Convergence of Fourier Sine and Cosine Series

1. Introduction

This notebook is a modification of an earlier notebook, Convergence of Fourier Series. In the present notebook, the techniques of the earlier notebook are modified to produce visualizations of both the sine and cosine series of a given function. To use the code for a particular function, you must define the function in Section 2 below. Section 2 is the only place where specific information about the function is given. Section 3 contains the definitions of the terms and partial sums of the series, and Section 4 defines functions which produce graphs of the nth partial sums for any n. Section 5 defines a computationally efficient routine for producing sequences of all partial sums up to a specified n. In section 6, we look at the command Manipulate as an alternative way to show the dependence of the partial sums on the number of terms retained. In section 7 we give a brief summary of how to construct the Manipulate display of the partial sums for a given function.

2. Definition of Function and Fourier Coefficients

In the definitions below, we first define the function over a basic interval [0, L] and then specify the value of L. This function is called fbas. The even and odd extensions are then defined. They are called fbasev and fbasod. Finally the periodic extensions of the even and odd extensions are defined. They are called fev and fod. To illustrate the procedure, we carry it through for the example $x + x^2$ with $L = 1$.

```
In[131]:= fbas[x_] := x + x^2
In[132]:= L = 1;
In[133]:= fbasev[x_] := If[(x > 0), fbas[x], fbas[-x]]
In[134]:= fbasod[x_] := If[(x > 0), fbas[x], -fbas[-x]]
```

Now we periodically extend these, using the Mathematica function Mod. The function Mod[x, 2*L, -L] (1) adds L to x, (2) calculates the remainder on dividing (1) by 2*L, (3) subtracts L from (2). This has the effect of shifting the original x repeatedly by $\pm 2L$ until it falls into the range $[-L,L]$. This modified $x$ is then used as the argument of the original functions fbasev[x] and fbasod[x].

```
In[135]:= fev[x_] := fbasev[Mod[x, 2*L, -L]]
In[136]:= fod[x_] := fbasod[Mod[x, 2*L, -L]]
```

To check our definitions, we plot both fev and fod over three periods. We make the line for the plotted function slightly thicker than the Mathematica default by using the Thickness function with the Option PlotStyle. We first set an ImageSize Option for the entire notebook for the Plot command.

```
In[137]:= SetOptions[Plot, ImageSize -> 250];
```
We now calculate here the Fourier coefficients for the function f. For most of the simple functions we work with (including this one), the coefficient integrals could be evaluated 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 function is called. If that is done, calculations involving the series, such as plotting, would be unacceptably slow. We evaluate the coefficients once and for all and store them in the functions a[n], for the cosine series, and b[n] for the sine series. Although we don’t know a priori how many coefficients will be needed in a particular problem, we calculate the first ncoeffmax coefficients, where we set the default value of ncoeffmax below to 101. If we need more coefficients than that, we might want to think about finding another way to do the problem! For convenience in later calculations, we define b[0] = 0. The function calcoeff, defined below, calculates and stores all of the Fourier coefficients up to order ncoeffmax.

```mathematica
In[142] = ncoeffmax = 101;

In[143] = calcoeff := Module[{}, a[0] = (1.0 / L) NIntegrate[fbas[x] , {x, 0, L}, AccuracyGoal -> 8]; b[0] = 0.0;
Do[(a[n] = (2.0 / L) NIntegrate[fbas[x] Cos[(n π x) / L], {x, 0, L}, AccuracyGoal -> 8];
    b[n] = (2.0 / L) NIntegrate[fbas[x] Sin[(n π x) / L], {x, 0, L}, AccuracyGoal -> 8]), {n, 1, ncoeffmax}]];
```

Now we execute calcoeff to calculate and store the coefficients.

```mathematica
In[144] = calcoeff;
```
3. Terms and Partial Sums of Fourier Series

Now we define the nth term of the Fourier cosine series, called fourtermev, and the nth term of the Fourier sine series, called fourtermod.

\[
\text{In[145]} := \text{fourtermev}[x_\_,n_] := \text{N[a[n]*Cos[(n*\pi*x)/L]]}
\]

\[
\text{In[146]} := \text{fourtermod}[x_\_,n_] := \text{N[b[n]*Sin[(n*\pi*x)/L]]}
\]

The nth partial sums of both series are defined as functions of x and n.

\[
\text{In[147]} := \text{foursumev}[x_\_,n_] := \text{Sum[fourtermev}[x_,k],\{k,0,n\}] \\
\text{In[148]} := \text{foursumod}[x_\_,n_] := \text{Sum[fourtermod}[x_,k],\{k,0,n\]}
\]

4. Graphs of fev[x], fod[x] and Partial Sums of Fourier Series

The function picev[n] gives a graph of the function fev[x] and of the nth partial sum of the Fourier cosine series. The function picod[n] gives a graph of the function fod[x] and of the nth partial sum of the Fourier sine series. To get a plot of the function fev only, use picfuncev. To get a plot of the function fod only, use picfuncod. The functions fev and fod are in blue, and the partial sums of the Fourier series are in red. The range that is plotted can be user-specified, but the default which is used here works for most examples. The left end-point is lfend, the right end-point is rtend. The default, as set below, gives a half-period extension on either end of the basic period. We define a command setrange that sets up the default. It is incorporated into the graphics commands.

\[
\text{In[149]} := \text{setrange} := (\lfend = -1.5*L; \rtend = 1.5*L; \frange = \fmax - \fmin; \plrange = (\lfend,\rtend), (\fmin-0.2*\frange,\fmax+0.2*\frange))
\]

\[
\text{In[150]} := \text{picfuncev} := (\text{setrange}; \text{Plot[fev}[x],\{x,\lfend,\rtend\},\text{PlotStyle} -> \{\text{RGBColor}[0,0,1],\text{Thickness}[0.004]\}, \text{PlotRange} -> \text{plrange}, \text{AxesLabel} -> \{"x","fev[x]","AspectRatio"} -> 1.0))
\]

\[
\text{In[152]} := \text{picfuncod} := (\text{setrange}; \text{Plot[fod}[x],\{x,\lfend,\rtend\},\text{PlotStyle} -> \{\text{RGBColor}[0,0,1],\text{Thickness}[0.004]\}, \text{PlotRange} -> \text{plrange}, \text{AxesLabel} -> \{"x","fod[x]","AspectRatio"} -> 1.0))
\]

\[
\text{In[153]} := \text{picev}[n_] := (\text{setrange}; \text{Plot[\{\text{foursumev}[x,n],fev}[x]\},\{x,\lfend,\rtend\}, \text{PlotStyle} -> \{\text{RGBColor}[1,0,0],\text{Thickness}[0.004]\},\text{RGBColor}[0,0,1],\text{Thickness}[0.004]\}, \text{PlotRange} -> \text{plrange}, \text{AxesLabel} -> \{"x","fev[x]\","AspectRatio"} -> 1.0, \text{PlotLabel} -> \text{Row[\{"n \="},\text{PaddedForm}[n,2]\}]\}))
\]

\[
\text{In[154]} := \text{picod}[n_] := (\text{setrange}; \text{Plot[\{\text{foursumod}[x,n],fod}[x]\},\{x,\lfend,\rtend\}, \text{PlotStyle} -> \{\text{RGBColor}[1,0,0],\text{Thickness}[0.004]\},\text{RGBColor}[0,0,1],\text{Thickness}[0.004]\}, \text{PlotRange} -> \text{plrange}, \text{AxesLabel} -> \{"x","fod[x]\","AspectRatio"} -> 1.0, \text{PlotLabel} -> \text{Row[\{"n \="},\text{PaddedForm}[n,2]\}]\)))
\]

Let's try this out. We produce first graphs of the basic functions, then graphs of the partial sums for \(n = 5\), and then a sequence of graphs of partial sums for each series, up to and including \(n = 5\). We produce the sequences by Do loops.
\textbf{In[155]} = \texttt{picfunc ev}

\texttt{Out[155]=}

\textbf{In[156]} = \texttt{picfuncod}

\texttt{Out[156]=}
In[157]:= picev[5]

Out[157]=

In[158]:= picod[5]

Out[158]=

In[159]:= Do[Print[picev[n]], {n, 1, 5}];
\[ n = 4 \]
\[ \text{lev}(x) \]

\[ n = 5 \]
\[ \text{lev}(x) \]

\[ \text{In}[160] = \text{Do}[\text{Print}[\text{picod}[n]],\{n,1,5\}]; \]
The extended \textit{fod} has discontinuities, and we can see from its graphs that many more terms in the partial sum are needed to get a result closely resembling the original \textit{fod}[x]. 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 store the result for partial sum \(k\) and use it to compute partial sum \(k+1\). 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 \(k\)th 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. We must then change our graphing technique, though, because \textit{Plot} works only for functions defined analytically. For the present case, we can use \textit{ListPlot}, which plots a given numerical set of points. The routines defined here are called \textit{picarrayev} and \textit{picarrayod}, with integer arguments \([\text{first, last, grinc}]\). \textit{First} is the \(n\) value of the first partial sum in the sequence and \textit{last} is the last \(n\) value. The variable \textit{grinc} specifies the step between displayed graphs. All the partial sums are calculated, but by choosing \textit{grinc} greater than 1, you can display every "grincth" graph. The functions defined below are used by \textit{picarray} to calculate coordinate lists and to produce graphs.
In[161]:= SetOptions[ListPlot, ImageSize -> 250];

In[162]:= npoints = 500;

In[163]:= mksumlistev[n_] := Module[{ans, x, inc, j},
ans = {{lfend, foursumev[lfend, n]});
inc = (rtend - lfend)/npoints;
Do[{x, x, lfend + j*inc};
ans = Append[ans, {x, foursumev[x, n]}], {j, 1, npoints}];
ans]

In[164]:= mksumlistod[n_] := Module[{ans, x, inc, j},
ans = {{lfend, foursumod[lfend, n]});
inc = (rtend - lfend)/npoints;
Do[{x, x, lfend + j*inc};
ans = Append[ans, {x, foursumod[x, n]}], {j, 1, npoints}];
ans]

In[165]:= mktermlistev[n_] := Module[{ans, x, inc, j},
ans = {{0.0, fourtermev[lfend, n]});
inc = (rtend - lfend)/npoints;
Do[{x, x, lfend + j*inc};
ans = Append[ans, {x, fourtermev[x, n]}], {j, 1, npoints}];
ans]

In[166]:= mktermlistod[n_] := Module[{ans, x, inc, j},
ans = {{0.0, fourtermod[lfend, n]});
inc = (rtend - lfend)/npoints;
Do[{x, x, lfend + j*inc};
ans = Append[ans, {x, fourtermod[x, n]}], {j, 1, npoints}];
ans]

In[167]:= funcliste v := Module[{ans, x, inc, j},
ans = {{lfend, fev[lfend]});
inc = (rtend - lfend)/npoints;
Do[{x, x, lfend + j*inc};
ans = Append[ans, {x, fev[x]}], {j, 1, npoints}];
ans]

In[168]:= funclistod := Module[{ans, x, inc, j},
ans = {{lfend, fod[lend]});
inc = (rtend - lfend)/npoints;
Do[{x, x, lfend + j*inc};
ans = Append[ans, {x, fod[x]}], {j, 1, npoints}];
ans]

In[169]:= mkgraph[list_, rcol_, gcol_, bcol_] := (setrange; ListPlot[list,
PlotJoined -> True, PlotStyle -> {RGBColor[rcol,gcol,bcol],Thickness[0.004]},
PlotRange -> pltrange, ImageSize -> 250])

In[170]:= picarrayev[first_, last_, grinc_] := Module[
{sumlist, termlist, k, grph, grph0}, setrange;
sumlist = mksumlistev[first];
grph0 = mkgraph[funcliste v, 0, 0, 1];
grph = mkgraph[sumlist, 1, 0, 0];
Print[Show[grph0, grph], AxesLabel -> {"x", "fev[x]"},
AspectRatio -> 1.0, PlotLabel ->
Row["n =", PaddedForm[first, 3]]]);
Do[sumlist = sumlist + mktermlistev[k];
If[0, (grph = mkgraph[sumlist, 1, 0, 0];
Print[Show[grph, grph0], AxesLabel -> {"x", "fev[x]"},
AspectRatio -> 1.0, PlotLabel ->
Row["n =", PaddedForm[k, 2]]);]
{k, first + 1, last}]
As an example, we execute `picarrayev[1,5,4]` and `picarrayod[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, for both the cosine and sine series.
Now we use this new faster method to display the partial sums for the cosine series up to \( n = 20 \), and for the sine series up to \( n = 51 \). We display every graph in the sequence by setting grinc = 1. We collect all of the graphs into a single cell. You can animate the graph sequence by selecting the cell and using the Menu option **Graphics -> Rendering -> Animate Selected Graphics**.

\[
\text{In[175]} = \text{picarrayv[1, 20, 1];}
\]
For visualization in the printed version of this notebook, we show every second graph of the first 9.

\[ n = \frac{1}{\text{fev}[x]} \]
Now we look at the Fourier sine series for the odd extension. In the printed version of the notebook we see only the first graph, which is the top graph in the cell containing all of the graphs. To play the graph sequence as a movie, select the cell and then go to the menu item Graphics->Rendering->Animate Selected Graphics.

\[ \text{In}[177]= \text{picarrayod}[1, 51, 1]; \]

For visualization of the sine series in the printed version of this notebook, we again construct a sequence running up to \( n = 51 \), displaying every 10th graph.

\[ \text{In}[178]= \text{picarrayod}[1, 51, 10]; \]
The Gibbs phenomenon (the overshoot near the discontinuities) shows clearly here.
The Gibbs phenomenon (the overshoot near the discontinuities) shows clearly here.

6. Use of Manipulate to Visualize the Graph Sequences

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 two new commands, manpicarrayev[first,last,grinc] and manpicarrayod[first,last,grinc]. They do exactly what picarrayev and picarrayod except that the outputs are now displayed in Manipulate panels.

```
In[179]= manpicarrayev[first_, last_, grinc_] := DynamicModule[
    {sumlist, termlist, k, grph, grph0, mangraph}, setrange;
    sumlist = mksumlistod[first];
    grph0 = mkgraph[funclistod, 0, 0, 1];
    grph = mkgraph[sumlist, 1, 0, 0];
    mangraph[first] = Show[{grph0, grph}, AxesLabel -> {"x", "fev[x]"},
        AspectRatio -> 1.0, PlotLabel ->
        Row[{"n =", PaddedForm[first, 3]}]];
    Do[sumlist = sumlist + mtclistod[k];
        If[(Mod[k - first, grinc] == 0),
            (grph = mkgraph[sumlist, 1, 0, 0];
                mangraph[k] = Show[{grph0, grph}, AxesLabel -> {"x", "fev[x]"},
                    AspectRatio -> 1.0, PlotLabel ->
                    Row[{"n =", PaddedForm[k, 2]}])],
            {k, first + 1, last}];
    Manipulate[mangraph[n], {n, first, last, grinc}]

In[180]= manpicarrayod[first_, last_, grinc_] := DynamicModule[
    {sumlist, termlist, k, grph, grph0, mangraph}, setrange;
    sumlist = mksumlistod[first];
    grph0 = mkgraph[funclistod, 0, 0, 1];
    grph = mkgraph[sumlist, 1, 0, 0];
    mangraph[first] = Show[{grph0, grph}, AxesLabel -> {"x", "fod[x]"},
        AspectRatio -> 1.0, PlotLabel ->
        Row[{"n =", PaddedForm[first, 3]}]];
    Do[sumlist = sumlist + mtclistod[k];
        If[(Mod[k - first, grinc] == 0),
            (grph = mkgraph[sumlist, 1, 0, 0];
                mangraph[k] = Show[{grph0, grph}, AxesLabel -> {"x", "fod[x]"},
                    AspectRatio -> 1.0, PlotLabel ->
                    Row[{"n =", PaddedForm[k, 2]}])],
            {k, first + 1, last}];
    Manipulate[mangraph[n], {n, first, last, grinc}]
```

Now we use our new commands to produce partial sum sequences for both the cosine and sine series.
The production process is quick, and both of the Manipulate panels have sliders and movies which are very responsive -- no lags for computation.
7. Summary

We summarize the process of creating Manipulate displays of partial sums of Fourier series by doing one more example. We will obtain the displays of the sine and cosine series for the base function \( fbas[x] = x(2L - x) \), with \( L = 2 \). We define this for \textit{Mathematica}, after clearing any earlier definitions.

\begin{verbatim}
In[183]:= Clear[fev, fod, fbasev, fbasod, a, b];
In[184]:= fbas[x_] := x(2 L - x)
In[185]:= L = 2.0;
\end{verbatim}

Now we define the even and odd extensions, and then extend both periodically.

\begin{verbatim}
In[186]:= fbasev[x_] := If[(x ≥ 0), fbas[x], fbas[-x]]
In[187]:= fbasod[x_] := If[(x ≥ 0), fbas[x], -fbas[-x]]
In[188]:= fev[x_] := fbasev[Mod[x, 2*L, -L]]
In[189]:= fod[x_] := fbasod[Mod[x, 2*L, -L]]
\end{verbatim}

The function \( fbas \) has a maximum absolute value of 4 on the interval (at \( x = L \)), so we assign \( fmin \) and \( fmax \) as

\begin{verbatim}
In[190]:= fmin = -4.0; fmax = 4.0;
\end{verbatim}

Now we check our work by plotting our two extended functions.

\begin{verbatim}
In[191]:= picfuncod
\end{verbatim}

\begin{center}
\includegraphics[width=0.5\textwidth]{picfuncod.png}
\end{center}
We calculate the Fourier coefficients.

Finally we create the two Manipulate panels with up to 20 terms in the partial sum of the cosine series, and up to 51 terms in the partial sum of the sine series.
In[195]:= manpicarrayod[1, 51, 1]

Out[195]=

\[ n - 3 - 2 - 1 \]

\[ 2 - 4 - 2 \]

\[ fOD[x] \]

\[ n = \frac{1}{fOD[x]} \]