

Polynomial Approximation - I

Legendre vs Taylor Polynomials

Worksheet by Mike May, S.J., maymk@slu.edu

> **restart;**

Overview

One of the uses of the Gram Schmidt process is to produce orthogonal projections of a vector onto a subspace. This can also be thought of as finding the best approximation in the subspace under the given norm.

Finding Fourier series of functions is an example of this process. When we find a Fourier polynomial for a function, we are actually approximating a periodic function by the best fitting trig polynomials of specified degree. Best fitting is defined by a least sum of squares rule. To measure the distance between two functions we integrate the square of the difference of the two functions.

Thus $\langle f(x), g(x) \rangle = \int_{-1}^1 f(x) g(x) dx$. This is the easiest generalization of the usual way we do

"best fit" in the discrete case, with a sum of squares.

Today we want to look at approximating functions by polynomials for specified domains and norms. In terms used in the literature, we want to project a function onto the span of a set of orthogonal polynomials.

Maple has a standard package for orthogonal polynomials. We want to compare how effective these are when compared to Taylor polynomials.

>

Technical details

We start by loading the packages we need.

> **with(orthopoly); with(plots):**

[G, H, L, P, T, U]

(2.1)

Warning, the name changecoords has been redefined

For this worksheet we will work with Legendre polynomials. They are orthogonal in the space of continuous functions on $[-1, 1]$ with the usual integration inner product. Look at the help page for details.

> **?P**

Time for some technical commands. We want to assume that i and j are integers to simplify the integrations we need to do. We also don't want them printed with a trailing tilde.

> **assume('i',integer): assume('j',integer):
interface(showassumed=0):**

An extended example, approximating the function $f(x) = \sin(3\pi x)$

We will start with the function $\text{func}(x) = \sin(3\pi x)$.

```
> func := x -> sin(5*Pi*x);
```

$$\text{func} := x \rightarrow \sin(5 \pi x) \quad (3.1)$$

Computing Taylor and Legendre polynomials

We would like to start by computing coefficients. We will find the coefficients for both the Legendre polynomials and for Taylor polynomials. If $P(n,x)$ is the n th Legendre polynomial, the

projection coefficient of $f(x)$ onto $P(n,x)$ should be $\int_{-1}^1 f(x) P(n,x) dx$ divided by

$$\int_{-1}^1 P(n,x) P(n,x) dx .$$

```
> func := x -> sin(5*Pi*x);
normP[0] := evalf(Int((P(0,x))^2, x=-1..1)):
coeffP[0] := evalf(Int(P(0,x)*func(x)*1.0, x=-1..1)):
taylco[0] := evalf(subs(x=0,func(x))):
for i from 1 to 20 do
normP[i] := evalf(Int((P(i,x))^2, x=-1..1)):
coeffP[i] := evalf(Int(simplify(P(i,x)*func(x)*1.0), x=-1..1)):
taylco[i] := evalf(subs(x=0,diff(func(x),x$i))/i!):
od:
```

$$\text{func} := x \rightarrow \sin(5 \pi x) \quad (3.1.1)$$

Now we define procedures that define the approximating polynomials for specified degrees.

```
> Lapprox := proc(m,x)
local count:
sum(P(count,x)*coeffP[count]/normP[count], count=0..m):
end;
Tapprox := proc(m,x)
local count:
sum(x^count*taylco[count], count=0..m):
end;
```

$$\text{Lapprox} := \text{proc}(m, x) \quad (3.1.2)$$

```
local count;
sum((P(count,x)*coeffP[count])/normP[count], count=0..m)
end proc
Tapprox := proc(m,x)
local count;
sum(x^count*taylco[count], count=0..m)
end proc
```

Comparing the fit of Taylor and Legendre polynomials

Compare the 3rd degree approximations and note that they are substantially different..

```
> Lapprox(3,x);
Tapprox(3,x);
```

$$-0.4368278955 x + 1.046356378 x^3 \quad (3.2.1)$$

$$15.70796327 x - 645.9640976 x^3$$

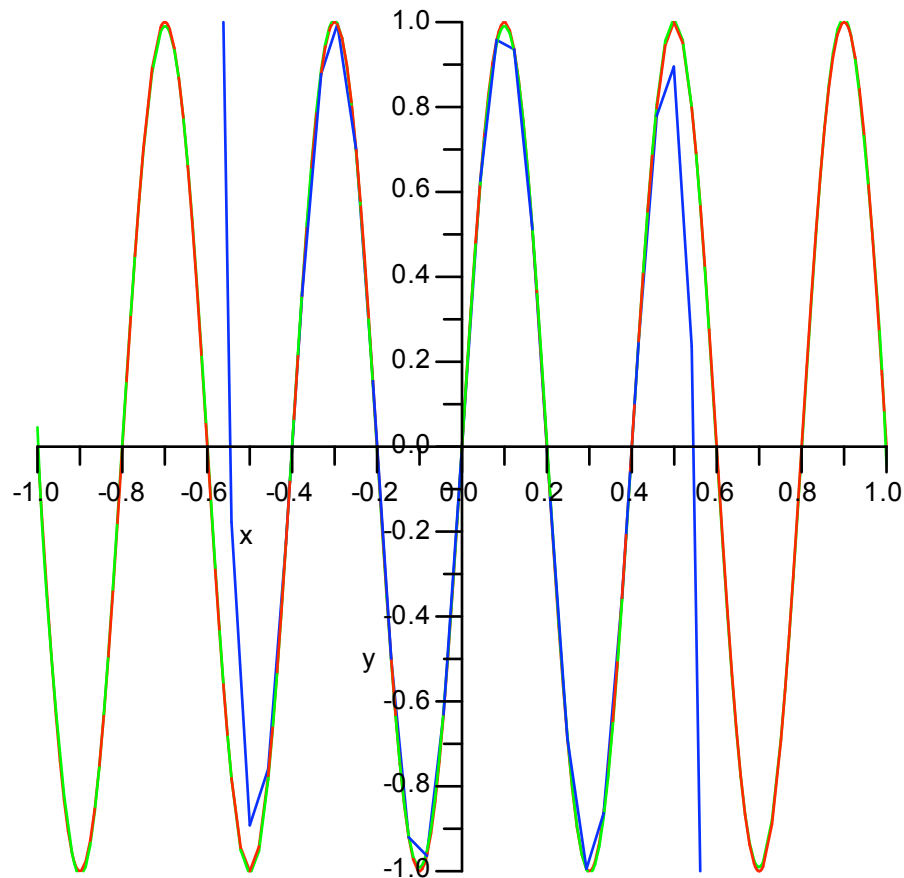
It is worth recalling that Taylor polynomials are "best fitting" in the sense of fitting very well at the point of expansion. Compare the graph of the function and the two approximations.

```
> n:=20:
```

```

p11 := plot(func(x), x=-1..1,y=-1..1, color=red):
p12 := plot(Lapprox(n,x), x=-1..1,y=-1..1, color=green):
p13 := plot(Tapprox(n,x), x=-1..1,y=-1..1, color=blue):
display({p11, p12, p13});
print("func with its 3rd degree Legendre and Taylor
approximations."):

```



(3.2.2)

"func with its 3rd degree Legendre and Taylor approximations."

Note that neither approximation seems very good at this point. We shouldn't be too surprised by the poorness of fit. The graph of the original function has 6 bumps, so we should expect a bad approximation until the degree is 7. It is worth noting that Taylor polynomials are very good close to the point of expansion, then get very bad. Legendre polynomials, on the other hand, never get too far off anywhere in the domain.

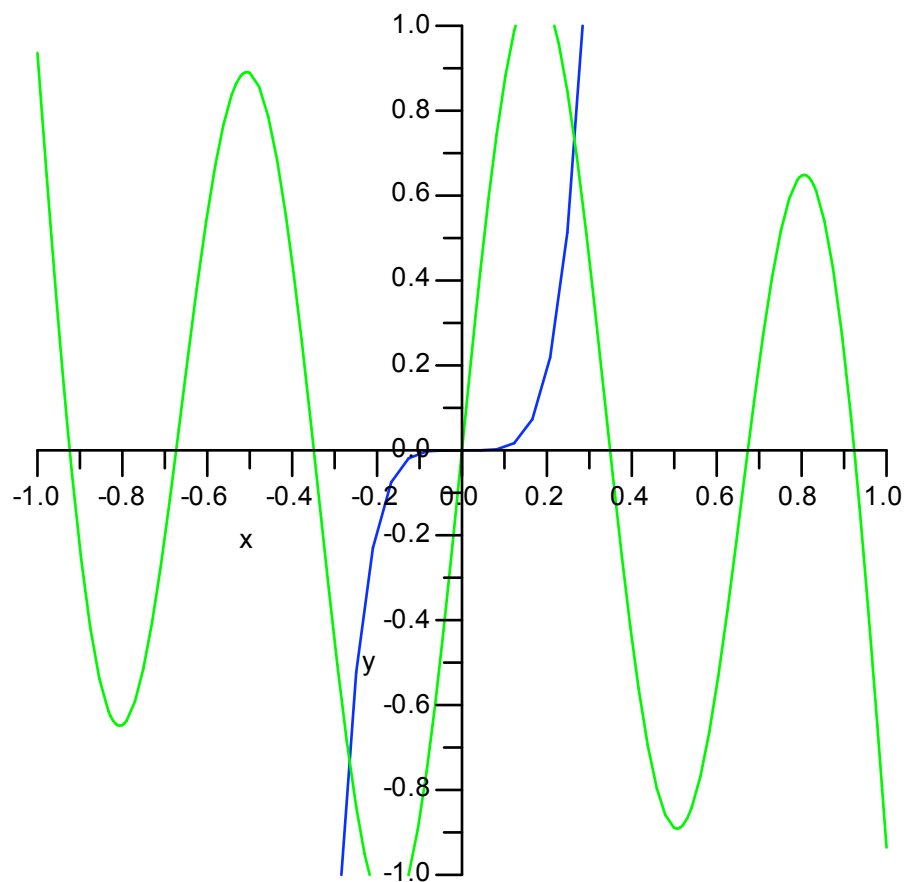
When looking at approximations, it is always good to look at graphs of the error between the approximation and the original function.

```

> err1 := plot(func(x) - Lapprox(n,x), x=-1..1,y=-1..1, color=
green):
err2 := plot(func(x) - Tapprox(n,x), x=-1..1,y=-1..1, color=
blue):
display({err1, err2});
print("The error in the 3rd degree Legendre and Taylor

```

approximations."):



(3.2.3)

"The error in the 3rd degree Legendre and Taylor approximations."

>

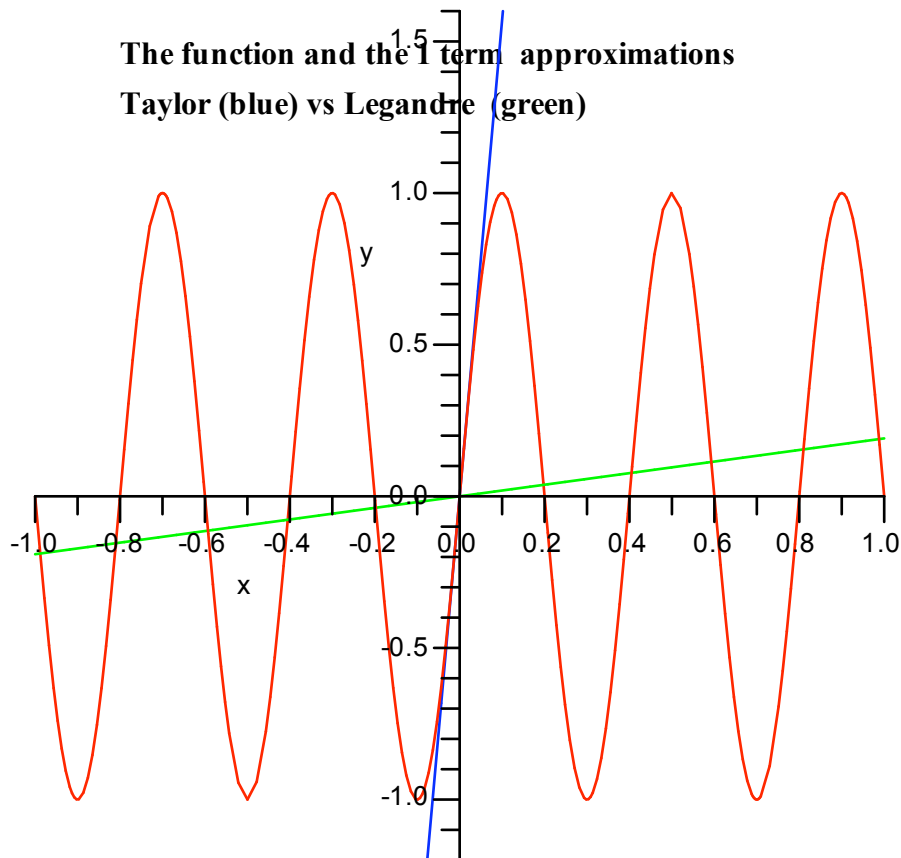
Animating the approximations and the errors

We can also compare the results with an animation. To prepare for the animation we create a procedure that will produce a single frame.

```
> framer := proc(n,x, lowy, highy)
  local p1, p2, p3, B, C, D:
  p1 := plot(func(x), x=-1..1,y=lowy..highy, color=red):
  p2 := plot(Lapprox(n,x), x=-1..1,y=lowy..highy, color=green)
  :
  p3 := plot(Tapprox(n,x), x=-1..1,y=lowy..highy, color=blue):
  B := textplot({[-0.8,highy-(highy-lowy)/20,
    `The function and the `|n||` term approximations`]
  ,
    [0-0.8,highy-(highy-lowy)/9, `Taylor (blue) vs Legendre
    (green)`]},
    align=RIGHT, font = [TIMES, BOLD, 12] ):
  display({p1, p2, p3, B},view=[-1..1, lowy..highy]);
end:
```

Using that procedure, it is easy to produce an animation of the approximations of degree 1 through 20.

```
> highy := 1.6: lowy := -1.2:  
display([seq(ramer(counta,x, lowy, highy), counta=1..20)],  
insequence = true);
```



If you step through the frames, it is noteworthy that the Legendre approximation becomes reasonable at degree 7, and that the graph of the function we are approximating has 6 bumps, and thus looks like a 7th degree polynomial.

In our example the Legendre approximation becomes very good when the degree is 13.

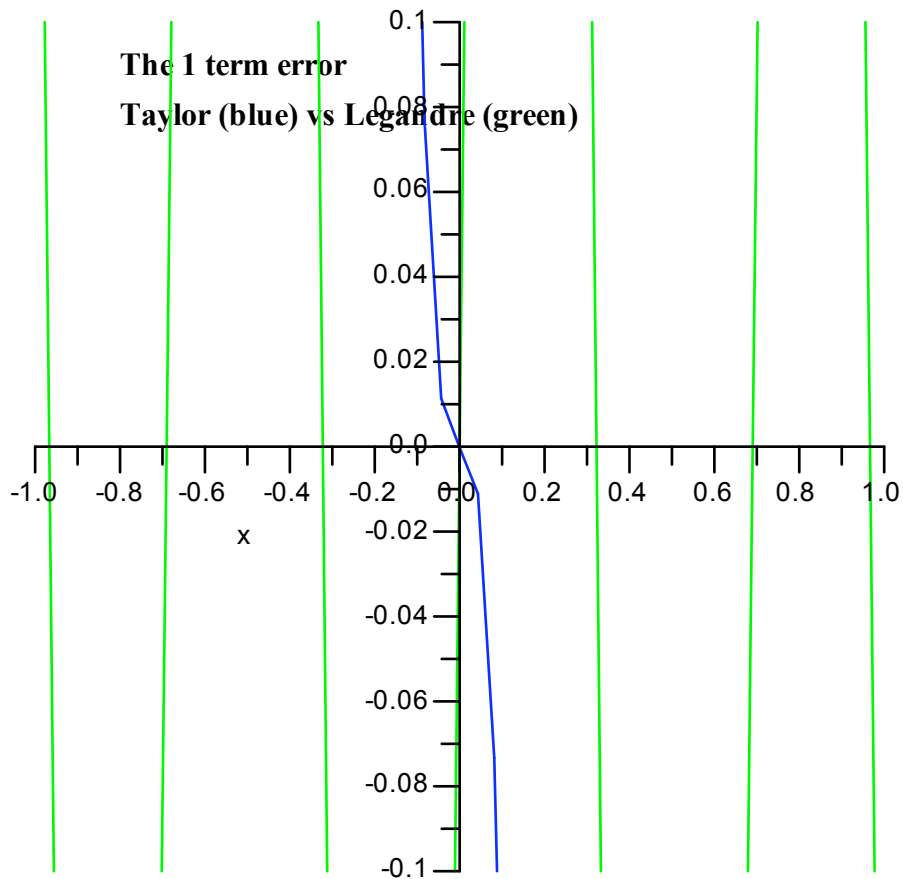
Since we are looking at approximations it is worthwhile to also look at the error between the approximations and the original function. Once again we start with a procedure for producing a single frame.

```
> ramererr := proc(n,x, lowy, highy)  
local pl1, pl2, B:  
pl1 := plot(func(x) - Lapprox(n,x), x=-1..1, color=green):  
pl2 := plot(func(x) - Tapprox(n,x), x=-1..1, color=blue):  
B := textplot({  
[-0.8,highy-(highy-lowy)/20, `The `||n||` term error`],  
[-0.8,highy-(highy-lowy)/9, `Taylor (blue) vs Legendre`]
```

```
(green)`}],
      align=RIGHT, font = [TIMES, BOLD, 12] ):
display({p11, p12, B}, view=[-1..1, lowy..highy]);
end:
```

We now look at the animation of the error terms. Initially we set the the window to see errors under .1.

```
> highy := .1: lowy := -.1:
display([seq(ramererr(counta,x, lowy, highy), counta=1..20)]
,
      insequence = true);
```



Exercises:

1) For our example function, what degree Taylor approximation do we need to be within 0.1 on the domain $[-0.5, 0.5]$? What degree Legendre approximation do we need to be within 0.1 on the domain $[-0.5, 0.5]$?

>

2) With the same function what is the maximum error of a 20th degree Taylor approximation on $[-1, 1]$?

>

3) With the same function, what degree approximation do we need to be within 0.1 on the domain $[-1, 1]$? What degree for an accuracy of 0.01 on the same domain? What degree for an accuracy of 0.001 on the same domain?

>

4) Repeat the process above for the function $\sin(2\pi x) + \cos(\pi x)$. What degree Legendre approximation gives an accuracy of 0.01 on the whole domain? (Be warned that you need to recompute the coefficients.)

>

>