Problem 5 [5 points]:

5. The Chebyshev polynomials $T_j(x) = \cos(j \arccos x)$, $j = 0, 1, \ldots$, 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 $T_0$, $T_1$, and $T_2$.

\[
q_0(x) = \frac{1}{\int_{-1}^{1} (1 - x^2)^{-1/2} \, dx} = \frac{1}{\int_{-1}^{1} \arcsin x |_{-1}^{1}/2} \cdot \frac{1}{\sqrt{\pi}} = \frac{1}{\sqrt{\pi}}.
\]

\[
\hat{q}_1(x) = x - \left( \int_{-1}^{1} (1 - x^2)^{-1/2} x \cdot \frac{1}{\sqrt{\pi}} \, dx \right) \cdot \frac{1}{\sqrt{\pi}} = x - \frac{1}{\pi} \int_{-1}^{1} (1 - x^2)^{1/2} \, dx = x.
\]

\[
q_1(x) = \frac{x}{\int_{-1}^{1} (1 - x^2)^{-1/2} x^2 \, dx} = \frac{x}{\int_{-1}^{1} \left(- \frac{x}{\pi} \sqrt{1 - x^2} + \frac{1}{2} \arcsin x \right) \, dx} \cdot \frac{1}{\sqrt{\pi}}.
\]

\[
= \sqrt{\frac{2}{\pi}} x.
\]

\[
\hat{q}_2(x) = x^2 - \left( \int_{-1}^{1} (1 - x^2)^{-1/2} x^2 \cdot \frac{1}{\sqrt{\pi}} \, dx \right) \cdot \frac{1}{\sqrt{\pi}} - \left( \int_{-1}^{1} (1 - x^2)^{-1/2} x^2 \cdot \sqrt{\frac{2}{\pi}} x \, dx \right) \cdot \sqrt{\frac{2}{\pi}}
\]

\[
= x^2 - \frac{1}{2} \text{ (since the coefficient of } x \text{ is the integral of an odd function, hence 0)}
\]

\[
q_2(x) = \frac{x^2 - \frac{1}{2}}{\int_{-1}^{1} (1 - x^2)^{-1/2} \left(x^2 - \frac{1}{2}\right) \, dx} = \frac{x^2 - \frac{1}{2}}{\int_{-1}^{1} (1 - x^2)^{-1/2} \left(x^4 - x^2 + \frac{1}{4}\right) \, dx} \cdot \frac{1}{\sqrt{\pi}}
\]

\[
= \sqrt{\frac{2}{\pi}} (x^2 - 1)
\]

Thus $q_0(x) = \frac{1}{\sqrt{\pi}} T_0(x)$ and $q_j(x) = \sqrt{\frac{2}{\pi}} T_j(x)$ for $j = 1, 2$ (and also for all other values of $j$).
Problem 7 [5 points]:

7. Write a MATLAB code to approximate

\[ \int_0^1 \cos(x^2) \, dx \]

using the composite trapezoidal 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 an input to each code. Turn in listings of your codes.

Do a convergence study to verify the second order accuracy of the composite trapezoidal rule and the fourth order accuracy of the composite Simpson’s rule; that is, run your code with several different \( h \) values and make a table showing the error \( E_h \) with each value of \( h \) and the ratios \( E_h/h^2 \) for the composite trapezoidal rule and \( E_h/h^4 \) for the composite Simpson’s rule. These ratios should be nearly constant for small values of \( h \). You can determine the error in your computed integral by comparing your results with those of MATLAB routine \texttt{quad}. To learn about routine \texttt{quad}, type \texttt{help quad} in MATLAB. When you run \texttt{quad}, ask for a high level of accuracy, say,

\[ q = \texttt{quad}('\cos(x.^2)',0,1,[1.e-12 1.e-12]), \]

where the last argument \([1.e-12 1.e-12]\) indicates that you want an answer that is accurate to \(10^{-12}\) in both a relative and an absolute sense. (Note that when you use routine \texttt{quad} you must define a function, either inline or in a separate file, that evaluates the integrand \( \cos(x^2) \) at a \textit{vector} of values of \( x \); hence you need to write \( \cos(x.^2) \), instead of \( \cos(x^2) \).)

to \(10^{-12}\) in both a relative and an absolute sense. (Note that when you use routine \texttt{quad} you must define a function, either inline or in a separate file, that evaluates the integrand \( \cos(x^2) \) at a \textit{vector} of values of \( x \); hence you need to write \( \cos(x.^2) \), instead of \( \cos(x^2) \).)

Running the code below with 4, 8, 16, 32, and 64 subintervals, gave the following values for the error in the composite trapezoid rule and composite Simpson’s rule:

<table>
<thead>
<tr>
<th>n</th>
<th>err trap</th>
<th>err simp</th>
<th>err trap/h^2</th>
<th>err simp/h^4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8.77e-03</td>
<td>7.87e-08</td>
<td>1.4e-01</td>
<td>2.0e-05</td>
</tr>
<tr>
<td>8</td>
<td>2.19e-03</td>
<td>-1.47e-08</td>
<td>1.4e-01</td>
<td>6.0e-05</td>
</tr>
<tr>
<td>16</td>
<td>5.48e-04</td>
<td>-1.22e-09</td>
<td>1.4e-01</td>
<td>8.0e-05</td>
</tr>
<tr>
<td>32</td>
<td>1.37e-04</td>
<td>-8.06e-11</td>
<td>1.4e-01</td>
<td>8.5e-05</td>
</tr>
<tr>
<td>64</td>
<td>3.42e-05</td>
<td>-5.11e-12</td>
<td>1.4e-01</td>
<td>8.6e-05</td>
</tr>
</tbody>
</table>

It is easy to see the \( O(h^2) \) error in the composite trapezoid rule. It is less easy to see the \( O(h^4) \) error in composite Simpson’s rule: \( E_h/h^4 \) appears to be approaching a constant as \( h \) goes to 0, but with 64 subintervals Simpson’s rule gives almost as much accuracy as we are requiring of the \texttt{quad} routine to which it is being compared, so we would need a more accurate value to compare it to if we went to 128 subintervals, and we would soon hit the machine precision.

Following is the MATLAB code that produced these results:
% Compute an approximation to the integral from 0 to 1
% of cos(x^2) dx using the composite trapezoid rule and
% composite Simpson's rule.

n = input('Enter number of subintervals: ');
dx = 1/n;

trap = 0;
simp = 0;
for i=0:n,                     % Loop over nodes
    x = i*dx;
    cosxsq = cos(x^2);        % Evaluate integrand at x

    % Composite trapezoid rule
    if i==0 | i==n,
        trap = trap + .5*cosxsq; % Add .5*value at endpoints
    else
        trap = trap + cosxsq;   % Add 1*value at interior points
    end;

    % Composite Simpson's rule
    if i==0 | i==n,
        simp = simp + cosxsq/6; % Add (1/6)*value at endpoints
    else
        simp = simp + cosxsq/3; % Add (1/3)*value at interior points
    end;
    if i < n,
        xmid = x + .5*dx;
        cosxmdsq = cos(xmid^2);
        simp = simp + (2/3)*cosxmdsq; % Add (2/3)*value at midpoints
    end;
end;
trap = trap*dx;               % Multiply results by dx
simp = simp*dx;

% Compare with "exact" solution returned by quad.
q = quad('cos(x.^2)',0,1,[1.e-12 1.e-12]);

fprintf('%8.12e %8.12e %8.12e %8.2e %8.2e \\
',q,trap,simp)
fprintf('
%8.12e %8.12e %8.2e %8.2e %7.1e \\
',err_trap,err_simp, err_trap/h^2, err_simp/h^4)
err_trap = q-trap; err_simp = q-simp;
fprintf('
%8.2e %8.2e %7.1e %7.1e
',err_trap,err_simp,...
    abs(err_trap)/dx^2, abs(err_simp)/dx^4)