

ME 201/MTH 281ME400/CHE400

Convergence of Bessel Expansions

Mathematica 7

■ 1. Introduction

This notebook provides a general framework for the construction and visualization of partial sums of Fourier-Bessel series. To use it for a specific function, you must define the function in Section 2 below. The Fourier-Bessel basis functions are defined in Section 3, and the Fourier-Bessel coefficients are constructed in Section 4. Section 5 contains the definitions of the terms and partial sums of the series, and Section 6 defines a function which produces a graph of the n th partial sum for any n . Section 7 defines a computationally efficient routine for producing a sequence of all partial sums up to a specified value n . The function expanded to illustrate the technique is $f(r) = (r/a)(1 - r/a)$. In section 8, a second example is considered, namely $f(r) = 1$. In section 9, we use the functions J_1 and again expand $f(r) = 1$.

■ 2. Definition of Function

The function to be expanded is defined below. The interval is $[0,a]$, with the value of a being assigned below.

```
f[r_] := (r/a)*(1-r/a)
a = 3.0;
```

■ 3. Specification of Fourier-Bessel Basis Functions

We define here the un-normalized basis functions, the normalization integral, and the eigenvalues. These will depend on the interval and the boundary conditions. In the example given below, the interval has left end-point 0, right end-point a , and the boundary condition of zero function at the right end-point. The eigenfunctions (which in this case are J_0 's) are denoted by $\Psi[r,n]$. The normalization integral is denoted by $\text{norm}[n]$. The maximum number of terms used in any partial sum is n_{max} .

```
nmax = 51;
```

We begin by getting the first n_{max} zeros of J_0 .

```

beszer = N[Table[BesselJZero[0,i],{i,1,nmax}]]

{2.40483, 5.52008, 8.65373, 11.7915, 14.9309, 18.0711, 21.2116, 24.3525,
 27.4935, 30.6346, 33.7758, 36.9171, 40.0584, 43.1998, 46.3412, 49.4826,
 52.6241, 55.7655, 58.907, 62.0485, 65.19, 68.3315, 71.473, 74.6145, 77.756,
 80.8976, 84.0391, 87.1806, 90.3222, 93.4637, 96.6053, 99.7468, 102.888, 106.03,
 109.171, 112.313, 115.455, 118.596, 121.738, 124.879, 128.021, 131.162,
 134.304, 137.446, 140.587, 143.729, 146.87, 150.012, 153.153, 156.295, 159.437}

```

Now we construct a list with the square roots of the eigenvalues.

```
eiger = beszer/a;
```

Now we can define the eigenfunctions.

```
Ψ[r_,n_] := BesselJ[0,r*eiger[[n]]]
```

Finally we define the normalization integral, and calculate the first nmax normalization integrals by NIntegrate. They also can be found analytically.

```

norm[n_] := NIntegrate[r*(Ψ[r,n])^2,{r,0,a}]

normlist = Module[{ans}, ans = {}; Do[ans = Append[ans, norm[n]], {n, 1, nmax}]; ans];

```

■ 4. Specification of Fourier-Bessel Coefficients

The Fourier-Bessel coefficients are now obtained numerically by using NIntegrate. The coefficients are denoted by fbc[n], and are then calculated and stored in a list named coeff.

```

fbc[n_] := NIntegrate[r*Ψ[r,n]*f[r],{r,0,a}]/normlist[[n]]

coeff = Module[{ans}, ans = {}; Do[ans = Append[ans, fbc[n]], {n, 1, nmax}]; ans];

```

■ 5. Terms and Partial Sums of Fourier-Bessel Series

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

```
fourterm[r_,n_] := coeff[[n]]*Ψ[r,n]
```

The nth partial sum of the series is foursum, given by

```
foursum[r_,n_] := Sum[fourterm[r,k],{k,1,n}]
```

■ 6. Graphs of $f[r]$ and Partial Sums of Fourier-Bessel Series

The function `pic[n]` gives a graph of the function $f[r]$ and of the n th partial sum of the Fourier-Bessel series. To get a plot of the function f only, use `picfunc`. The function f is in blue, and the Fourier-Bessel series is in red. The range that is plotted is user-specified. The number of points plotted in each graph is specified by the variable `npoints`. The default, set below, is 500. If your computation times seem excessive, you can make this number smaller. The graph size is set by the `SetOptions` command below. The value chosen for `ImageSize`, namely 250, is appropriate for printed graphs. For computer display a value of 350 would be better.

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

SetOptions[ListPlot, ImageSize -> 250];

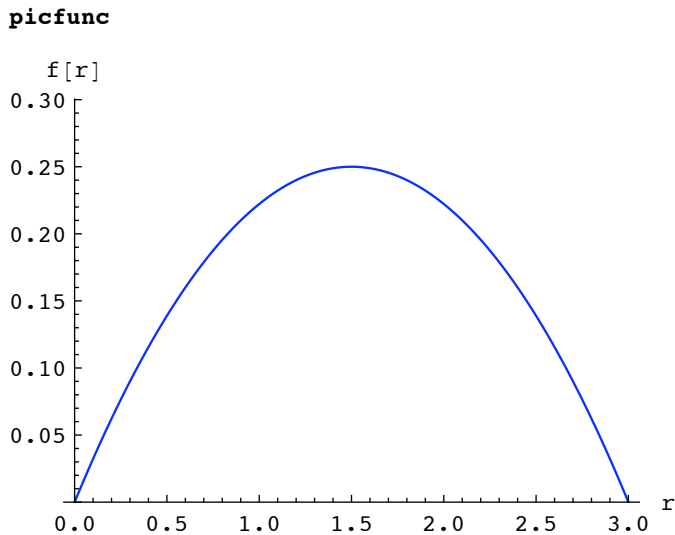
pltrange = {0.0,0.301};

npoints = 500;

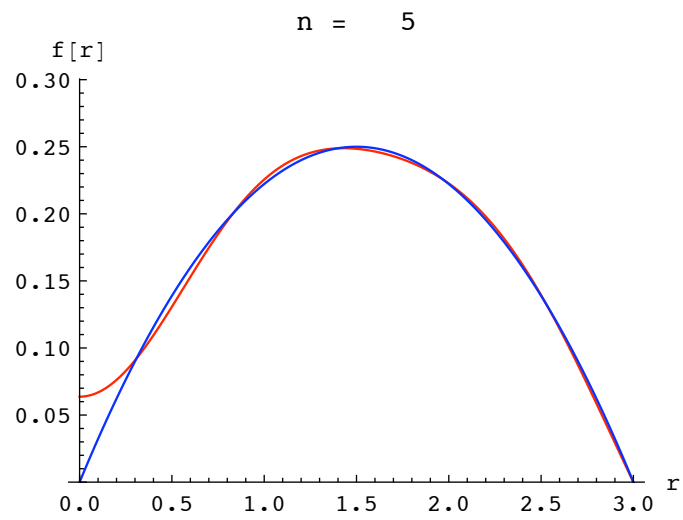
picfunc := Plot[f[r],{r,0,a},PlotStyle ->
  {RGBColor[0,0,1],Thickness[0.004]},PlotPoints -> npoints,
  PlotRange -> pltrange,
  AxesLabel -> {"r","f[r]"},AspectRatio -> 0.7]

pic[n_] := Plot[{foursum[r,n],f[r]},{r,0,a},
  PlotStyle ->
  {{RGBColor[1,0,0],Thickness[0.004]},{RGBColor[0,0,1],Thickness[0.004]}},
  PlotPoints -> npoints,PlotRange -> pltrange,
  AxesLabel -> {"r","f[r]"},AspectRatio -> 0.7,
  PlotLabel -> Row[{"n = ",PaddedForm[n,2]}]]
```

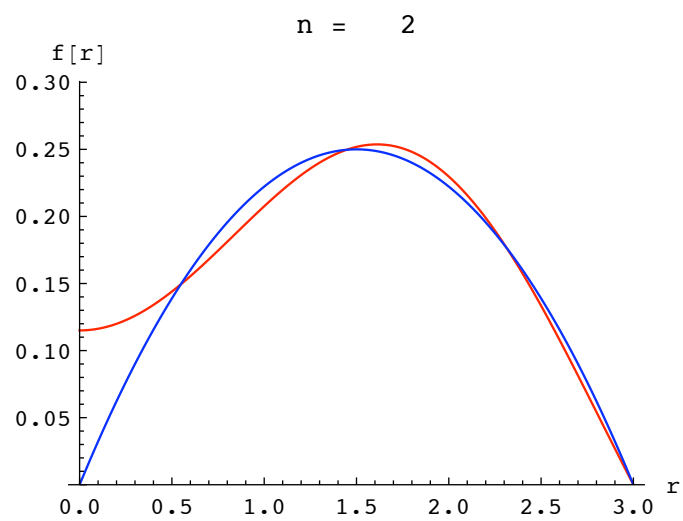
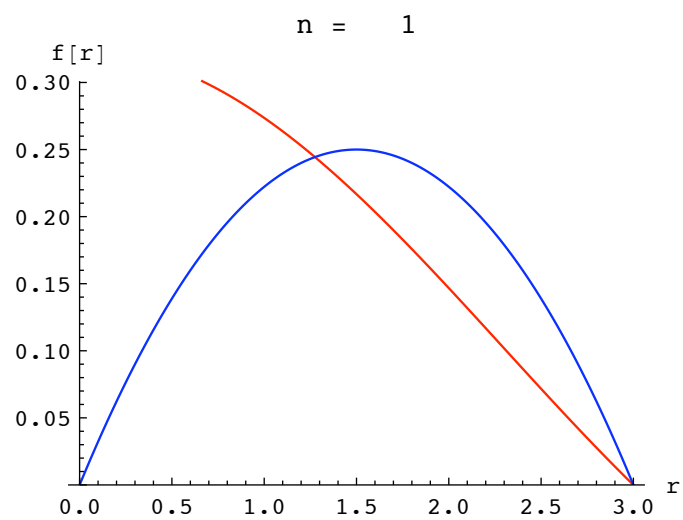
Let's try this out. We produce first a graph of the basic function, then a graph of the partial sums for $n = 5$, and then a sequence of graphs of partial sums up to and including $n = 5$. We produce the sequence by a `Do` loop.

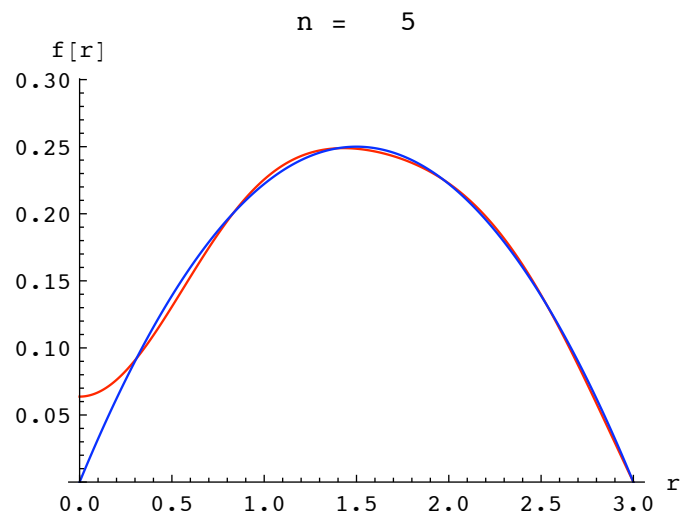
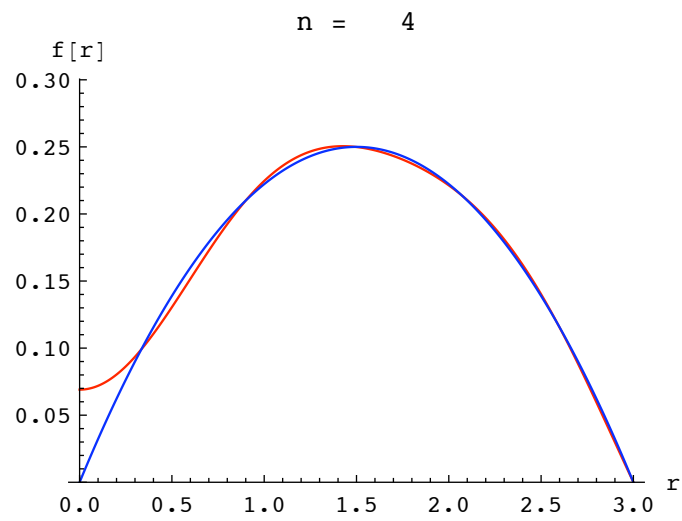
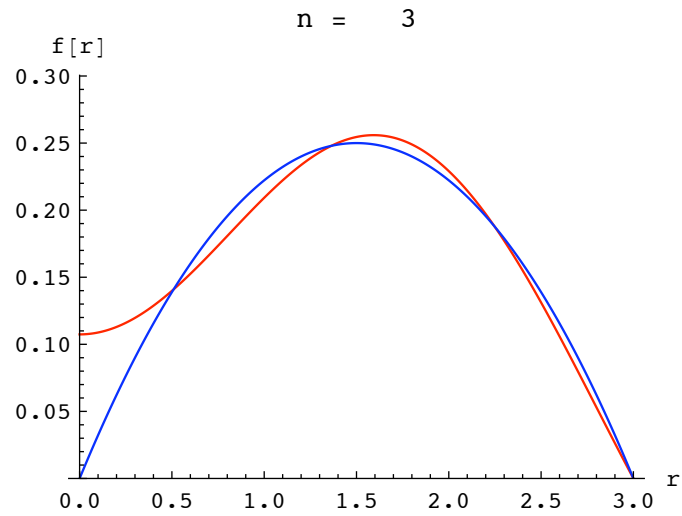


```
pic[5]
```



```
Do[Print[pic[n]],{n,1,5}];
```





If you select and animate this last sequence of graphs, you will get a nice dynamic view of the convergence process. You will also see that more terms are needed to get a result closely resembling the original $f[r]$. 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.

■ 7. 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. However, we must then change our graphing technique, 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], 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.

```

mksumlist[n_] := Module[{ans,r,inc,j},
  ans = {{0,foursum[0,n]}};
  inc = (a)/npoints;
  Do[r = j*inc;
  ans = Append[ans,{r,foursum[r,n]}],{j,1,npoints}];
  ans]

mktermlist[n_] := Module[{ans,r,inc,j},
  ans = {{0.0,fourterm[0,n]}};
  inc = a/npoints;
  Do[r = j*inc;
  ans = Append[ans,{0.0,fourterm[r,n]}],{j,1,npoints}];
  ans]

funclist := Module[{ans,r,inc,j},
  ans = {{0,f[0]}};
  inc = a/npoints;
  Do[r = j*inc;
  ans = Append[ans,{r,f[r]}],{j,1,npoints}];
  ans]

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

```

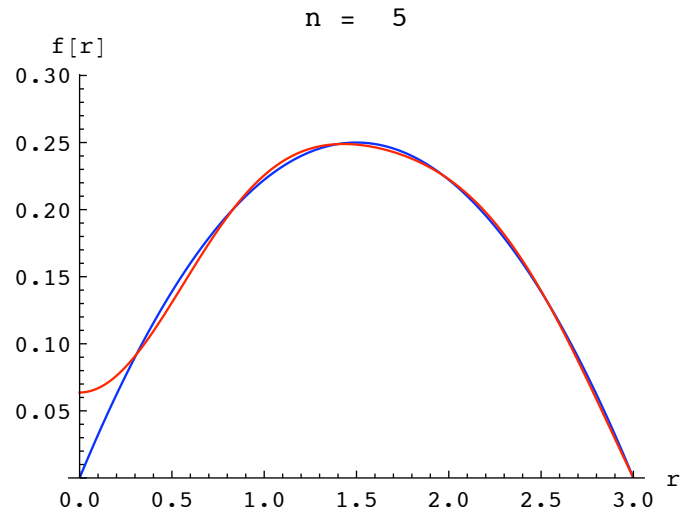
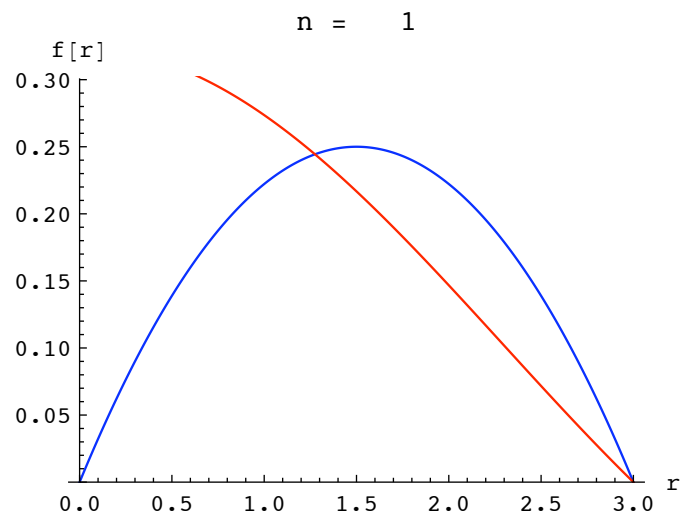
```

picarray[first_,last_,grinc_] := Module[
  {sumlist,termlist,k,grph,grph0},
  sumlist = mksumlist[first];
  grph0 = mkgraph[funclist,0,0,1];
  grph = mkgraph[sumlist,1,0,0];
  Print[Show[{grph0,grph},AxesLabel -> {"r","f[r]"},
  AspectRatio -> 0.7,
  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 -> {"r","f[r]"},
      AspectRatio -> 0.7,
      PlotLabel ->
      Row[{"n = ",PaddedForm[k,2}]]]]),
    {k,first+1,last}]]

```

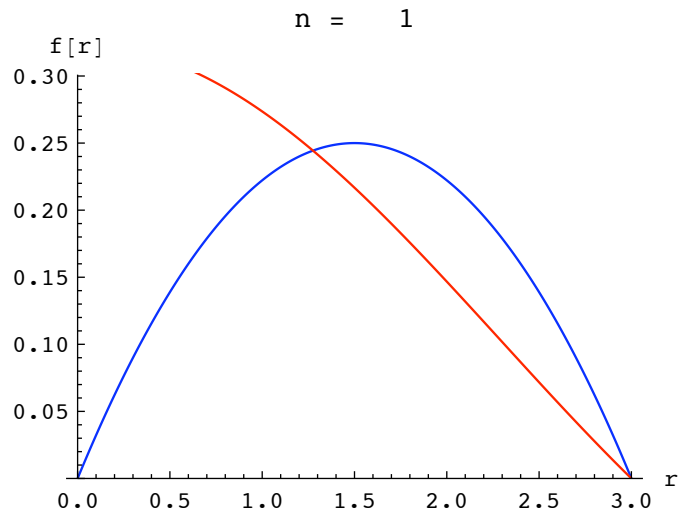
As an example, we execute `picarray[1,5,4]`. This will start with $n = 1$ and then display every 4th graph up to $n = 5$ -- i.e., graphs 1 and 5.

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



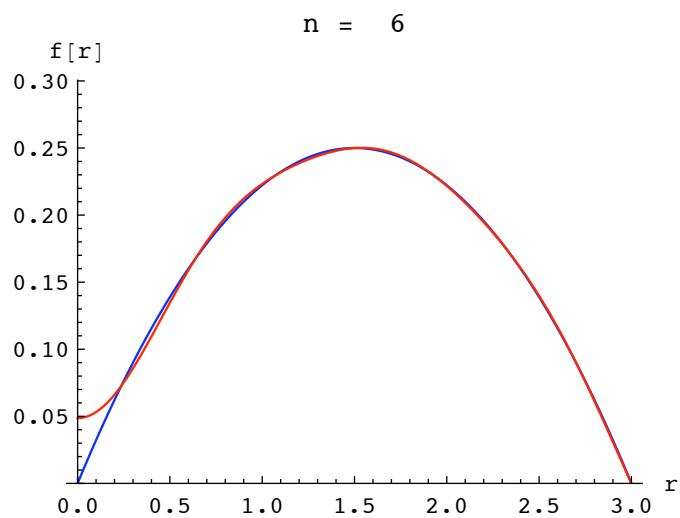
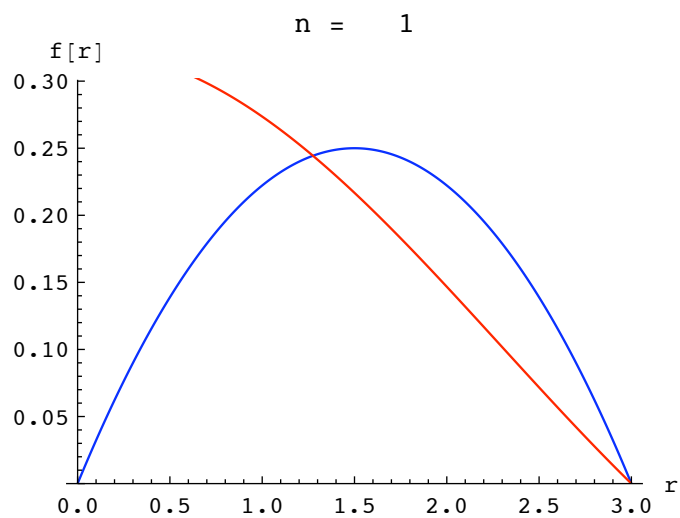
Now we use this new faster method to display the partial sums for our function up to and including $n = 26$. We display every graph in the sequence by setting `grinc = 1`. In the printed version of the notebook, the graphs of the sequence are combined into a single cell, so only the first graph is visible. To animate the graph sequence, select the cell containing the graphs and use the menu item **Graphics->Rendering->Animate Selected Graphics**.

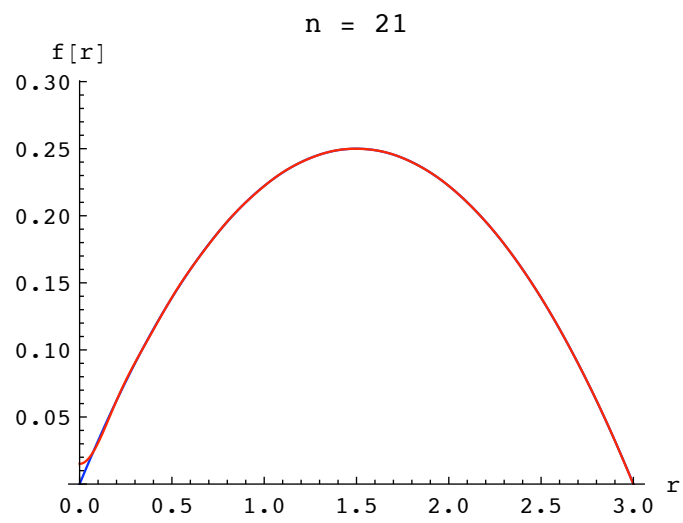
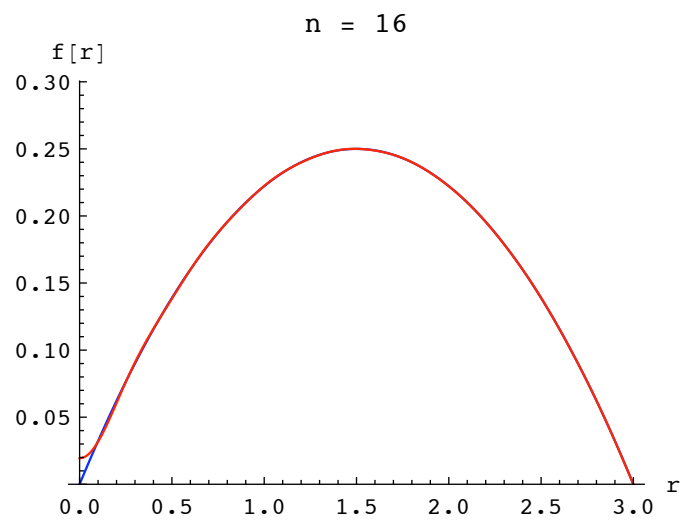
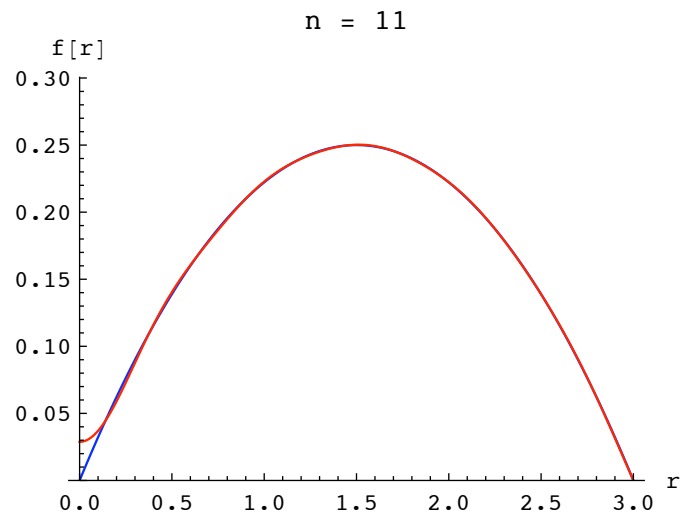
```
picarray[1,26,1];
```

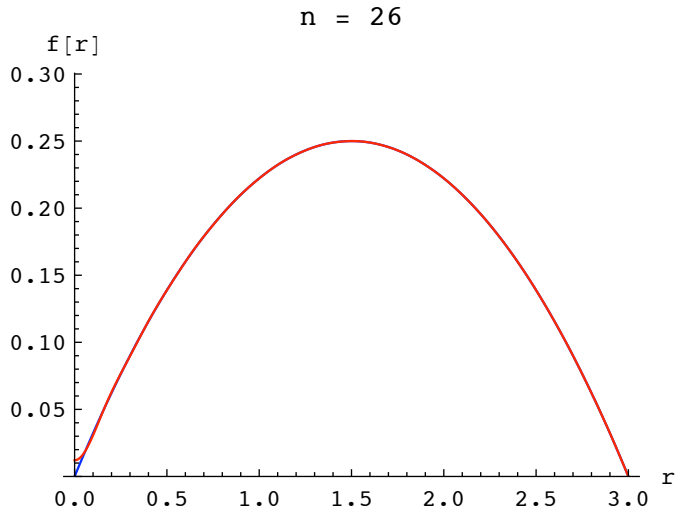


For visualization in the printed version of this notebook, we show every 5th graph in the sequence.

```
picarray[1, 26, 5];
```







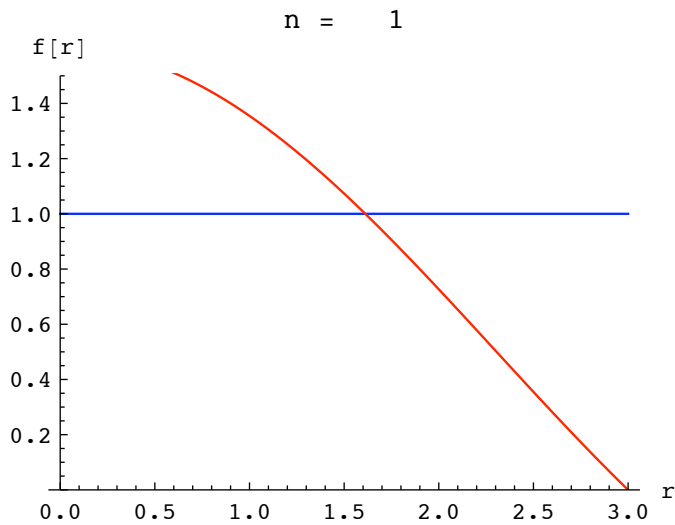
■ 8. Application to $f(r) = 1$.

We now apply all of this to a different function, namely $f(r) = 1$. We must change the definition of f and recalculate the Fourier-Bessel coefficients.

```
f[r_] := 1
a = 3.0;
coeff = Module[{ans}, ans = {}; Do[ans = Append[ans, fbc[n]], {n, 1, nmax}]; ans];
pltrange = {0, 1.5};
```

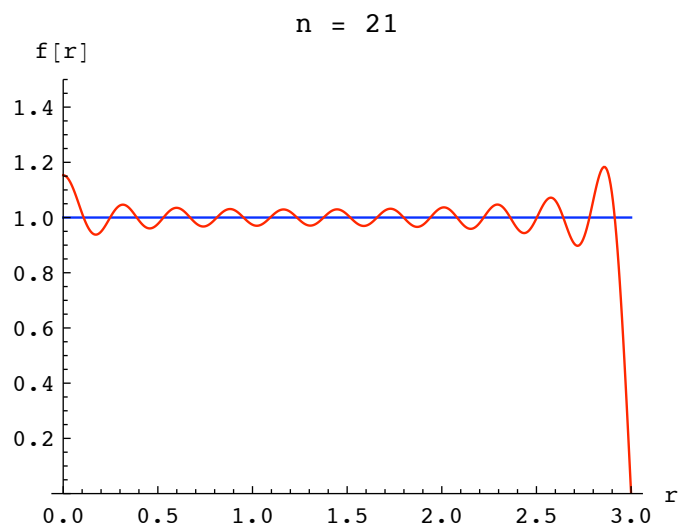
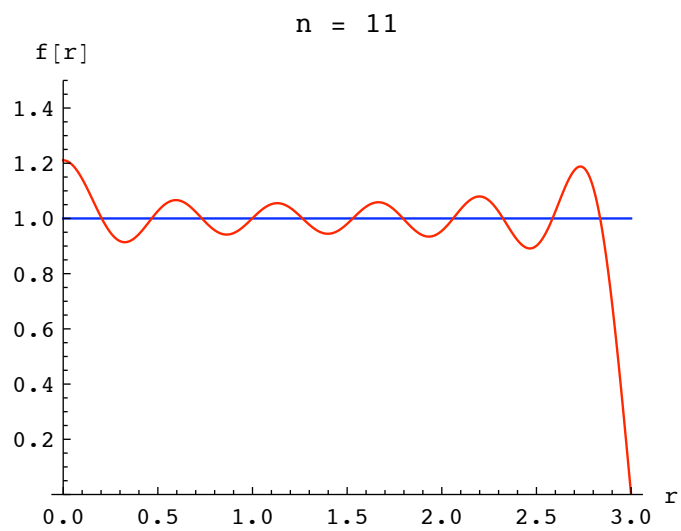
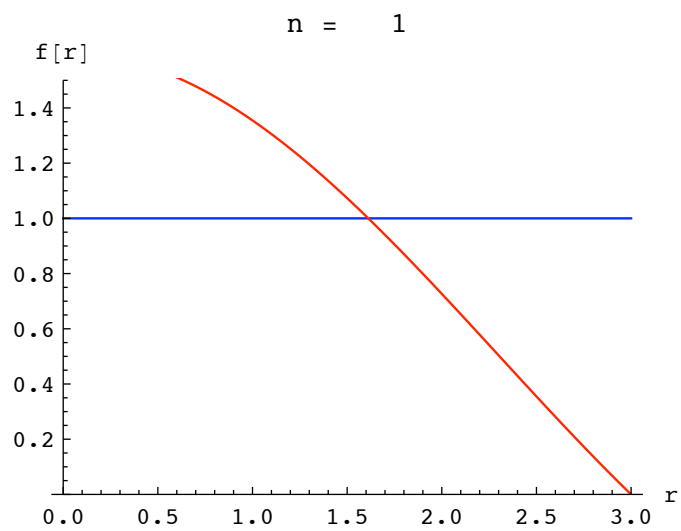
We now construct a graph sequence showing the first 51 partial sums of the series.

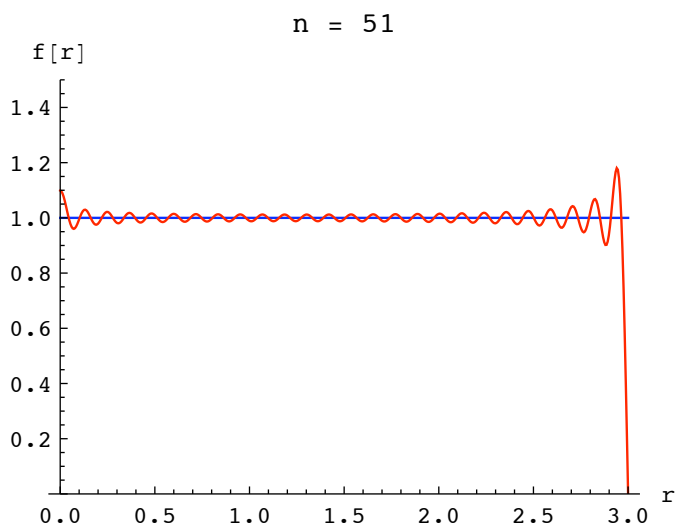
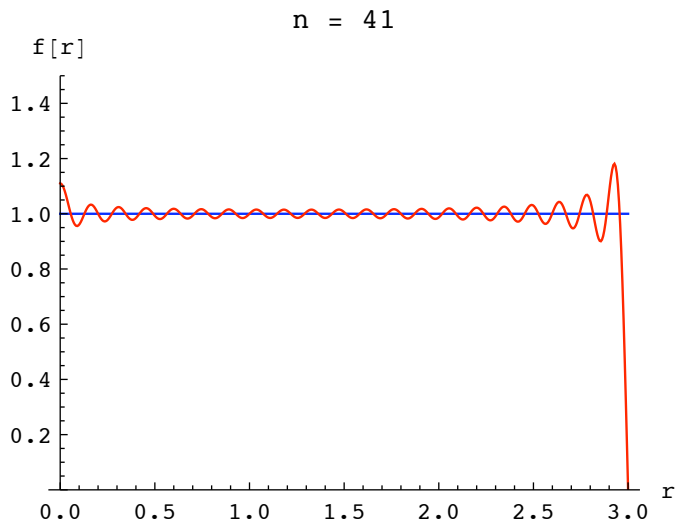
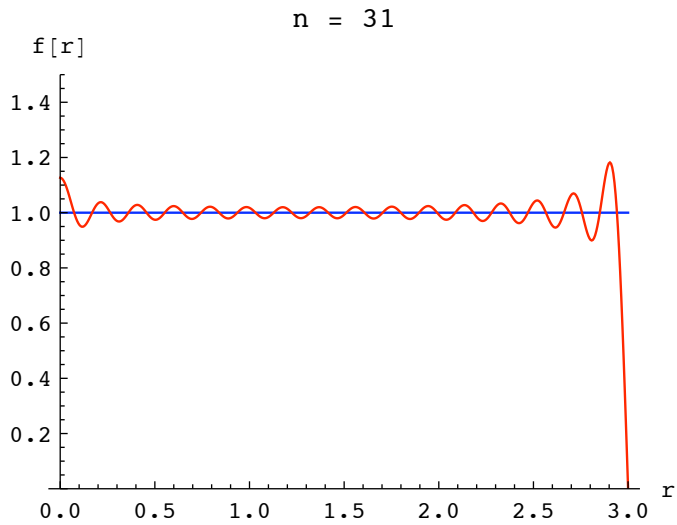
```
picarray[1, 51, 1];
```



For visualization in the printed form of the notebook, we show every 10 th graph.

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





Animating the graph sequence gives a dynamic picture of the convergence. We see the Gibbs overshoot at the right endpoint, and some serious oscillations near $r = 0$. Because the Bessel functions for large argument (hence large n) are

asymptotic to trig functions with a decaying amplitude, the mathematics of the Gibbs overshoot is qualitatively the same here as for the simpler Fourier series we studied earlier.

■ 9. Using J_1 's to expand $f(r) = 1$.

We will now use the eigenfunctions $J_1(\alpha_i r/a)$ where α_i is the i th root of J_1 . We must redefine the eigenfunctions and the zeros for *Mathematica*.

```
beszer = N[Table[BesselJZero[1, i], {i, 1, nmax}]]

{3.83171, 7.01559, 10.1735, 13.3237, 16.4706, 19.6159, 22.7601, 25.9037, 29.0468,
 32.1897, 35.3323, 38.4748, 41.6171, 44.7593, 47.9015, 51.0435, 54.1856, 57.3275,
 60.4695, 63.6114, 66.7532, 69.8951, 73.0369, 76.1787, 79.3205, 82.4623,
 85.604, 88.7458, 91.8875, 95.0292, 98.171, 101.313, 104.454, 107.596, 110.738,
 113.879, 117.021, 120.163, 123.304, 126.446, 129.588, 132.729, 135.871,
 139.013, 142.154, 145.296, 148.438, 151.579, 154.721, 157.863, 161.004}

eiger = beszer / a;

Ψ[r_, n_] := BesselJ[1, r * eiger[[n]]]
```

Now we define the normalization integral and calculate the first n_{\max} values. These can also be found analytically.

```
norm[n_] := NIntegrate[r * (Ψ[r, n])^2, {r, 0, a}]

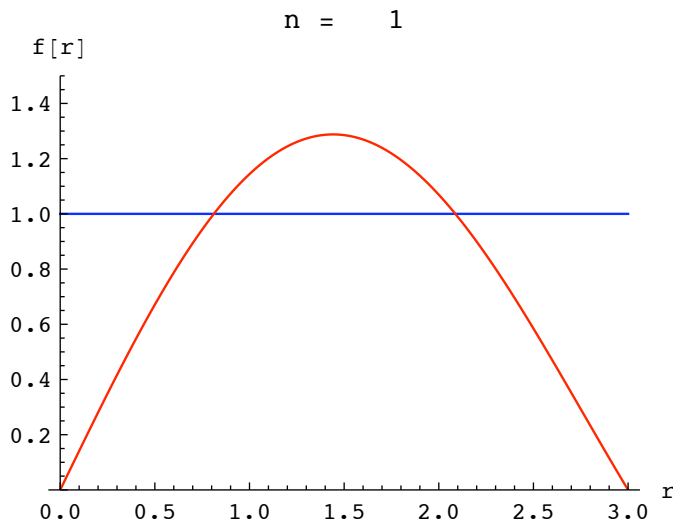
normlist = Module[{ans}, ans = {}; Do[ans = Append[ans, norm[n]], {n, 1, nmax}]; ans];
```

Next we calculate the Fourier-Bessel coefficients, as before using *NIntegrate*. The coefficients are stored in the list *coeff*.

```
coeff = Module[{ans}, ans = {}; Do[ans = Append[ans, fbc[n]], {n, 1, nmax}]; ans];
```

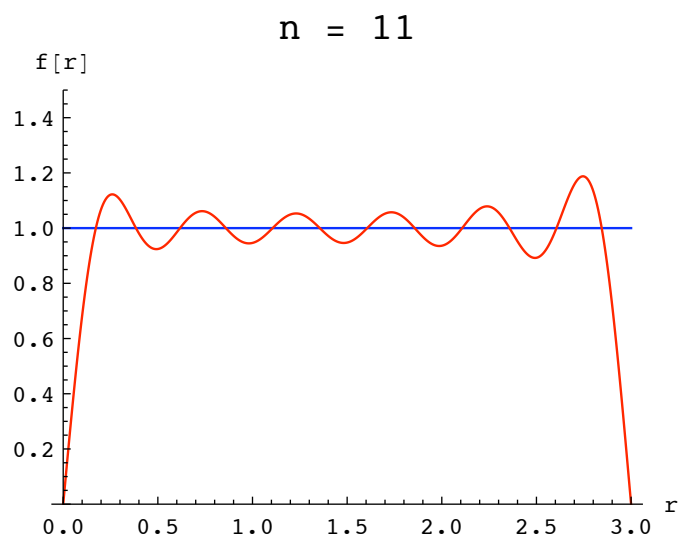
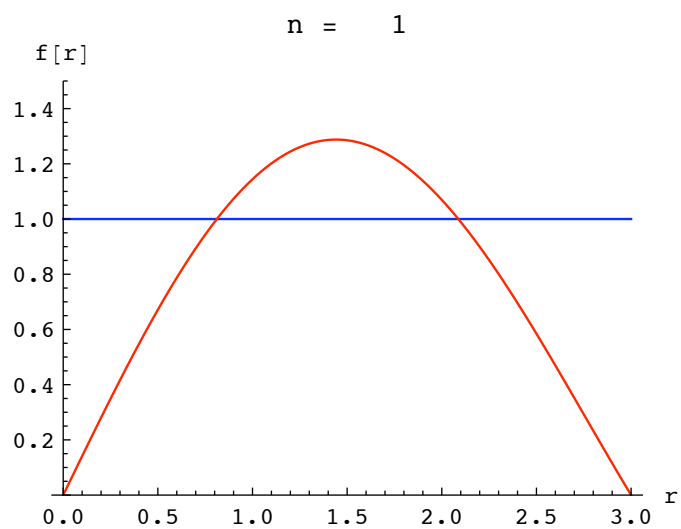
Now we construct a sequence of graphs showing the first 50 partial sums.

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

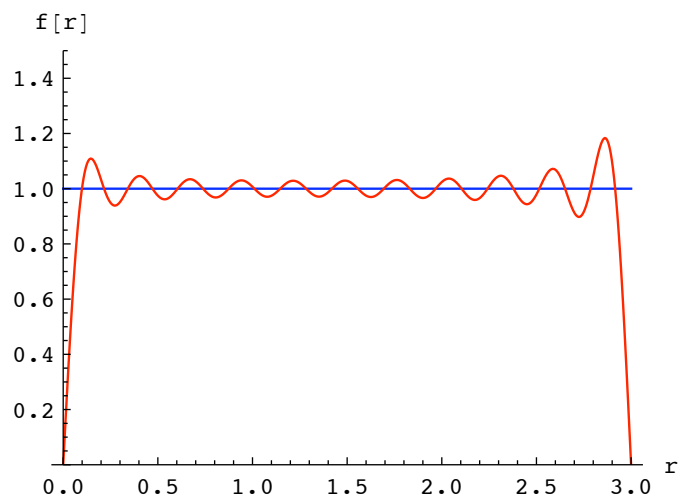


For visualization in the printed form of the notebook, we show every 10th graph.

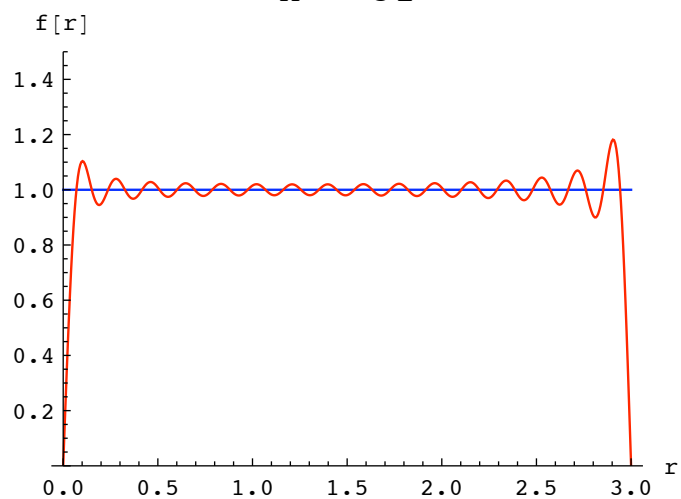
```
picarray[1, 51, 10];
```



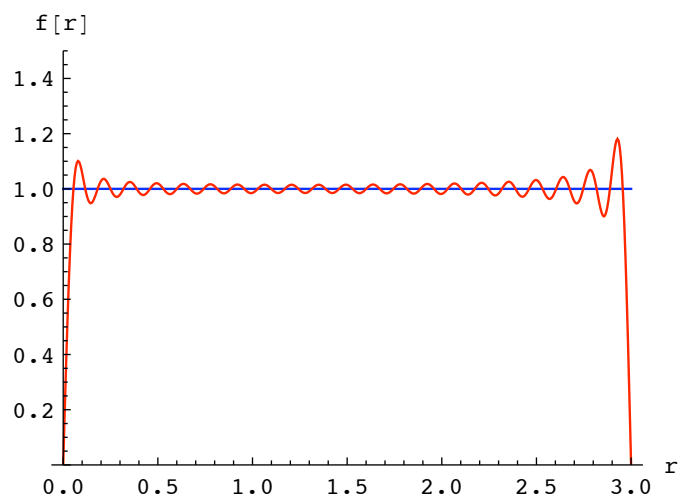
n = 21

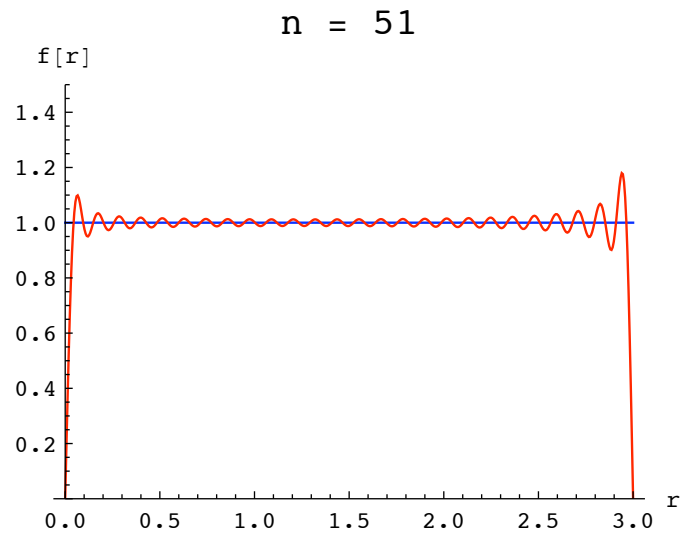


n = 31



n = 41





Again we see the Gibbs overshoot at the right endpoint. There are oscillations at $r = 0$ but not as pronounced as at the right endpoint.