Max Kramer

I affirm that I have adhered to the honor code on this assignment.

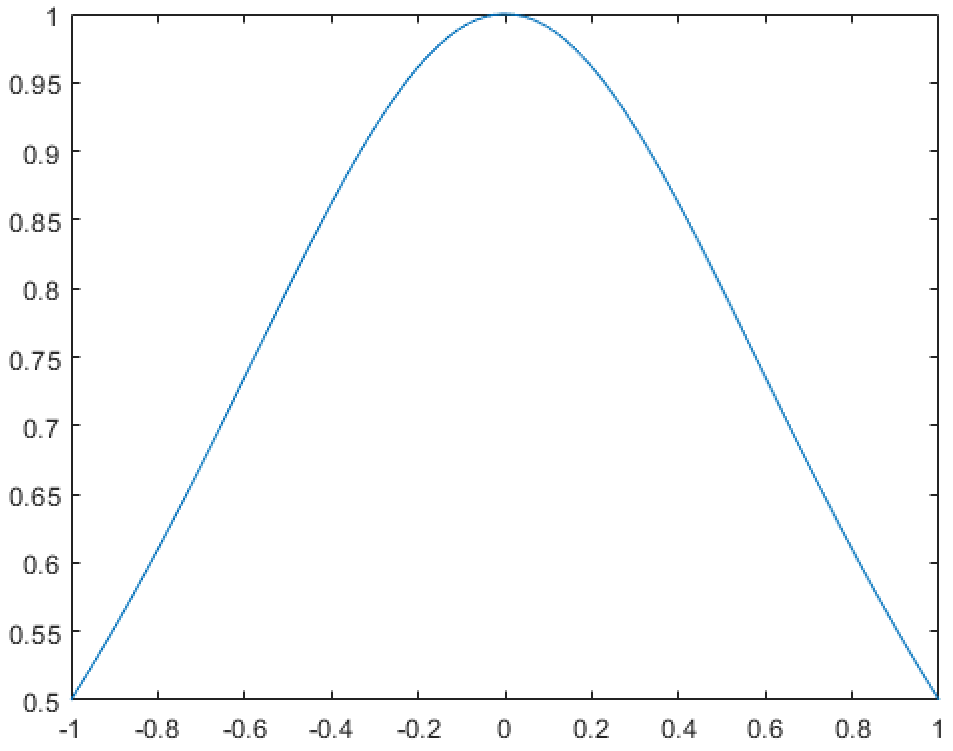
Hello again, scientist! I'll do all my writing in italics, and problems for you will be in **bold.** *Comment your code, and* explain your ideas in plaintext. *As a general rule, I expect you to do at least as much writing as I do. Code should be part of your solution, but I expect variables to be clear and explanation to involve complete sentences. Cite your sources; if you work with someone in the class on a problem, that's an extremely important source. Don't work alone.*

Problem F.10: Stop the ringing.

This problem assumes that you have read Lay 6.7, "Inner Product Spaces."

Hey, remember the Runge function?

```
syms x;
runge = 1/(1+x^2);
fplot(runge,[-1,1])
```



Way back in Week 5 you discovered that naive polynomial interpolation (that is, using the Vandermonde to fit a polynomial exactly) is a bad idea. It's prone to ringing, and it gets much worse at the edges. But that's because polynomial interpolation uses a really awkward inner product. Let's fix that. Here's the standard basis for P_2.

```
f0 = 1+0*x; f1 = x; f2 = x^2; % note: f0 is just the function 1, but
 MATLAB is confused by that
```

Do Gram-Schmidt by hand to find an orthonormal basis [ff0 ff1 ff2] for P_2 with respect to the L^2 inner product on [-1,1]. Show me the basis, correct to four decimal places, with vpa([ff0 ff1 ff2],4). *It'll go quickly once you know what you're doing. (The L^2 inner product (f,g) in MATLAB code is int(f*g,-1,1).)*

```
one = [1;1;1]; % evaluate from -1 to 1
ex = [-1;0;1]; % evaluate from -1 to 1
exsq = [1;0;1]; % evaluate from -1 to 1

orthotest = dot(one,ex) % 1 and x are orthogonal

p0 = one;
p1 = ex;

a = int(f2*f0,-1,1);
b = int(f0*f0,-1,1);
c = int(f2*f1,-1,1);

newf2 = f2 - (a/b)*p0 - (c * p1); % ff2 is x^2 - (1/3)

ff0 = f0;
ff1 = f1;
ff2 = x^2 - (1/3);

test1 = int(ff0*ff2,-1,1)
test2 = int(ff1*ff2,-1,1)

ff0 = ff0/sqrt(int(ff0*ff0,-1,1)); % normalize
ff1 = ff1/sqrt(int(ff1*ff1,-1,1)); % normalize
ff2 = ff2/sqrt(int(ff2*ff2,-1,1)); % normalize

basis = vpa([ff0 ff1 ff2],4)
```

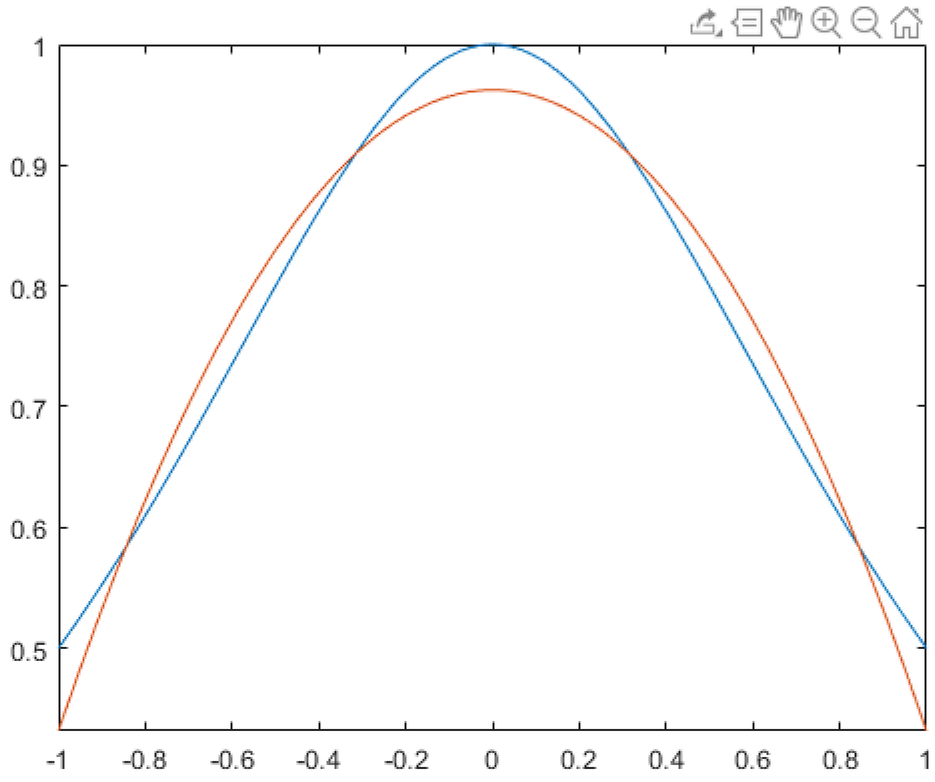
```
orthotest =

     0


test1 =

0


test2 =

0


basis =

[ 0.7071, 1.225*x, 2.372*x^2 - 0.7906]
```

To compute the Gram Schmidt with respect to the L^2 inner product, we evaluate the given basis at [-1 1], producing three vectors. The first and second vectors are already orthogonal, so we only need to apply Gram Schmidt to find ff2. We calculate the L^2 inner products between f2 and f0, f0 and f0, and f2 and f1 to project x^2 onto span{1,x}. We find the function X^2-1/3. The inner products of ff0 and ff2 and ff1 and ff2 demonstrate that ff0 through ff2 is an orthogonal basis for P^2. Finally, the basis must be normalized. Each basis vector is normalized by dividing itself by the square root of its inner product with itself. The resulting set {ff0 ff1 ff2} is an orthonormal basis for P^2 with respect to the L^2 inner product on [-1 1].

Compute F2, the projection of runge onto P_2 with respect to the L^2 inner product on [-1,1]. Show me F2, correct to four decimal places, with vpa(F2,4). Plot F2 and runge on the same set of axes. *Hint: F2 is approximately 1-0.5x^2.*

```
a2 = int(runge*ff0,-1,1) * ff0;
b2 = int(runge*ff1,-1,1) * ff1;
c2 = int(runge*ff2,-1,1) * ff2;

F2 = a2 + b2 + c2;
vpa(F2,4)

fplot(runge,[-1 1])
hold on
fplot(F2,[-1 1])
```

```
ans =

0.9624 - 0.531*x^2
```



In order to project runge onto P_2 with respect to the L^2 inner product on [-1 1] we use the orthonormal basis [ff0 ff1 ff2] calculated in the previous step. To project runge, we multiply the L^2 inner product on [-1 1] with each function against the function itself, resulting in three functions a2 b2 c2. These functions, when added together, produce the closest function to runge in P_2 with respect to the L^2 inner product on [-1 1], which in this case is 0.9624 - 0.531x^2, which is approximately 1-0.5x^2.

Not bad, right? Let's do better. Here's a (row) vector of the first 11 orthogonal basis polynomials for P on [-1,1] with respect to the L^2 inner product for [-1,1]; they're called the Legendre polynomials. There's a way to write them down quickly without doing Gram-Schmidt, but they really are exactly what you get if you apply Gram-Schmidt to the standard basis for P. Nothing spooky.

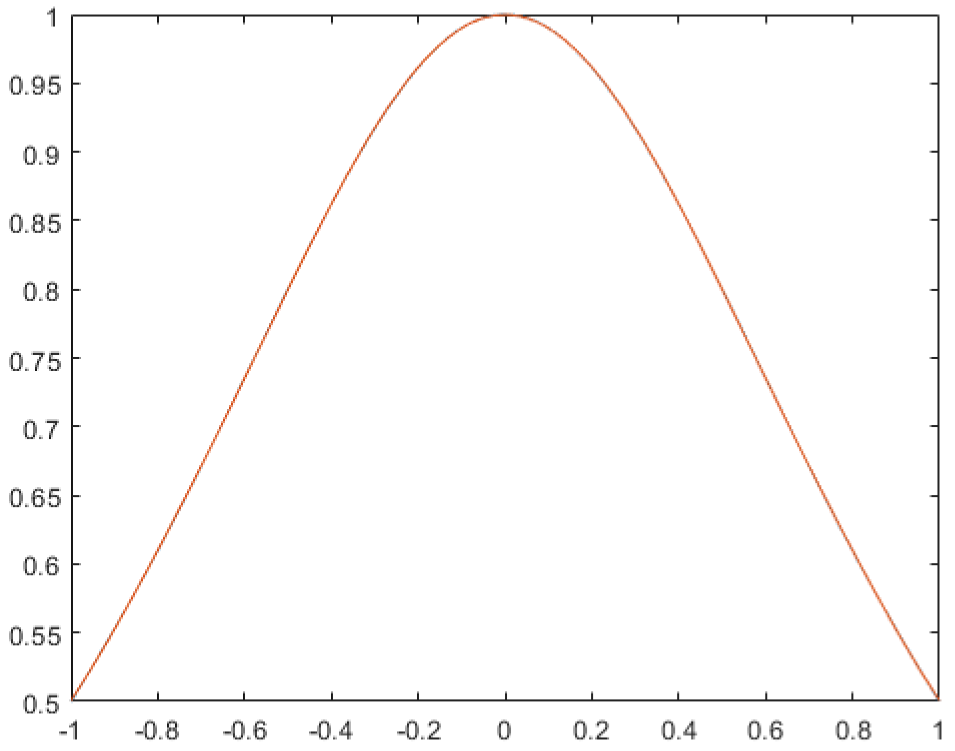
```
L = legendreP(0:10,x);
L = L./sqrt(int(L.^2,-1,1));
vpa(L(1:3),4) % see? same thing that you got by hand!
```

```
ans =

[ 0.7071, 1.225*x, 2.372*x^2 - 0.7906]
```

Compute F10, the projection of runge onto P_10 with respect to the L^2 inner product on [-1,1]. Show me F10, correct to four decimal places, with vpa(F10,4). Plot F10 and runge on the same set of axes. *Now you're cooking with gas. I don't want this one to be tedious: there's a way to do the projection in one line of code, no loops required. Think about what L is, and note that you can integrate all the terms of a vector at once using int(). Note that the coefficients on 1 and x^2 are different for F2 and F10. This isn't a* *mistake; the first three components of the orthogonal basis L are the same, but higher-order components of L continue to use lower-order polynomial terms.*

```
F10 = int(runge*L(1:10),-1,1) * L(1:10)';

vpa(F10,4)

hold off
fplot(runge,[-1 1])
hold on
fplot(F10,[-1 1])
```

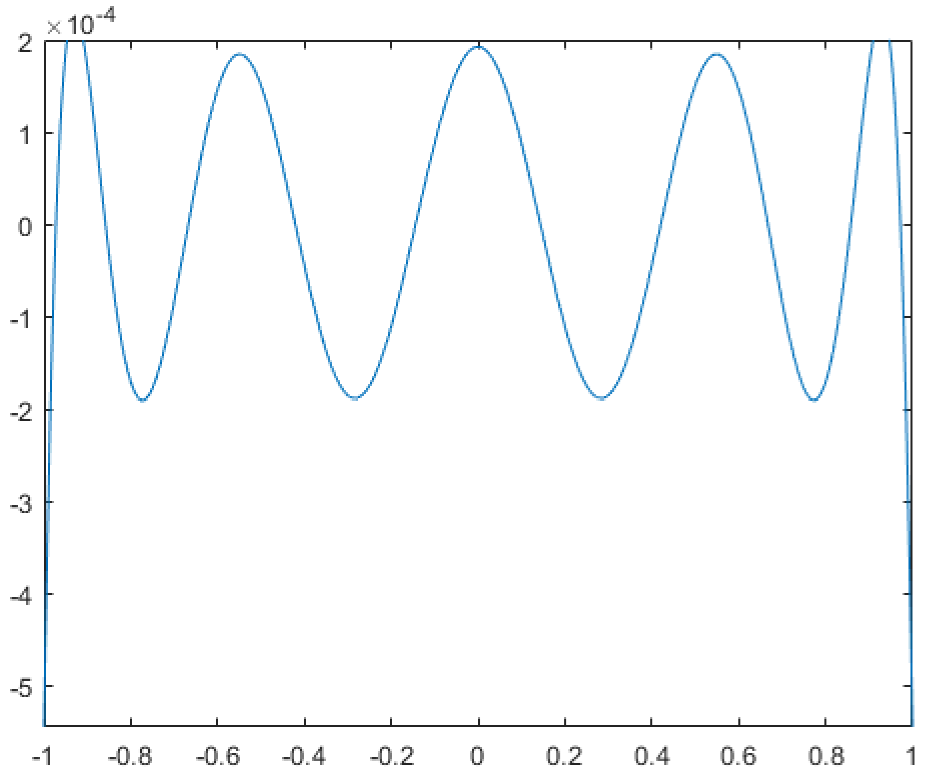
```
ans =

0.8856*conj(x)^4 - 0.9885*conj(x)^2 - 0.5668*conj(x)^6 +
 0.1705*conj(x)^8 + 0.9998
```



The projection F10 was calculated in one action by creating a vector of inner products between the runge function and the first 10 Legendre polynomials and then multioplying that against the same polynomials. The resulting function F10 lines up almost perfectly with runge when plotted on the same axes.

Here's a way to check that you've done this correctly, since the graphs of F and runge should line up almost exactly: let's look at the residual. ***Once you've done the previous part, uncomment the next line of code.*** *If your answer to the previous part is correct, the maximum value of this function will be about 0.0001. (You'll also be able to see the remaining ringing on this scale.)*

```
hold off; Fdiff = runge-F10; fplot(Fdiff,[-1,1])
```



The maximum value of the residual graph is 0.0001925.

For those of you coming from Calc II who think "hey, this is dumb, Taylor series can do that!" I invite you to plot the first 200 terms of the Taylor series of the Runge function and compare your results.

```
%rungetaylor = taylor(runge,x,'Order',200)
```

Published with MATLAB® R2019b