# MATH 2270−3
# MAPLE PROJECT 2b − Inner product spaces
November 2, 2009

We will explore two interesting inner product spaces. Refer to section 5.5 of the text. This part of your project replaces homework problems from that section.

Even though you thought you were only doing classical geometry with the dot product in $R^n$, it turns out we only needed three algebraic properties (symmetry, linearity in each factor, positivity) of the dot product in order to talk about distance, orthogonality, projection, Pythagorean Theorem, etc. If we have an operator <f,g> which inputs any two elements f, g from a linear space *V*, and outputs real scalars, then this operator is called an "**inner product**" if it satisfies the four axioms of Definition 5.5.1, page 234.

## Example 1: A good dot product and interval for orthogonal polynomials

The first inner product and interval we mention are used in numerical analysis algorithms. We'll call the inner product ''dot1''. It is the inner product of example 7, page 238.

```
> restart:with(plots):
```

```
> dot1:=(f,g)->int(f(t)*g(t),t=-1..1);
     #our first inner product, for the
     #interval -1<t<1
```

```
> dot1(t->t,t->1);
     #the dot product of f(t)=t with g(t)=1
     #you should get zero.  Meaning what?
```

Meaning you just showed that f(t)=t and g(t)=1 are orthogonal for the dot1 inner product! Does this surprise you? (Think of the graph of these two functions for −1<t<1 and think about what the integral of f(t)*g(t) is measuring.) In fact, once the inner product is defined, you can define magnitude, distance, angle between vectors, and you can even do projection problems. This is discussed in detail in section 5.5.

```
> mag1:=f->sqrt(dot1(f,f)):
     # the magnitude of a vector
```

```
> dist1:=(f,g)->mag1(f-g):
     # the ''distance'' between two functions
```

```
> cosangle1:=(f,g)->(dot1(f,g)/(mag1(f)*mag1(g))):
     #computes the "cos of the angle" between functions
```

```
> fangle1:=(f,g)->evalf(arccos(cosangle1(f,g))):
     #computes angle between functions
```

For example we may use Gram−Schmidt to find an orthonormal basis for the polynomial subspace P2= span{1, t, t^2}, for our inner product.

```
> P0:=t->1;
  P1:=t->t;
  P2:=t->t^2;
     #our usual "natural" basis
> U0:=P0/mag1(P0):   #first orthonormal vector
  Z1:=P1-dot1(P1,U0)*U0:
     #P1 was orthogonal to P0, so don't really
     #need the usual projection formula, but
     #here it is anyway.
  U1:=Z1/mag1(Z1):
     #second orthonormal vector
  Z2:=P2-dot1(P2,U0)*U0-dot1(P2,U1)*U1:
  U2:=Z2/mag1(Z2):
     #third orthonormal vector
> U0(t);U1(t);U2(t);
     #the orthonormal polynomials we get
> plot({U0(t),U1(t),U2(t)},t=-1..1, title='orthonormal basis
  for P2, using dot1');
```

## Projection:

Now that we have an orthonormal basis for P2 we can do projection problems. We will try to find the closest degree 2 polynomial to f(t)=exp(t), using our inner product dot1 to measure distance:

```
> proj2:=f->evalf(dot1(f,U0)*U0+dot1(f,U1)*U1+dot1(f,U2)*U2);
     #this is the usual projection formula, but we use evalf to
     #get decimals rather than messy algebraic numbers.
> f:=t->exp(t);
> g:=t->proj2(f)(t);
```

This polynomial g is "closer" to exp(t) than the usual Taylor polynomial p(t)=1+t+t^2/2, when we use the distance which we get from dot1. We can compare distances:

```
> p:=t->1+t+t^2/2;
> evalf(dist1(f,g));
  evalf(dist1(f,p));
```

So, at least for our distance, the function g does about three times as well as the Taylor polynomial. You can maybe see this geometrically, by plotting the three graphs on the interval from −1 to 1:

```
> with(plots):
> actual:=plot(f(t),t=-1..1,color=red):
  best:=plot(g(t),t=-1..1,color=black):
  tayl:=plot(p(t),t=-1..1,color=blue):
  display({actual,best,tayl});
```

The orthogonal polynomials which you have been constructing are called the Legendre polynomials. Maple knows about them. To see the first few you can load the 'orthopoly' package. By the way, if you define different weighted inner products you get different (famous to experts) orthogonal polynomial families, you can read about some of them on the help windows, starting at 'orthopoly'. Orthogonal polynomials are used in approximation problems, as you might expect.

```
> with(orthopoly):
> P(0,x);P(1,x);P(2,x);P(3,x);P(4,x);P(5,x);
     #these should look familiar, after what you just did!
     #they haven't been normalized, though.
```

## Example 2: Fourier Series

Here's an interval and dot product which makes the usual trig functions into an orthonormal family! See page 239 of the text!

```
> dot2:=(f,g)->1/Pi*int(f(t)*g(t),t=-Pi..Pi);
```

For example:

```
> f:=t->sin(5*t);
  g:=t->cos(3*t);
  dot2(f,f);
  dot2(g,g);
  dot2(f,g);
```

We add the constant function to our collection, but it will need to be normalized:

```
> dot2(1,1);
  dot2(f,1);
  dot2(g,1);
    #the constant function has norm squared
    #equal to 2, however.  But it is orthogonal
    #to all cos(kt),sin(kt), k a natural number
```

You can show!!! that for any n, the family {1/sqrt(2), cos(t), cos(2t), ..., cos(nt), sin(t), sin(2t), ...sin(nt)} is an orthonormal basis of a 2n+1 dimensional subspace of functions. So it is easy to project onto this subspace using our usual projection formulas. It is an amazing fact that if f(t) is any piecewise continuous function on the interval −Pi<=t<=Pi, then as as n approaches infinity the distances between these projections and f converges to zero.

Here's an example (Example 8 page 242):

```
> f:=t->t:
    #the function we shall decompose into trigonometric pieces

> a0:=1/Pi*int(f(t),t=-Pi..Pi);
    #could you have predicted the answer?
> #now get projection coefficients
  #AKA Fourier coefficients
  for i from 1 to 10 do
  g1:=t->cos(i*t):
  g2:=t->sin(i*t):
  a[i]:=dot2(g1,f):
  b[i]:=dot2(g2,f):
  od:

> fsum:=t-> a0/2+ sum(a[j]*cos(j*t),j=1..10) +
       sum(b[j]*sin(j*t),j=1..10);
> fsum(t);
    #why are there no cosine terms?
> plot1:=plot(fsum(t),t=-Pi..Pi,color=black):
  plot2:=plot(t,t=-Pi..Pi,color=red):
  display({plot1,plot2});
```

There are problems for you to do!  Download the file 2270proj2b.mw from the project 2 web page.