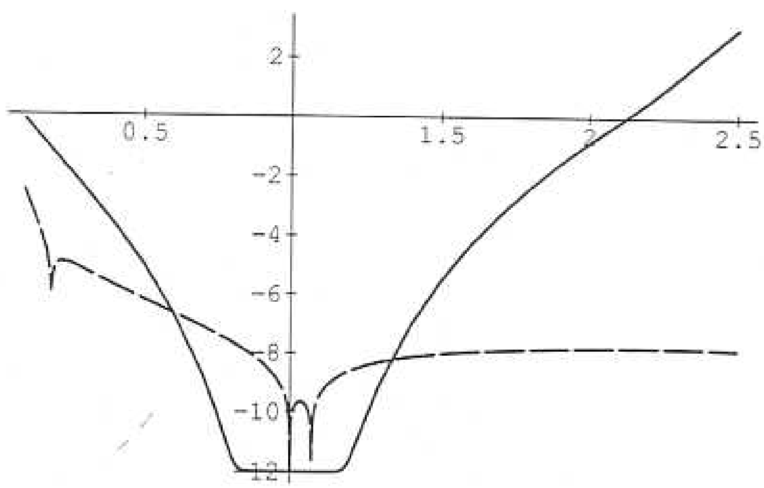
We now demonstrate just how fast these expansions can be computed and how good they can be by repeating these operations with $n = 15$, thereby generating 15'th degree approximations.

```
In[5]:= n = 15;

In[6]:= Clear[ctay,cpade]
        Timing[ctay[x_] = ctayfunc[alpha,gamma,rho,n];]

Out[6]:= {28.9167 Second, Null}

In[7]:= Timing[cpade[x_] = cpadefunc[ctay,1,n]]

Out[7]=
                                                                  2
{1.71667 Second, (0.2 + 0.778477 (-1 + x) + 1.22781 (-1 + x)  +

                      3                  4                  5
   1.00546 (-1 + x)  + 0.454781 (-1 + x)  + 0.111466 (-1 + x)  +

                      6                  7
   0.0134483 (-1 + x)  + 0.000634302 (-1 + x)  +

                      8
   0.00000625817 (-1 + x) ) /
```

```
Out[7] (cont.)

                                                      2                       3
 (1. + 3.58198 (-1 + x) + 5.11923 (-1 + x)  + 3.71602 (-1 + x)

                        4                      5                        6
 + 1.44084 (-1 + x)  + 0.286181 (-1 + x)  + 0.0250533 (-1 + x)

                          7
 + 0.000638291 (-1 + x)  ))
```

When we plot the residuals, we find that both expansions are excellent:

```
In[8]:= eps = 0.000000000001;
        Plot[{Log[residpade[x]+eps]/Log[10.],
                Log[residtay[x]+eps]/Log[10.]},{x,.1,2.5},
                PlotStyle -> {{Dashing[{0.05,0.01}]},{}},
            PlotRange -> All]
```




```
Out[8]:= -Graphics-
```

Note that in this case the Padé expansion is excellent over a very large range of capital stocks; in fact, the residual is of the order 10^{-7} or better over almost all capital stocks examined. The Taylor expansion is better near the steady state, as it should be since it is the best possible local expansion. However, its advantage in that region is truly trivial, existing only past the seventh significant digit, whereas the Taylor expansion becomes of questionable value outside of [.5, 1.5].

We have seen that perturbation methods can be used to compute two kinds of approximation to the continuous-time deterministic growth problem, and that *Mathematica* programs can automate the necessary algebra. We will next turn to other models which can also be analyzed in this way.

4.4 Continuous-Time Stochastic Growth

For many interesting questions concerning economic growth and dynamics, it is necessary to add uncertainty to the analysis. We will next show how to use the approximation to $C(k)$ around kss in the deterministic case to compute an

approximate policy function in a similar model with small amounts of uncertainty. While the assumption of small shocks may seem limiting, we will find that the approximations do well with empirically relevant levels of risk.

The problem we will examine is just the stochastic version of the previous problem:

$$V(k_0) = \max_c E\left\{\int_0^\infty e^{-\rho t}u(c)\,dt\right\}$$
$$dk = (f(k) - c)\,dt + \frac{1}{2}\sqrt{\sigma}\,k\,dz$$
$$k(0) = k_0$$

We define the value function to be

$$V(k) = \sup_{c\in\mathcal{F}} E\left\{\int_0^\infty e^{-\rho t}u(c)\,dt\right\}$$

where $\mathcal{F}$ is the set of feasible consumption processes with $k(0) = k_0$. If $V(k)$ is C^2, then stochastic optimization theory demonstrates that the value function solves the partial differential equation

$$0 = \max_c \left[-\rho V(k) + u(c) + V'(k)\,(f(k) - c) + \sigma\,k^2V''(k)\right]$$

Again, the optimal choice of consumption depends solely on the current capital stock. While the dependence of V and C on σ is normally suppressed in the notation, we will add σ as a parameter and express the policy function as $C(k,\sigma)$. This emphasizes the fact that we are using the $\sigma = 0$ case as the basis of our approximation. Formally, we are looking for the terms of the Taylor expansion of C:

$$C(k,\sigma) \doteq C(kss,0) + C_k(kss,0)(k - kss) + C_\sigma(kss,0)\sigma$$
$$+C_{kk}(kss,0)(k - kss)^2/2 + C_{\sigma k}(kss,0)\sigma(k - kss) + C_{\sigma\sigma}(kss,0)\sigma^2/2 + \cdots$$

The Bellman equation implies that the policy and value functions satisfy the system

$$0 = -\rho V + u(C) + V_k\,(f - C) + \sigma\,k^2V_{kk} \tag{4}$$
$$0 = u'(C) - V_k \tag{5}$$

Differentiating (4,5) with respect to k and using (5), we find a single equation for the consumption policy function, C:

$$0 = u'(f' - \rho) + u''C_k(f - C) + 2\sigma\,ku''C_k + \sigma k^2(u''C_kC_k + u''C_{kk}) \tag{6}$$

Equation (6) will be the centerpiece of our analysis. We know that at $\sigma = 0$ and $k = kss$

$$C(kss,0) = f(kss).$$

Our earlier analysis has computed all of the derivatives

$$\frac{\partial^i C}{\partial k^i}\,(kss,0)\,,\ i = 1,\cdots,n.$$

We will next move our derivatives

$$\frac{\partial}{\partial \sigma}\left(\frac{\partial^i C}{\partial k^i}\right)(kss, 0)$$

We first differentiate (6) with respect to σ, yielding

$$\begin{aligned} 0 = {} & u''C_\sigma(f'-\rho) + u'''C_\sigma(f-C)C_k + u''C_{k\sigma}(f-c) + u''(-C_\sigma)C_k \\ & + 2ku''C_k + 2\sigma k u'''C_\sigma C_k + 2\sigma k u''C_{k\sigma} \\ & + k^2(u'''C_kC_k + u''C_{kk}) \\ & + \sigma k^2(u''''C_\sigma C_kC_k + 2u'''C_{k\sigma}C_k + u'''C_\sigma C_{kk} + u''C_{kk\sigma}) \end{aligned} \tag{7}$$

At $(kss.0)$ this reduces to

$$0 = -u''C_\sigma C_k + 2ku''C_k + k^2(u'''C_{kk} + u''C_{kk})$$

showing that C_σ can be solved linearly in terms of C, C_k, and C_{kk} at kss.

Subsequent differentiation of (4) with respect to k will yeild similar expressions for $C_{\sigma k}$, $C_{\sigma kk}$, etc.

However, note that C_σ needs C_{kk}, $C_{\sigma k}$ needs C_{kkk}, etc. Hence, if we know only the first n derivatives of C with respect to k, we can only compute the first $n-2$ derivatives of C_σ with respect to k. This restricts what σ derivatives can be computed. For example, the following is a maximal collection of computable derivatives if we only know up to C_{kkkk}:

$$\begin{array}{cccc} C & C_k & C_{kk} & C_{kkkk} \\ C_\sigma & C_{\sigma k} & C_{\sigma kk} & \\ C_{\sigma\sigma} & & & \end{array}$$

This tableau is maximal since $C_{\sigma kkk}$ needs C_{kkkkk}, which is absent, and the resulting absence of $C_{\sigma kkk}$ then makes it infeasible to compute $C_{\sigma\sigma k}$.

We now implement this procedure in *Mathematica*. We first initialize the critical parameters,

```
In[1]:= kss=1; alpha = .25; rho = .05; gamma = -2;
```

define the production function,

```
In[2]:= f[k_] = rho/alpha k^alpha;
```

define the utility function,

```
In[3]:= u[c_] = c^(1+gamma)/(1+gamma);
```

define some useful auxiliary functions,

```
In[4]:= u1[c_] = Simplify[u'[c]/u''[c]];
```

```
In[5]:= u2[c_] = Simplify[u'''[c]/u''[c]];
```

and set the desired degree of approximation,

```
In[6]:= n = 4;
```

The following procedure builds a function, `conss[k,s]`, which approximates $C(k,\sigma)$:

```
In[7]:= Block[{sbell,root,cons},
         Clear[conss];
        For[j=0,j<=n,j++,
           For[p=0,p<=n,p++,
            Derivative[j,p][cons][kss,0] = .;
             ];
            ];
         sbell[0,0,k_,s_] = (f'[k] - rho) u1[cons[k,s]] +
            D[cons[k,s],k] (f[k]-cons[k,s]) +
            s k k (u2[cons[k,s]] (D[cons[k,s],k])^2 +
            D[cons[k,s],{k,2}]);
        sbell[1,0,k_,s_] = D[sbell[0,0,k,s],k];
        cons[kss,0] = f[kss];
        conss[k_,s_] = cons[kss,0];
        root = Solve[sbell[1,0,kss,0]==0,
        Derivative[1,0][cons][kss,0]];
        Print[root];
        Derivative[1,0][cons][kss,0] = root[[2]][[1]][[2]];
        conss[k_,s_] = conss[k,s]
                       + Derivative[1,0][cons][kss,0] (k-kss);
        For[j=2,j<n+1,j++,
          sbell[j,0,k_,s_] = D[sbell[j-1,0,k,s],k];
          root = Solve[sbell[j,0,kss,0]==0,
          Derivative[j,0][cons][kss,0]];
          Print[root];
          Derivative[j,0][cons][kss,0] = root[[1]][[1]][[2]];
          conss[k_,s_] = conss[k,s] +
            Derivative[j,0][cons][kss,0] (k-kss)^j / j!
          ];

        For[p=1,p<=n/2,p++,
          sbell[0,p,k_,s_] = D[sbell[0,p-1,k,s],s];
          root = Solve[sbell[0,p,kss,0]==0,
          Derivative[0,p][cons][kss,0]];
          Print[root];
          Derivative[0,p][cons][kss,0] = root[[1]][[1]][[2]];
          conss[k_,s_] = conss[k,s] +
            Derivative[0,p][cons][kss,0] s^p / p!;
          For[j=1, j <= n - 2 p,j++,
              sbell[j,p,k_,s_] = D[sbell[j-1,p,k,s],k];
              root = Solve[sbell[j,p,kss,0]==0,
              Derivative[j,p][cons][kss,0]];
              Print[root];
              Derivative[j,p][cons][kss,0] = root[[1]][[1]][[2]];
              conss[k_,s_] = conss[k,s] +
              Derivative[j,p][cons][kss,0]
                  (k-kss)^j s^p/(j! p!)
              ];
          ]
        ]

                (1,0)                                    (1,0)
{{cons      [1, 0] -> 0.0911438}, {cons      [1, 0] -> -0.0411438}}

                (2,0)
{{cons      [1, 0] -> 0.0293714}}
```

```
        (3,0)
{{cons      [1, 0] -> -0.0481816}}

        (4,0)
{{cons      [1, 0] -> 0.127661}}

        (0,1)
{{cons      [1, 0] -> -0.0967163}}

        (1,1)
{{cons      [1, 0] -> 0.0617278}}

        (2,1)
{{cons      [1, 0] -> 0.0392422}}

        (0,2)
{{cons      [1, 0] -> -5.30455}}
```

To evaluate the quality of the approximation, we substitute it into the stochastic bellman equation,

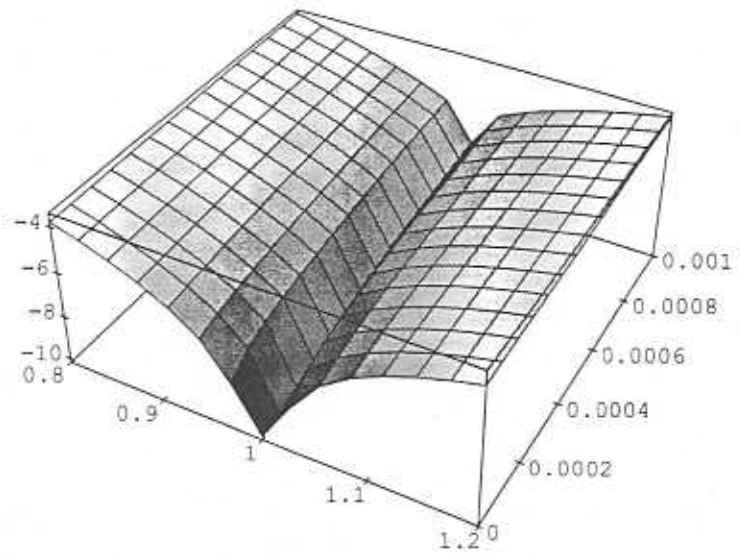
```
In[8]:= sbells[0,0,k_,s_] = (f'[k] - rho) u1[conss[k,s]] +
          D[conss[k,s],k] (f[k]-conss[k,s]) +
          s k k (u2[conss[k,s]] (D[conss[k,s],k])^2 +
          D[conss[k,s],{k,2}]);
```

and define a unit-free residual function,

```
In[9]:= resid[k_,s_] = sbells[0,0,k,s]/(rho u1[conss[kss,0]]);
```

We plot a three-dimensional surface of `resid`:

```
In[10]:= Plot3D[Log[Abs[resid[k,s]]+.0000000001]/Log[10],
         {k,.8,1.2},{s,0,.001}]
```

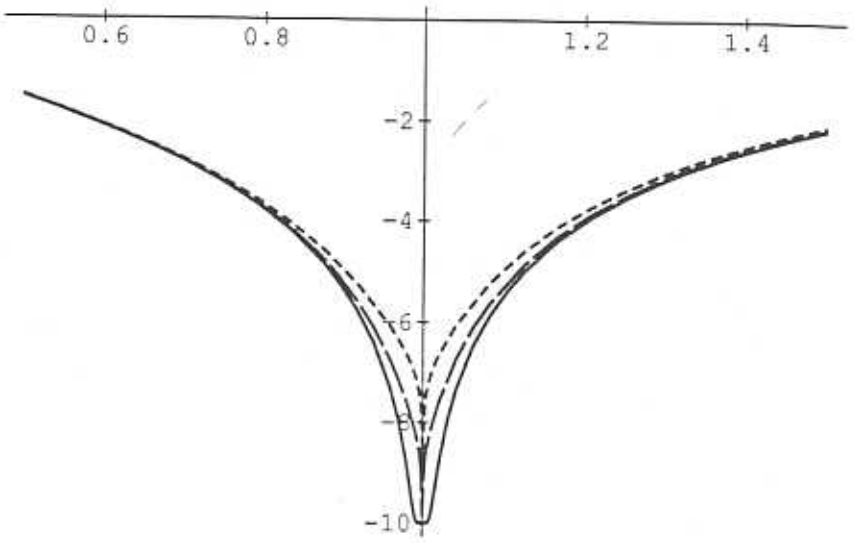



```
Out[10]:= -SurfaceGraphics-
```

This graph shows that the residual function is quite small over a wide range of capital stocks and an economically significant range of s values. To get some perspective on what s means, recall that the standard deviation of output in the U.S. economy is on the order of 2% of GNP and that capital-output ratio is about 2, implying that the standard deviation of output is 1% of wealth and implying a variance, s, equal to .0001.

We next plot the residual function for some specfic values of s which are similar to output variability in U.S. aggregate data:

```
In[11]:= eps = .0000000001;
         Plot[{Log[Abs[resid[k,0]]+eps]/Log[10],
               Log[Abs[resid[k,.0001]]+eps]/Log[10],
               Log[Abs[resid[k,.0004]]+eps]/Log[10]},
              {k,.5,1.5},
              PlotStyle -> {{Dashing[{.06,.0}]},
                            {Dashing[{.05,.01}]},
                            {Dashing[{.01,.01}]}}]
```




```
Out[11]:= -Graphics-
```

Note that the residuals are quite small, considering the low degree of approximation and the wide range of capital stocks. Also note that the residuals are practically equal for all three values of the variance, s.

4.5 Discrete-Time Growth

In some problems, it is more natural to use a discrete-time formulation of the problem. We will next apply these ideas to a discrete-time growth model. In this model, we assume that at the beginning of each period the current stock of

capital, k, is split between consumption uses, c, and investment, $k - c$, which is used to produce gross output, $f(k-c)$, which will be available the next period. More specifically,

$$k(t+1) = f(k(t) - c(t))$$

is the law of motion.

The optimal growth problem is expressed as

$$\max_{c_t} \sum_{t=0}^{\infty} \beta^t u(c_t)$$

$$k(t+1) = f(k(t) - c(t))$$
$$k(0) = k_0$$

We make the standard assumptions that $u(c)$ is a concave utility function and $f(k)$ is a concave gross production function.

Because of the stationarity of the problem, the optimal consumption in each period is given by a policy function, $C(k)$, satisfying the Euler Equation

$$u'(C(k)) = \beta u'\Big(C\big(f(k - C(k))\big)\Big) f'(k - C(k)) \tag{8}$$

At the steady state, kss, we have $f(kss - C(kss)) = kss$, implying

$$u'(C(kss)) = \beta u'(C(kss)) f'(kss - C(kss))$$

which in turn implies

$$1 = \beta f'(kss - C(kss))$$

all of which uniquely determines kss. Furthermore

$$kss = f(kss - C(kss))$$

Taking the derivative of (8) with respect to k implies

$$\begin{aligned} u''(C(k))\, C'(k) = {} & \beta u''\Big(C\big(f(k - C(k))\big)\Big) C'\big(f(k - C(k))\big) \\ & \times f'(k - C(k))(1 - C'(k))\, f'(k - C(k)) \\ & + \beta u'\Big(C\big(f(k - C(k))\big)\Big) f''(k - C(k))\, (1 - C'(k)) \end{aligned}$$

At $k = kss$, this reduces to (we will now drop all arguments)

$$u''C' = \beta u''C' f'(1 - C')\, f' + \beta u'\, f''(1 - C')$$

This is a quadratic equation with the solution

$$C' = \frac{1}{2}\left(1 - \beta - \beta^2 \frac{u'}{u''} f'' + \sqrt{\left(1 - \beta - \beta^2 \frac{u'}{u''} f''\right)^2 + 4 \frac{u'}{u''} \beta^2 f''}\right)$$

We now define the critical functions and parameters:

```
In[1]:=  n=7; Clear[cd]; kss=1.;beta=0.96;alpha=0.25;
         gamma=-10.0; a=kss/(alpha beta kss)âlpha;
         f[k_]=a kâlpha; u[y_]=y^(1+gamma)/(1+gamma);
         u1[y_]=u'[y]; u2[k_]=u'[cd[f[k-cd[k]]]];
         cd[kss]=kss*(1-alpha beta);
```

The Euler equation for this model is

```
In[2]:=  euler[0,k_]=beta*u2[k]*f'[k-cd[k]]-u'[cd[k]]
Out[2]:=   -1.                         0.342893
         -------- + ---------------------------------------------------
            10.              0.75                          0.25 10.
         cd[k]       (k - cd[k])     cd[1.42872 (k - cd[k])     ]
```

This equation states that the utility from saving one unit of capital and consuming the gross proceeds tomorrow has the same marginal yield as consuming that unit today.

While this equation is intuitive, this form of the Euler equation generates unnecessarily complex algebraic expressions. More efficient is the equivalent form which follows from the CRRA specification of the utility function which we specified above:

```
In[3]:=  euler1[0,k_]=cd[f[k-cd[k]]]*(beta f'[k-cd[k]])^(1/gamma)-
         cd[k]
Out[3]:=
                                    0.075                          0.25
         -cd[k] + 1.11297 (k - cd[k])      cd[1.42872 (k - cd[k])     ]
```

The gain in simplicity is seen by examining the output cells of `euler` and `euler1`. The denominator of the second term in `euler` involves the tenth power of `cd[k]` composed with itself raised to a power. In `euler1`, the tenth power is missing from the comparable term. Rewriting the Euler equation so as to minimize the complexity of the most complex term will help us keep down the complexity of the computation.

We sequentially solve for the Taylor series solution around the steady state. Successive differentiation and solving for derivatives yields these series:

```
In[4]:= Derivative[1][cd][kss] = .;
        euler1[1,k_]=D[euler1[0,k],k]; euler1[0,k] = .;
        root=Solve[euler1[1,kss]==0, Derivative[1][cd][kss]];
        r1=root[[1]][[1]][[2]]; r2=root[[2]][[1]][[2]];
        If[r1>0,rt=r1,rt=r2]; Derivative[1][cd][kss]=rt;
        For[j=1,j<n,j++,
             Derivative[j][cd][kss] = .;
             euler1[j,k_]=D[euler1[j-1,k],k];
             euler1[j-1,k] = .;
             sol=Solve[euler1[j,kss]==0,
                       Derivative[j][cd][kss]];
             vt=Derivative[j][cd][kss]/.sol[[1]];
             Derivative[j][cd][kss]=vt;];
        Do[cee[ii]=1/(ii!)
                Derivative[ii][cd][kss],{ii,1,n-1}]
        Do[cd[k_]=cd[kss]+
                Sum[cee[iii] (k-kss)îii,{iii,1,jj}],
                       {jj,1,n-1}]
        cd[k]
```

```
Out[4]:= 0.76 + 0.392658 (-1. + k) - 0.286785 (-1. + k)^2 +

    0.195456 (-1. + k)^3 - 0.0727102 (-1. + k)^4 -

    0.0639789 (-1. + k)^5 + 0.166206 (-1. + k)^6
```

In this code, we eliminate `euler1[j,k]`, the jth derivative of the Euler equation, as soon as we have finished using it. This is done to economize on space.

We can again solve for the Padé approximation.

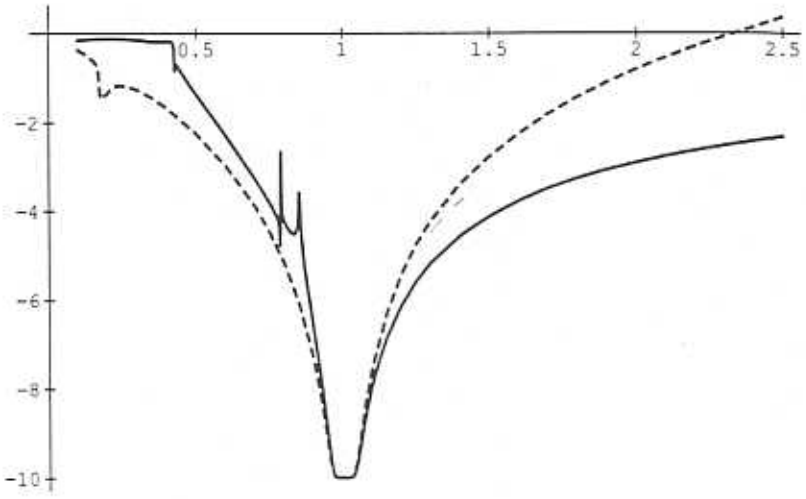
```
In[5]:= mpade = Floor[n/2];
        npade = n-mpade;
        fcn[x_] = cd[x+1];
        numpol[x_] = Sum[numcoef[i] x^(i-1),{i,npade+1}];
        denpol[x_] = Sum[dencoef[i] x^(i-1),{i,mpade+1}];
        taylor[x_] = Series[fcn[x],{x,0,npade+mpade}];
        diff = numpol[x] - taylor[x] denpol[x];
        dencoef[1] = 1;

In[6]:= s1 = Solve[LogicalExpand[diff == 0]];
        Do[numcoef[i]=numcoef[i]/.s1[[1]],{i,npade+1}];
        Do[dencoef[i]=dencoef[i]/.s1[[1]],{i,mpade+1}];
        pade[x_] = numpol[x]/denpol[x];
        cdpade[k_] = pade[k-1]
```

```
Out[6]= (0.76 + 6.9806 (-1 + k) + 13.9395 (-1 + k)^2 + 8.83725 (-1 + k)^3 +

    0.397994 (-1 + k)^4) /

    (1 + 8.66834 (-1 + k) + 14.2402 (-1 + k)^2 + 7.28448 (-1 + k)^3)
```

We can again compute residual functions to represent the error of our approximation. In this case, the residual will be `euler1[0,k]/cd[kss]`. This expresses the Euler equation error as a fraction of steady state consumption. While this is similar to what we did in the continuous-time case, the meaning of the residual is more clear here. If the residual is .01 at some k, then we conclude that the difference between what our approximation says current consumption should be and what it would be if an agent optimized today believing that our approximation would be used in the future equals `.01*c[kss]`. This is therefore a one-period error. Over the life of an economic agent he may make several such errors. However, if this relative error is on the order of, say, 10^{-6}, then this one-period error is so small that the cumulative error is also small given the ability of the human brain to do intricate calculations.

We next compute and plot the residual functions for both the Taylor and Padé approximations:

```
In[7]:= cdtay[k_] = cd[k];
        residtay[k_]=Abs[
        (cdtay[f[k-cdtay[k]]]*(beta f'[k-cdtay[k]])^(1/gamma)-
          cdtay[k]) / cdtay[kss]];
        residpade[k_]=Abs[
        (cdpade[f[k-cdpade[k]]]*(beta f'[k-cdpade[k]])^(1/gamma)-

          cdpade[k]) / cdpade[kss]];
In[8]:= eps = 0.0000000001;
        Plot[{Log[residpade[x]+eps]/Log[10.],
                Log[residtay[x]+eps]/Log[10.]},{x,.1,2.5},
            PlotStyle -> {{},{Dashing[{.01,.01}]}}]
```




```
Out[8]= -Graphics-
```

Again we find that the Euler equation errors are quite small. The difference in the discrete-time case is that the Taylor expansion slightly dominates for capital stocks below the steady state, but is much worse above the steady state.

Overall, we conclude that our approximations are as good as one could reasonably expect real-life individuals to compute. We also find that the Padé approximation is good over a wider range of capital stocks than the Taylor expansion.

4.6 Extensions and Conclusions

We have demonstrated that perturbation methods can be implemented in *Mathematica*, and can produce excellent approximate solutions to simple economic growth models. Other applications are under development. Adding taxes to our analysis is straightforward and will facilitate analysis of the effects of taxation on growth and welfare. In Judd [1991], there are discussions of, for example, applications with several state variables and applications to dynamic games.

In our examples, we have focussed on computing particular examples. *Mathematica* is rather slow if we wanted to compute hundreds of examples, as would be the case inside a maximum likelihood estimation procedure. These methods could still be used. Using the same procedures, one can compute the derivatives in terms of the underlying parameters, ρ, α, γ and any others, and then, using `FortranForm` commands, write Fortran statements which could be used in Fortran programs to rapidly compute the coefficients. In this way one can combine the symbolic tools of *Mathematica* with the computational power and software base of Fortran.

In general, we anticipate that perturbation methods will become as useful in economics as they have been in science generally, particularly when symbolic manipulation software becomes more powerful and widespread.

4.7 References

Bensoussan, A. *Perturbation Methods in Optimal Control*. John Wiley and Sons, 1988.

Cuyt, A. and L. Wuytack, *Nonlinear Numerical Methods: Theory and Practice*, Amsterdam: North-Holland, 1986.

Judd, K. L. *Numerical Methods in Economics*, December, 1991, Hoover Institution.

Judd, K. L. *Perturbation Methods for Solving Dynamic Economic Models*, November, 1991, Hoover Institution.

Judd, K. L., and Sy-Ming Guu, "Asymptotic Methods in Aggregate Growth Models," *Journal of Economic Dynamics and Control* (forthcoming).

Taylor, J., and H. Uhlig. "Solving Nonlinear Stochastic Growth Models: A Comparison of Alternative Solution Methods," *Journal of Business and Economic Statistics* 8 (January, 1990): 1–18.