Problem 10.5: The Chebyshev polynomials $T_j(x) = cos(jarccos(x))$, $j = 0, 1, \ldots,$ are orthogonal with respect to the weight function $(1-x^2)^{-1/2}$ on $[-1, 1]$; that is $\int_{-1}^{1} T_j(x)T_k(x)(1-x^2)^{-1/2}dx = 0$ if $j \neq k$. Use the Gram-Schmidt process applied to the linearly independent set $\{1, x, x^2\}$ to construct the first three orthonormal polynomials for this weight function and show that they are indeed scalar multiples of T_0, T_1, T_2

Solution:
First let $w = (1 - x^2)^{-1/2}$. Then using the Gram-Schmidt process to construct q_0, q_1, q_2 w-orthogonal polynomials on the set $\{1, x, x^2\}$. Let $q_0 = 1$, then

$$\begin{aligned} q_1(x) &= x - \frac{< x, 1 >_w}{< 1, 1 >_w} 1 \\ &= x - \frac{\int_{-1}^{1} x(1 - x^2)^{-1/2}dx}{\int_{-1}^{1} x(1 - x^2)^{-1/2}dx} \\ &= x - \frac{0}{\pi} = x. \end{aligned}$$

Solving for q_2 that is orthogonal to q_0, q_1,

$$\begin{aligned} q_2(x) &= x^2 - \frac{< x^2, 1 >_w}{< 1, 1 >_w} 1 - \frac{< x^2, x >_w}{< x, x >_w} x, \\ &= x^2 - \frac{\int_{-1}^{1} x^2(1 - x^2)^{-1/2}dx}{\int_{-1}^{1} x(1 - x^2)^{-1/2}dx} - \frac{\int_{-1}^{1} x^3(1 - x^2)^{-1/2}dx}{\int_{-1}^{1} x^2(1 - x^2)^{-1/2}dx} x, \\ &= x^2 - \frac{(\pi/2)}{\pi} - \frac{0}{\pi} x, \\ &= x^2 - \frac{(\pi/2)}{\pi}. \end{aligned}$$

Finally normalizing each polynomial we get the following,

$$\hat{q}_0 = \frac{1}{< 1, 1 >} = \frac{1}{\sqrt{(\int_{-1}^{1} 1dx)}} = \frac{1}{\sqrt{2}},$$

$$\hat{q}_1 = \frac{x}{< x, x >} = \frac{1}{\sqrt{(\int_{-1}^{1} x^2dx)}} = \frac{\sqrt{3}x}{\sqrt{2}},$$

$$\hat{q}_2 = \frac{x^2 - \frac{(\pi/2)}{\pi}}{< q_2, q_2 >} = \frac{x^2 - \frac{(\pi/2)}{\pi}}{\sqrt{(\int_{-1}^{1} (x^2 - \frac{(\pi/2)}{\pi})^2 dx)}} = \frac{\sqrt{30}(x^2 - \frac{(\pi/2)}{\pi})}{\sqrt{7}}.$$

Calculating the first three Chebyshev Polynomials.

$$T_0 = cos(0arccos(x)) = 1,$$

$$T_1 = cos(arccos(x)) = x.$$

Using the double angle identity, and pythagorean theorem,

$$\begin{aligned} T_2 &= cos(2arccos(x)), \\ &= cos^2(arccos(x)) - sin^2(arccos(x)), \\ &= x^2 - (1 - x^2), \\ &= 2x^2 - 1. \end{aligned}$$

Now we can see that we get the following scalar multiples,

$$\begin{aligned} T_0 &= \frac{1}{\sqrt{2}}\hat{q}_0, \\ T_1 &= \frac{\sqrt{2}}{\sqrt{2}}\hat{q}_1, \\ T_2 &= \frac{\sqrt{30}}{2\sqrt{7}}\hat{q}_2. \end{aligned}$$

Problem 10.7: Write a MATLAB code to approximate,

$$\int_0^1 cos(x^2)dx$$

Using composite trapezoid rule and one to approximate the integral using the composite Simpson's rule, with equally-spaced, nodes. The number of intervals $n = 1/h$ should be and input to each code.
Do a convergence study to verify the second-order accuracy of the composite trapezoid rule and the fourth-order accuracy of the composite Simpson's Rule;

Solution:
Function:

```
function [X] = Trap(f,a,b,n)
%This function takes a function f, an interval a,b
%and a number of intervals n and returns the approximate
%integral using the composite trapezoid rule

x = linspace(a,b,n+1);
X = 0;
for i = 1:n
    X = X + (x(i+1) - x(i))*((f(x(i+1)) + f(x(i)))/2);
end
end
```

Function:

```
function [X] = Simpsons(f,a,b,n)
%This function takes a function f, an interval a,b
%and a number of intervals n and returns the approximate
%integral using the composite Simpsons rule

x = linspace(a,b,n+1);
X = 0;
for i = 1:n
    X = X + ((x(i+1) - x(i))/6) *
    (f(x(i)) + 4*f(((x(i+1) + x(i))/2)) + f(x(i+1)));
end
end
```

Console:

```
f =

  function_handle with value:

    @(x)cos(x.^2)

>> n = [1,10,100,1000,10000,100000];

>> T_h = []
    for i = 1: 6
        int = Trap(f,0,1,n(i));
        T_h = [T_h,int];
    end

>> q = quad(f,0,1,[1.e-12 1.e-12]);
>> E_h = abs(T_h' - q)'
>> ratio = E_h.*n.^2
>> table(n',T_h',E_h',ratio',
    'VariableNames',{'1\h','T_h','E_h','E_h\h^2'})
```

$1/h$	T_h	E_h	E_h/h^2
1	0.77015	0.13437	0.13437
10	0.90312	0.0014025	0.14025
100	0.90451	1.4025e-05	0.14025
1000	0.90452	1.4025e-07	0.14025
10000	0.90452	1.4025e-09	0.14025
100000	0.90452	1.4039e-11	0.14039

Console:

```
f =

  function_handle with value:

    @(x)cos(x.^2)

>> n = [2, 4, 8, 16, 32, 64, 128, 256];

>> T_h = []
    for i = 1: 8
        int = Simpson(f,0,1,n(i));
        T_h = [T_h,int];
    end

>> q = quad(f,0,1,[1.e-12 1.e-12]);
>> E_h = abs(T_h' - q)'
>> ratio = E_h.*n.^4
>> table(n',T_h',E_h',ratio',
    'VariableNames',{'1\h','T_h','E_h','E_h\h^2'})
```

$1/h$	T_h	E_h	E_h/h^4
2	0.9045	2.2972e-05	0.00036755
4	0.90452	7.8693e-08	2.0145e-05
8	0.90452	1.4667e-08	6.0077e-05
16	0.90452	1.2157e-09	7.967e-05
32	0.90452	8.0618e-11	8.4534e-05
64	0.90452	5.1057e-12	8.5659e-05
128	0.90452	3.1475e-13	8.449e-05
256	0.90452	1.41e-14	6.0558e-05

Supplimental 1: Continuing with the theme that some sample points are better than others, recall that polynomial interpolation with high-order polynomials is prone to making large oscillation errors, but that this can be minimized using Chebyshev polynomials, which are the Lagrange polynomials associated with the sample points

$$x_j = \cos(\pi + (\pi j/n)), \qquad j = 0, \ldots, n$$

on the interval $[-1, 1]$. Clenshaw-Curtis integration is integration using polynomial interpolation at these sample points.

Use the MATLAB `polyfit` function to perform polynomial interpolation at these sample points for $n = 4, 6, 10$ and then use the resulting polynomials to approximate

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

Compare your approximations to the exact answer (which you should compute by hand). Integration by parts!

Solution:
Computing the value of the integral by hand using integration by parts,

$$\begin{aligned}\int_{-1}^{1} x \sin(x)\, dx &= -x\cos(x)|_{-1}^{1} - \int_{-1}^{1} -\cos(x)\, dx,\\ &= -x\cos(x) - (-\sin(x))|_{-1}^{1},\\ &= -x\cos(x) + \sin(x)|_{-1}^{1},\\ &= -2\cos(1) + 2\sin(1).\end{aligned}$$

Using the following MATLAB function to approximate the integral, with a Chebyshev polynomial interpolant of degree 4,6, and 10.
Function:

```
function [inter] = cheby_inter(f,n)
%This function takes in a function f and
%the desired order for the approximating
%polynomial interpolant n. Returns the integral
%of the polynomial interpolant from -1,1

x = [];
    for i = 0:n
        x = [x, cos(pi+((pi*i)/n))];
    end
p = polyfit(x, f(x),n);
i = polyint(p);
inter = diff(polyval(i,[-1 1]));
end
```

Console:

```
>> f = @(x) x.*sin(x)

f =
```

```
   function_handle with value:

    @(x)x.*sin(x)

>> x = -2*cos(1)+2*sin(1)

x =

    0.602337357879513

>>  c_4 = cheby_inter(f,4);
    c_6 = cheby_inter(f,6);
    c_10 = cheby_inter(f,10);

>> c = [c_4 c_6 c_10];
>> error = abs(c - x)'

error =

    1.0e-03 *

    0.154362643490780
    0.000148674157208
    0.000000000211609
```

Supplimental 2: Recall that 5 point Gauss-Legendre integration uses sample points $[-\beta, -\alpha, 0, \alpha, \beta]$ where

$$\alpha = \frac{1}{3}\sqrt{5 - 2\sqrt{\frac{10}{7}}}$$

$$\beta = \frac{1}{3}\sqrt{5 + 2\sqrt{\frac{10}{7}}}.$$

Write a code that performs composite Gauss-Legendre integration with these sample points. Your code should have the signature

```
function q=glquad(f,a,b,N)
...
end
```

were f is the function to integrate, a and b are the endpoints of integration, and N is the

number of subintervals. Your code should perform Gauss-Legendre integration on each subinterval and add them up. Then apply your function to compute

$$\int_{-1}^{1} x\sin(x)\,dx$$

using $N = 1, 2, 4, 10$. Compare the results of Gauss-Legendre integration to the results you saw using Clenshaw-Curtis integration.

Solution:

Function:

```
function [Q] = glquad(f,a,b,n)
% This fuction takes a function f, an interval [a,b]
% and a number of intervals n and returns the composite
% gauss-legendre integration.

% Pulling the roots for a 5th degree legendre polynomial
syms x
roots = vpasolve(legendreP(5,x) == 0);

% intitilizing the endpoints of each sub-interval
ep = linspace(a,b,n+1);
%Casting the roots as floats instead of a symbolic variable
roots = cast(roots,'like',ep);

% Finding the normalized roots on each sub interval.
% Each column of X is the set of roots on the ith sub-interval
X = [];
for i = 1:n
    legSub = roots*((ep(i+1) - ep(i))/2) + ((ep(i+1) + ep(i)))/2;
    X = [X legSub;];
end


% Performing guassian quadrature on each subinterval.
Q = 0;
for i = 1:n
    x = X(:,i)';

    B = [x.^0; x; x.^2; x.^3; x.^4];
    bb = [];
    A = [];
```

```
    for j = 1:5
        bb(j) = ((ep(i+1))^j - (ep(i))^j)/j;
    end

    A = B\bb';
    Q = Q + dot(A,f(x));
end


end
```

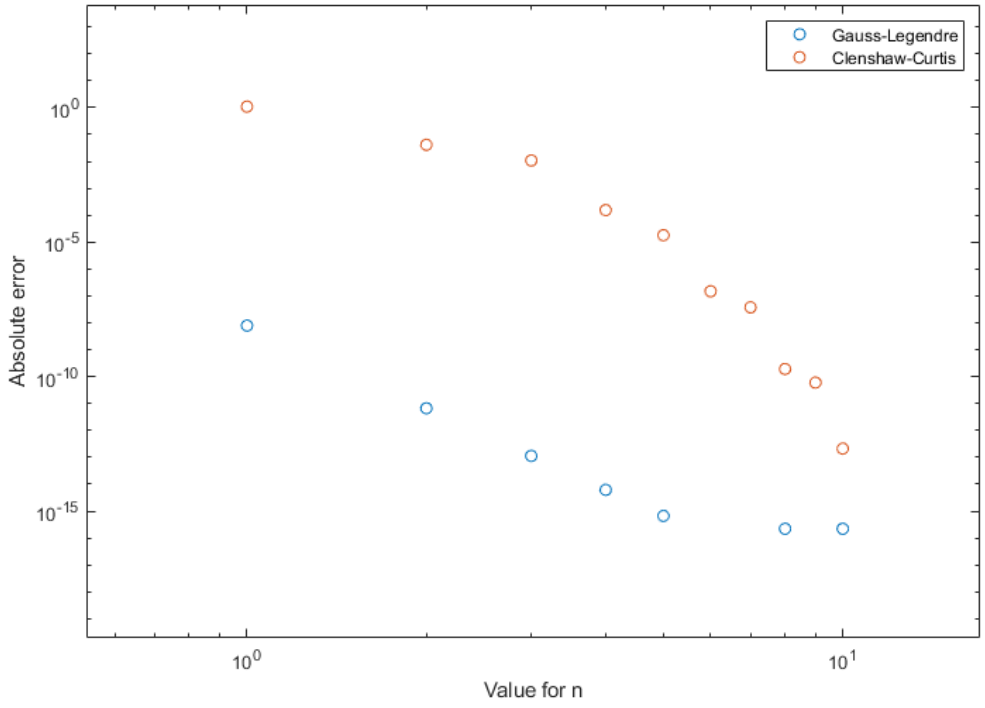
Console:

```
f =

    @(x)x.*sin(x)

>> gl_1 = glquad(f,-1,1,1)
>> gl_2 = glquad(f,-1,1,2)
>> gl_4 = glquad(f,-1,1,4)
>> gl_10 = glquad(f,-1,1,10)
>> gl = [gl_1 gl_2 gl_4 gl_10]'

gl =

   0.602337349998268
   0.602337357872820
   0.602337357879507
   0.602337357879514

>> x = -2*cos(1)+2*sin(1)

   0.602337357879513

>> error = abs(gl - x)

error =

   1.0e-08 *

   0.788124521200473
   0.000669309052626
```

0.000000621724894
0.000000022204460

Clearly we are seeing faster convergence using the Gauss-Legendre integration. Doing an approximation for $n = 1, 2, \ldots, 10$ and plotting the absolute errors against our value of n we can see that Gauss-Legendre integration reaches higher precision at lower values of n.

Figure 1: Integration Technique Comparison



Problem 9.1: Use MATLAB to evaluate the second-order-accurate approximation to $f''(x)$ for $f(x) = sin(x)$ and $x = \pi/6$. Try $h = 10^{-1}, 10^{-2}, \ldots, 10^{-16}$ and make a table of values of h, the computed finite difference quotient, and the error. Explain your results.

Solution:

Function:

```
function [X] = diff_2(f, h, x)
% This function takes a function f,
% a vector of values for h, an x value of
% interest and returns an approximation for the
% second derivative at x.

X = [];
for i = 1:length(h)
    X(i) = ((f(x + h(i)) - 2.*f(x) + f(x - h(i)))./h(i).^2);
end

end
```

Console:

```
f =
    @(x)sin(x)

>> h = 10.^(-1.*[1:16]);
>> x = pi/6

>> X = diff_2(f,h,x);
>> E_d2 = abs(X - -.5)'
```

h	$f''(pi/6)$	E_{d2}
0.1	-0.49958	0.00041653
0.01	-0.5	4.1667e-06
0.001	-0.5	4.1674e-08
0.0001	-0.5	3.0387e-09
1e-05	-0.5	5.9648e-07
1e-06	-0.49993	6.6572e-05
1e-07	-0.49405	0.0059508
1e-08	-1.1102	0.61022
1e-09	55.5112	56.0112
1e-10	0	0.5
1e-11	0	0.5
1e-12	0	0.5
1e-13	5551115123.1258	5551115123.6258
1e-14	-555111512312.5782	555111512312.0782
1e-15	0	0.5
1e-16	-5551115123125783	5551115123125782

From our error analysis we can see that our best approximation comes when $h = 10^{-4}$. Looking at the Taylor's Theorem formula for $f''(x)$ in Example 9.1.2 we get that,

$$f''(x) = \frac{f(x+h) - 2f(h) + f(x-h)}{h^2} - \frac{h^2}{12} f''''(v)$$

We see that the error will be $O(h^2)$. Following a similar analysis as example 9.1.1 we see that the minimum computed difference quotient is on the order of $O(\epsilon/h^2)$. Solving for h we get that $h = \epsilon^{\frac{1}{4}} \approx 10^{-4}$.

Problem 9.5: Using Taylor series, derive the error term for the approximation,

Solution:

$$f'(x) \approx \frac{1}{2h}[-3f(x) + 4f(x+h) - f(x+2h)].$$

Applying the Taylor series formula to each function, we ge the following system,

$$-3f(x) = -3f(x).$$

$$4f(x+h) = 4f(x) + 4f'(x)h + \frac{4f'(x)h^2}{2!} + \frac{4f'(x)h^3}{3!} + O(h^4).$$

$$-f(x+2h) = -f(x) - 2f'(x)h - \frac{4f'(x)h^2}{2!} - \frac{8f'(x)h^3}{3!} + O(h^4).$$

Summing over the equation we get that,

$$[-3f(x) + 4f(x+h) - f(x+2h)] = 2f'(x)h - \frac{4f'(x)h^3}{3!} + O(h^4).$$

Finally we divide the whole equation by $2h$ to get,

$$\frac{[-3f(x) + 4f(x+h) - f(x+2h)]}{2h} = f'(x) - \frac{2f'(x)h^2}{3!} + O(h^3).$$

Thus the error term is,

$$error = -\frac{2f'(x)h^2}{3!} + O(h^3).$$