ME 201/MTH 281/ME400/CHE400
Convergence of Legendre Expansions

1. Introduction

This notebook provides a general framework for the construction and visualization of partial sums of expansions in Legendre polynomials. In general structure, it is very similar to the earlier notebook Fourier, which produced partial sum sequences for Fourier series. To use the routines for a specific function, you must define the function in Section 2 below. Section 3 contains the definitions of the terms and partial sums of the series, and Section 4 defines a function which produces a graph of the nth partial sum for any n. Section 5 defines a computationally efficient routine for producing a sequence of all partial sums up to a specified value n. In section 6 we show how to send the output of this more efficient routine to a Manipulate panel. The Manipulate panel gives us a very good tool for an interactive study of the convergence of the expansion. In section 7, we summarize the use of the efficient routines and the Manipulate panel by considering a second example.

As we showed in class from the Legendre differential equation, the Legendre polynomials are orthogonal on the interval [-1,1]:

\[ \int_{-1}^{1} P_m(\eta) P_n(\eta) \, d\eta = 0 \quad \text{for} \quad m \neq n. \]

It may be shown that the normalization integral is given by

\[ \int_{-1}^{1} P_n(\eta) P_n(\eta) \, d\eta = \frac{2}{2n+1}. \]

These polynomials form a complete set on the interval [-1,1], and any piecewise smooth function may be expanded in a series of the polynomials. The series will converge at each point to the usual mean of the right and left-hand limits. The coefficients are easily calculated using the orthogonality property and the normalization integral, and the basic expansion theorem is

\[ f(\eta) = \sum_{n=0}^{\infty} C_n P_n(\eta), \quad \text{where} \quad C_n = \frac{2n+1}{2} \int_{-1}^{1} f(\eta) P_n(\eta) \, d\eta. \]

2. Definition of Function and Expansion Coefficients

By way of example, we consider the function \( f(\eta) = -1 + 2H(\eta) \), where \( H(\eta) \) is the unit step function. We may write this a number of ways in Mathematica. The most direct is

\[ f[\eta_] := \text{Sign}[\eta] \]

The plot range in this notebook is user-specified, and is assigned to the variable pltrange. We choose a value appropriate for this function. We allow for some overshoot, which will no doubt happen with the earlier partial sums in the sequence.

\[ \text{pltrange} = (-1.5, 1.5); \]

We check our definition by plotting the function. We first set an option to make our plots uniform in size at 250.

\[ \text{SetOption[Plot, ImageSize -> 250];} \]
We define \( A_{\mu} \) as the coefficient of \( \sqrt{\xi(x)} \) in the expansion of \( f(x) \). Now we will compare the coefficient \( A_{\mu} \) calculated numerically with the coefficient \( A_{\mu} \) obtained analytically. However, we write our code in terms of numerical integration, so that it will work for (almost) any integrable function. It is important NOT to calculate the integral for each coefficient each time the partial sum is evaluated for a given \( \eta \). If that is done, plotting will be unacceptably slow. We evaluate the coefficients once and for all and store the values in the function \( \text{coeff}[n] \). Although we don’t know a priori how many coefficients will be needed in a particular problem, we calculate the first \( n_{\text{coeffmax}} \) coefficients, where we set the default value of \( n_{\text{coeffmax}} \) to 101. If we need more coefficients than that, we might want to think about finding another way to do the problem! The function \( \text{calcoeff} \), defined below, calculates and stores all of the coefficients up to order \( n_{\text{coeffmax}} \). The accurate numerical integration of higher order Legendre polynomials requires a lot of digits of precision. If you have analytical expressions for the coefficients, you can use those instead of the numerical routine below. You need to assign those values to the function \( \text{coeff}[n] \) instead of executing \( \text{calcoeff} \). We calculate the coefficients both ways below for this particular function, and we compare the results.

\[
In[142]:= n_{\text{coeffmax}} = 101;
\]

\[
In[143]:= \text{calcoeff} := \text{Module}[(\text{faux}, \text{faux}[\eta] = \text{SetPrecision}[f[\eta], 50];
\quad \text{Do}[(\text{coeff}[n] = (n + 0.5) \text{NIntegrate}[\text{faux}[\eta] \text{LegendreP}[n, \eta],
\quad \quad (\eta, -1, 0, 1), \text{PrecisionGoal} \rightarrow 40, \text{AccuracyGoal} \rightarrow 8, \text{MinRecursion} \rightarrow 3,
\quad \quad \text{MaxRecursion} \rightarrow 20, \text{WorkingPrecision} \rightarrow 50]), (n, 0, n_{\text{coeffmax}})]);
\]

Now we execute \( \text{calcoeff} \) to calculate the coefficients.

\[
In[144]:= \text{calcoeff}
\]

As shown in class, the \( n \)-th Legendre coefficient for the expansion of this particular function is given by \( P_{n-1}(0) - P_{n+1}(0) \). We define these as an alternate set of coefficients which we will compare with the coefficients obtained by numerical integration.

\[
In[145]:= \text{Do}[\text{coeffanalyt}[n] = \text{LegendreP}[n - 1, 0.] - \text{LegendreP}[n + 1, 0.], (n, 1, 101)]; \text{coeffanalyt}[0] = 0;
\]

We compare them by constructing a table in which the first element in each trio is \( n \), the second element is \( \text{coeff}[n] \), and the third element is \( \text{coeff}[n]-\text{coeffanalyt}[n] \).
A brief study of the table shows that the coeff and coeffanalyt are in excellent agreement. Of course we have used serious brute force in the numerical calculation by asking for 50 digits of working precision and 40 digits for a precision goal. Even so, the calculations take little time, and have to be done only once for each function studied. In the rest of this notebook, we will use the coefficients coeff[n] calculated by numerical integration. If we wanted to use the values of coeffanalyt[n] instead, we would have to first move those values to coeff[n].

3. Terms and Partial Sums of Fourier-Legendre Series

We keep the notation for eigenfunctions the same as in similar notebooks for Fourier series and Fourier-Bessel series. The (non-normalized) eigenfunctions are denoted by \( \Phi[\eta,n] \):

\[
\Phi[\eta,n] := \text{LegendreP}[n, \eta]
\]

Now we define the nth term of the series, called fourterm, and the partial sum foursum.

\[
\text{fourterm}[\eta_-, n_] := \text{N}[\text{coeff}[n]*\Phi[\eta,n]]
\]

The nth partial sum of the series (the sum of all terms up to and including polynomials of order n) is foursum, given by

\[
\text{foursum}[\eta_-, n_] := \text{Sum}[\text{fourterm}[\eta,k],\{k,0,n\}]
\]
4. Graphs of \( f[\eta] \) and Partial Sums of the Legendre Series

The function \( \text{pic}[n] \), defined below, gives a graph of the function \( f[\eta] \) and of the \( n \)th partial sum of the series. To get a plot of the function \( f \) only, use \( \text{picfunc} \), also defined below. The function \( f \) is in blue, and the series partial sum is in red. The number of points plotted in each graph is specified by the variable \( \text{npoints} \). The default, set below, is 500. If your computation times seem excessive, you can make this number smaller.

\[
\text{npoints} = 500;
\]

\[
\text{picfunc} := \text{Plot}\{f[\eta], \{\eta, -1, 1\}, \text{PlotStyle} \to \{\text{RGBColor}[0,0,1], \text{Thickness}[0.004]\},
\quad \text{PlotPoints} \to \text{npoints}, \text{PlotRange} \to \text{pltrange},
\quad \text{AxesLabel} \to \{"\eta","f[\eta]"\}, \text{AspectRatio} \to 0.7\}
\]

\[
\text{pic}[n_] := \text{Plot}\{\{\text{foursum}[\eta,n],f[\eta]\}, \{\eta, -1, 1\}, \text{PlotStyle} \to \{\{\text{RGBColor}[1,0,0], \text{Thickness}[0.004]\}, \{\text{RGBColor}[0,0,1], \text{Thickness}[0.004]\}\},
\quad \text{PlotPoints} \to \text{npoints}, \text{PlotRange} \to \text{pltrange},
\quad \text{AxesLabel} \to \{"\eta","f[\eta]"\}, \text{AspectRatio} \to 0.7,
\quad \text{PlotLabel} \to \text{Row}[\{\text{n} = \_, \text{PaddedForm}[n,2]\}]\}
\]

Let's try this out. We produce first a graph of the basic function, then a graph of the function and the 7th partial sum, and finally a sequence of graphs of the partial sums up to \( n = 7 \). We use \text{Table} and \text{GraphicsGrid} to produce the graphics sequence, and to display the graphs two per line. We display only the odd partial sums, because all of the even Legendre coefficients are zero for this odd function. Hence each even partial sum will be identical to the odd one immediately preceding it.

\[
\text{picfunc}
\]

\[
\text{pic}[7]
\]
In[156]:= GraphicsGrid[Table[{pic[i], pic[i + 2]}, {i, 1, 5, 4}]]

Out[156]=

We see that many more terms are needed to get a result closely resembling the original f[\eta]. It is possible to produce a much longer sequence of graphs by this same technique, but the computation time becomes large. The problem is that our technique is very inefficient. For each graph in the sequence we are recomputing all of the previous terms. An efficient technique would save the partial sums rather than recomputing them at each step. We develop such a technique next, and use it to generate a large graph sequence.

5. Efficient Production of a Sequence of Partial Sum Graphs

Our technique is to find and save the values of the kth partial sums at every plotted point. Then to get the partial sums for k+1, we only have to increment those values by the value of the new term. Thus each succeeding graph requires the evaluation of only one term in the series. However, we must then change our graphing technique, because Plot works only for functions defined analytically. For the new algorithm, we can use ListPlot, which plots a given numerical set of points. The routine defined here is called picarray[first, last, grinc], where all arguments are integers. The argument "first" is the n value of the first partial sum in the sequence and the argument "last" is the last n value. The argument "grinc" specifies the step between displayed graphs. All the partial sums are calculated, but by choosing grinc greater than 1, you can display every "grincth" graph. The first four functions defined below are used by picarray to calculate coordinate lists and to produce graphs.

In[157]:= SetOption[ListPlot, ImageSize -> 250];

In[158]:= mksumlist[n_] := Module[{ans, \eta, inc, j},
   ans = {{-1., foursum[-1., n]}},
   inc = 2./npoints;
   Do[\eta = -1+j*inc;
      ans = Append[ans, {\eta, foursum[\eta, n]}], {j, 1, npoints}];
   ans]
As an example, we execute `picarray[1,5,4]`. This will start with \( n = 1 \) and then display every 4th graph up to \( n = 5 \) -- i.e., graphs 1 and 5.
Now we use this new faster method to display the partial sums for our function up to and including $n = 51$. We display every second graph in the sequence by setting $\text{grinc} = 2$. The reason for displaying only every second graph is that for this function, the even Legendre expansion coefficients are zero. Thus if $n$ is odd, graphs $n$ and $n+1$ are the same. The printed version of the notebook shows only the first graph in the sequence.

```math
\text{In[164]} = \text{picarray[1, 51, 2];}
```

You can visualize the convergence process by selecting and animating the above group of cells. The animation is through the menu sequence Graphics -> Rendering -> Animate Selected Graphics.

For visualization in the printed version of this notebook, we show every 10th graph in the sequence.

```math
\text{In[165]} = \text{picarray[1, 51, 10];}
```
6. Use of Manipulate to View the Sequence of Partial Sums

We saw in section 5 above that many lines of *Mathematica* code were required to define a function which would produce efficiently a graph sequence of partial sums. Here we accomplish the same thing with a single command Manipulate, although to be honest we make use here also of the many lines of code defined earlier in our use of Manipulate. The power of manipulate is only evident in a fully interactive mode, so you need to execute this *Mathematica* notebook to appreciate what Manipulate can do.
We start by modifying the earlier code for picarray[first, last, grinc] to produce a Manipulate panel as an output, rather than a sequence of printed graphs. For convenience, we repeat the definition of the arguments of picarray here. The argument first is the \( n \) value of the first partial sum in the sequence and the argument last is the last \( n \) value. The variable grinc specifies the step between displayed graphs. All the partial sums are calculated, but by choosing grinc greater than 1, you can display every "grincth" graph. Now we define a new command manpicarray[first, last, grinc]. It does exactly what picarray does except that the output is now displayed in a Manipulate panel.

```
In[166]:= manpicarray[first_, last_, grinc_] := DynamicModule[
{s, u, m, l, i, s, t, r, e, m, l, i, s, t, k, g, r, p, h, g, r, p, h, 0, m, a, n, g, r, a, p},

sumlist = mksumlist[first];
grph0 = picfunc;
grph = mkgraph[sumlist, 1, 0, 0];
mangraph[first] = Show[{grph0, grph}, AxesLabel -> {"\( \eta \)", "f[\( \eta \)]"},
AspectRatio -> 0.7, PlotLabel ->
Row[{"n =", PaddedForm[k, 2]}]];]

Do[sumlist = sumlist + mktermlist[k];
If[(Mod[k-first, grinc] == 0),

mangraph[k] = Show[{grph0, grph}, AxesLabel -> {"\( \eta \)", "f[\( \eta \)]"},
AspectRatio -> 0.7, PlotLabel ->
Row[{"n =", PaddedForm[k, 2]}]];]

Manipulate[mangraph[n], {n, first, last, grinc}]
```

Now we use this to look at our square wave in a Manipulate panel. We include partial sums up to order 101.

```
In[167]:= manpicarray[1, 101, 2]
```

There is a slight delay while the graphs are constructed, but then the Manipulate panel gives us a very smooth interactive view of the partial sums with both the slider and the movie mode.

7. Summary

This has been a rather long excursion in Mathematica. We give here a brief summary of exactly how to generate a Manipulate plot of the partial sums of a Legendre expansion. We do this in terms of one last example. We choose a rather exotic function which is neither even nor odd, and which has a discontinuity at \( \eta = 0 \).
\[ f[\eta_] := \text{If}[ (\eta < 0), (1 + \eta), (-4 \eta (1 - \eta)) ] \]

We plot the function.

\[ \text{In[169]} = \text{picfunc} \]

We tighten the plot range slightly.

\[ \text{In[170]} = \text{pltrange} = (-1.2, 1.2); \]

\[ \text{In[171]} = \text{picfunc} \]

Now we execute \text{calcoeff} to construct the Legendre expansion coefficients.

\[ \text{In[172]} = \text{calcoeff} \]

Finally we use \text{manpicarray} to output the partial sums to a Manipulate panel.
By using the slider and/or the movie, we can see that the expansion is working. There still some small oscillations. The Gibbs phenomenon is present here because of the discontinuity at \( \eta = 0 \), so oscillations will persist no matter how many terms we use, although they will be increasingly localized around \( \eta = 0 \) as the number of terms increases. If we did increase the number of terms, we would very likely have to increase the WorkingPrecision and PrecisionGoal settings in the integration routine for the coefficients.