Use the Fourier Series to Approximate Discontinuous Functions and to Interpret Music

Note: You may notice differences between this Maple worksheet and the equivalent Mathematica notebook. These differences were introduced to preserve the content of these modules and were necessary because of major functional differences between Maple and Mathematica.

Introduction

OBJECTIVE: Use *Maple* to calculate Fourier series and to build even and odd Fourier representations of selected functions.

In signal processing and communications, it is necessary to construct periodic functions, some with discontinuities. The Fourier series provides us with a tool to analyze such functions. One very important type of signal that you probably receive every day is music. We can use the Fourier series to build mathematical models of musical tones and to look at their graphs.

As you have probably noticed, the computations involved in arriving at a Fourier series approximation to a function can be tedious. Fortunately, the computer can perform such computations for us, enabling you to not only get the results but to visualize the results.

Maple has a package that allows you to merely call upon the Fourier Series. The package is called by typing **with(inttrans).** You can get details on this from the Help window; by typing **?inttrans**. This module does not call upon inttrans; instead, we use the **sum( )** command to compute the Fourier coefficients.

Technology Guidelines

NOTE: If you have just finished a worksheet, **restart** *Maple* before executing a new worksheet.
TO OPEN SECTIONS,
Click on the **PLUS** sign at the left hand side of the screen *or* select **Expand All Sections** from the **View** drop down menu.

TO STOP AN EXECUTION
Click on **STOP** button from the toolbar.

ORDER OF EXECUTION
Execute commands in the order given. Do not skip any *Maple* Input lines within a given worksheet

Alternatively, you can execute the entire worksheet by selecting the **Execute Worksheet** command from the **Edit** drop down menu.

SAVING WORKSHEETS.

You can save anytime to any directory you choose, and it is wise to save often.
EXPERIENCING MAJOR PROBLEMS
Save if appropriate, and then shut down *Maple* and start it up again.

Part I: Fourier Series Approximations for Non-Periodic Functions

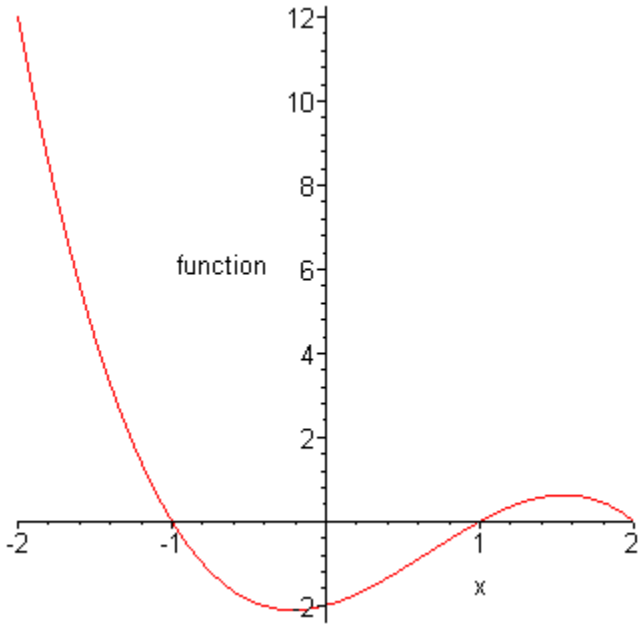
The following is an example of how you can use a Fourier series to approximate a continuous function. If you have not done so already, read Section 11.11 in your text before proceeding. We define the coefficients for the sine and cosine functions as they are defined in your text.

The function we look at first is $f(x) = (1-x)(x+1)(x-2)$ and we will find its Fourier series approximation over the interval $x = -2$ to $+2$. We begin by defining the function and looking at its plot.

```
> restart;
  f := x -> (1 - x)*(x + 1)*(x - 2);
  endpoint := 2;
  plot( f(x), x=-endpoint..endpoint, labels=["x", "function"] );
```

$$f := x \to (1 - x)\,(x + 1)\,(x - 2)$$

$$endpoint := 2$$



Next, we compute the Fourier coefficients using the formulations in your text.

```
> a := n -> 1/endpoint*int( sin(n*Pi/endpoint*x)*f(x), x=-endpoint..endpoint ):
  b := n -> 1/endpoint*int( cos(n*Pi/endpoint*x)*f(x), x=-endpoint..endpoint ):
```

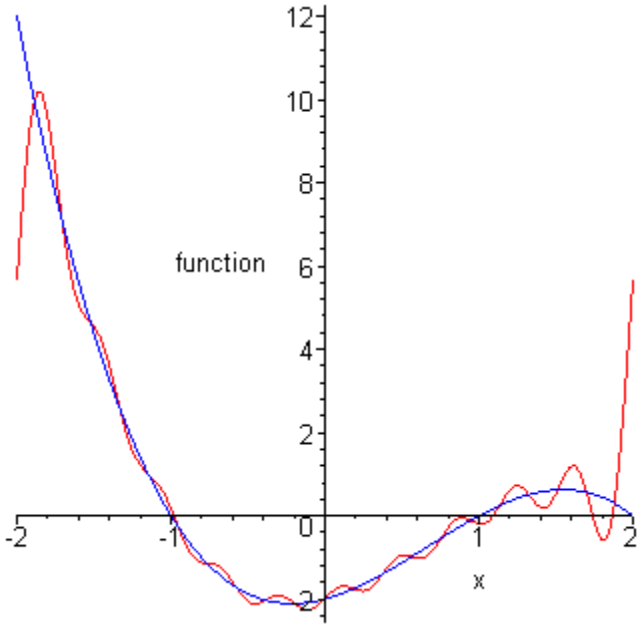
Now, we put these coefficients into the series formulas and go out to the $n = 10$ term in both the sine and cosine components.

```
> FourierSin := (x, n) -> sum( a(i)*sin(i*Pi/endpoint*x), i=1..n ):
  FourierCos := (x, n) -> b(0)/2 + sum( b(i)*cos(i*Pi/endpoint*x), i=1..n ):
  FourierAll := (x, n) -> FourierSin(x, n) + FourierCos(x, n):
  simplify( FourierSin(x, 10) );
```

$$-\frac{1}{166698000}\left(2000376000\sin\left(\frac{\pi x}{2}\right)\pi^2 - 16003008000\sin\left(\frac{\pi x}{2}\right) - 1000188000\sin(\pi x)\pi^2\right.$$
$$+ 2000376000\sin(\pi x) + 666792000\sin\left(\frac{3\pi x}{2}\right)\pi^2 - 592704000\sin\left(\frac{3\pi x}{2}\right) - 500094000\sin(2$$
$$+ 250047000\sin(2\pi x) + 400075200\sin\left(\frac{5\pi x}{2}\right)\pi^2 - 128024064\sin\left(\frac{5\pi x}{2}\right) - 333396000\sin(3$$
$$+ 74088000\sin(3\pi x) + 285768000\sin\left(\frac{7\pi x}{2}\right)\pi^2 - 46656000\sin\left(\frac{7\pi x}{2}\right) - 250047000\sin(4\pi$$
$$+ 31255875\sin(4\pi x) + 222264000\sin\left(\frac{9\pi x}{2}\right)\pi^2 - 21952000\sin\left(\frac{9\pi x}{2}\right) - 200037600\sin(5\pi$$
$$\left.+ 16003008\sin(5\pi x)\right) / \pi^3$$

I expect that you are glad that you did not have to calculate that by hand! Now, let's look at a plot of the function together with its Fourier series approximation.

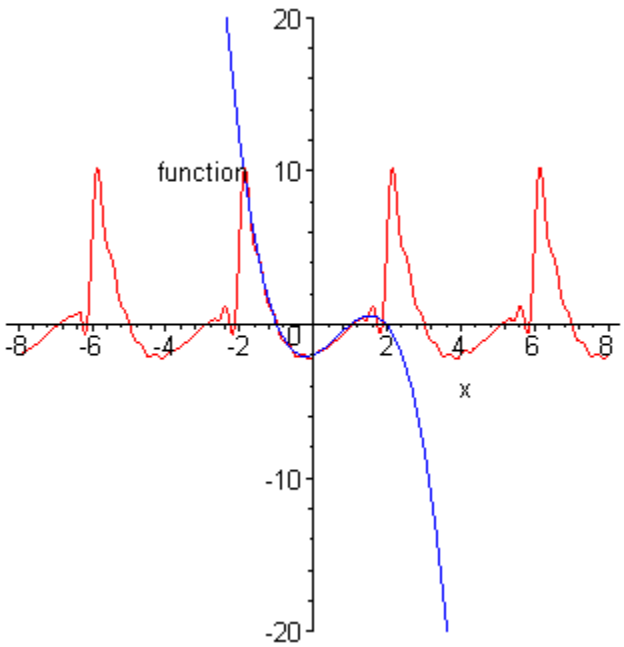
```
> plot( [f(x), FourierAll(x, 10)], x=-endpoint..endpoint, color=[blue,red], labels=["x",
  "function"] );
  printf( "The function is plotted in blue and its Fourier series approximation over the interv
  from x=-2 to x=2 is plotted in red");
```



The function is plotted in blue and its Fourier series approximation over the interval from x=-2 to x=2 is plotted in red

What happens if you extend the plot further? You will notice that the Fourier Series repeats the pattern of the function over the interval [-2, 2], whereas the polynomial is not at all periodic.

> **plot([f(x), FourierAll(x, 10)], x=-4*endpoint..4*endpoint, view=[DEFAULT, -20..20] , colo**
[blue,red],labels=["x", "function"]);



The Fourier series gives a reasonable approximation to the function over the interval [-2, 2], but

outside that interval, anything can happen. The extended Fourier series is called the periodic extension of the function.

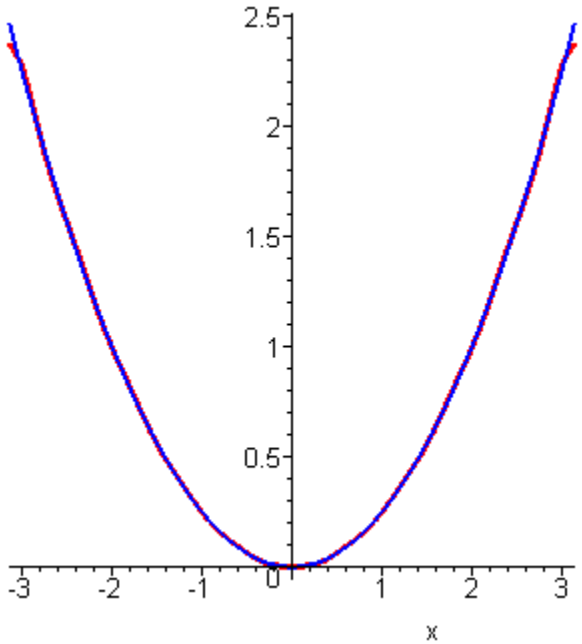
You Try It: Part I

Find the Fourier series for the function $\frac{x^2}{4}$ over the interval $-\pi$ to π. The following will get you started.

```
> f := x -> x^2/4;
  endpoint := Pi;
  plot( [f(x), FourierAll(x, 10)], x=-endpoint..endpoint, thickness=[2,3], color=[blue,red] );
  ### WARNING: %x or %X format should be %y or %Y if used with floating point argun
  printf( "The function is plotted in blue and its Fourier series approximation over the inter
  x=-%a to x=%a is plotted in red", endpoint, endpoint );
  FourierAll(x, 10);
```

$$f := x \to \frac{1}{4}x^2$$

$$endpoint := \pi$$



The function is plotted in blue and its Fourier series approximation over the interval x=-Pi to x=Pi is plotted in red

$$\frac{\pi^2}{12} - \cos(x) + \frac{1}{4}\cos(2x) - \frac{1}{9}\cos(3x) + \frac{1}{16}\cos(4x) - \frac{1}{25}\cos(5x) + \frac{1}{36}\cos(6x) - \frac{1}{49}\cos(7x) +$$

$$-\frac{1}{81}\cos(9x)+\frac{1}{100}\cos(10x)$$

Use this result to verify that the series that converges to $\frac{\pi^2}{6}$ is $1+\frac{1}{4}+\frac{1}{9}+\frac{1}{16}+\frac{1}{25}\ \ldots.$

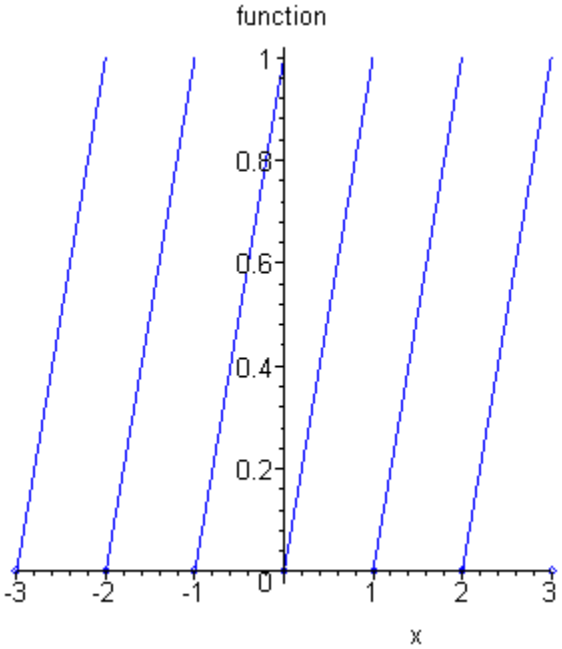
What should you let x equal in your $f(x)$ and in your Fourier series to get the desired result?

Can you come up with the sum of any other infinite series by this method?

Part II: Fourier Coefficients for the Sawtooth Function

Consider the Sawtooth Function to be essentially the line $y = x$ over the interval from 0 to 1, plus that segment repeated over and over. Let's look at a graph of this function.

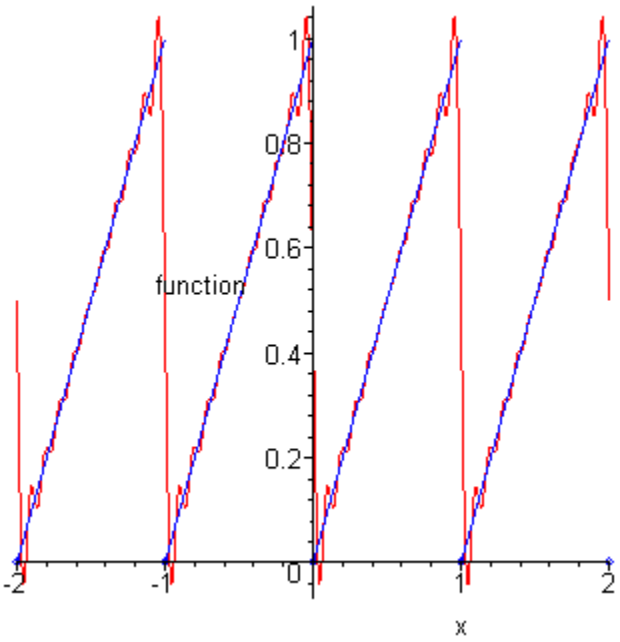
```
> f := x -> x - floor(x):
  endpoint := 1:
  plot( f(x), x=-3..3 , color=blue,title="function",discont=true );
```



To compute Fourier coefficients for this function, we need to focus on the function over the interval -1 to 1. The function is $(x+1)$ when x goes from -1 to 0 and then becomes x when x goes from 0 to +1. Here we will put all our commands together and show the results at the end.

```
> plot( [f(x), FourierAll(x, 20)], x=-2*endpoint..2*endpoint, color=[blue, red], discont=true,
  labels=["x","function"] );
  ### WARNING: %x or %X format should be %y or %Y if used with floating point argun
  printf( `The function is plotted in blue and its Fourier series approximation over the interv
```

```
x=-%a to x=%a is plotted in red`, endpoint, endpoint );
printf(`The Fourier series approximation out to n=20 is`);
FourierCos(x,20)+FourierSin(x,20);
```



The function is plotted in blue and its Fourier series approximation over the interval x=-1 to x=1 is plotted in red

The Fourier series approximation out to n=20 is

$$-\frac{\sin(2\pi x)}{\pi}-\frac{1}{2}\frac{\sin(4\pi x)}{\pi}-\frac{1}{3}\frac{\sin(6\pi x)}{\pi}-\frac{1}{4}\frac{\sin(8\pi x)}{\pi}-\frac{1}{5}\frac{\sin(10\pi x)}{\pi}-\frac{1}{6}\frac{\sin(12\pi x)}{\pi}-\frac{1}{7}\sin$$
$$-\frac{1}{8}\frac{\sin(16\pi x)}{\pi}-\frac{1}{9}\frac{\sin(18\pi x)}{\pi}-\frac{1}{10}\frac{\sin(20\pi x)}{\pi}+\frac{1}{2}$$

Since this is neither an even nor an odd function, we need both the sine and cosine series to fit the function. However, note that there is only one nonzero term in the Fourier cosine series.

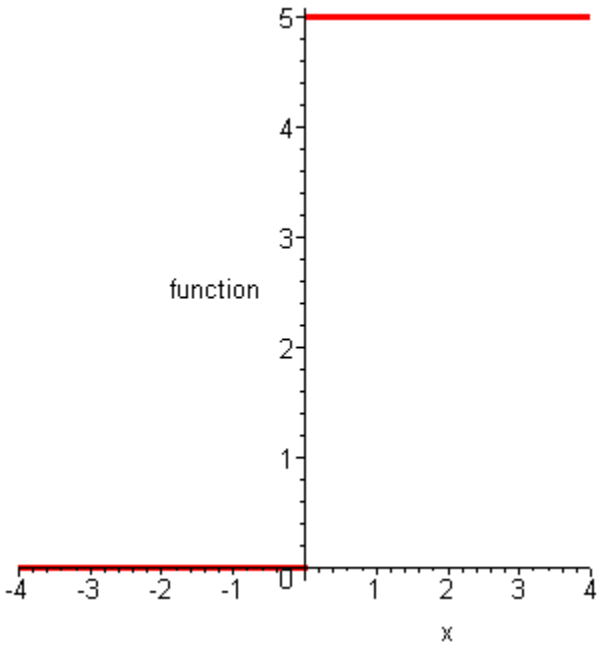
You should notice what is referred to as Gibbs' phenomenon. That refers to the poor (wiggly) fit at the points of discontinuity. There are far fewer wiggles over the continuous sections. This phenomenon would occur no matter how many terms we extended our series.

You Try It: Part II

Let's suppose you have a step function that takes on the value 0 from -3 to 0 and then jumps up to 5 when x is between 0 and 3. First we will define the function and look at its graph.

```
> f :=x->piecewise(  x>0, 5, x<0, 0);
  plot(f(x), x=-4..4, color=red, thickness=3,discont=true,labels=["x","function"]);
```

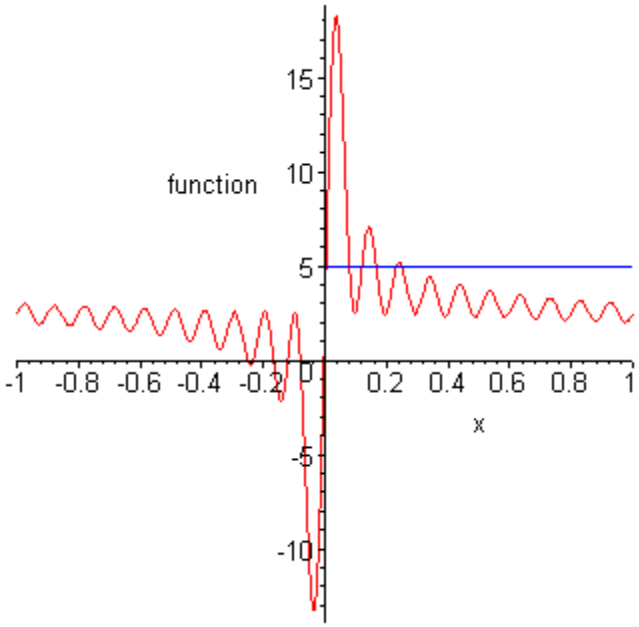
$$f := x \rightarrow \text{piecewise}(0 < x, 5, x < 0, 0)$$



Alter the **endpoint** and the function **f** to generate the series you want for this function. For more efficient computation, we define the series out to 20 terms. If you want it out to more or fewer terms, change the terms in **FourierAll**..

```
> f:=x->eval(piecewise( t < 0,0,5 ), t=x);
  endpoint := 1:
  plot( [f(x), FourierAll(x, 20)], x=-endpoint..endpoint,color=[blue, red] ,labels=["x","functi
  printf(`The function is plotted in blue and its Fourier series approximation over the interv
  from -%a to %a is plotted in red.`, endpoint, endpoint );
  print( `The Fourier series approximation out to n = 20 is`, evalf( FourierAll(x, 20) ) );
```

$$f := x \rightarrow \text{piecewise}(t < 0, 0, 5)\Big|_{t = x}$$



The function is plotted in blue and its Fourier series approximation over the interval from -1 to 1 is plotted in red.

The Fourier series approximation out to n = 20 is, 1.061032954 sin(3.141592654 x)
+ 1.061032954 sin(6.283185308 x) + 1.061032954 sin(9.424777962 x) + 1.061032954 sin(12.56
+ 1.061032954 sin(15.70796327 x) + 1.061032954 sin(18.84955592 x) + 1.061032954 sin(21.99
+ 1.061032954 sin(25.13274123 x) + 1.061032954 sin(28.27433389 x) + 1.061032954 sin(31.41
+ 1.061032954 sin(34.55751919 x) + 1.061032954 sin(37.69911185 x) + 1.061032954 sin(40.84
+ 1.061032954 sin(43.98229716 x) + 1.061032954 sin(47.12388981 x) + 1.061032954 sin(50.26
+ 1.061032954 sin(53.40707512 x) + 1.061032954 sin(56.54866777 x) + 1.061032954 sin(59.69
+ 1.061032954 sin(62.83185308 x) + 2.500000000

Part III: Define a Function to be Either Even or Odd

Sometimes, all we want is a Fourier series approximation for a function over an interval and we don't care how the periodic extension of the function behaves. Consequently, we have the choice of constructing a Fourier series that is either even (cosine terms only) or odd (sine terms only). There may be a reason that you would choose to have only cosine terms or only sine terms to approximate the function. The following is an example of how you can do either.

Suppose that the function for which we want the Fourier series takes on the value x for $0 \le x < \frac{1}{2}$ and the value $\frac{1}{2}$ for $(\frac{1}{2} \le x < 1)$. We will begin by defining

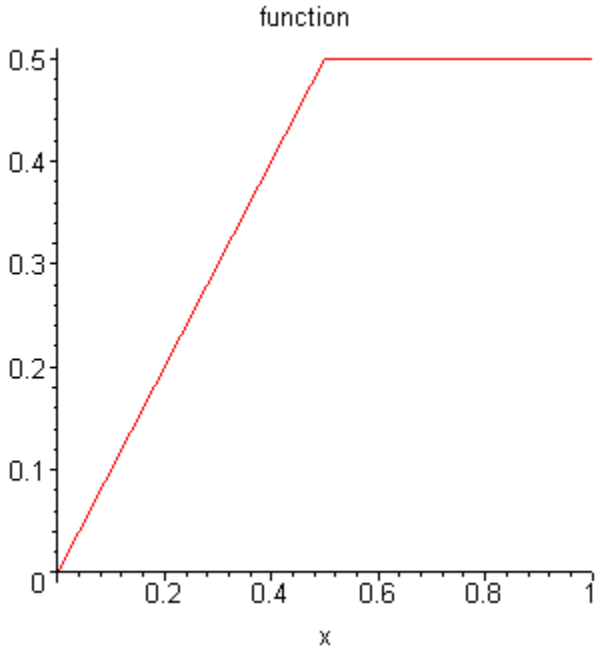
$g(x)$ over the interval 0 to 1, and then we will extend the definition to an odd function and to an even function.

> **endpoint := 1;**

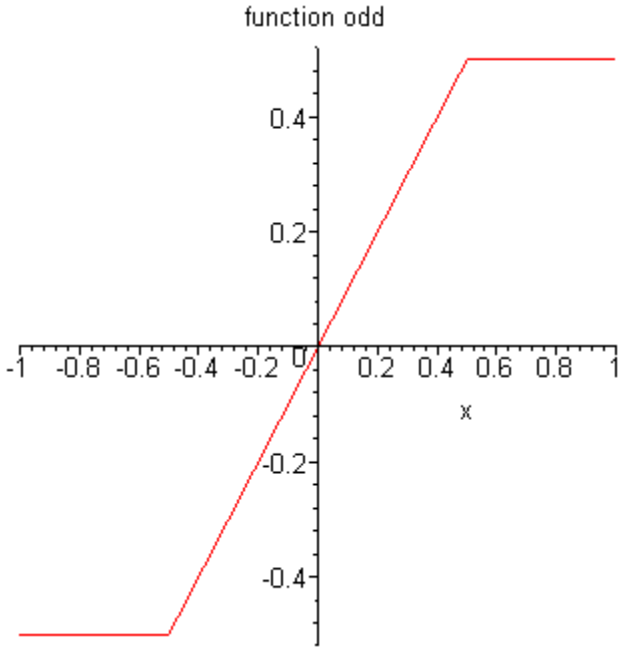
```
g := x -> eval( piecewise( t < 0, undefined, t <= 0.5, t, t <= 1, 0.5, undefined ), t=x );
plot( g(x), x=0..endpoint,labels=["x",""], title="function");
godd := x -> eval( piecewise( t < -1, undefined, t <= -0.5, -0.5, t <= 0.5, t, t <= 1, 0.5, undefin
t=x );
plot( godd(x), x=-endpoint..endpoint ,labels=["x",""],title="function odd");
geven := x -> eval( piecewise( t < -1, undefined, t <= -0.5, 0.5, t < 0, -t, t <= 0.5, t, t <= 1, 0.5,
undefined ), t=x );
plot( geven(x), x=-endpoint..endpoint ,labels=["x",""], title="function even");
```

$$endpoint := 1$$

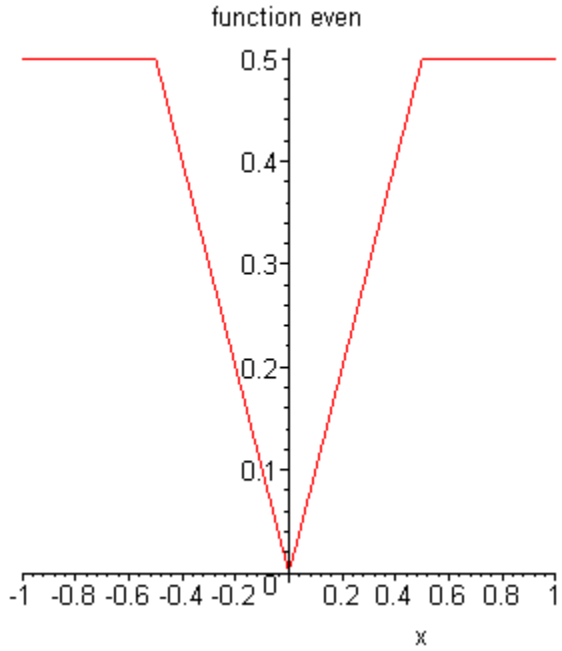
$$g := x \to \text{piecewise}(t < 0, undefined, t \le 0.5, t, t \le 1, 0.5, undefined)\Big|_{t=x}$$



$$godd := x \to \text{piecewise}(t < -1, undefined, t \le -0.5, -0.5, t \le 0.5, t, t \le 1, 0.5, undefined)\Big|_{t=x}$$



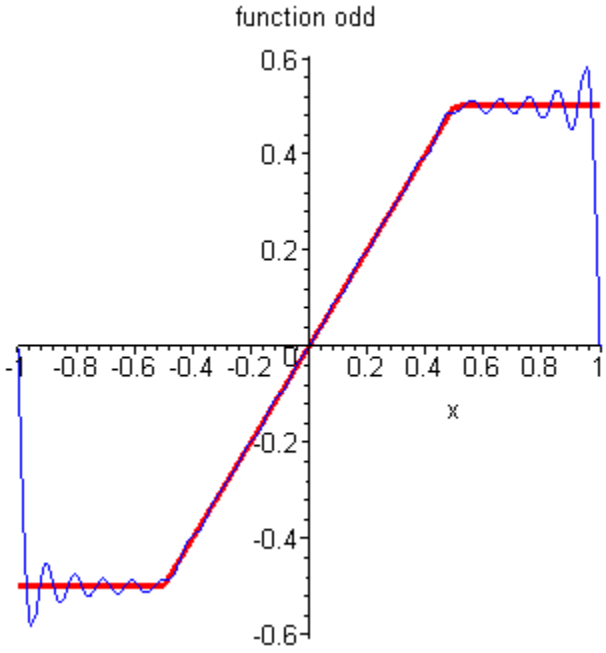
$$geven := x \rightarrow \text{piecewise}(t < -1, undefined, t \le -0.5, 0.5, t < 0, -t, t \le 0.5, t, t \le 1, 0.5, undefined)\Big|_t$$



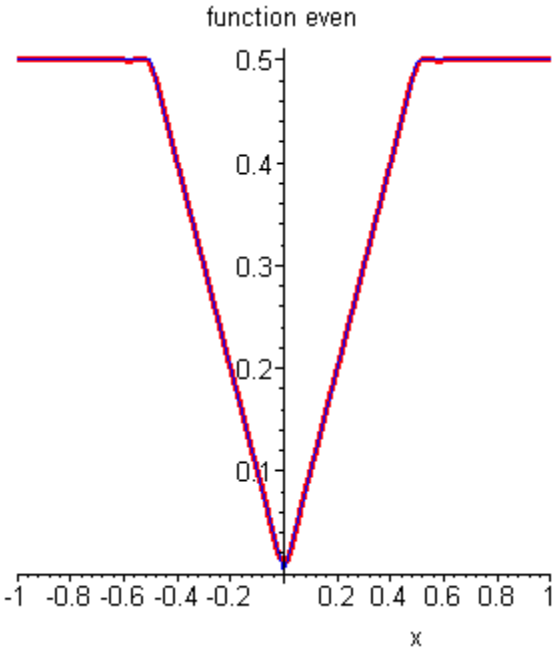
Here we will call upon the simplified formulas used when you are looking only for a sine or only for a cosine series. We double the integrals, but integrate over only half the interval and multiply the integral by 2. We will go to $n = 20$ in both the sine and cosine series for these evaluations.

```
> endpoint:=1:
  a := n -> 2/endpoint*int( sin(n*Pi/endpoint*x)*x, x=0..endpoint/2 )+2/endpoint*int( sin
  (n*Pi/endpoint*x)*0.5, x=endpoint/2..endpoint ):
  b := n -> 2/endpoint*int( cos(n*Pi/endpoint*x)*x, x=0..endpoint/2 )+2/endpoint*int( cos
  (n*Pi/endpoint*x)*0.5, x=endpoint/2..endpoint ):
```

```
FourierSin := (x, n) -> sum( a(i)*sin(i*Pi/endpoint*x), i=1..n ):
FourierCos := (x, n) -> b(0)/2 + sum( b(i)*cos(i*Pi/endpoint*x), i=1..n ):
FourierAll := (x, n) -> FourierSin(x, n) + FourierCos(x, n):
plot( {godd(x), FourierSin(x, 20)}, x=-1..1 ,color=[blue, red] ,thickness=[1,3],labels=
["x",""],title="function odd" );
print( `The Fourier sine series approximation out to n = 20 is`, evalf( FourierSin(x, 20) ) );
plot( [geven(x), FourierCos(x, 20)], x=-endpoint..endpoint ,color=[blue, red], thickness=
[1,3],labels=["x",""],title="function even" );
print( `The Fourier cosine series approximation out to n = 20 is`, evalf( FourierCos(x, 20) )
```



The Fourier sine series approximation out to n = 20 is, $0.5209522533 \sin(3.141592654\,x) - 0.1591549432 \sin(6.283185308\,x) + 0.08358747689 \sin(9.424777962\,x) - 0.07957747152 \sin(12.56637062\,x) + 0.07176767186 \sin(15.70796327\,x) - 0.05305164771 \sin(18.84955592\,x) + 0.04133728249 \sin(21.99114858\,x) - 0.03978873576 \sin(25.13274123\,x) + 0.03786952279 \sin(28.27433389\,x) - 0.03183098863 \sin(31.41592654\,x) + 0.02726253222 \sin(34.55751919\,x) - 0.02652582384 \sin(37.69911185\,x) + 0.02568444308 \sin(40.84070450\,x) - 0.02273642044 \sin(43.98229716\,x) + 0.02032002630 \sin(47.12388981\,x) - 0.01989436788 \sin(50.26548246\,x) + 0.01942529553 \sin(53.40707512\,x) - 0.01768388258 \sin(56.54866777\,x) + 0.01619181573 \sin(59.69026043\,x) - 0.01591549430 \sin(62.83185308\,x)$



The Fourier cosine series approximation out to n = 20 is, $0.3750000000 - 0.2026423674 \cos(3.14 - 0.1013211834 \cos(6.283185308\,x) - 0.02251581863 \cos(9.424777962\,x) + 0.3835963200\,10^{-9} \cos(12.56637062\,x) - 0.008105694851 \cos(15.70796327\,x) - 0.01125790930 \cos(18.84955592\,x) - 0.004135558732 \cos(21.99114858\,x) - 0.1734459809\,10^{-9} \cos(25.13274123\,x) - 0.002501757902 \cos(28.27433389\,x) - 0.004052847148 \cos(31.41592654\,x) - 0.001674730315 \cos(34.55751919\,x) + 0.2244413769\,10^{-9} \cos(37.69911185\,x) - 0.001199067352 \cos(40.84070450\,x) - 0.002067778925 \cos(43.98229716\,x) - 0.0009006328706 \cos(47.12388981\,x) + 0.2549769804\,10^{-10} \cos(50.26548246\,x) - 0.0007011848380 \cos(53.40707512\,x) - 0.001250878579 \cos(56.54866777\,x) - 0.0005613363876 \cos(59.69026043\,x) + 0.6528643381\,10^{-10} \cos(62.83185308\,x)$

Note that both the sine and the cosine expansions fit the function over the interval 0 to 1, but, over a broader range, one is even and one is odd. The cosine series seems to be a better fit.

You Try It: Part III

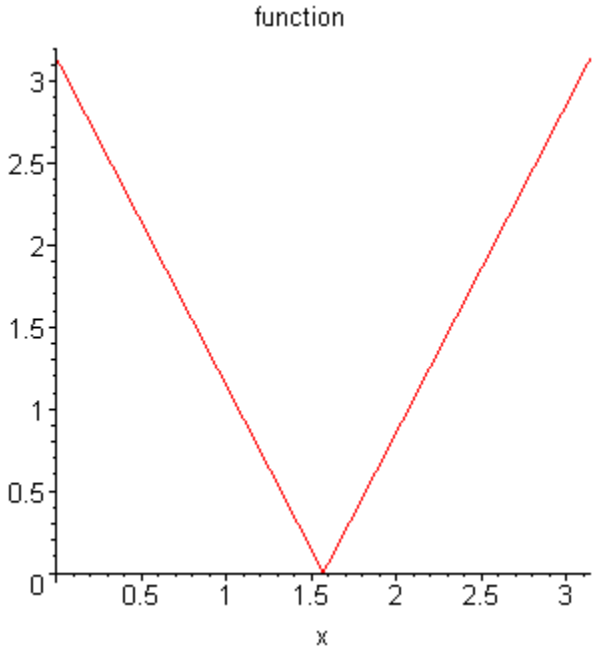
Find both a Fourier sine series and cosine series expansion for the function $|2x - \pi|$ over the interval 0 to π. We could simply define the function as given, but, for clarity in plotting, we will define both its even counterpart and its odd counterpart over the interval $-\pi$ to π. Note that this can be done by shifting the function to the left a distance of π units.

```
> endpoint := Pi;
  g := x -> eval( piecewise( t < 0, Pi, t <= Pi/2, Pi-2*t, t <= Pi, 2*t-Pi, Pi ), t=x );
```

```
plot( g(x), x=0..Pi, labels=["x",""],title="function" );
godd := x -> eval( piecewise( t < -Pi, undefined, t <= -Pi/2, 2*t+Pi, t < 0, -Pi-2*t, t <= Pi/2, P
t <= Pi, 2*t-Pi, undefined ), t=x );
plot( godd(x), x=-Pi..Pi , labels=["x",""],title="even function" );
geven := x -> eval( piecewise( t < -Pi, undefined, t <= -Pi/2, -Pi-2*t, t < 0, 2*t+Pi, t <= Pi/2, I
2*t, t <= Pi, 2*t-Pi, undefined ), t=x );
plot( geven(x), x=-Pi..Pi, labels=["x",""],title="odd function"  );
```

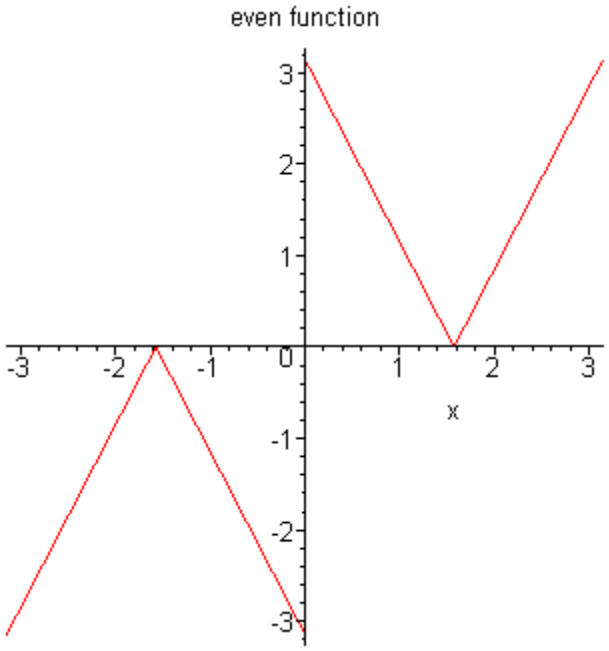
$$endpoint := \pi$$

$$g := x \to \text{piecewise}\left(t < 0, \pi, t \le \frac{\pi}{2}, \pi - 2\,t, t \le \pi, 2\,t - \pi, \pi\right)\Big|_{t=x}$$

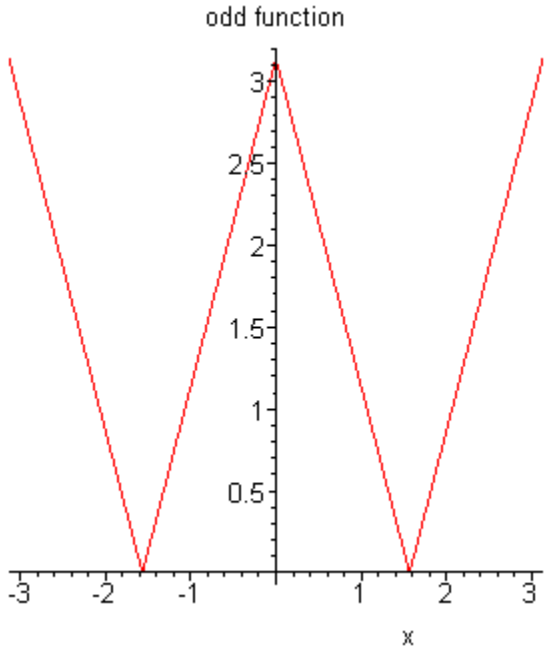


$$godd := x \to$$

$$\text{piecewise}\left(t < -\pi, undefined, t \le -\frac{\pi}{2}, 2\,t + \pi, t < 0, -\pi - 2\,t, t \le \frac{\pi}{2}, \pi - 2\,t, t \le \pi, 2\,t - \pi, undefin\right.$$



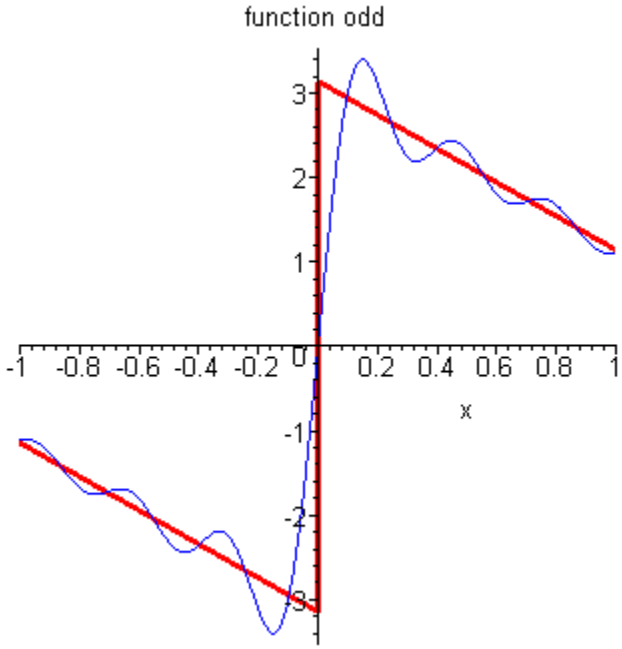
$$geven := x \rightarrow \text{piecewise}\left(t < -\pi, undefined, t \le -\frac{\pi}{2}, -\pi - 2\,t, t < 0, 2\,t + \pi, t \le \frac{\pi}{2}, \pi - 2\,t, t \le \pi, 2\,t - \pi, undefin\right.$$



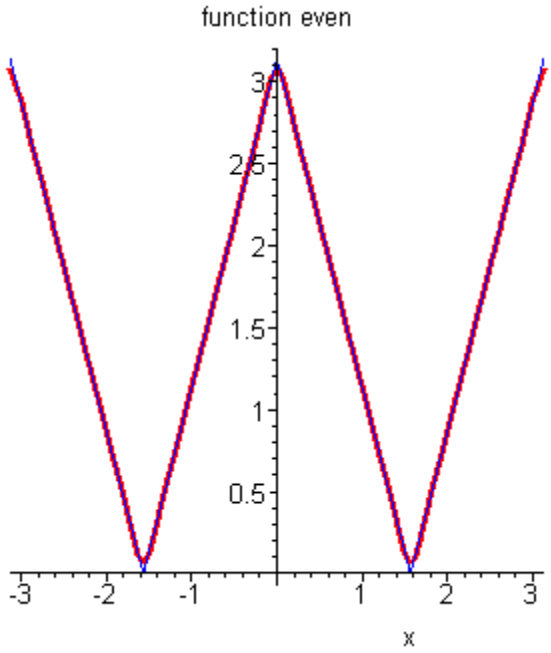
Note that this function is -(2x - π) between 0 and $\frac{\pi}{2}$ but +(2x - π) from $\frac{\pi}{2}$ to π . Look at the following code and see if you can tell how we have adjusted the code for evaluation of the coefficients a and b in order to adapt to the given function rather than the one in Part III above.

Then execute the cell and see if you get what you expected.

```
> unassign('a,b');
  endpoint:=Pi:
  a := n -> 2/endpoint*int( sin(n*Pi/endpoint*x)*(-2*x+Pi), x=0..endpoint/2 )+2/endpoint*int
  (n*Pi/endpoint*x)*(2*x-Pi), x=endpoint/2..endpoint ):
  b := n -> 2/endpoint*int( cos(n*Pi/endpoint*x)*(-2*x+Pi), x=0..endpoint/2 )+2/endpoint*int
  (n*Pi/endpoint*x)*(2*x-Pi), x=endpoint/2..endpoint ):
  FourierSin := (x, n) -> sum( a(i)*sin(i*Pi/endpoint*x), i=1..n ):
  FourierCos := (x, n) -> b(0)/2 + sum( b(i)*cos(i*Pi/endpoint*x), i=1..n ):
  FourierAll := (x, n) -> FourierSin(x, n) + FourierCos(x, n):
  plot( {godd(x), FourierSin(x, 20)}, x=-1..1 ,color=[blue, red] ,thickness=[1,3],labels=
  ["x",""],title="function odd" );
  print( `The Fourier sine series approximation out to n = 20 is`, evalf( FourierSin(x, 20) ) );
  plot( [geven(x), FourierCos(x, 20)], x=-endpoint..endpoint ,color=[blue, red], thickness=
  [1,3],labels=["x",""],title="function even" );
  print( `The Fourier cosine series approximation out to n = 20 is`, evalf( FourierCos(x, 20) )
```



The Fourier sine series approximation out to n = 20 is, 1.453520911 sin(x) + 1.616275454 sin(3. x
+ 0.6981408363 sin(5. x) + 0.6233975324 sin(7. x) + 0.4130064310 sin(9. x) + 0.3846816453 sin
+ 0.2926243841 sin(13. x) + 0.2779843515 sin(15. x) + 0.2264827714 sin(17. x) + 0.217580274



The Fourier cosine series approximation out to n = 20 is, $1.570796327 + 1.273239544\cos(2.x) + 0.1414710605\cos(6.x) + 0.05092958178\cos(10.x) + 0.02598448050\cos(14.x) + 0.01571900672\cos(18.x)$

Is the cosine function a better fit again?

Try this with other functions.

Part IV: Analyze Musical Tones of a Clarinet

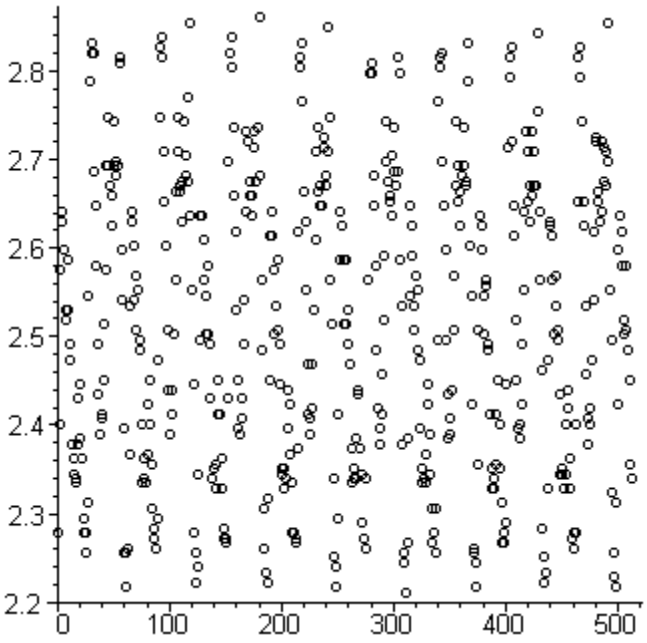
The harmonies in mathematics are heard in musical instruments such as the piano and clarinet and are seen in the Fourier Series of the form:

$$a(0) + a(1)\cos\left(\frac{2\pi 1 t}{\lambda}\right) + b(1)\sin\left(\frac{2\pi 1 t}{\lambda}\right) + a(2)\cos\left(\frac{2\pi 2 t}{\lambda}\right) + b(2)\sin\left(\frac{2\pi 2 t}{\lambda}\right)$$

$$+ \ldots + a(n)\cos\left(\frac{2\pi n t}{\lambda}\right) + b(n)\sin\left(\frac{2\pi n t}{\lambda}\right)$$

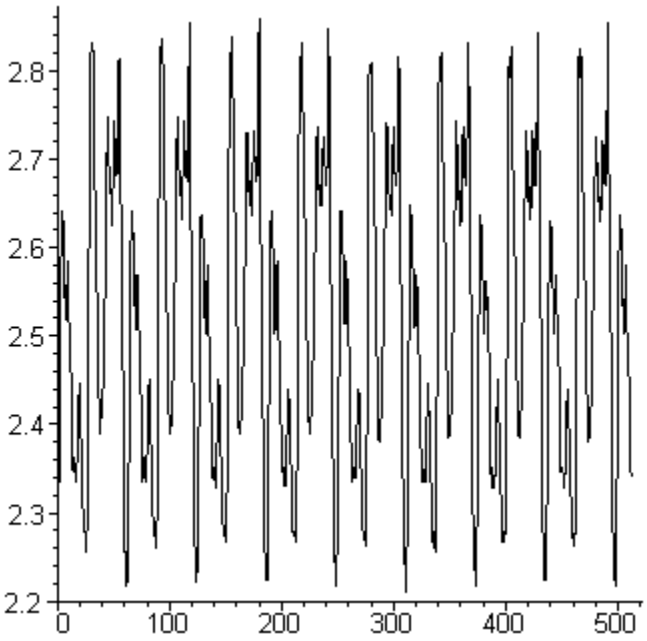
A group of students from Carroll College in Montana recorded musical tones from a clarinet and from a piano using a sound sensor that interfaces with a graphing calculator. The recorded signals represent a measure of the loudness of the tones as a function of time. After sampling the tones and graphically observing the periodic pattern, the students selected what appeared to be approximately one period of the signal and then used Riemann sums to approximate the integrals for the Fourier coefficients. The following data sets are those that were collected by the students. Lambda (λ) represents the length of the interval selected for what appears to be one period in the recorded signal. Increasing the number of Fourier terms gives better approximations of the signal.

```
> clarinet := [2.27817, 2.40131, 2.57483, 2.64200, 2.63081, 2.59722, 2.53005, 2.51886, 2.58603,
2.53005, 2.49087, 2.47408, 2.37892, 2.34534, 2.36213, 2.33974, 2.33414, 2.37892, 2.42930, 2.4
2.38452, 2.36213, 2.29496, 2.27817, 2.27817, 2.25578, 2.31175, 2.54684, 2.78753, 2.82112, 2.8
2.82112, 2.68678, 2.64760, 2.58043, 2.49087, 2.43489, 2.39012, 2.40691, 2.41250, 2.45169, 2.5
2.57483, 2.69238, 2.74835, 2.69238, 2.66999, 2.65879, 2.62521, 2.69238, 2.74276, 2.69798, 2.6
2.69238, 2.81552, 2.80992, 2.59722, 2.54125, 2.39571, 2.25578, 2.25578, 2.21659, 2.26137, 2.3
2.53565, 2.64200, 2.63081, 2.60282, 2.54125, 2.50766, 2.56923, 2.55244, 2.49647, 2.48527, 2.4
2.33414, 2.36213, 2.33974, 2.33414, 2.36773, 2.42370, 2.45169, 2.40131, 2.35653, 2.30615, 2.2
2.28376, 2.26137, 2.29496, 2.47408, 2.74835, 2.82672, 2.81552, 2.83791, 2.70917, 2.65320, 2.6
2.50766, 2.44049, 2.39012, 2.39012, 2.41250, 2.44049, 2.50206, 2.56364, 2.66439, 2.74835, 2.7
2.66439, 2.66999, 2.63081, 2.67559, 2.74276, 2.70357, 2.68118, 2.67559, 2.77074, 2.85470, 2.6
2.55244, 2.44609, 2.27817, 2.25578, 2.22219, 2.23898, 2.34534, 2.49647, 2.63640, 2.63640, 2.6
2.56364, 2.50206, 2.54684, 2.58043, 2.50206, 2.49087, 2.42930, 2.33974, 2.35093, 2.35093, 2.3
2.35653, 2.41250, 2.45169, 2.41250, 2.36213, 2.32854, 2.27257, 2.28376, 2.26697, 2.27257, 2.4
2.69798, 2.82112, 2.80433, 2.83791, 2.73716, 2.65879, 2.61961, 2.53005, 2.45169, 2.39571, 2.3
2.40691, 2.42930, 2.49087, 2.54125, 2.64200, 2.73156, 2.72037, 2.65879, 2.67559, 2.63640, 2.6
2.73156, 2.71477, 2.67559, 2.67559, 2.73716, 2.86030, 2.68118, 2.56364, 2.48527, 2.30615, 2.2
2.23339, 2.22219, 2.31735, 2.45169, 2.61401, 2.64200, 2.61401, 2.57483, 2.50206, 2.53565, 2.5
2.50766, 2.49087, 2.44609, 2.35093, 2.34534, 2.35093, 2.32854, 2.33974, 2.39571, 2.44049, 2.4
2.36773, 2.33414, 2.27817, 2.27817, 2.27257, 2.26697, 2.37332, 2.61961, 2.81552, 2.80433, 2.8
2.76514, 2.66439, 2.63081, 2.55244, 2.46848, 2.41250, 2.39012, 2.40691, 2.41810, 2.46848, 2.5
2.60842, 2.70917, 2.73716, 2.66439, 2.66999, 2.64760, 2.64760, 2.71477, 2.72596, 2.68118, 2.6
2.70917, 2.84911, 2.74835, 2.56364, 2.51326, 2.33974, 2.25018, 2.23898, 2.21659, 2.29496, 2.4
2.58603, 2.64200, 2.62521, 2.58603, 2.51326, 2.51326, 2.58603, 2.53005, 2.49087, 2.46848, 2.3
2.33414, 2.35093, 2.33974, 2.33974, 2.38452, 2.43489, 2.44049, 2.37332, 2.34534, 2.28936, 2.2
2.27257, 2.26137, 2.33974, 2.56364, 2.79873, 2.79873, 2.80992, 2.79873, 2.68118, 2.64760, 2.5
2.48527, 2.41810, 2.37892, 2.39571, 2.41250, 2.45728, 2.51886, 2.59162, 2.69798, 2.74276, 2.6
2.65879, 2.65320, 2.63640, 2.70357, 2.73716, 2.68678, 2.66999, 2.68678, 2.81552, 2.79873, 2.5
2.53565, 2.37892, 2.25578, 2.24458, 2.21100, 2.26697, 2.38452, 2.54684, 2.64760, 2.62521, 2.5
2.53565, 2.50766, 2.56923, 2.55244, 2.48527, 2.47408, 2.39571, 2.33414, 2.35093, 2.33974, 2.3
2.36773, 2.42370, 2.44609, 2.39012, 2.34534, 2.30615, 2.26697, 2.27817, 2.25578, 2.30615, 2.4
2.76514, 2.81552, 2.80433, 2.82112, 2.69798, 2.64760, 2.59722, 2.49647, 2.43489, 2.38452, 2.3
2.40691, 2.44049, 2.50766, 2.56923, 2.66999, 2.74276, 2.69238, 2.65320, 2.66439, 2.62521, 2.6
2.73716, 2.69238, 2.66999, 2.67559, 2.78753, 2.83231, 2.60282, 2.54684, 2.42370, 2.26137, 2.2
2.21659, 2.24458, 2.35653, 2.50766, 2.63640, 2.62521, 2.59722, 2.54684, 2.50206, 2.55804, 2.5
2.49087, 2.48527, 2.41250, 2.32854, 2.35093, 2.33974, 2.32854, 2.35653, 2.41250, 2.45169, 2.4
2.35093, 2.31175, 2.26697, 2.27817, 2.26697, 2.28936, 2.44609, 2.71477, 2.81552, 2.79313, 2.8
2.72037, 2.64760, 2.61401, 2.51886, 2.45169, 2.39571, 2.38452, 2.40131, 2.42370, 2.49087, 2.5
2.64200, 2.73156, 2.70917, 2.65320, 2.66999, 2.63081, 2.65879, 2.73156, 2.70917, 2.66999, 2.6
2.75395, 2.84351, 2.64200, 2.56364, 2.46288, 2.28376, 2.25018, 2.22219, 2.23339, 2.32854, 2.4
2.62521, 2.63081, 2.61401, 2.56364, 2.50206, 2.53565, 2.56923, 2.50766, 2.49647, 2.43489, 2.3
2.35093, 2.34534, 2.32854, 2.34534, 2.40131, 2.44049, 2.41810, 2.36213, 2.32854, 2.27257, 2.2
2.26137, 2.27817, 2.40131, 2.65320, 2.81552, 2.79313, 2.82672, 2.74276, 2.65320, 2.62521, 2.5
2.45728, 2.40691, 2.37892, 2.40131, 2.41810, 2.47408, 2.54125, 2.61961, 2.72037, 2.72596, 2.6
2.66439, 2.63081, 2.64200, 2.72037, 2.71477, 2.67559, 2.66999, 2.70917, 2.85470, 2.69798, 2.5
2.49647, 2.32295, 2.25578, 2.22779, 2.21659, 2.31175, 2.42370, 2.59722, 2.63640, 2.61961, 2.5
2.50206, 2.51886, 2.58043, 2.50766, 2.48527, 2.45169, 2.35653, 2.33974]:
plots[listplot]( clarinet, style=POINT, symbol=CIRCLE );
```



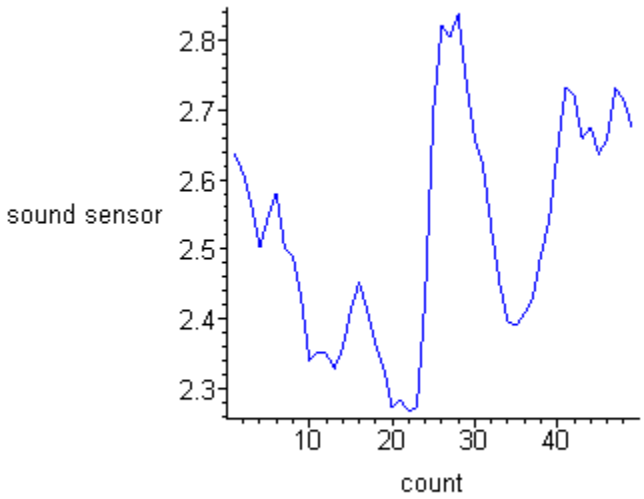
To get a clearer understanding of the signal you are hearing, let's look at a graph of the signal with the points connected. Note the periodic behavior of the musical tone. The units on the vertical axis represent the sound levels in the units recorded by the particular sound sensor.

> **plots[listplot](clarinet);**



The students chose the points 129 to 177 to represent one period. Does that look about right? That means that the interval will be of length λ = 177 - 129.

```
> initial := 129:
  final := 177:
  period := clarinet[initial..final]:
  lambda := nops( period ):
  clarinetplot := plots[listplot]( period, labels=["count", "sound sensor"], color=blue ):
  plots[display]( clarinetplot );
```



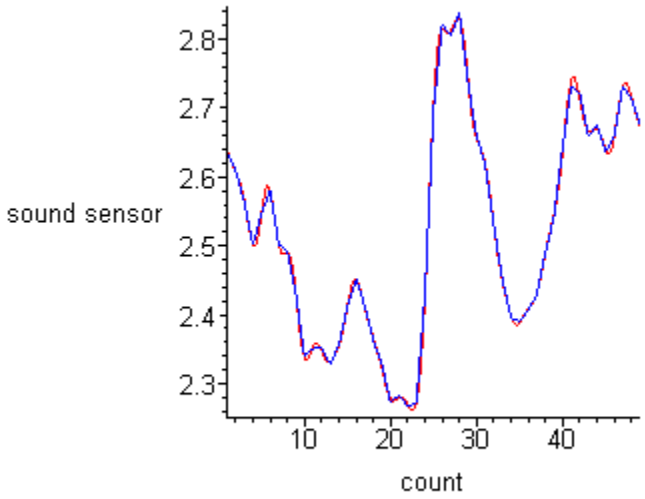
We want to get a Fourier series approximation for the set of points you see in the graph. In what follows, note the use of Riemann sums rather than integrals to approximate the Fourier coefficients. Because you have a set of discrete values (assigned to the symbol period) instead of a mathematical function to approximate, you can estimate the areas of each small rectangle to be the width of the rectangle ($\frac{1}{\lambda}$) times the appropriate height (sound sensor values as in the above graph). Adding up these areas gives approximations to the integrals over the interval of length λ .

```
> terms := 20:
  a := proc(n)
     option remember;

     if n = 0 then
        add( period[i], i=1..lambda ) / lambda;
     elif type( n, posint ) then
        2*add( period[i]*cos(2*Pi*n*i/lambda), i=1..lambda ) / lambda;
     else
        'procname'(n);
     fi;
  end:
  b := proc(n)
```

```
    option remember;

    if type( n, posint ) then
       2*add( period[i]*sin(2*Pi*n*i/lambda), i=1..lambda ) / lambda;
    else
       'procname'(n);
    fi;
end:
musicfour := proc(t)
    evalf( a(0) + add( a(i)*cos(2*Pi*i*t/lambda) + b(i)*sin(2*Pi*i*t/lambda), i = 1..terms ) );
end:
```

Now we can see what our simulated clarinet plot looks like.

```
> clarinetfourplot := plot( 'musicfour'(t), t=1..lambda, color=red ):
  plots[display]( [clarinetplot, clarinetfourplot] );
  printf( `The orignal plot is in blue and the Fourier series plot is in red.` );
```



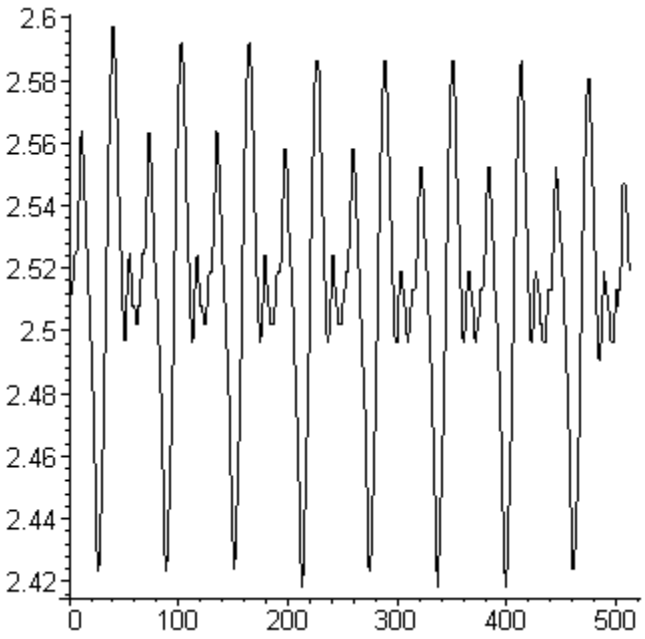
The orignal plot is in blue and the Fourier series plot is in red.

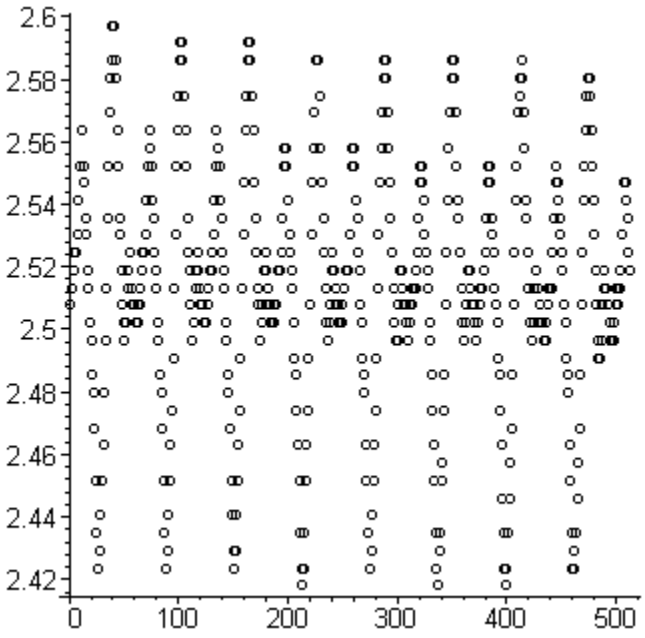
You Try It: Part IV: The Piano

As with the clarinet example, first we look at and listen to all the sample points collected when middle C is played on a piano.

```
> piano := [2.50766, 2.51326, 2.51326, 2.51886, 2.52445, 2.52445, 2.52445, 2.53005, 2.54125,
  2.55244, 2.56364, 2.56364, 2.55244, 2.54684, 2.53565, 2.53005, 2.51886, 2.51326, 2.50206, 2.4
  2.48527, 2.47967, 2.46848, 2.45169, 2.43489, 2.42370, 2.42370, 2.42930, 2.44049, 2.45169, 2.4
  2.47967, 2.49647, 2.51326, 2.53565, 2.55244, 2.56923, 2.58043, 2.58603, 2.59722, 2.59722, 2.5
  2.58043, 2.56364, 2.55244, 2.53565, 2.53005, 2.51886, 2.50766, 2.50206, 2.49647, 2.50206, 2.5
```

```
2.51886, 2.52445, 2.52445, 2.51326, 2.50766, 2.50766, 2.50766, 2.50206, 2.50206, 2.50766, 2.5
2.51326, 2.51886, 2.52445, 2.52445, 2.52445, 2.53005, 2.54125, 2.55244, 2.56364, 2.55804, 2.5
2.54125, 2.53565, 2.52445, 2.51886, 2.51326, 2.50206, 2.49647, 2.48527, 2.47967, 2.46848, 2.4
2.43489, 2.42370, 2.42370, 2.4293, 2.44049, 2.45169, 2.46288, 2.47408, 2.49087, 2.51326, 2.53
2.55244, 2.56364, 2.57483, 2.58603, 2.59162, 2.59162, 2.58603, 2.57483, 2.56364, 2.55244, 2.5
2.52445, 2.51886, 2.50766, 2.49647, 2.49647, 2.50206, 2.51326, 2.51886, 2.52445, 2.51886, 2.5
2.50766, 2.50766, 2.50766, 2.50206, 2.50206, 2.50206, 2.50766, 2.51326, 2.51886, 2.51886, 2.5
2.52445, 2.53005, 2.54125, 2.55244, 2.56364, 2.55804, 2.55244, 2.54125, 2.53565, 2.52445, 2.5
2.51326, 2.50206, 2.49647, 2.48527, 2.47967, 2.46848, 2.45169, 2.44049, 2.42930, 2.42370, 2.4
2.44049, 2.45169, 2.46288, 2.47408, 2.49087, 2.51326, 2.53005, 2.54684, 2.56364, 2.57483, 2.5
2.59162, 2.59162, 2.58603, 2.57483, 2.56364, 2.54684, 2.53565, 2.52445, 2.51326, 2.50766, 2.4
2.49647, 2.50206, 2.50766, 2.51886, 2.52445, 2.51886, 2.51326, 2.50766, 2.50766, 2.50206, 2.5
2.50206, 2.50206, 2.50766, 2.50766, 2.51886, 2.51886, 2.51886, 2.51886, 2.52445, 2.53565, 2.5
2.55804, 2.55804, 2.55244, 2.54125, 2.53005, 2.52445, 2.51886, 2.51326, 2.50206, 2.49087, 2.4
2.47408, 2.46288, 2.45169, 2.43489, 2.42370, 2.41810, 2.42370, 2.43489, 2.45169, 2.46288, 2.4
2.49087, 2.50766, 2.53005, 2.54684, 2.55804, 2.56923, 2.58603, 2.58603, 2.58603, 2.58603, 2.5
2.55804, 2.54684, 2.53565, 2.52445, 2.51326, 2.50206, 2.49647, 2.49647, 2.50206, 2.50766, 2.5
2.52445, 2.51886, 2.51326, 2.50766, 2.50766, 2.50206, 2.50206, 2.50206, 2.50206, 2.50766, 2.5
2.51326, 2.51886, 2.51886, 2.51886, 2.53005, 2.53005, 2.55244, 2.55804, 2.55804, 2.55244, 2.5
2.53565, 2.52445, 2.51886, 2.50766, 2.50206, 2.49087, 2.48527, 2.47967, 2.46288, 2.45169, 2.4
2.42370, 2.42370, 2.42930, 2.44049, 2.45169, 2.46288, 2.47408, 2.49087, 2.51326, 2.53005, 2.5
2.55804, 2.56923, 2.58043, 2.58603, 2.58603, 2.58043, 2.56923, 2.55804, 2.54684, 2.53565, 2.5
2.51326, 2.50206, 2.49647, 2.49647, 2.49647, 2.50766, 2.51886, 2.51886, 2.51886, 2.51326, 2.5
2.50766, 2.50206, 2.49647, 2.49647, 2.50206, 2.50766, 2.50766, 2.51326, 2.51326, 2.51326, 2.5
2.52445, 2.53565, 2.54684, 2.55244, 2.55244, 2.54684, 2.54125, 2.53005, 2.52445, 2.51886, 2.5
2.50206, 2.49647, 2.48527, 2.47408, 2.46288, 2.45169, 2.43489, 2.42370, 2.41810, 2.42930, 2.4
2.45169, 2.45728, 2.47408, 2.48527, 2.50766, 2.52445, 2.54125, 2.55804, 2.56923, 2.58043, 2.5
2.58603, 2.58043, 2.56923, 2.55244, 2.54125, 2.53565, 2.52445, 2.51326, 2.50206, 2.49647, 2.4
2.50206, 2.50766, 2.51886, 2.51886, 2.51886, 2.51326, 2.50766, 2.50766, 2.50206, 2.49647, 2.4
2.50206, 2.50206, 2.50766, 2.51326, 2.51326, 2.51326, 2.51886, 2.52445, 2.53565, 2.54684, 2.5
2.55244, 2.54684, 2.53565, 2.53005, 2.52445, 2.51326, 2.50766, 2.50206, 2.49087, 2.48527, 2.4
2.46288, 2.44609, 2.43489, 2.42370, 2.41810, 2.42370, 2.43489, 2.44609, 2.45728, 2.46848, 2.4
2.50766, 2.52445, 2.54125, 2.55244, 2.56923, 2.57483, 2.58043, 2.58603, 2.58043, 2.56923, 2.5
2.54125, 2.53565, 2.52445, 2.51326, 2.50206, 2.49647, 2.49647, 2.50206, 2.51326, 2.51886, 2.5
2.51326, 2.51326, 2.50766, 2.50206, 2.50206, 2.49647, 2.49647, 2.49647, 2.50206, 2.50766, 2.5
2.51326, 2.51326, 2.51326, 2.52445, 2.53565, 2.54684, 2.55244, 2.54684, 2.54125, 2.53565, 2.5
2.52445, 2.51886, 2.50766, 2.50206, 2.49087, 2.48527, 2.47967, 2.46288, 2.45169, 2.43489, 2.4
2.42370, 2.42930, 2.43489, 2.44609, 2.45728, 2.46848, 2.48527, 2.50766, 2.52445, 2.54125, 2.5
2.56364, 2.57483, 2.58043, 2.58043, 2.57483, 2.56364, 2.55244, 2.54125, 2.53005, 2.51886, 2.5
2.49647, 2.49087, 2.49087, 2.49647, 2.50766, 2.51326, 2.51886, 2.51326, 2.51326, 2.50766, 2.5
2.50206, 2.49647, 2.49647, 2.49647, 2.50206, 2.50766, 2.51326, 2.50766, 2.51326, 2.51326, 2.5
2.53005, 2.54684, 2.54684, 2.54684, 2.54125, 2.53565, 2.52445, 2.51886]:
plots[listplot]( piano );
plots[listplot]( piano, style=POINT, symbol=CIRCLE );
```





The students chose the points 90 to 152 to represent one period. Does this look about right?

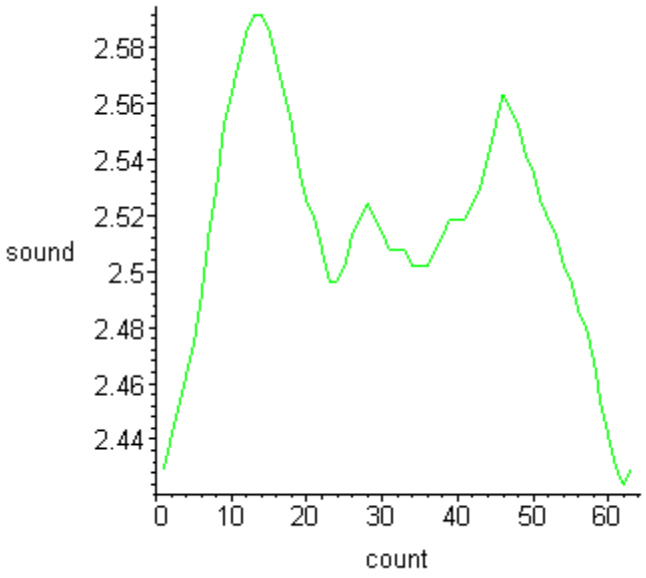
Note that this makes the period for the note C slightly longer than for the note A. Might this have anything to do with the notes themselves?

```
> initial := 90;
  final := 152;
  period := piano[initial..final]:
  lambda := nops( period );
  pianoplot := plots[listplot]( period, labels=["count", "sound"], color=green ):
  plots[display]( pianoplot );
```

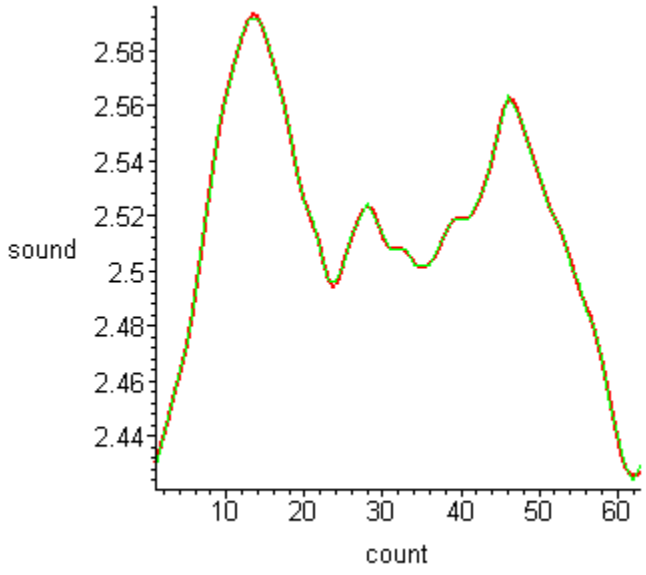
$$initial := 90$$

$$final := 152$$

$$\lambda := 63$$



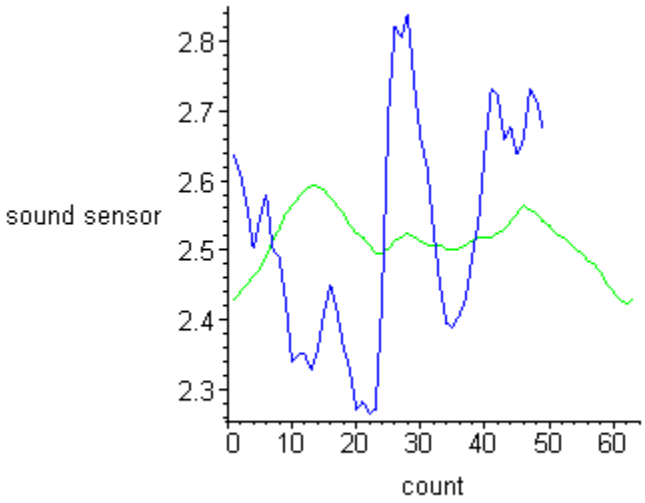

```
> forget( a );
  forget( b );
  pianofourplot := plot( 'musicfour'(t), t=1..lambda, color=red, thickness=2 ):
  plots[display]( [pianoplot, pianofourplot]  );
  printf( `The orignal plot is in green and the Fourier series plot is in red.` );
```



The orignal plot is in green and the Fourier series plot is in red.

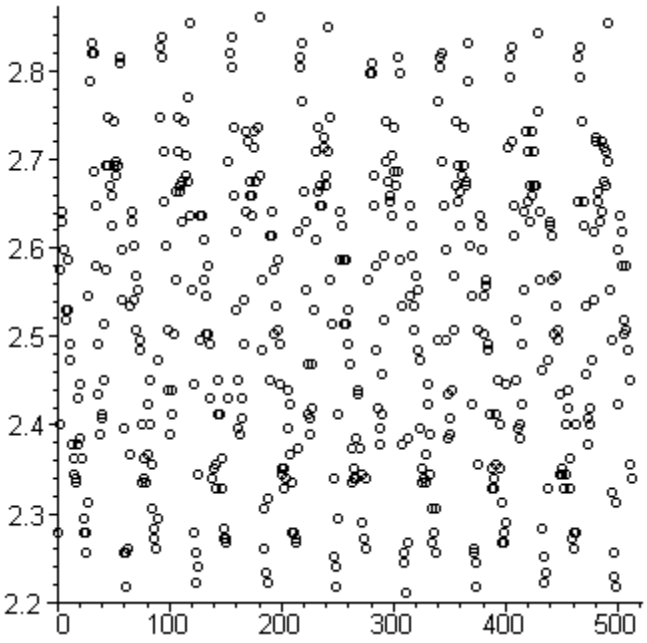
Compare the clarinet and the piano. The clarinet is in blue and the piano is in green in the following graph.

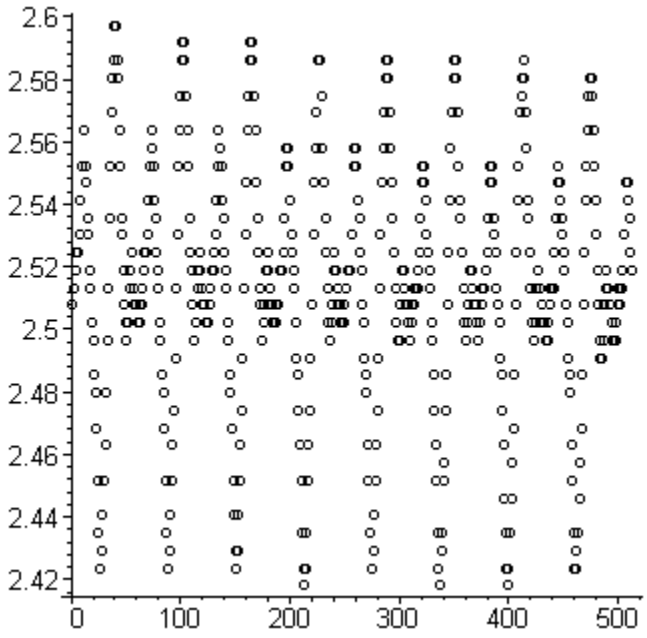
> **pianoplot := plots[listplot](period, labels=["count", "sound"],color=green):**
> **plots[display]([clarinetplot, pianoplot]);**



Why might the piano tone be smoother than the clarinet tone?

> **plots[listplot](clarinet, style=POINT, symbol=CIRCLE);**
> **plots[listplot](piano, style=POINT, symbol=CIRCLE);**


2.8
2.7
2.6
2.5
2.4
2.3
2.2
0
100
200
300
400
500


2.6
2.58
2.56
2.54
2.52
2.5
2.48
2.46
2.44
2.42
0
100
200
300
400
500

```
> ?

>
```