

ME 201/MTH 281/ME400/CHE400

Convergence of Fourier Series

Mathematica 7

Version 1.6

■ 1. Introduction

This notebook provides a general framework for the visualization of partial sums of Fourier series. To use it for a specific 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 a function which produces a graph of the n th 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 . A number of examples are given in Section 6. In Section 7, we take a different approach to the visualization by using the powerful *Mathematica* command `Manipulate`, which is new to version 6. As you will see there, one line of code produces the graph sequence along with a convenient set of controls for visualizing the sequence.

■ 2. Definition of Function and Fourier Coefficients

In the definitions below, we first define the function over a basic interval $[-L,L]$, specify the value of L , and then extend the function periodically. To illustrate the procedures as we go, we will use the square wave as an example. The name used for the basic function defined over $[-L,L]$ is `fbas`. The periodically extended function is called `f`.

```
fbas[x_] := -1 /; -L ≤ x < 0 (* Square Wave *)
```

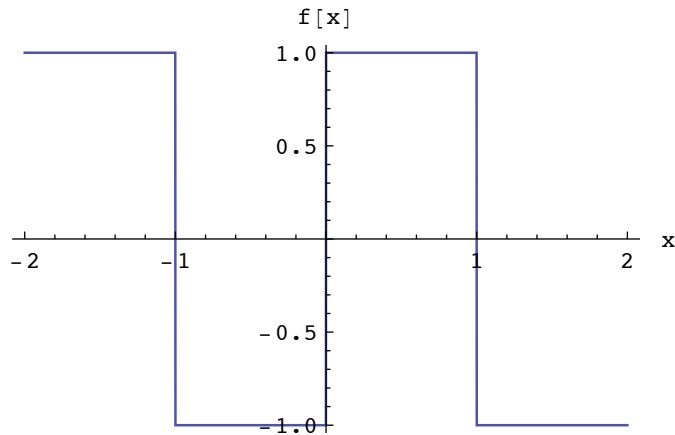
```
fbas[x_] := 1 /; 0 ≤ x ≤ L
```

```
L = 1;
```

```
f[x_] := fbas[Mod[x, 2 * L, -L]]
```

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 by $\pm 2L$ until it falls into the range $[-L,L]$. This modified x is then used as the argument of the original function `f[x]`. We plot this over $[-2L, 2L]$.

```
Plot[f[x], {x, -2 L, 2 L}, AxesLabel -> {"x", "f[x]"}, PlotStyle -> Thickness[0.004]]
```



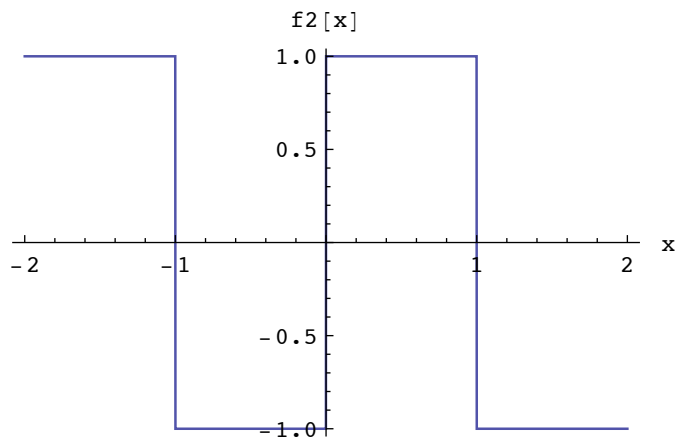
An alternative way to extend fbas periodically is by recursion. The definition is given below.

```
f2[x_] := fbas[x] /; -L ≤ x ≤ L
```

```
f2[x_] := f2[x - 2 L] /; x > L
```

```
f2[x_] := f2[x + 2 L] /; x < -L
```

```
Plot[f2[x], {x, -2 L, 2 L}, AxesLabel -> {"x", "f2[x]"}, PlotStyle -> Thickness[0.004]]
```



For purposes of plotting yet to come, we specify the minimum and maximum of the function f , denoted by f_{\min} and f_{\max} .

```
fmin = -1.0;
```

```
fmax = 1.0;
```

We give here the Fourier coefficients for the function f (in this case the square wave). We assume these have been calculated elsewhere, and the values are supplied here in the definitions of the coefficients. We denote the cosine coefficients by $a[n]$ and the sine coefficients by $b[n]$. For convenience in later calculations, we define $b[0] = 0$.

```
a[n_] := 0.0;
```

```

b[0] := 0;

b[n_] := If[EvenQ[n], 0, (4 / (n * π))]

```

EvenQ is a logical function of an integer which returns True if its argument is even, False otherwise.

■ 3. Terms and Partial Sums of Fourier Series

Now we define the n th term of the Fourier series, called fourterm.

```

fourterm[x_, n_] :=
  N[a[n] * Cos[(n * π * x) / L] + b[n] * Sin[(n * π * x) / L]]

```

The n th partial sum of the series is foursum, given by

```

foursum[x_, n_] := Sum[fourterm[x, k], {k, 0, n}]

```

■ 4. Graphs of $f[x]$ and Partial Sums of Fourier Series

The function pic[n] gives a graph of the function $f[x]$ and of the n th partial sum of the Fourier series. To get a plot of the function f only, use picfunc. The function f is in blue, and the Fourier series is in red. The range that is plotted is user-specified. 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.

```

lfend = -1.5*L;

rtend = 1.5*L;

frange = fmax - fmin;

pltrange =
{ {lfend, rtend}, {fmin - 0.2*frange, fmax + 0.2*frange} };

picfunc := Plot[f[x], {x, lfend, rtend}, PlotStyle ->
  {RGBColor[0, 0, 1], Thickness[0.004]}, PlotRange -> pltrange,
  AxesLabel -> {"x", "f[x]"}, AspectRatio -> 1.0]

pic[n_] := Plot[{foursum[x, n], f[x]}, {x, lfend, rtend},
  PlotStyle ->
{ {RGBColor[1, 0, 0], Thickness[0.004]}, {RGBColor[0, 0, 1], Thickness[0.004]} },
  PlotRange -> pltrange,
  AxesLabel -> {"x", "f[x]"}, AspectRatio -> 1.0,
  PlotLabel -> Row[{"n =", PaddedForm[n, 2]}]]

```

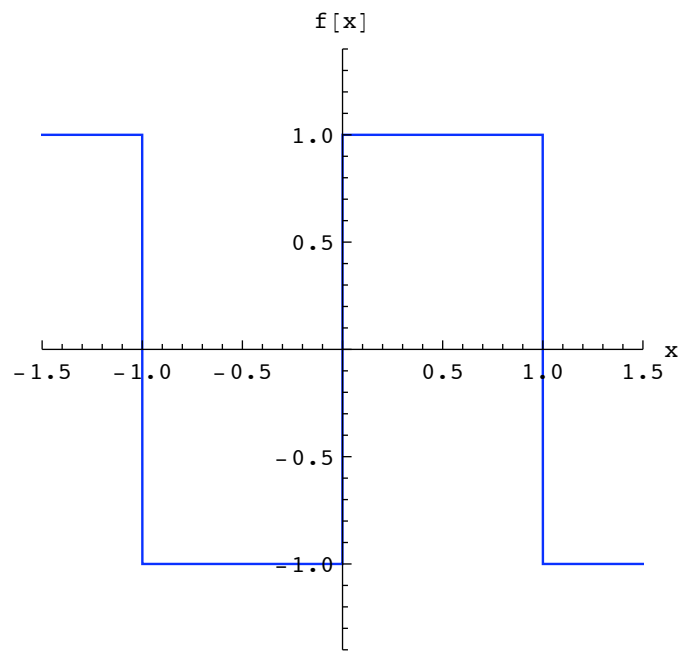
Let's try this out. We produce first a graph of the basic function, then a graph of the partial sums for $n = 5$, and then a sequence of graphs of partial sums up to and including $n = 5$. We produce the sequence by a Do loop, and we increment in steps of two, because only the odd- n coefficients are non-zero for this function. We first set the graph size to 250.

```

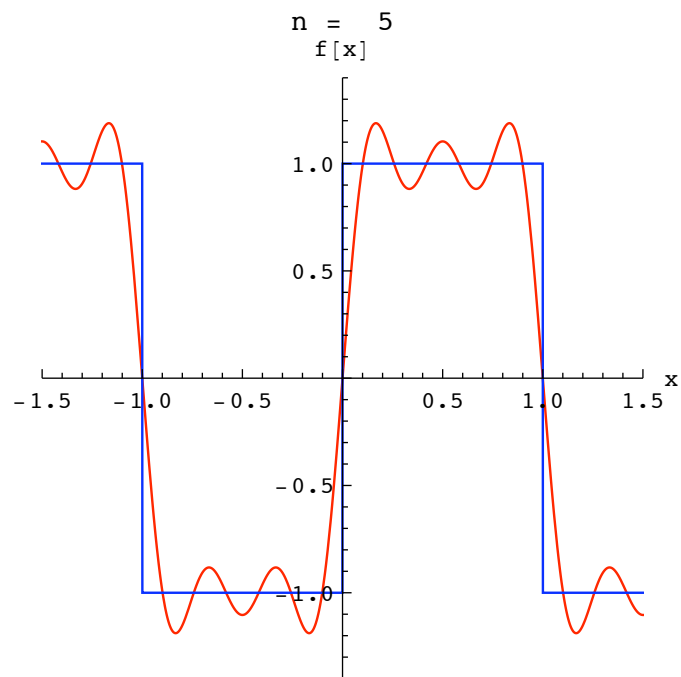
SetOptions[Plot, ImageSize -> 250];

```

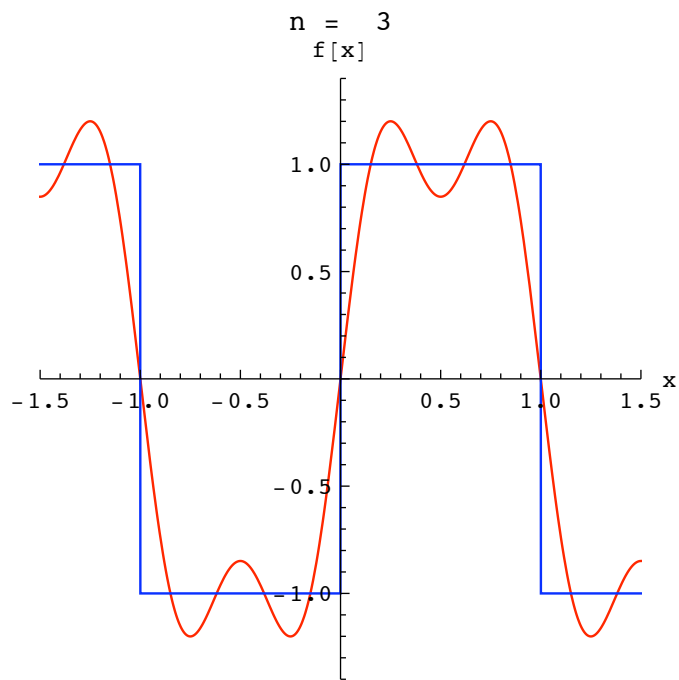
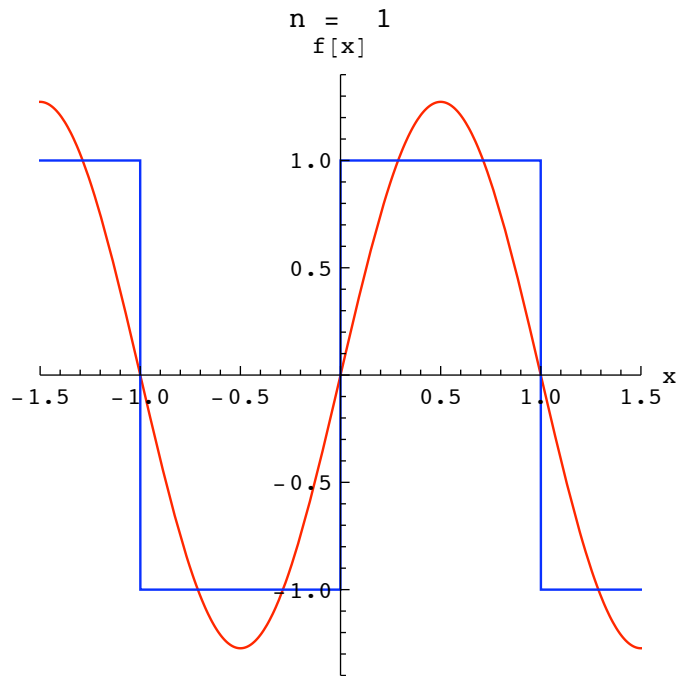
picfunc

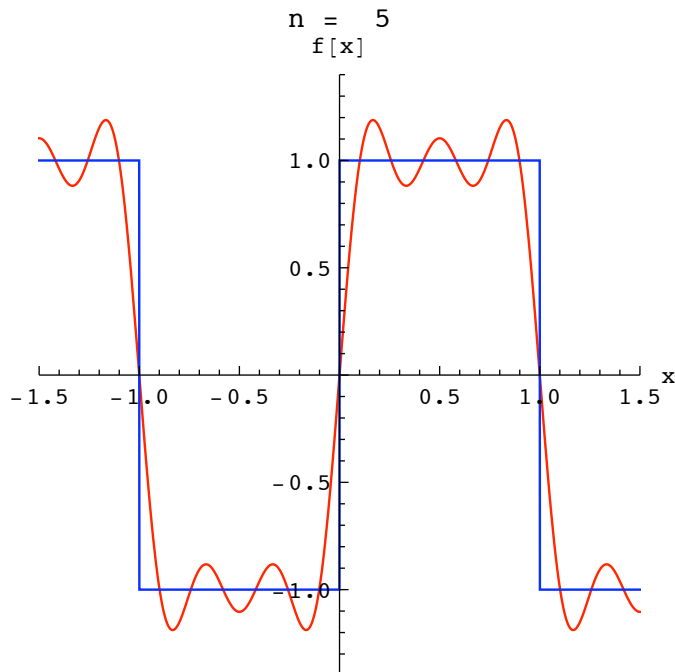


pic[5]



Do[Print[pic[n]], {n, 1, 5, 2}]





If you select and animate this last sequence of graphs (after selecting the graphs, choose the menu item **Graphics -> Rendering -> Animate Selected Graphics**), you will get a nice dynamic view of the convergence process. You will also see that many more partial sums are needed to get a result closely resembling the original $f[x]$. 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 Plot works only for functions defined analytically. For the present case, we can use ListPlot, which plots a given numerical set of points. The routine defined here is called picarray[first, last, grinc, ninc], 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 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. The variable ninc is normally set equal to 1. It specifies the interval at which partial sums are calculated. For the square wave for example, only the odd Fourier coefficients are non-zero. In that case we can specify ninc = 2, and it will save the calculation of the zero terms for even n . The first four functions defined below are used by picarray to calculate coordinate lists and to produce graphs.

```
SetOptions[ListPlot, ImageSize -> 250];
```

```
npoints = 500;
```

```

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

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

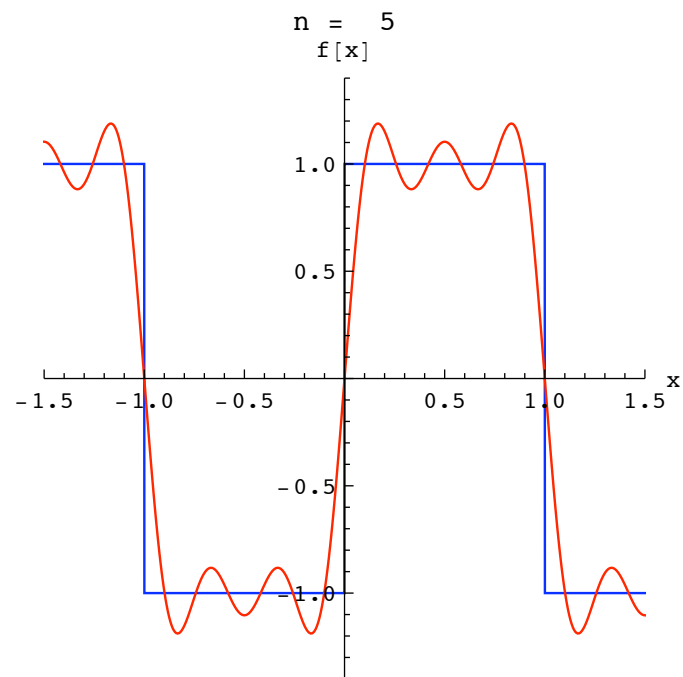
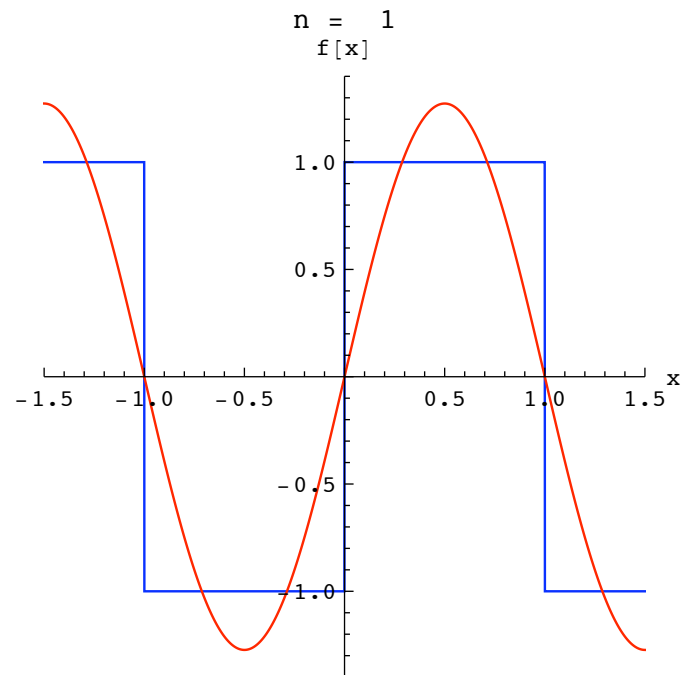
mkgraph[list_, rcol_, gcol_, bcol_] := ListPlot[list, Joined -> True,
  PlotStyle -> {RGBColor[rcol, gcol, bcol], Thickness[0.004]}, PlotRange -> pltrange]

picarray[first_,last_,grinc_,ninc_] := Module[
  {sumlist,termlist,k,grph,grph0},
  sumlist = mksumlist[first];
  grph0 = picfunc;
  grph = mkgraph[sumlist,1,0,0];
  Print[Show[{grph0,grph},AxesLabel -> {"x","f[x]"},
  AspectRatio -> 1.0, PlotLabel ->
  Row[{"n =",PaddedForm[first,2]}]];
  Do[sumlist = sumlist+mktermlist[k];
    If[(Mod[k-first,grinc]== 0),
      (grph = mkgraph[sumlist,1,0,0];
      Print[Show[{grph0,grph},AxesLabel -> {"x","f[x]"},
      AspectRatio -> 1.0, PlotLabel ->
      Row[{"n =",PaddedForm[k,2]}]]]),
    {k,first+ninc,last,ninc}]]

```

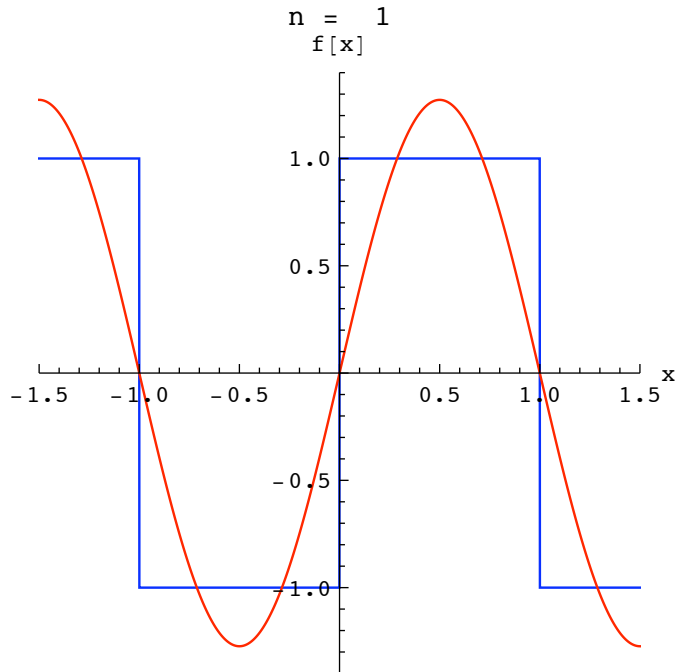
As an example, we execute `picarray[1,5,4,2]`. This will start with $n = 1$ and then display every 4th graph up to $n = 5$ -- i.e., graphs 1 and 5. The last integer 2 tells `picarray` that Fourier coefficients are nonzero only for every second n . This saves computing the zero terms to add to the partial sums.

```
picarray[1,5,4,2]
```



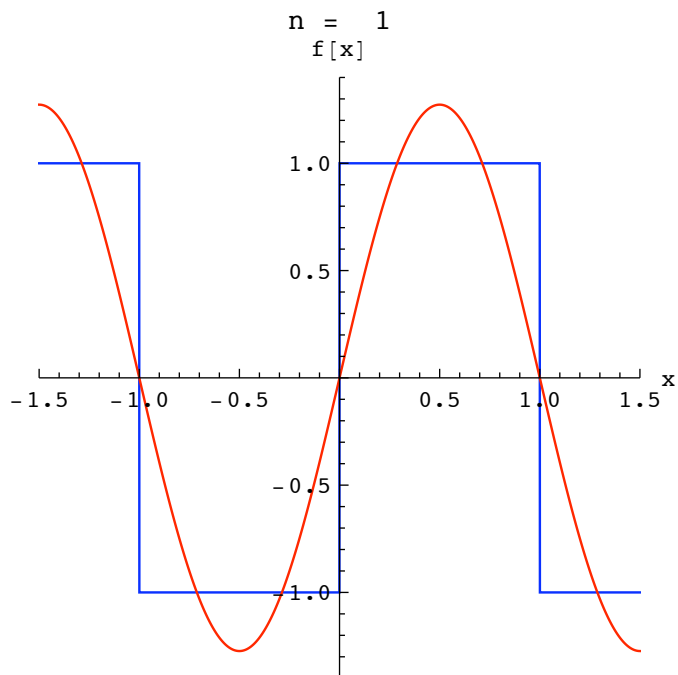
Now we use this new faster method to display the partial sums for the square wave up to and including $n = 51$. We display every graph in the sequence by setting `grinc = 1`. In the printed version of this notebook we collect all of the graphs in a single cell and display only the first graph. To animate the graph sequence as a movie, select the group and go to the menu **Graphics-Rendering->Animate Selected Graphs**

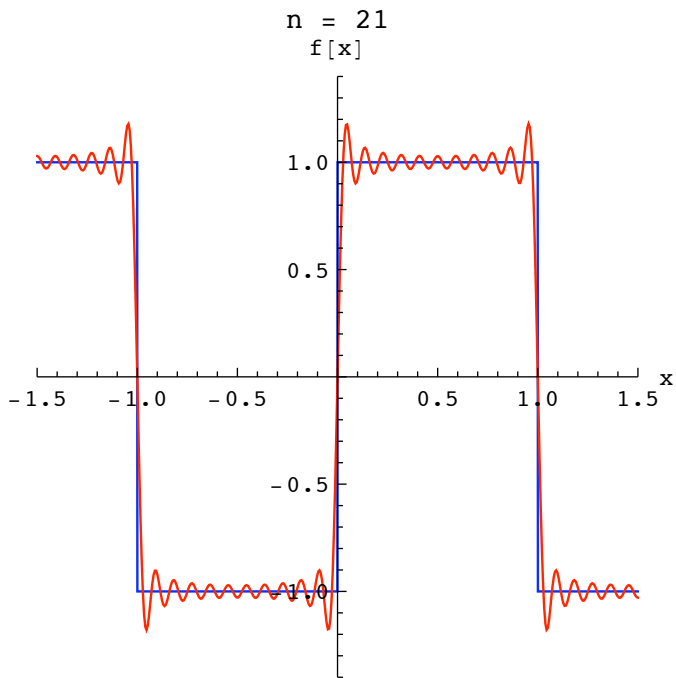
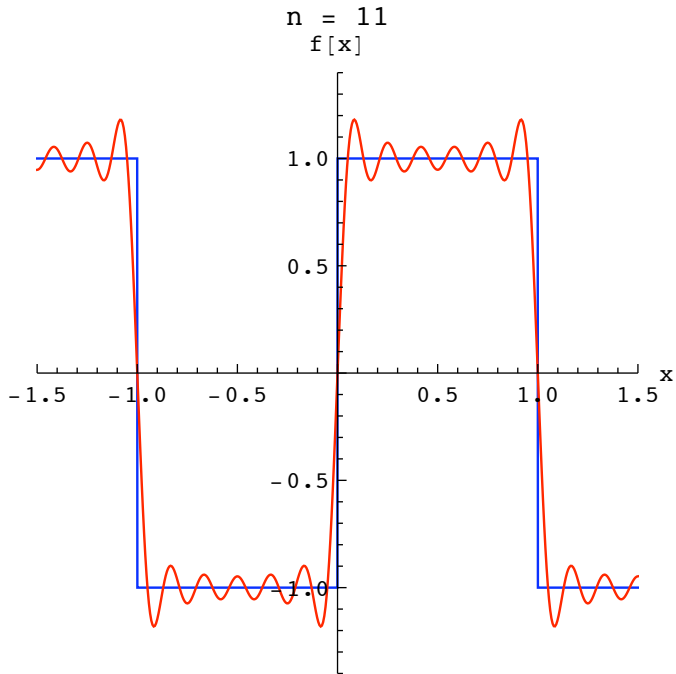
```
picarray[1,51,1,2];
```

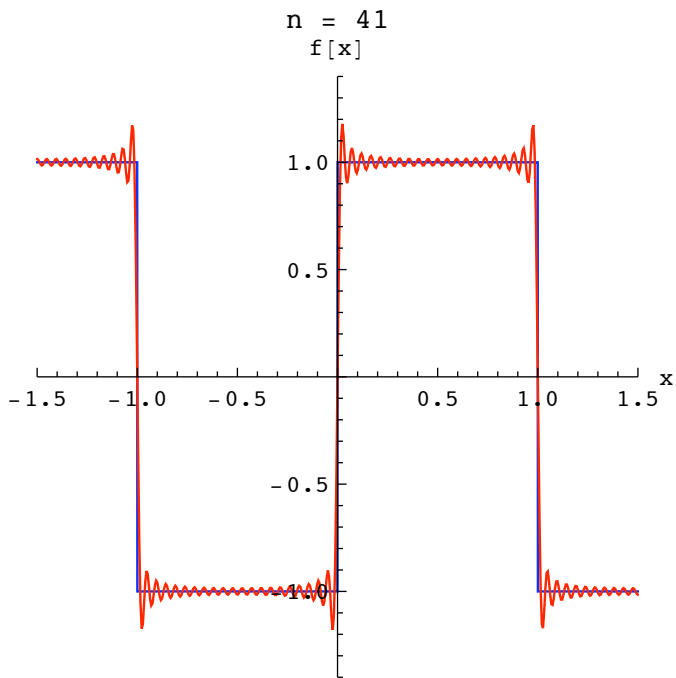
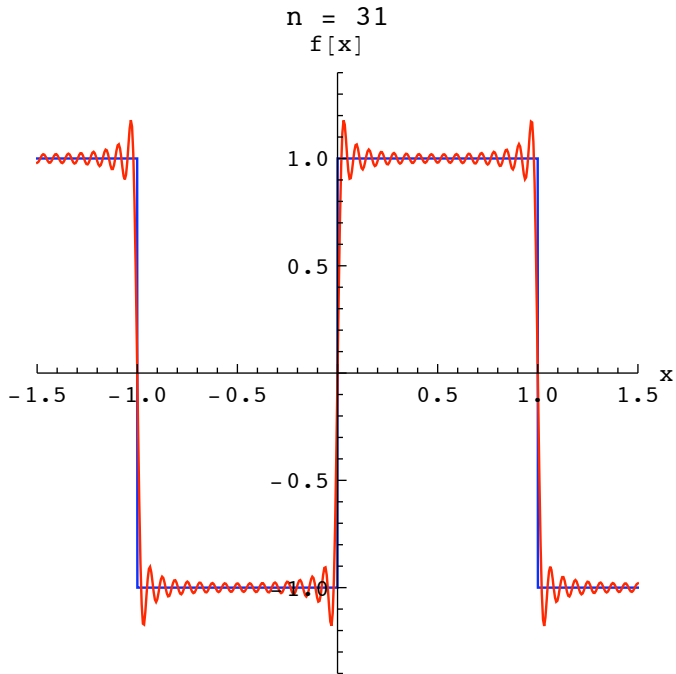


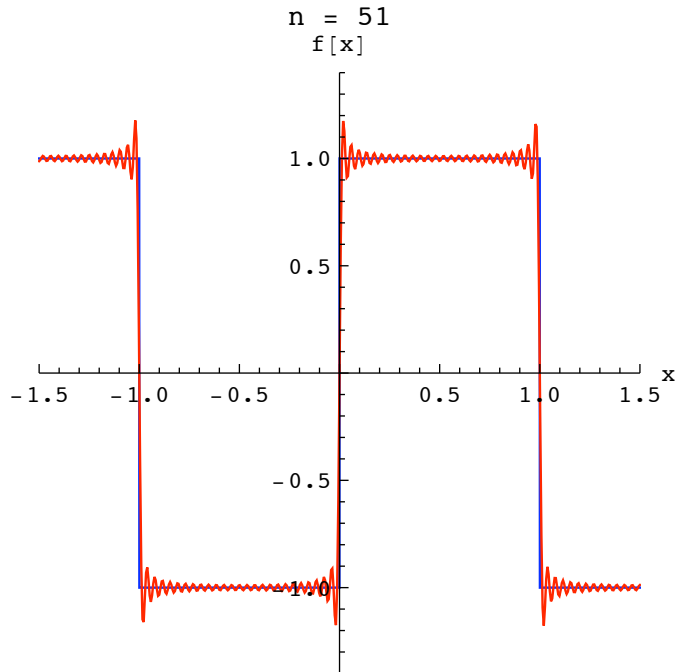
For visualization in the printed version we again construct a sequence running up to $n = 51$, but we display only every 10th graph.

```
picarray[1, 51, 10, 2];
```









The Gibbs phenomenon (the overshoot near the discontinuities) shows clearly here.

■ 6. Other Examples

To illustrate the discussion of convergence given in class, we look at some other examples here. The square wave, which we have just examined, has rather poor convergence (like $1/n$) because the periodically extended function is discontinuous. The Gibbs phenomenon is directly associated with the discontinuities. We look at several other examples here to illustrate the connection between function behavior and convergence.

■ Sawtooth Wave

For our next example, we take the linear function $f_{bas}[x] = x$ on $[-1,1]$, and periodically extended. We must redefine the function and the Fourier coefficients.

```
Clear[fbas, f]

fbas[x_] := x

L = 1;

f[x_] := fbas[Mod[x, 2 * L, -L]]

fmin = -L;

fmax = L;
```

We use the values calculated in class for the Fourier coefficients.

```
Clear[a, b];
```

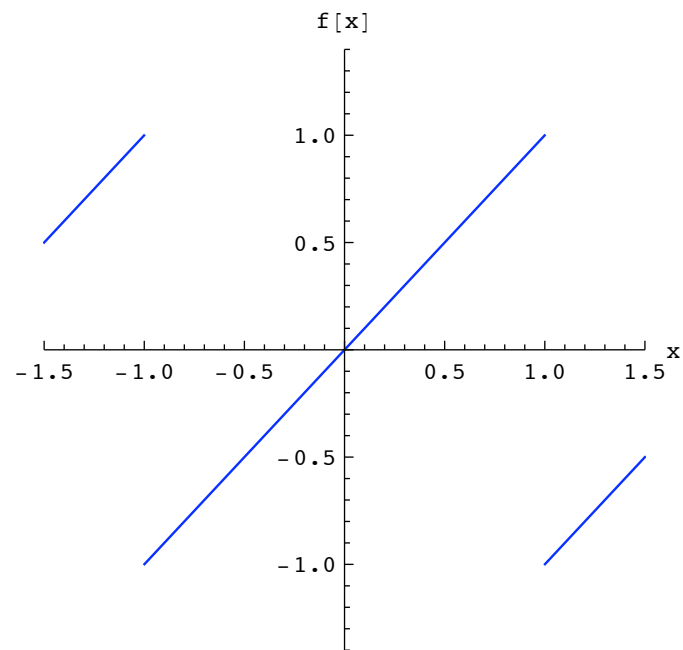
```
a[n_] := 0.0;
```

```
b[0] := 0;
```

```
b[n_] := ((2 * L) / (n * π)) * (-1)n+1
```

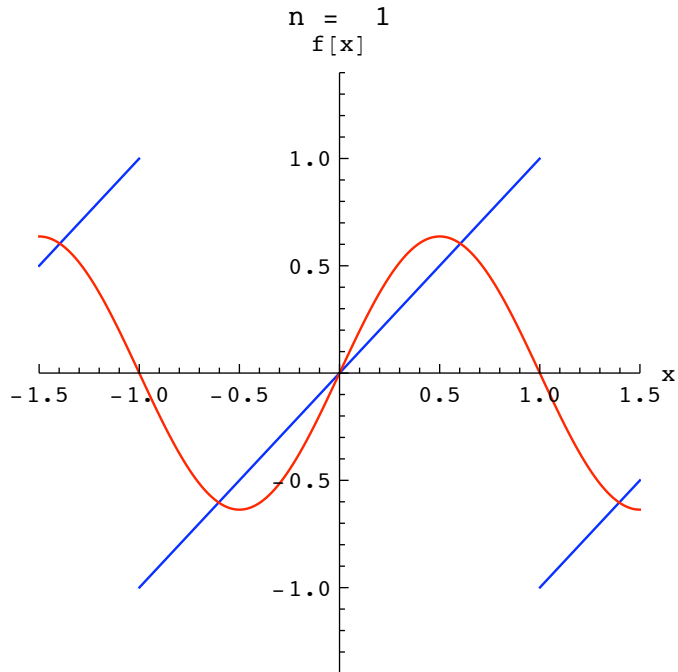
As a check, we plot the periodically extended function.

```
picfunc
```



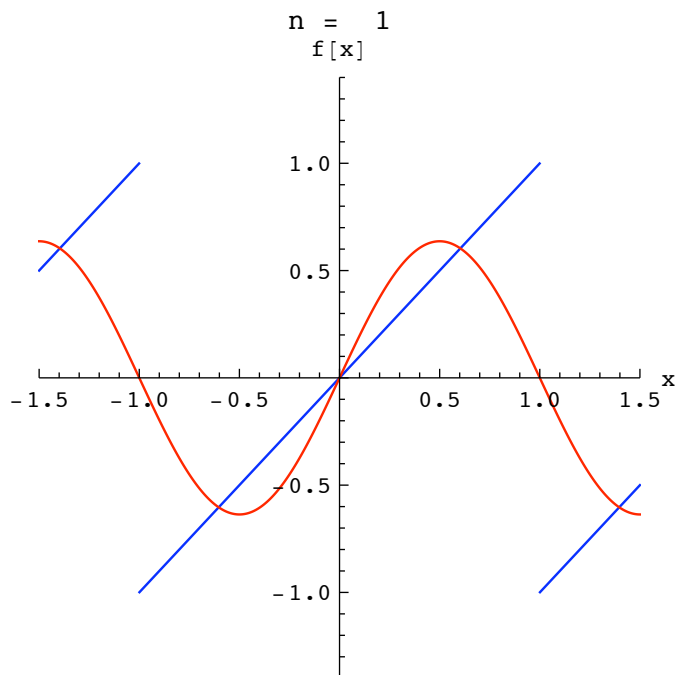
Now we construct an array of partial sums, running from $n = 1$ to $n = 51$. In the printed version we show only the first graph.

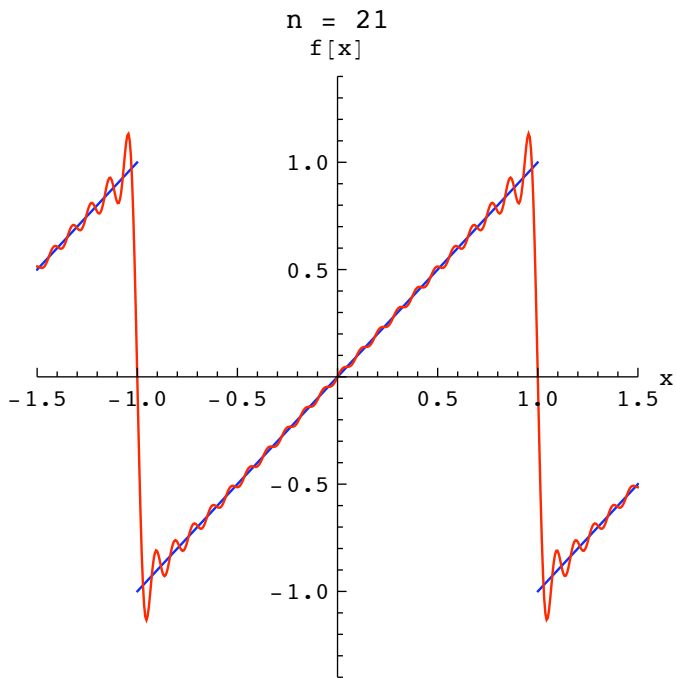
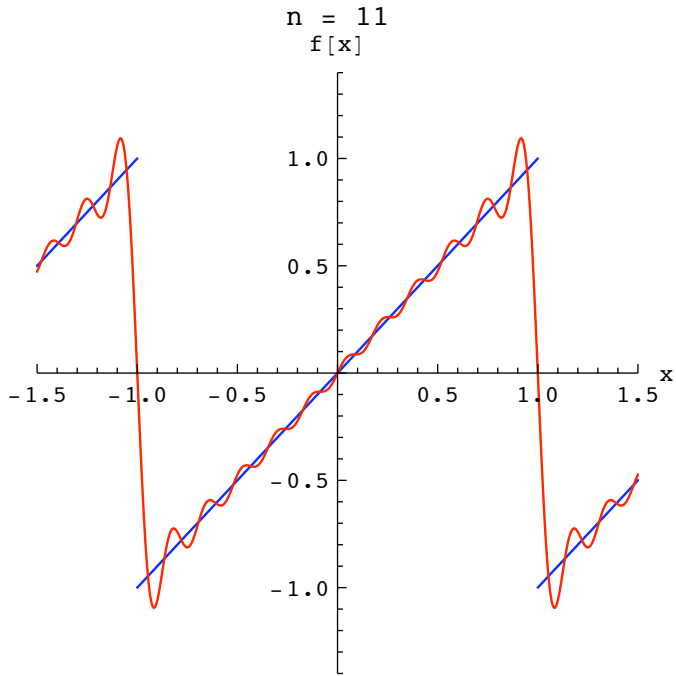
```
picarray[1, 51, 1, 1];
```

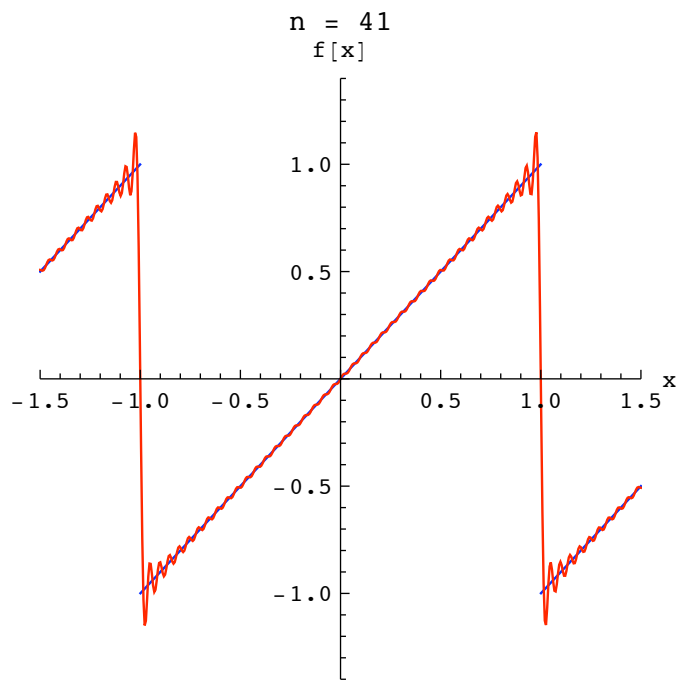
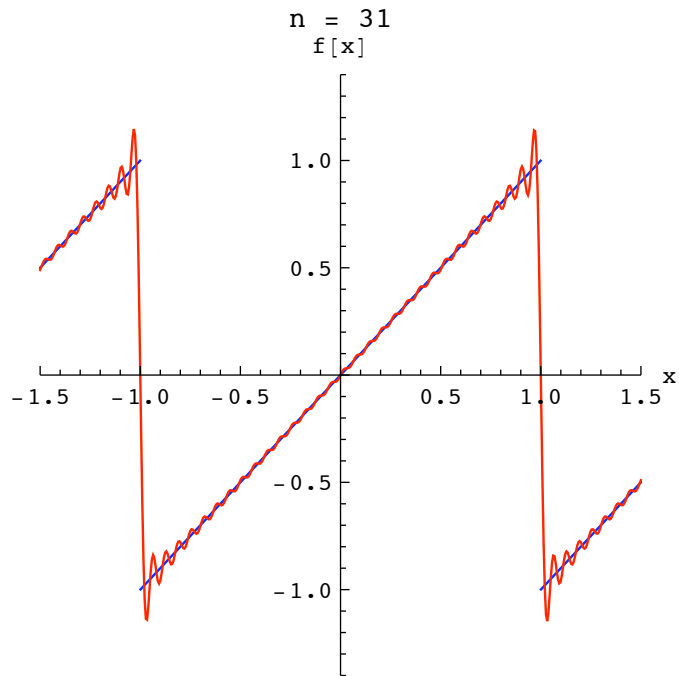


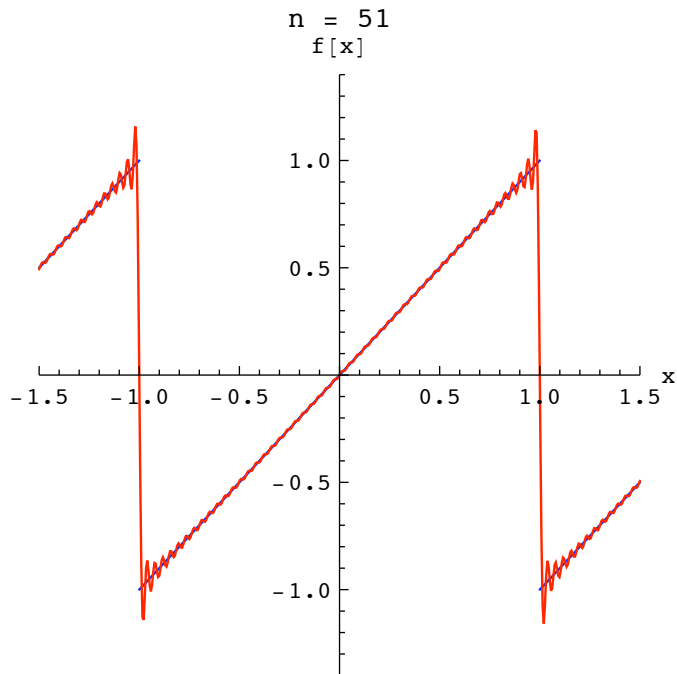
For visualization in the printed version, we again construct a sequence running up to $n = 51$, but we display only every 10th graph.

```
picarray[1, 51, 10, 1];
```









We see again the Gibbs phenomenon. This function, unlike the square wave, is continuous in the basic interval, but the periodically extended function has discontinuities at the end points, so the convergence behavior is no better than that of the square wave.

■ Quadratic Function

Now we consider a function which, when periodically extended, is continuous. The function is $f_{\text{bas}}[x] = x^2$. We expect more rapid convergence and no Gibbs phenomenon.

```

Clear[fbas, f]

fbas[x_] := x^2

L = 1;

f[x_] := fbas[Mod[x, 2 * L, -L]]

fmin = 0;

fmax = L^2;

```

We use the values calculated in class for the Fourier coefficients.

```

Clear[a, b];

a[0] := L^2 / 3;

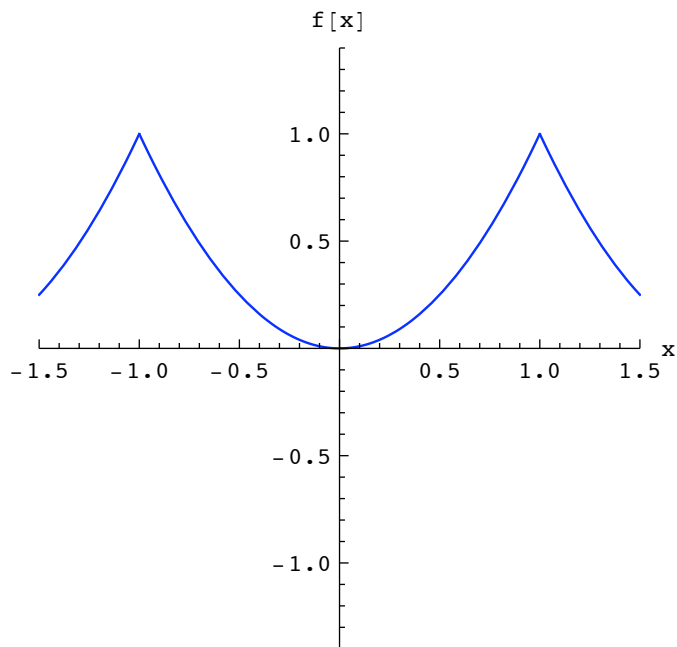
a[n_] := ((4 * L^2) / (n^2 * π^2)) * (-1)^n

b[n_] := 0;

```

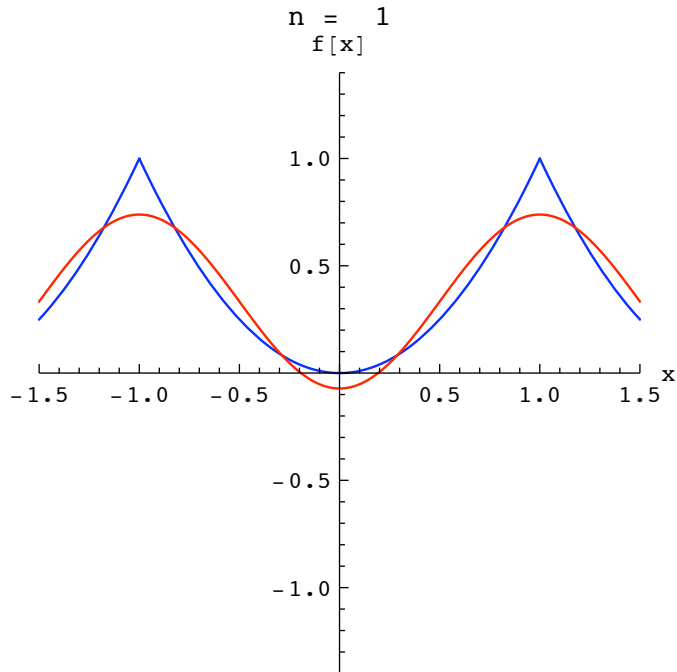
As a check, we plot the periodically extended function.

```
picfunc
```



We see that the periodically extended function is continuous. We also see that its derivative is discontinuous at the endpoints. This is consistent with the $1/n^2$ behavior of the Fourier coefficients for large n . Roughly speaking, with $1/n^2$ convergence, 10 terms will be enough for to reduce the truncation error to about 1%. We look at a sequence of partial sums from $n = 1$ to $n = 21$. Animate the graph group to see the convergence. In the printed version of the notebook only the first graph is shown.

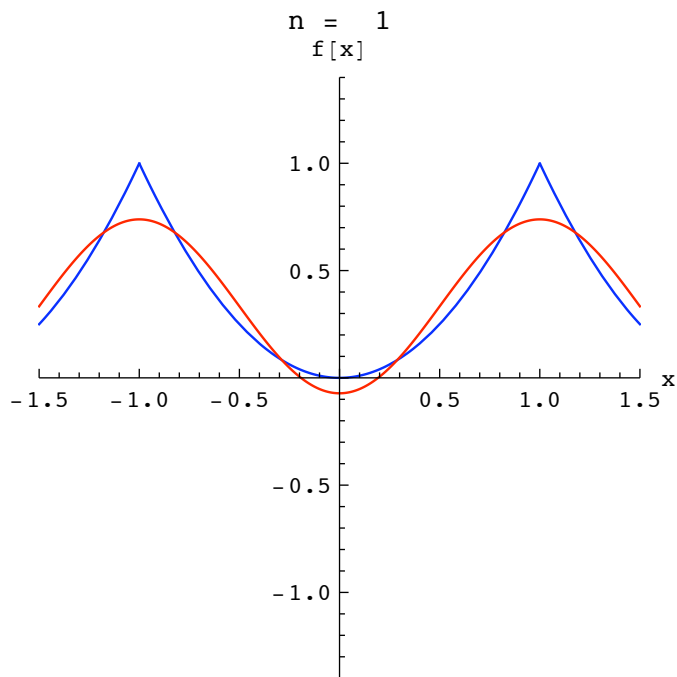
```
picarray[1, 21, 1, 1];
```

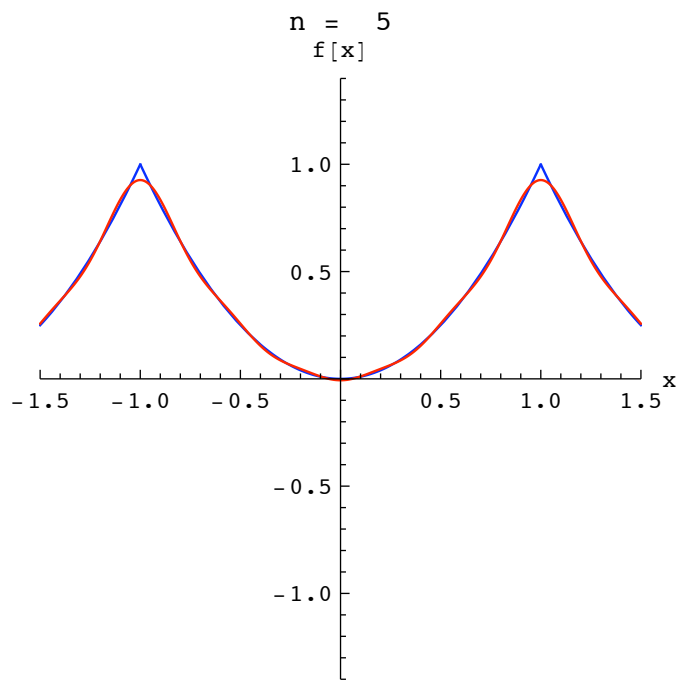
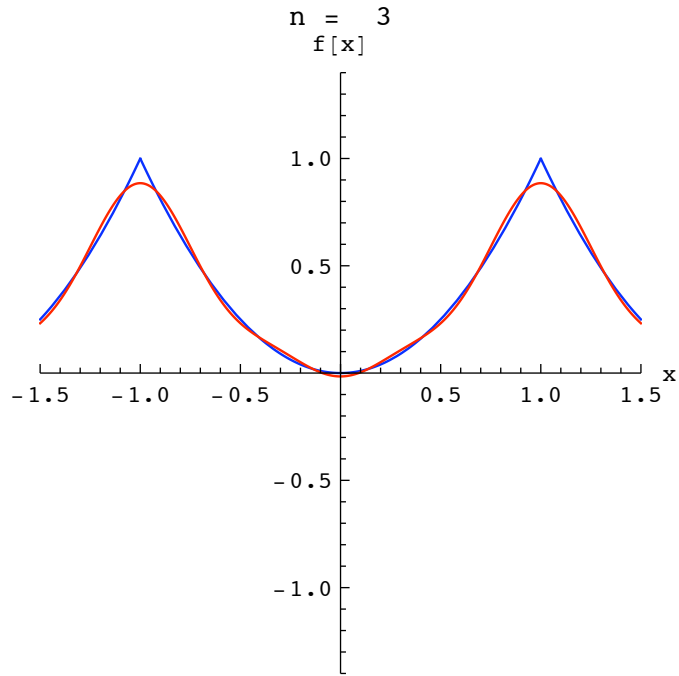


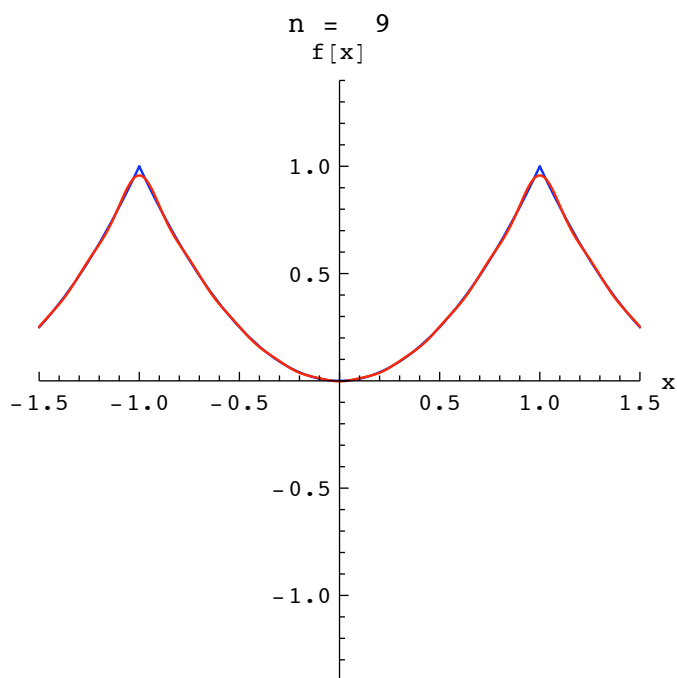
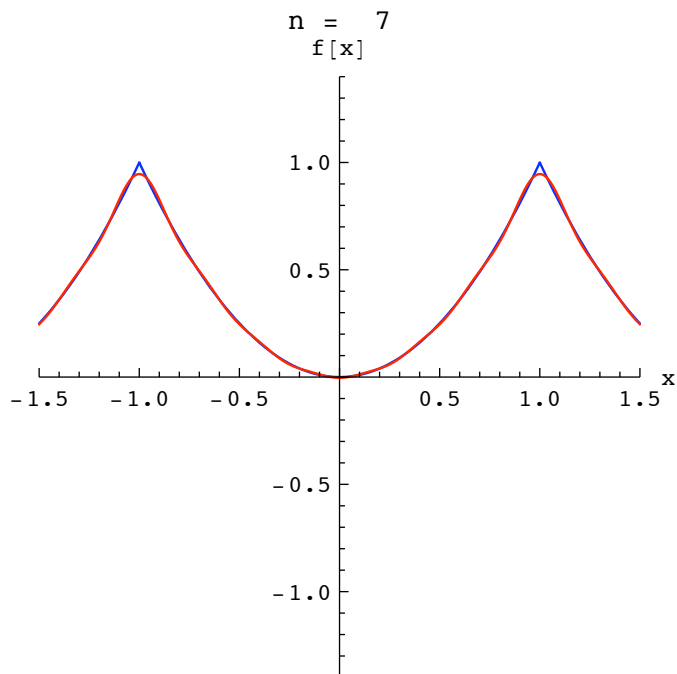
We see that the convergence is indeed rapid. For most values of x , 5 or even 3 terms is enough for graphical accuracy, the exception being the neighborhoods of the discontinuities in the derivative.

For visualization in the printed version, we show the graphs up to $n = 1, 3, 5, 7$ and 9 .

```
picarray[1, 9, 2, 1];
```







■ 7. Use of the Command Manipulate to Visualize the Partial Sums

In the notebook *Convergence of Fourier Series*, there were many lines of *Mathematica* code 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`. The power of this command is only evident in a fully interactive mode, so you need to execute this *Mathematica* notebook to appreciate what `Manipulate` can do. We will illustrate the use of `Manipulate` with the sawtooth wave, so we repeat the definition of the function first.

```
Clear[fbas, f]

fbas[x_] := x

L = 1;

f[x_] := fbas[Mod[x, 2 * L, -L]]

fmin = -L;

fmax = L;
```

We use the values calculated in class for the Fourier coefficients.

```
Clear[a, b];

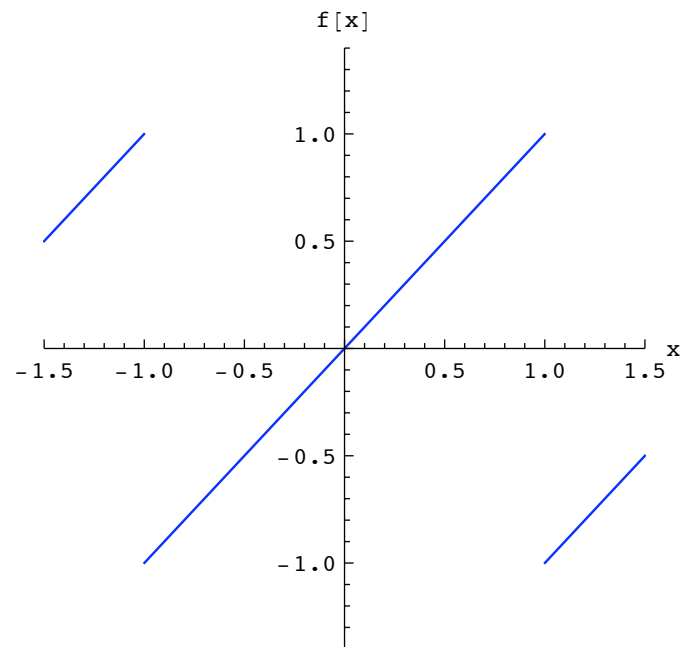
a[n_] := 0.0;

b[0] := 0;

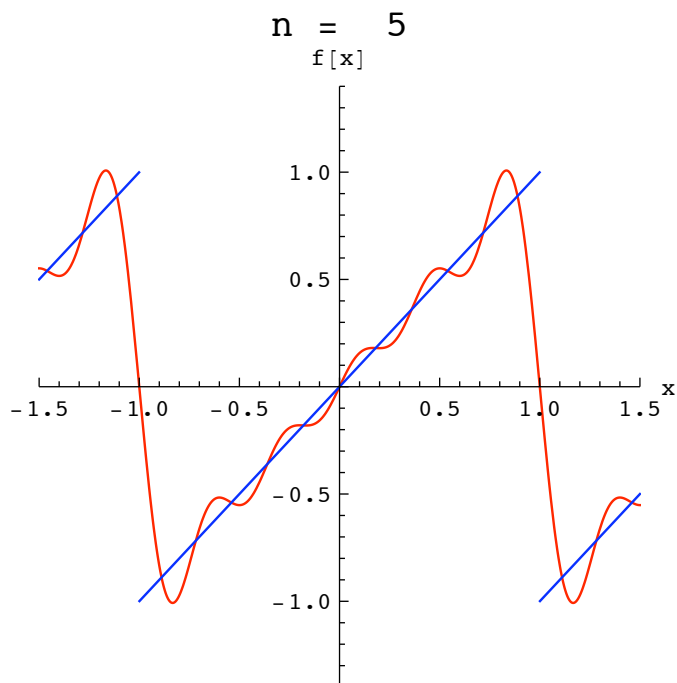
b[n_] := ((2 * L) / (n *  $\pi$ )) * (-1)n+1
```

As a check, we plot the periodically extended function.

picfunc

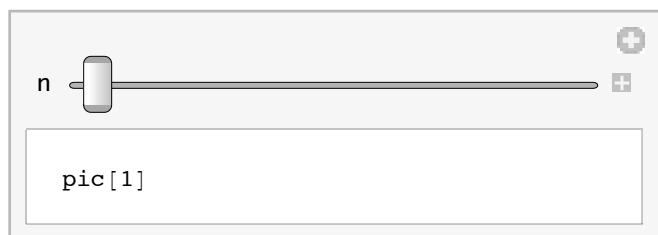


`pic[5]`



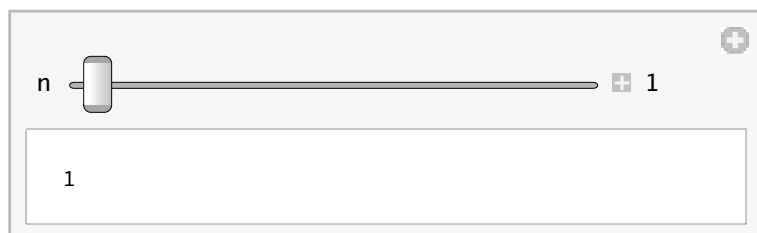
Now we execute the `manipulate` command, and then give some description of what it does. The specific command given below tells *Mathematica* to construct the graphs `pic[n]` for n from 1 to 50, with n incremented by 1.

`Manipulate[pic[n], {n, 1, 50, 1}]`



As soon as you execute this command, you see the graph above, which gives the basic function and the first partial sum. Notice the slider above the graph. By dragging the slider button with the mouse, you will get graphs for different values of n . You can drag the button back and forth to get the graph for any value of n between 1 and 50. Basically this command produces a table of the graphs from $n = 1$ to $n = 50$, and we display different elements in the table by moving the slider. Perhaps the way this works is clearer from a simpler example.

`Manipulate[n^2, {n, 1, 50, 1, Appearance -> "Labeled"}]`



Now as we move the slider, we get the table elements in succession, the elements in this case being the squares of the integers from 1 to 50. The Option setting Appearance->"Labeled" causes the value of n to be displayed just to the right of the slider bar.

Now go back to the graph produced by the earlier Manipulate command and click on the small plus sign just to the right of the slider bar. You get a set of controls to display the graph sequence as a movie. When you pass the cursor over any of the control symbols, a pop-up box will tell you the function of that control. Press the play control and you will get a movie of the graph sequence. This is clearly a much easier way to produce a movie than in the earlier *Mathematica* notebook Convergence of Fourier Series. There is a lot of computation going on during the display. On an older slower machine, the movie may appear a bit slow and jerky. It appeared this way on my old G4, but it is smooth on my MacBook Pro. Be warned though that you will not get a smooth movie from Manipulate if your graph sequence requires a lot of computation. In that case, the best method is to create the sequence with a Do loop and to display it with **Animate Selected Graphics**.