

ME 201/MTH 281/ME400/CHE400

Convergence of Legendre Expansions

Mathematica 7

■ 1. Introduction

This notebook provides a general framework for the construction and visualization of partial sums of series in Legendre polynomials. To use it for a specific function, you must define the function and the expansion coefficients 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 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 . The function introduced in Section 2 to illustrate the technique is $f(\eta) = -1 + 2H(\eta)$, where H is the unit step function. In section 6, a second example is considered, namely $f(\eta) = (1 + \eta)/(1 + \eta^2)$. For this example, the expansion coefficients are calculated by numerical integration.

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

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

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

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

■ 2. Definition of Function and Expansion Coefficients

We now apply all of this to a particular function, namely $f(\eta) = -1 + 2H(\eta)$.

$$\mathbf{f}[\eta_] := \mathbf{If}[\eta < 0, -1, 1]$$

The plot range in this notebook is user-specified, and is assigned to the variable `pltrange`. We choose a value appropriate for this function.

```
pltrange = {-1.5, 1.5};
```

This function is discontinuous and so we expect that many terms will be required in the expansion. We set the maximum polynomial order at 51:

```
nmax = 51;
```

The coefficients for this function were derived analytically in the notebook entitled Legendre Polynomials. We take the analysis from that notebook to define an array coeff that contains all of the coefficients for polynomials from order 0 to order nmax = 51.

```
coeff = Module[{i, Co}, Co = {0}; Do[Co = Append[Co,
  (LegendreP[i - 1, 0] - LegendreP[i + 1, 0])], {i, 1, nmax}]; Co];
```

■ 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 $\Psi[\eta, n]$:

```
 $\Psi[\eta_, n_] := LegendreP[n, \eta]$ 
```

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

```
fourterm[\eta_, n_] :=
  N[coeff[[n+1]]*\Psi[\eta, n]]
```

Note the shift in index. The element n+1 of coeff is the coefficient of P_n in the expansion.

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

```
foursum[\eta_, n_] := Sum[fourterm[\eta, k], {k, 0, n}]
```

■ 4. Graphs of $f[\eta]$ and Partial Sums of the Legendre Series

The function pic[n] gives a graph of the function $f[\eta]$ and of the nth partial sum of the series. To get a plot of the function f only, use picfunc. 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 npoints. The default, set below, is 500. If your computation times seem excessive, you can make this number smaller. The size of the graphs in this notebook is controlled by the variable imsize below. The value 200 is appropriate for a printed version, and the value 350 for computer display.

```
imsize = 250;
```

```
npoints = 500;
```

```
picfunc := Plot[f[\eta], {\eta, -1, 1}, PlotStyle ->
  {RGBColor[0, 0, 1], Thickness[0.004]}, ImageSize->imsize,
  PlotPoints -> npoints, PlotRange -> pltrange,
  AxesLabel -> {"\eta", "f[\eta]"}, AspectRatio -> 0.7]
```

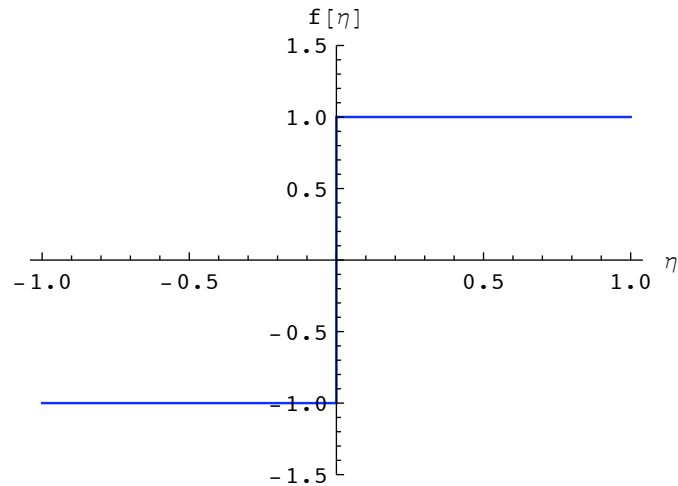
```

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

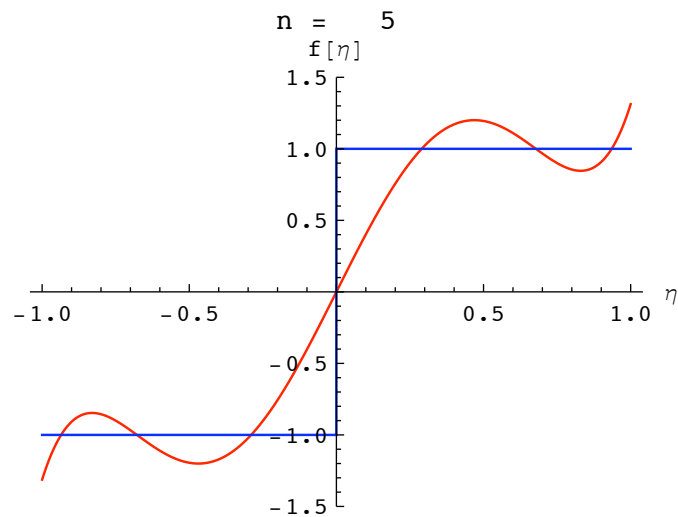
```

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.

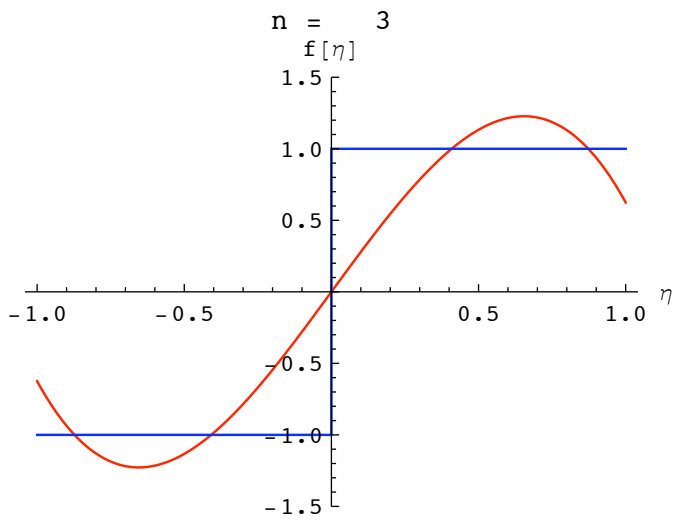
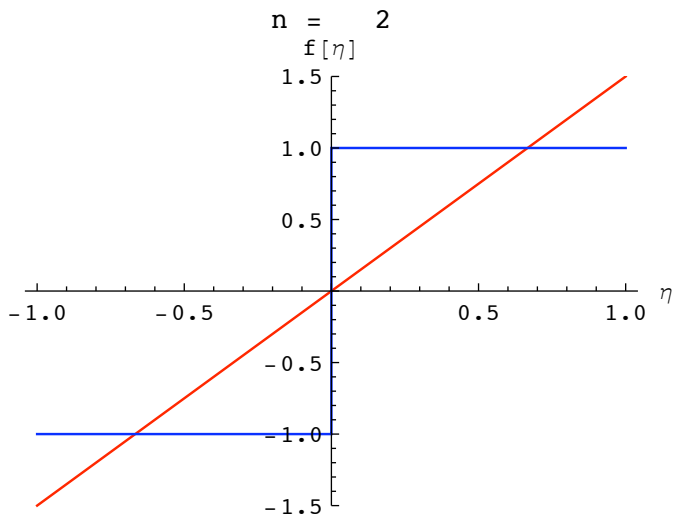
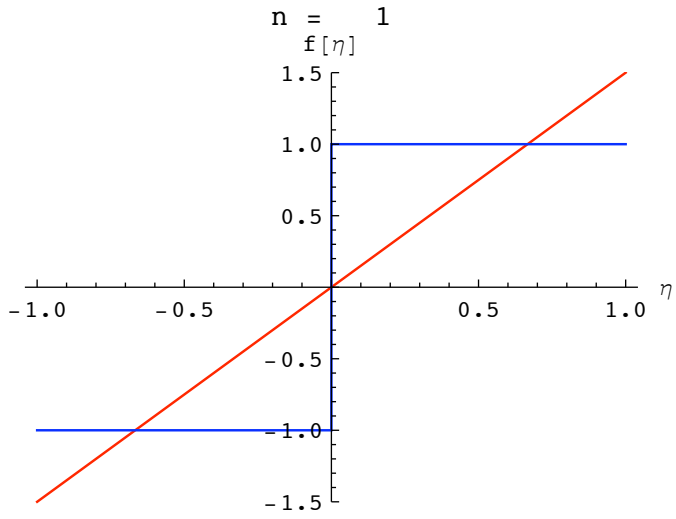
picfunc

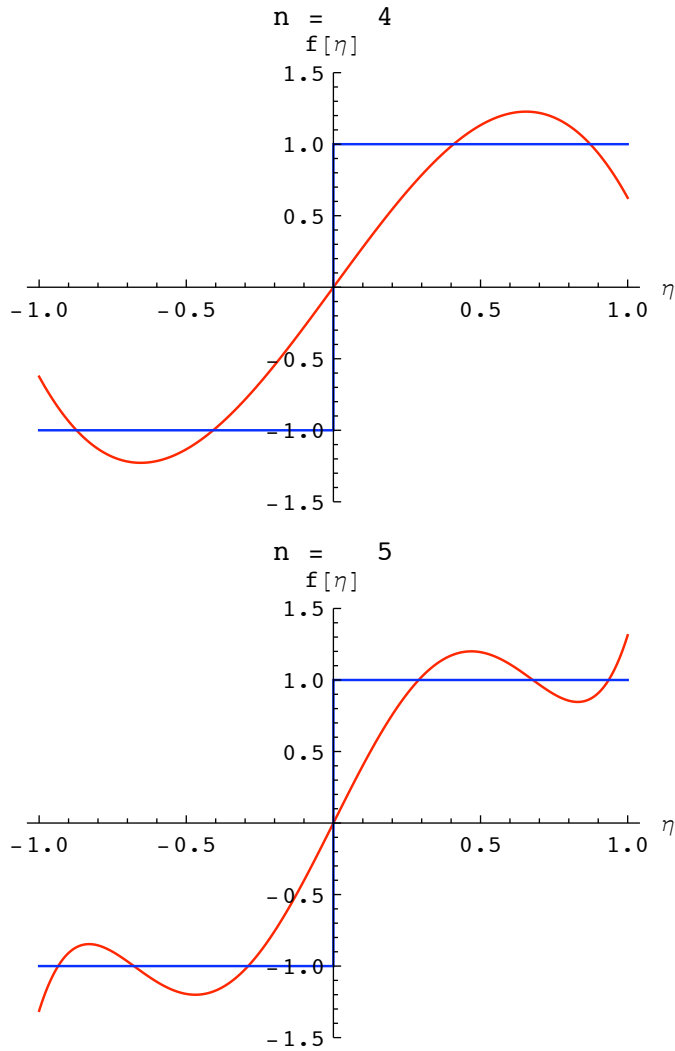


pic[5]



```
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 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. Another minor inefficiency in this case is that successive graphs are identical. Because the original function is odd, the even expansion coefficients are zero. An efficient technique would (1) avoid the computation of the zero terms and (2) 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 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 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.

```

mksumlist[n_] := Module[{ans, η, inc, j},
  ans = {{-1., foursum[-1., n]}};
  inc = 2./npoints;
  Do[η = -1+j*inc;
  ans = Append[ans, {η, foursum[η, n]}], {j, 1, npoints}];
  ans]

mktermlist[n_] := Module[{ans, η, inc, j},
  ans = {{0.0, fourterm[-1., n]}};
  inc = 2./npoints;
  Do[η = -1+ j*inc;
  ans = Append[ans, {0.0, fourterm[η, n]}], {j, 1, npoints}];
  ans]

funclist := Module[{ans, η, inc, j},
  ans = {{-1, f[-1]}};
  inc = 2./npoints;
  Do[η = -1+j*inc;
  ans = Append[ans, {η, f[η]}], {j, 1, npoints}];
  ans]

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

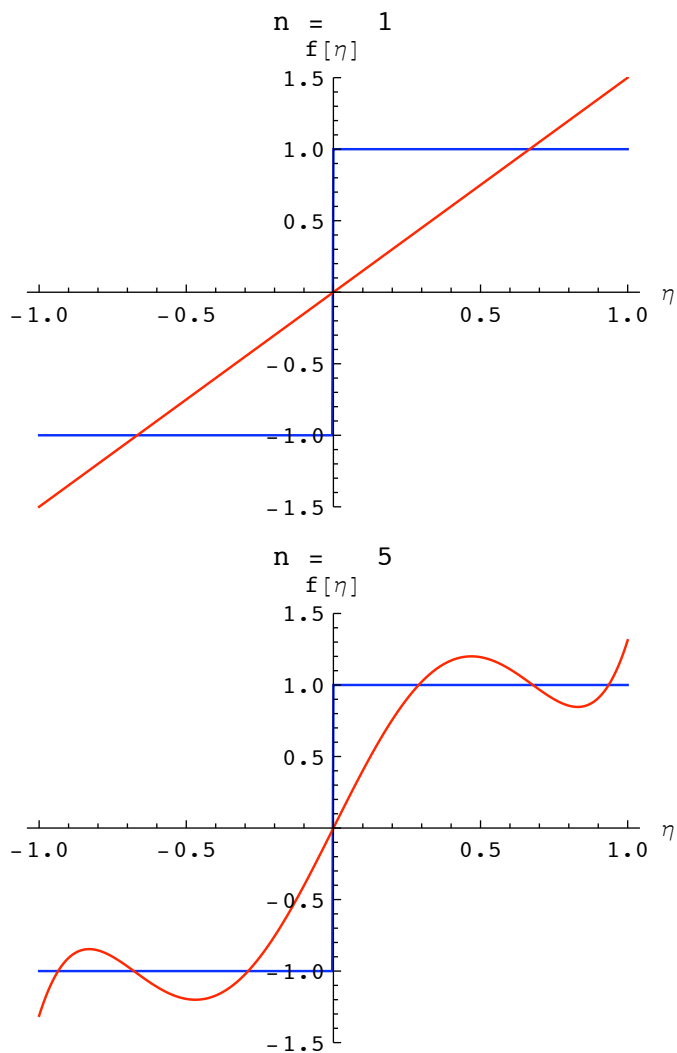
picarray[first_, last_, grinc_] := Module[
  {sumlist, termlist, k, grph, grph0},
  sumlist = mksumlist[first];
  grph0 = mkgraph[funclist, 0, 0, 1];
  grph = mkgraph[sumlist, 1, 0, 0];
  Print[Show[{grph0, grph}, AxesLabel -> {"η", "f[η]"},
  AspectRatio -> 0.7,
  PlotLabel -> Row[{"n = ", PaddedForm[first,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

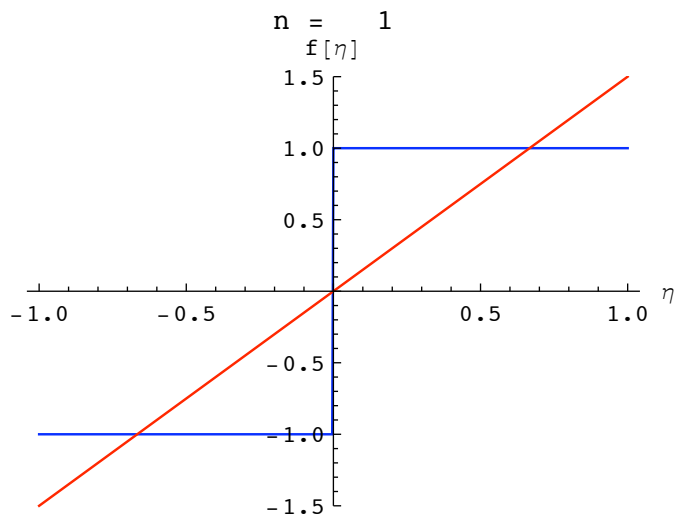
graphs 1 and 5.

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



Now we use this new faster method to display the partial sums for our function up to and including $n = 51$. We display every second graph in the sequence by setting `grinc = 2`. The printed version of the notebook shows only the first graph in the sequence.

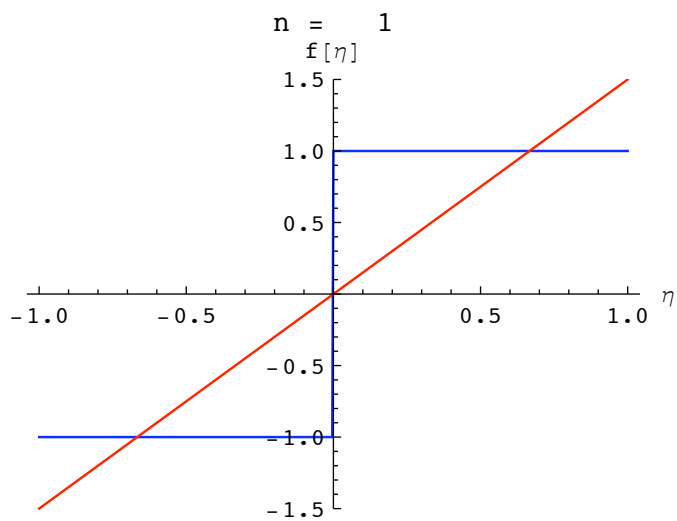
```
picarray[1,51,2];
```

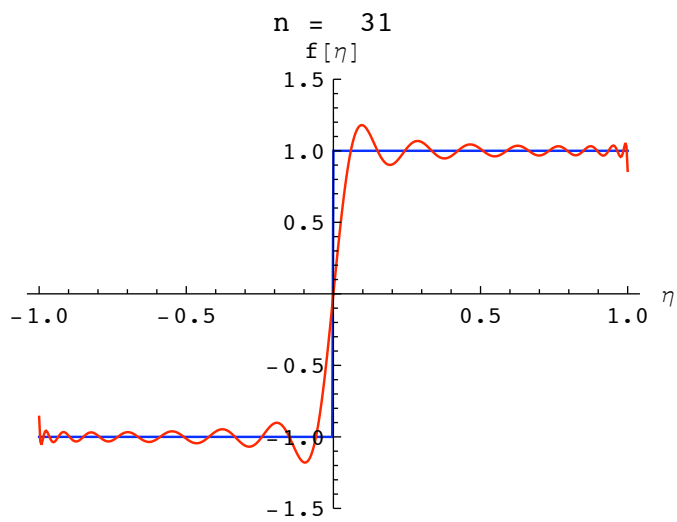
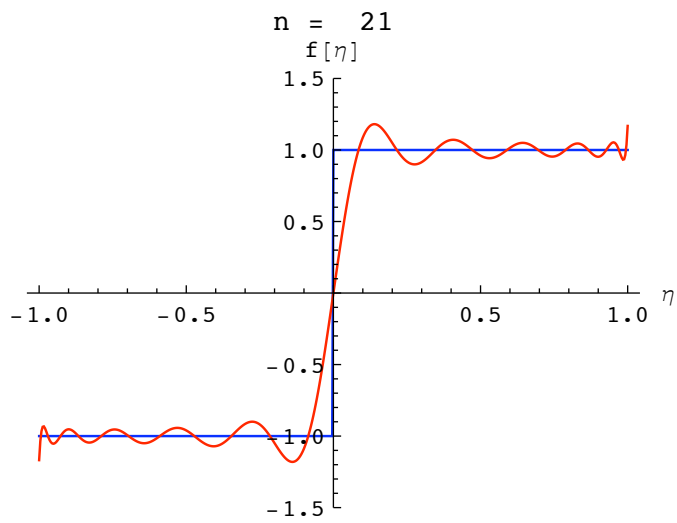
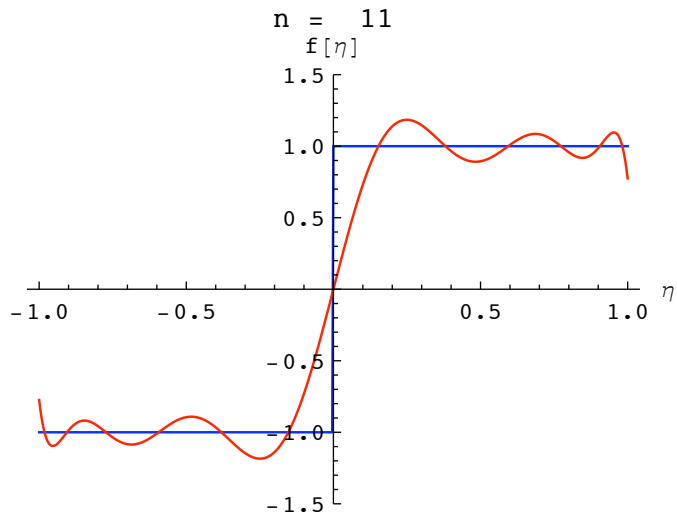


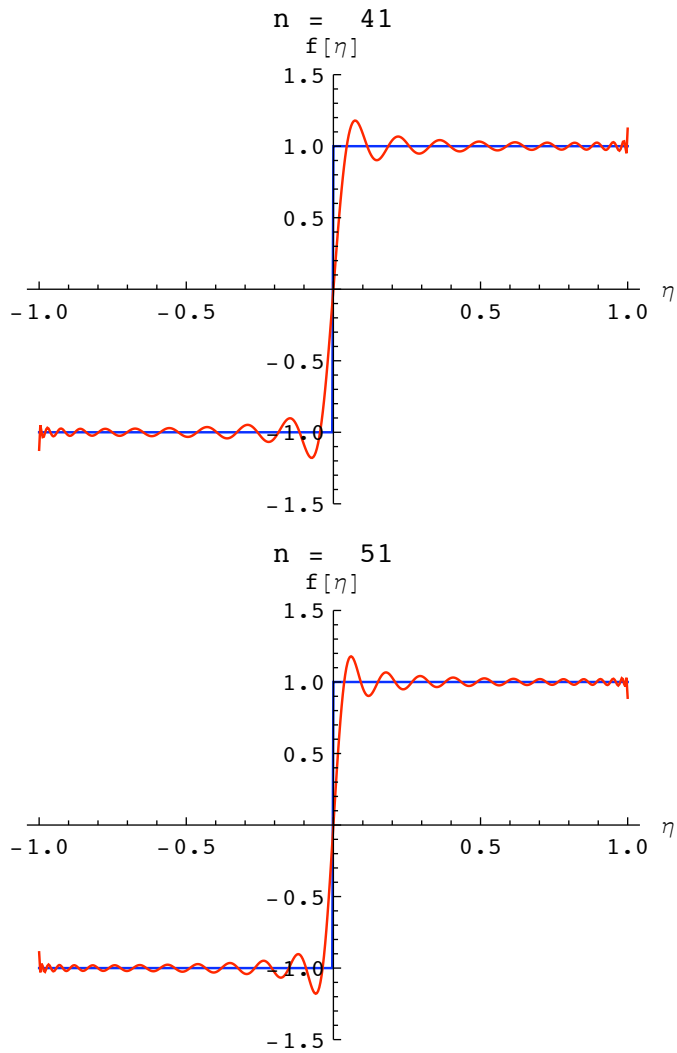
You can visualize the convergence process by selecting and animating the above group of cells. The animation is through the menu sequence **Graphics -> Rendering -> Animate Selected Graphics**.

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

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







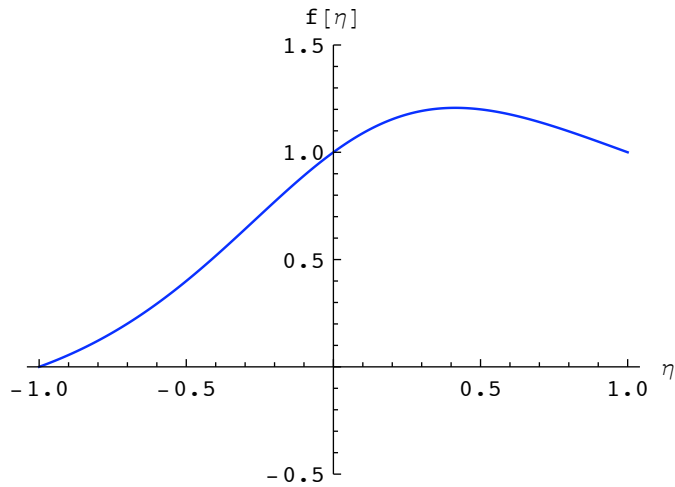
■ 6. Application to $f(\eta) = (1 + \eta)/(1 + \eta^2)$

Now we look at an example for which the expansion coefficients will have to be calculated numerically. We start by defining and plotting the function.

```
f[ $\eta_*$ ] := (1 +  $\eta$ ) / (1 +  $\eta^2$ )
```

```
pltrange = {-0.5, 1.5};
```

picfunc



We set the maximum polynomial order at 10.

```
nmax = 10;
```

The expansion coefficients will have to be defined by numerical integration. We define the nth coefficient as legc[n].

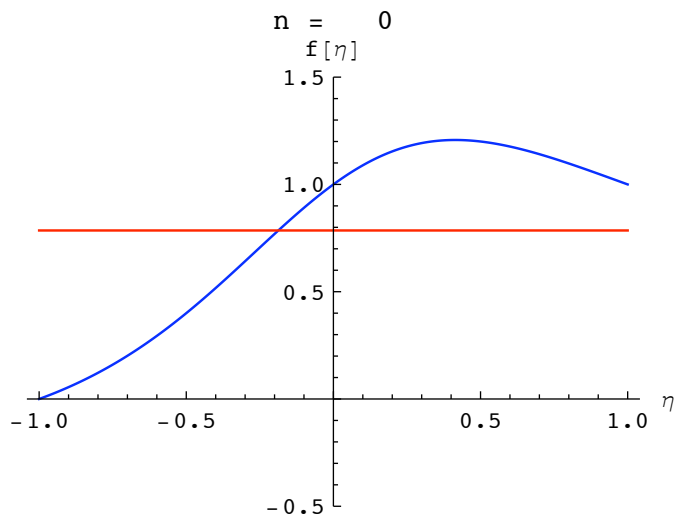
```
legc[n_] := (n + 0.5)*NIntegrate[Psi[eta,n]*f[eta], {eta, -1, 1}]
```

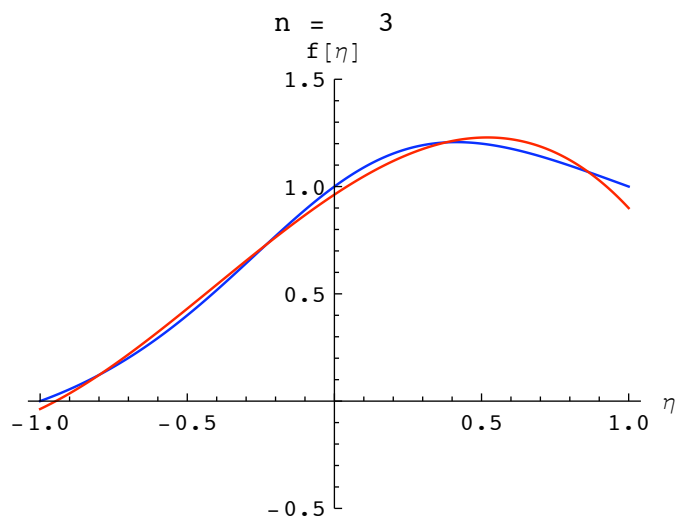
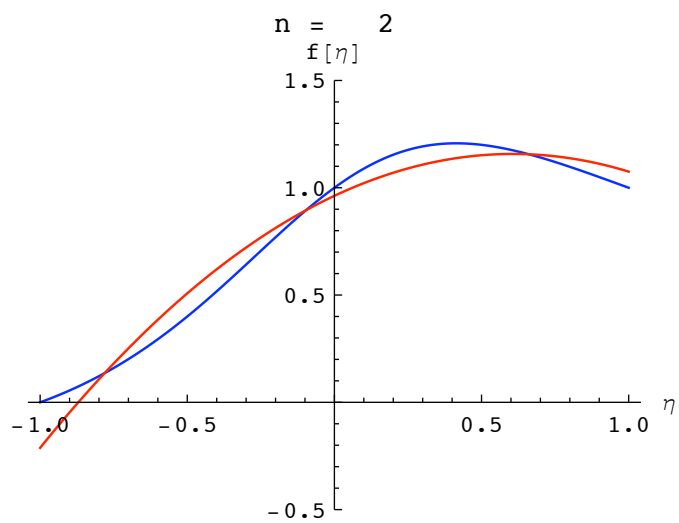
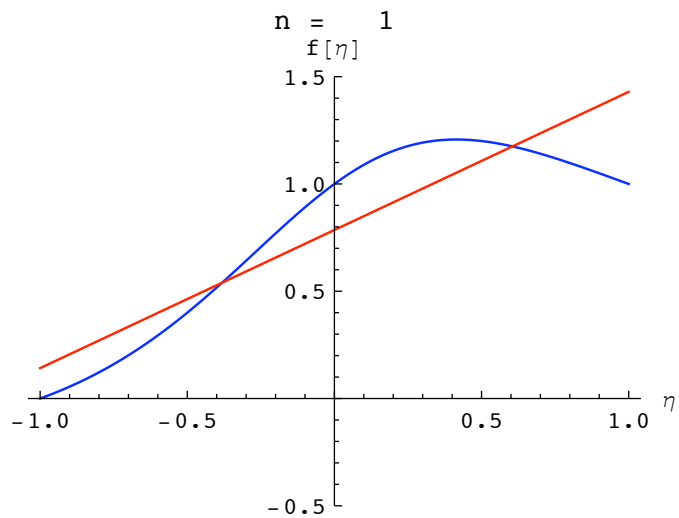
Now we create the array of coefficients.

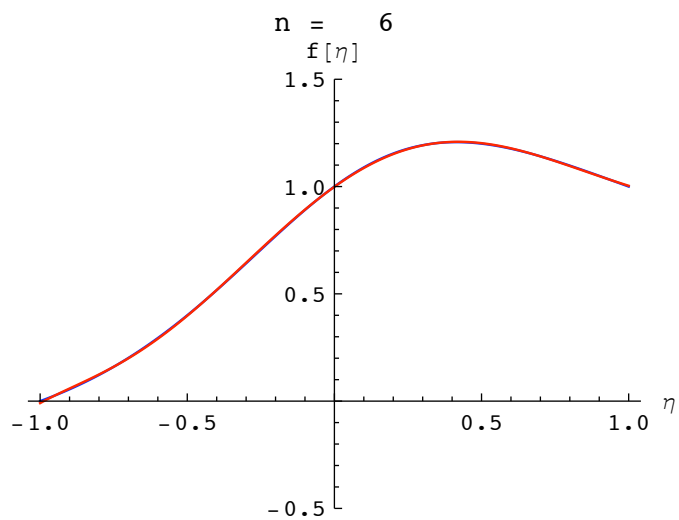
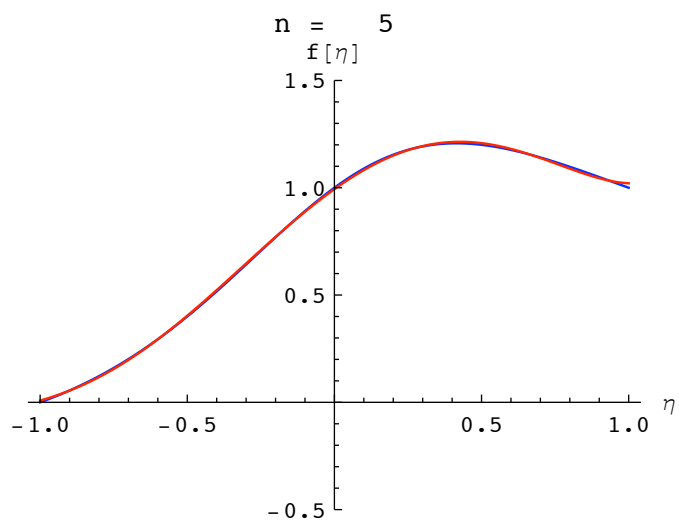
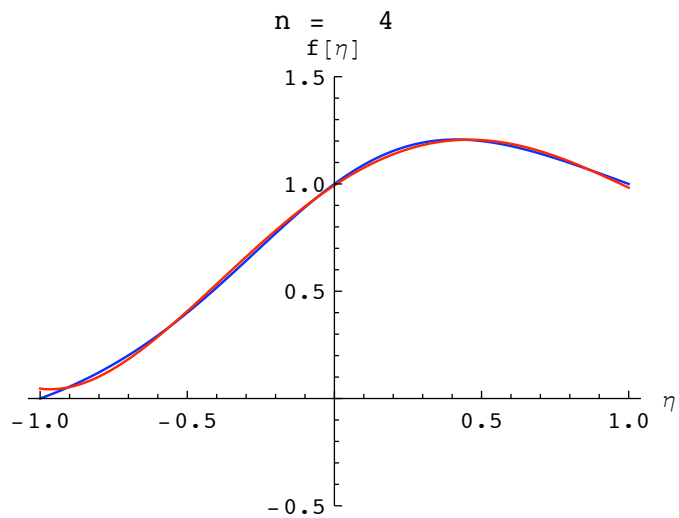
```
coeff = Module[{ans}, ans = {}; Do[ans = Append[ans, legc[n]], {n, 0, nmax}]; ans]  
{0.785398, 0.643806, -0.353982, -0.175518, 0.0829595, 0.0381404,  
-0.0172211, -0.00767737, 0.00339049, 0.00148644, -0.000647908}
```

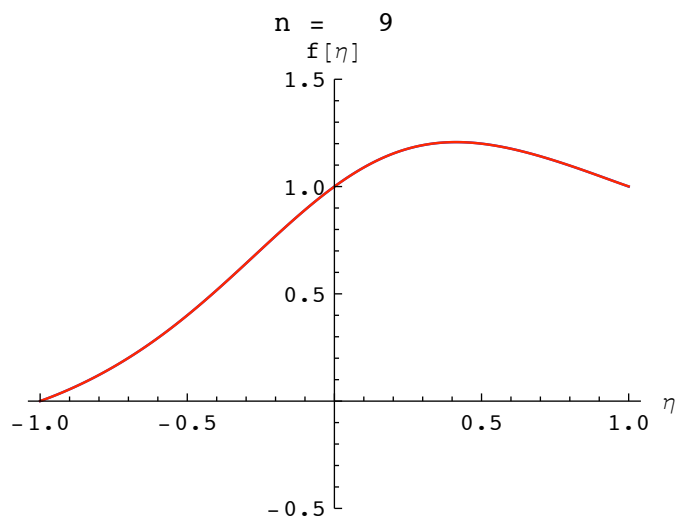
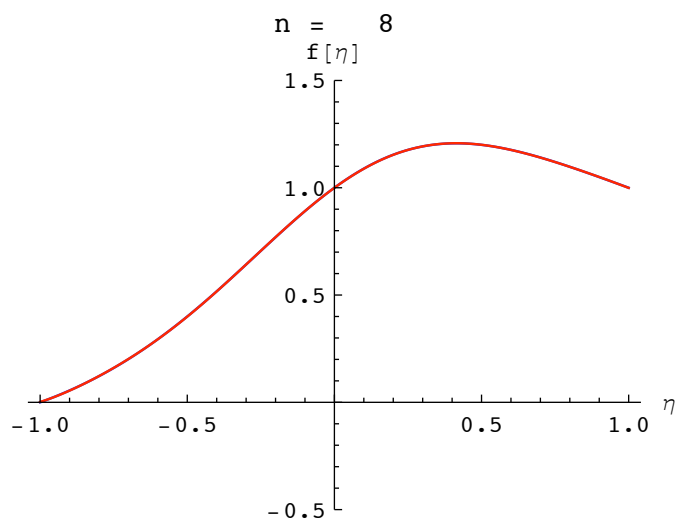
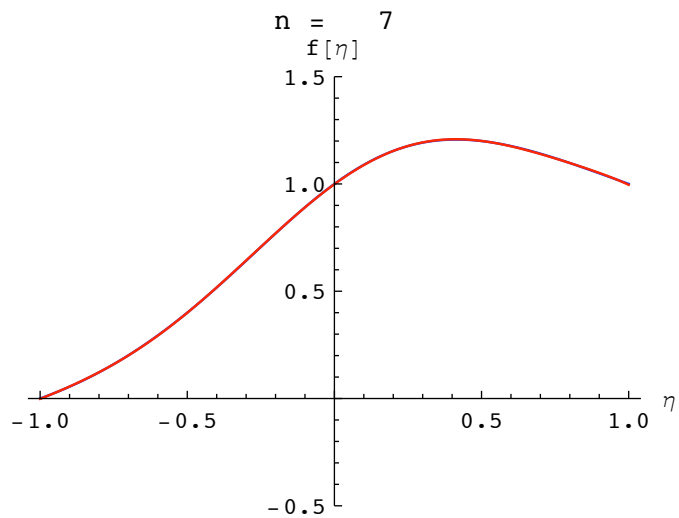
We create a graph sequence showing all the partial sums up to polynomial order 10.

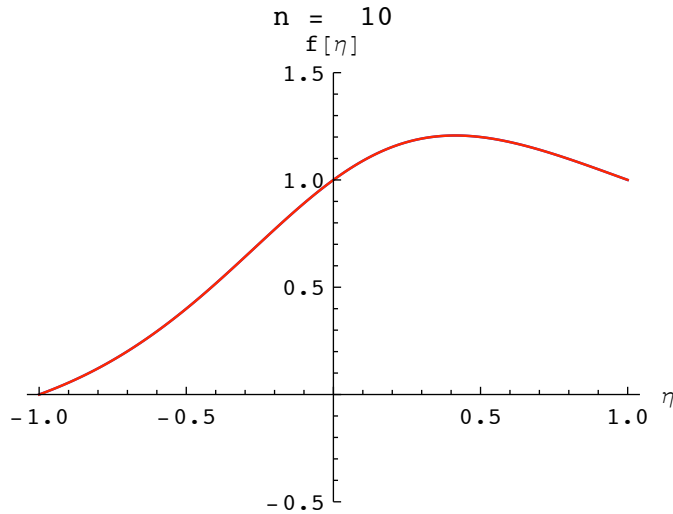
```
picarray[0, 10, 1];
```









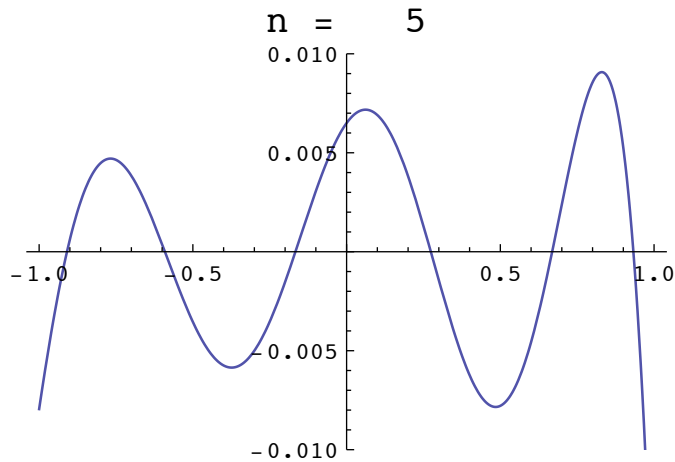


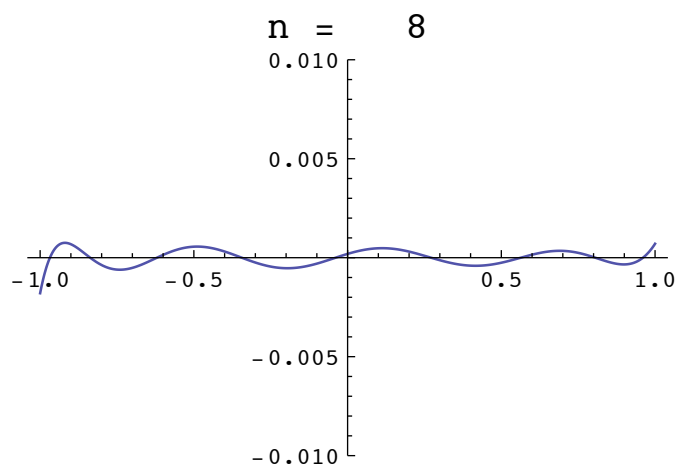
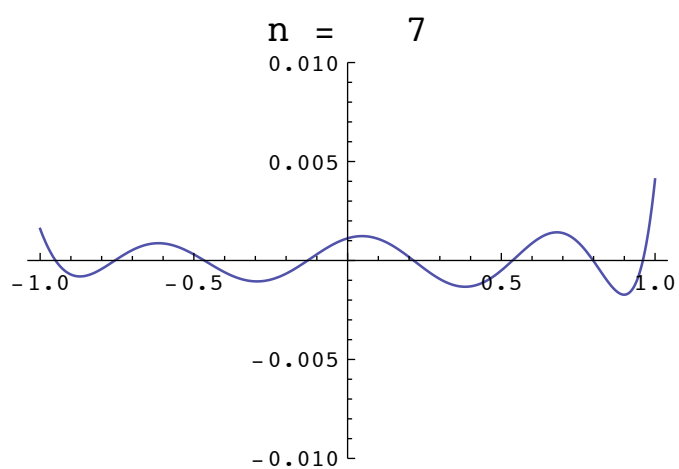
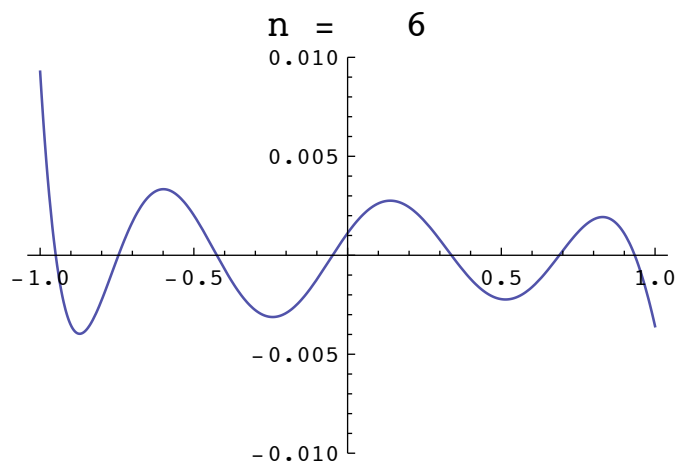
The function is smooth and we see that the convergence is excellent. You can animate the sequence to get a dynamic view of the convergence. To get a better look at the accuracy, we plot the difference between the exact solution and the representation by the series for $n = 5$ through 10. We define the difference between the original function and the n th partial sum as `diff[η,n]`.

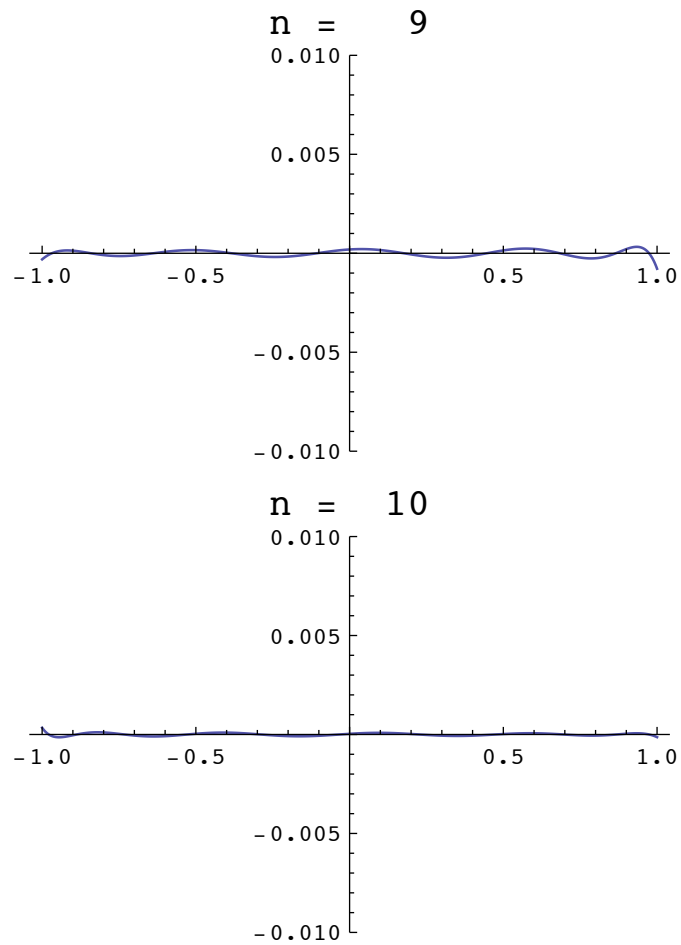
```
diff[η_, n_] := f[η] - foursum[η, n]
```

Now we plot this difference against η for $n = 5, 6, 7, 8, 9,$ and 10.

```
Do[Print[Plot[diff[η, n], {η, -1, 1}, PlotRange -> {-0.01, 0.01},  
PlotStyle -> Thickness[0.004], ImageSize -> imsize,  
PlotLabel -> Row[{"n = ", PaddedForm[n, 2]}]], {n, 5, 10}];
```







The error diminishes rapidly with n , and is, for most purposes, quite negligible for $n = 10$.