Exercises for Numerical Analysis, April 1st, 2005

1. Implement a Matlab function, `Simpson1` that will compute the integral using Simpson's Rule. The start of you function will be:

```
function y=Simpson1(f,a,b,n)
%FUNCTION y=Simpson1(f,a,b,n)
%  Computes the integral of f from a to b using n subintervals
%    and the Composite Simpson's Rule.  n should be an odd number.

if rem(n,2)==0
  error('n should be an odd number');
end
```

   HINT: See Matlab Note 1 on the next page. NOTE: n was even in class because we actually had $n+1$ points. Now we have n points, so now n needs to be odd.

   Come up with some test integrals to be sure that your algorithm is correct. (Hint: Simpson's Rule should be exact for what functions?)

2. Look up the recurrence relation for the Legendre Polynomials. Use this to write a Matlab function which has the following header:

```
function y=LegendrePoly(x,n)
%  Evaluates the nth Legendre Polynomial at the values given in
%  the vector x.  Output the result as y.
% RESTRICTIONS THAT ARE NOT CHECKED:
%     **The degree n should be bigger than 0 (1 or more)
%     **The vector x should only have values between -1 and 1
```

   HINT: You will need to run a loop, `for j=1:n`

   You will need this function before continuing. Check to be sure that it is correct for degrees 1 and 2 before going on.

3. Show (via examples) that in general, discretizing the Legendre Polynomials does not result in orthogonal vectors.

4. We will use the following approximation to the zeros of the Legendre polynomial. For the n^{th} degree polynomial, the n zeros are approximately:

$$x_i = \cos\left(\pi \frac{(i-0.25)}{(n+0.5)}\right), \quad i = 1, 2, \ldots, n$$

   In Matlab, `x=cos( pi*((1:n)-0.25)/(n+0.5) );` Show graphically that these are excellent approximations by plotting the 7th degree and 10th degree Legendre polynomials with these approximations. Show that these are not the exact roots by computing the Legendre polynomials of degrees 7 and 10 at these values.

5. Write up the Matlab function `LegendreQuad`, as attached. This will, given inputs a, b and n, output the Legendre quadrature points x_i and associated weights w_i for the approximation:

$$\int_a^b f(x)\,dx = \sum_{j=1}^{n} w_j f(x_j)$$

(Note that we're counting from 1 instead of zero).

6. Show that if we define the weighted inner product of two vectors as:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} w_i x_i y_i$$

then our discretized Legendre Polynomials form an orthogonal basis of $\mathbb{R}^n$. Thus, we maintained orthogonality in our move from $C[a, b]$ to $\mathbb{R}^n$.

7. Compare the values of

$$\int_1^1 \mathrm{e}^{-x^2}\,dx$$

using Simpson's Rule with n points versus using the n Legendre quadrature points.

Matlab Notes

1. We said in class that there are times when we would like to pass a function into a Matlab function. For example, Newton's Method would need to have $f(x)$ and $f'(x)$ and an initial point, x_0. Below is a "toy script" that doesn't do anything very useful, but does show you how to use functions within a Matlab function:

```
function x=NewtonStep(f,df,x0)
%x=NewtonMethod(f,df,x0)

x=x0-f(x0)/df(x0);
```

   Here is what we would write in the Command Window to call this function. Suppose I want to apply one step of Newton's Method to $f(x) = x^2 - 1$ with $x_0 = 3$. Then we would type:

```
>> f = inline('x.^2 - 1');
>> df= inline('2*x');
>> x = NewtonStep(f,df,3);
```

   (x should be returned as 1.666...)