MATH/CS 5466 NUMERICAL ANALYSIS
Homework 5

Luan Cong Doan
luandoan@vt.edu

April 18, 2016

Problem 1. When tasked with computing a definite integral $\int_a^b f(x)\mathrm{d}x$, one is not forced to apply some quadrature rule directly to f. Sometimes a bit of forethought can lead to a more effective approach.

1. Does the standard error bound derived in class for the composite trapezoid rule give any insight about the performance of that method applied to the Fresnel integral

$$\int_0^1 \frac{\sin(x)}{\sqrt{x}}\mathrm{d}x$$

$$f(x) = \frac{\sin(x)}{\sqrt{x}} \Rightarrow f'(x) = \frac{\cos(x)\sqrt{x} - \sin(x)\dfrac{1}{2\sqrt{x}}}{x} = \frac{2x\cos(x) - \sin(x)}{2x^{3/2}}$$

$$\begin{aligned} f''(x) &= \frac{(2\cos(x) - 2x\sin(x) - \cos(x))2x^{3/2} - (2x\cos(x) - \sin(x))3x^{1/2}}{4x^3} \\ &= \frac{2x^{3/2}\cos(x) - 4x^{5/2}\sin(x) - 6x^{3/2}\cos(x) + 3x^{1/2}\sin(x)}{4x^3} \\ &= \frac{3x^{1/2}\sin(x) - 4x^{3/2}\cos(x) - 4x^{5/2}\sin(x)}{4x^3} \\ &= \frac{3\sin(x) - 4x\cos(x) - 4x^2\sin(x)}{4x^{5/2}} \end{aligned}$$

$$f'''(x) = \frac{[-\cos(x) - 4xsin(x) - 4x^2\cos(x)]\,4x^{5/2} - [3\sin(x) - 4x\cos(x) - 4x^2\sin(x)]\,10x^{3/2}}{16x^5}$$

We have: $f'''(x) < 0$ for all $x \in [0,1] \Rightarrow f''(x)$ decreasing over $[0,1]$

Error based on trapezoid rule:

$$\begin{aligned} E = -\frac{b-a}{12}h^2.f''(\eta) &= -\frac{1}{12}h^2.\frac{3\sin(\eta) - 4\eta\cos(\eta) - 4\eta^2\sin(\eta)}{4\eta^{5/2}} \qquad \text{for some } \eta \in [0,1] \\ &= -\frac{1}{12}h^2.\left(\frac{3\sin(\eta)}{4\eta^{5/2}} - \frac{\cos(\eta)}{\eta^{3/2}} - \frac{\sin(\eta)}{\eta^{1/2}}\right) \end{aligned}$$

We easily see that: $\lim_{\eta\to 0} E = \infty$

$\Rightarrow$ NO insight for error bound for Fresnel integral based on composite trapezoid rule.

2. One can compute this Fresnel integral by expanding $\sin(x)$ in a Taylor series about $x = 0$ to obtain

$$\int_0^\varepsilon \left(\frac{x - \frac{1}{3!}x^3 + \ldots}{\sqrt{x}}\right)\mathrm{d}x + \int_\varepsilon^1 \frac{\sin(x)}{\sqrt{x}}\mathrm{d}x$$

To approximate this quantity, truncate the Taylor series after m terms and compute the resulting approximation to the first integral exactly; approximate the second integral using the composite trapezoid rule. Estimate the accuracy of this procedure as a function of m, ϵ, and the number of subintervals used in the composite trapezoid rule for the second integral.

Expansion of function $\sin(x)$ to m terms by using Taylor series:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ... + (-1)^m \frac{x^{2m+1}}{(2m+1)!} \qquad \text{for } m = 0, 1, 2, ...$$

We have the first integral with m terms:

$$\begin{aligned}
\int_0^\varepsilon \left(\frac{x - \frac{1}{3!}x^3 + ...}{\sqrt{x}} \right) \mathrm{d}x &= \int_0^\varepsilon \left(x^{1/2} - \frac{1}{3!}x^{5/2} + ... + (-1)^m \frac{1}{(2m+1)!} x^{(4m+1)/2} \right) \mathrm{d}x \\
&= \left(\frac{2}{3}x^{3/2} - \frac{2}{7}\frac{1}{3!}x^{7/2} + ... + (-1)^m \frac{2}{4m+1}\frac{1}{(2m+1)!} x^{(4m+1)/2} \right) \Bigg|_0^\varepsilon \\
&= \frac{2}{3}\varepsilon^{3/2} - \frac{2}{7}\frac{1}{3!}\varepsilon^{7/2} + ... + (-1)^m \frac{2}{4m+1}\frac{1}{(2m+1)!} \varepsilon^{(4m+1)/2}
\end{aligned}$$

Second integral can be compute via composite trapezoid rule: $h = \dfrac{1-\varepsilon}{N}$ with N is number of subinterval

$$\begin{aligned}
\int_\varepsilon^1 \frac{\sin(x)}{\sqrt{x}} \mathrm{d}x &= \frac{h}{2} \left[f(x_0) + 2\sum_{j=1}^{N-1} f(x_j) + f(x_N) \right] - \frac{1-\varepsilon}{12} h^2 . f''(\eta) \\
&= \frac{1-\varepsilon}{2N} \left[\frac{\sin(\varepsilon)}{\sqrt{\varepsilon}} + 2\sum_{j=1}^{N-1} \frac{\sin(\varepsilon + j\dfrac{1-\varepsilon}{N})}{\sqrt{\varepsilon + j\dfrac{1-\varepsilon}{N}}} + \frac{\sin(1)}{\sqrt{1}} \right] - \frac{(1-\varepsilon)^3}{12N^2} . f''(\eta)
\end{aligned}$$

Error: $E_2 = -\dfrac{(1-\varepsilon)^3}{12N^2} . f''(\eta) = -\dfrac{(1-\varepsilon)^3}{12N^2} \left[\dfrac{3\sin(\eta)}{4\eta^{5/2}} - \dfrac{\cos(\eta)}{\eta^{3/2}} - \dfrac{\sin(\eta)}{\eta^{1/2}} \right]$ with $\eta \in [\varepsilon, 1]$

Base on above result, we have $f''(\eta)$ decreasing over $[\varepsilon, 1] \Rightarrow \min(f''(\eta)) = f''(\varepsilon)$

$$\Rightarrow E_2 = -\frac{(1-\varepsilon)^3}{12N^2} . f''(\eta) \leq -\frac{(1-\varepsilon)^3}{12N^2} \left[\frac{3\sin(\varepsilon)}{4\varepsilon^{5/2}} - \frac{\cos(\varepsilon)}{\varepsilon^{3/2}} - \frac{\sin(\varepsilon)}{\varepsilon^{1/2}} \right]$$

So:

$$\int_\varepsilon^1 \frac{\sin(x)}{\sqrt{x}} \mathrm{d}x \leq \frac{h}{2} \left[\frac{\sin(\varepsilon)}{\sqrt{\varepsilon}} + 2\sum_{j=1}^{N-1} \frac{\sin(\varepsilon + jh)}{\sqrt{\varepsilon + jh}} + \sin(1) \right] - \frac{(1-\varepsilon)^3}{12N^2} \left[\frac{3\sin(\varepsilon)}{4\varepsilon^{5/2}} - \frac{\cos(\varepsilon)}{\varepsilon^{3/2}} - \frac{\sin(\varepsilon)}{\varepsilon^{1/2}} \right]$$

3. Construct a function $g(x)$ such that the integral in part (a) can be effectively computed:

$$\int_0^1 \left(\frac{\sin(x)}{\sqrt{x}} - g(x) \right) \mathrm{d}x + \int_0^1 g(x) \mathrm{d}x$$

where the first integral can be computed with the composite trapezoid rule and the second integral can be computed exactly without need for numerical quadrature.

As result from above, we have the composite trapezoid approximation for $\int_0^1 f(x)\mathrm{d}x$ is:

$$\begin{aligned}\int_0^1 \frac{\sin(x)}{\sqrt{x}}\mathrm{d}x &= \frac{h}{2}\left[\frac{\sin(0)}{\sqrt{0}} + 2\sum_{j=1}^{N-1}\frac{\sin(0+jh)}{\sqrt{0+jh}} + \sin(1)\right] - \frac{1}{12N^2}\left[\frac{3\sin(\eta)}{4\eta^{5/2}} - \frac{\cos(\eta)}{\eta^{3/2}} - \frac{\sin(\eta)}{\eta^{1/2}}\right] \\ &= \frac{h}{2}\left[2\sum_{j=1}^{N-1}\frac{\sin(jh)}{\sqrt{jh}} + \sin(1)\right] - \frac{1}{12N^2}\left[\frac{3\sin(\eta)}{4\eta^{5/2}} - \frac{\cos(\eta)}{\eta^{3/2}} - \frac{\sin(\eta)}{\eta^{1/2}}\right] \qquad \eta \in [0,1]\end{aligned}$$

We can not get the exact result of approximation because terms in error part are not bounded. So, utilizing Taylor expansion for $\sin(x)$ and $\cos(x)$ we have:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + ... = \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}x^{2n+1}$$

$$\cos(x) = x - \frac{x^2}{2!} + \frac{x^4}{4!} + ... = \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n)!}x^{2n}$$

$$\begin{aligned}Error &= -\frac{1}{12N^2}\left[\frac{3\sin(\eta)}{4\eta^{5/2}} - \frac{\cos(\eta)}{\eta^{3/2}} - \frac{\sin(\eta)}{\eta^{1/2}}\right] \\ &= -\frac{1}{12N^2}\left[\frac{1}{\eta^{5/2}}\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}\eta^{2n+1} - \frac{1}{\eta^{3/2}}\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n)!}\eta^{2n} - \frac{1}{\eta^{1/2}}\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}\eta^{2n+1}\right] \\ &= -\frac{1}{12N^2}\left[\frac{1}{\eta^{3/2}} + \sum_{n=1}^{\infty}\frac{(-1)^n}{(2n+1)!}\eta^{2n-3/2} - \frac{1}{\eta^{1/2}} - \sum_{n=1}^{\infty}\frac{(-1)^n}{(2n)!}\eta^{2n-3/2} - \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}\eta^{2n+1/2}\right]\end{aligned}$$

In order to get the right value of error, we have to neglect 2 terms: $\frac{1}{\eta^{3/2}}$, and $\frac{1}{\eta^{3/2}}$ in error function. So we have:

$$\int_0^1 g(x)\mathrm{d}x = -\frac{1}{12N^2}\left[\frac{1}{\eta^{3/2}} - \frac{1}{\eta^{3/2}}\right] \Leftrightarrow$$

Problem 2. The gamma function is defined as

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} \mathrm{d}x$$

1. Use the composite Simpson's rule to evaluate $\Gamma(5), \Gamma(10)$, and $\Gamma(0.5)$ to five digits of accuracy. You may use `MATLAB`'s built-in `gamma` function to verify your answer. Please report the number of integrand evaluations required. In particular, design a way to reliably compute $\Gamma(0.5)$ using fewer than 1000 function evaluation. (Consider further partitioning the domain into subdomains).

   The orthogonal polynomial on $[0, \infty)$ with weight function $w(x) = e^{-x}$ are called Gauss-Laguerre polynomials. These polynomials can be constructed from the recurrence
   $L_0(x) = 1, \;\; L_1(x) = x - \alpha_0 = x - 1, \;\; L_{k+1}(x) = (x - \alpha_k)L_k(x) - \beta_k L_{k-1}(x), \quad k = 1, 2, ...$
   where $\alpha_k = 2k + 1$ for $k = 0, 1, ...$ $\quad \beta_0 = 1$, and $\beta_k = k^2$ for $k = 1, 2, ...$

2. Use your answer from Question 3 of Problem Set 4 to design a `MATLAB` function:

   ```
   [x,w] = gauss_lag(n)
   ```

   that computes the nodes x and the weight w for $n+1$ point Gauss-Laguerre quadrature from the corresponding Jacobi matrix $\mathbf{J}$. Recall that the weights are given by $w_j = \beta_0 (\mathbf{v}_j)_1^2$
   where $(\mathbf{v}_j)_1^2$ is the square of the first entry of the eigenvector $\mathbf{v}_j$ of $\mathbf{J}$ corresponding to the eigenvalue x_j. (Here $\mathbf{v}_j$ should be normalized so that $\|\mathbf{v}_j\|_2 = 1$, which is automatically imposed upon the eigenvectors returned by `MATLAB's eig` command).
   $$\begin{aligned} L_{k+1}(x) = (x - \alpha_k)L_k(x) - \beta_k L_{k-1}(x) &= xL_k(x) - \alpha_k L_k(x) - \beta_k L_{k-1}(x) \\ &= xL_k(x) - (2k+1)L_k(x) - k^2 L_{k-1}(x) \end{aligned}$$
   and $L_0 = 1, L_1(x) = x - \alpha_0 = x - 1$
   As result from Question 3 - Problem set 4 we have the corresponding Jacobi matrix $\mathbf{J}$ is defined by:

$$\mathbf{J} = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & 0 & \dots & 0 \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & \dots & 0 \\ 0 & \sqrt{\beta_2} & \ddots & \ddots & \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \ddots & \alpha_{n-1} & \sqrt{\beta_n} \\ 0 & 0 & \dots & \sqrt{\beta_n} & \alpha_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & \dots & 0 \\ 1 & 3 & 2 & \dots & 0 \\ & 2 & \ddots & \ddots & \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \ddots & 2n-1 & n \\ 0 & 0 & \dots & n & 2n+1 \end{bmatrix}$$

```
function [x,w] = gauss_lag(n)
J = zeros(n+1); x = zeros(1,n+1);
for i = 1:n+1
    for j = 1:n+1
        if (i-j==0)          J(i,j) = 2*i-1;
        elseif (j-i==1)      J(i,j) = i;
        elseif (i-j==1)      J(i,j) = j;
        end
    end
end
[V,D] = eig(J);
for i = 1:size(D,1)
    for j = 1:size(D,2)
        if (i-j==0)      x(1,i) = D(i,j);     end
    end
end
    w = V(1,:).^2;
```

3. Produce a *single plot* comparing the nodes of all $n+1$ point Gauss-Laguerre rules for $n = 0, ..., 25$. For example, you can plot each set of nodes with

   `plot(x, n*ones(n+1,1), '.','markersize',24)`

   which displays the nodes horizontally at vertical level n. (Notice how the largest node, x_n, grows as n increases, as the rule integrates over the unbounded interval $[0, \infty)$.)

```
N = 1:1:25;
```

```
figure;
for i = 1:length(N)
    [x,~,~] = gauss_lag(i);
    plot(x, i*ones(i+1,1), '.','markersize',24); hold on; grid on;
end
xlabel('x'); ylabel('N'); title('Root estimation');
print('hw5_2c','-dpng');
```

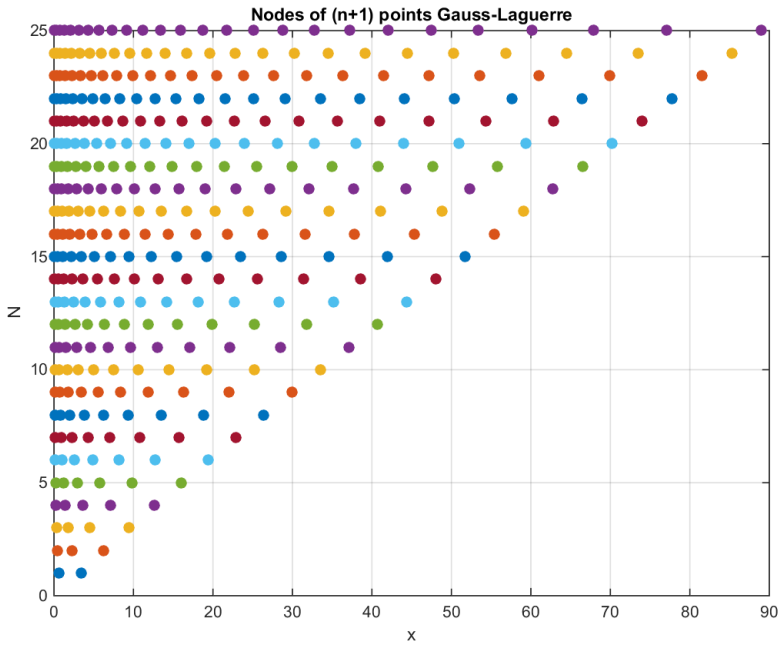

Figure 1: Nodes of $(n+1)$ points Gauss-Laguerre

4. Compute $\Gamma(5), \Gamma(10)$, and $\Gamma(0.5)$ using this $n+1$ point Gauss-Laguerre rule for $n = 5$. (Each of these integrals will only require six $f(x)$ evaluations.)

5. How does the accuracy of Gauss-Laguerre quadrature compare to that obtained by the composite Simpson's rule? Give pros and cons of each method for this problem.

Problem 3. How might the implementer of a mathematical function library (or even hardware designer) implement a function like `sqrt` using only basic operations (addition, subtraction, multiplication, division)? Various clever approaches exist. This question (from problems in Gautschi's book) considers several options using Newton's method.

1. Given some fixed $A > 0$, one can compute $x_* = \sqrt{A}$ via the root-finding problem: $x^2 - A = 0$ Explain why Newton's method applied to the first equation converges for

any initial value $x_0 > 0$. Be as rigorous as possible.

$$f_1(x) = x^2 - A \to f_1'(x) = 2x$$
$$f_1''(x) = 2 > 0 \to f_1(x) \text{ is convex on } \mathbb{R}_+$$

and increase monotonically from $-A$ to ∞.

Therefore, if $x_0 > \alpha$ ($\alpha = \sqrt{A}$), then x_n converges monotonically decreasing to α.

If $0 < x_0 < \alpha$, then $x_1 > \alpha$, and the same conclusion holds for $n \geq 1$.

2. Alternatively, one might compute $x_* = \sqrt{A}$ via the root-finding problem: $\dfrac{A}{x^2} - 1 = 0$

Show that the initial set of positive initial guesses $x_0 > 0$ can be partitioned into three sets:

$$x \in (0, \alpha) \Rightarrow x_k \to \sqrt{A} \text{ and all } x_k > 0$$
$$x \in (\alpha, \beta) \Rightarrow x_k \to \pm\sqrt{A},\ x_k \text{ do not all have the same sign;}$$
$$x \in (\beta, \infty) \Rightarrow x_k \text{ diverges}$$

(You should specify formulas for α and β as a function of A)

$$f_2(x) = \frac{A}{x^2} - 1 \to f_2'(x) = -\frac{2A}{x^3}$$
$$f_2''(x) = \frac{6A}{x^4} > 0 \to f_2(x) \text{ is convex on } \mathbb{R}_+$$

and decrease monotonically from ∞ to -1.

If $0 < x_0 < \alpha$ ($\alpha = \sqrt{A}$) then x_n converges monotonically increasing to α.

If $x_0 > \alpha$, we must make sure that $x_1 > 0$, which following Newton-method means that:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} > 0 \Leftrightarrow x_1 = x_0 - \frac{\frac{A}{x_0^2} - 1}{-2\frac{A}{x_0^3}} > 0 \Leftrightarrow x_0 + x_0\frac{A - x_0^2}{2A} > 0$$
$$\Leftrightarrow 3Ax_0 - x_0^3 > 0$$
$$\Leftrightarrow 3A - x_0^2 > 0$$
$$\Leftrightarrow x_0 < \sqrt{3A} =: \beta$$

3. Repeat part (a) and (b) for the analogous approaches to the cube root:

$$x^3 - A = 0; \qquad \frac{A}{x^3} - 1 = 0$$

In generalizing part (b), simplify identify the interval $(0, \alpha)$ (with $\alpha = \sqrt[3]{A}$) on which the iterates are all positive and coverage to $\sqrt{A}$

$$f_3(x) = x^3 - A \to f_3'(x) = 3x^2$$
$$f_3''(x) = 6x > 0 \text{ because } x > 0$$

$\rightarrow f_3(x)$ is convex on $\mathbb{R}_+$ and increasing monotonically from $-A$ to ∞.

Then, if $x_0 > \alpha$, then x_n converges monotonically decreasing to α.

If $0 < x_0 < \alpha$, then $x_1 > \alpha$, and the same conclusion holds for $n \geq 1$.

$$f_4(x) = \frac{A}{x^3} - 1 \rightarrow f_4'(x) = -\frac{3A}{x^4}$$
$$f_4''(x) = \frac{12A}{x^5} > 0 \text{ because } x > 0$$

$\rightarrow f_4(x)$ is convex on $\mathbb{R}_+$ and decrease monotonically from ∞ to -1

If $0 < x_0 < \alpha$ then x_n converges monotonically increasing to α.

If $x_0 > \alpha$, we must make sure that $x_1 > 0$, which following Newton-method means that:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} > 0 \Leftrightarrow x_1 = x_0 - \frac{\frac{A}{x_0^3} - 1}{-3\frac{A}{x_0^4}} > 0 \Leftrightarrow x_0 + x_0\frac{A - x_0^4}{3A} > 0$$
$$\Leftrightarrow 4Ax_0 - x_0^4 > 0$$
$$\Leftrightarrow 4A - x_0^3 > 0$$
$$\Leftrightarrow x_0 < \sqrt[3]{4A} =: \beta$$

4. One can also compute to compute $x_* = \sqrt{A}$ via the fixed point iteration:

$$x_{k+1} = \frac{x_k(x_k^2 + 3a)}{3x_k^2 + a}$$

(i) Show that *if the sequence converges to* $\sqrt{A}$, it converges *cubucally*

(ii) Determine the *asymptotic error constant*

(iii) Discuss (using graphical or numerical support) the convergence of this method

$x_0 > 0$.

Set $\varphi(x) = \frac{x_k(x_k^2 + 3a)}{3x_k^2 + a}$, $\qquad \alpha = \sqrt{a}$

Clearly see that: $\Rightarrow \varphi(\alpha) = \frac{\alpha(\alpha^2 + 3a)}{3\alpha^2 + a} = \frac{\alpha(a + 3a)}{3a + a} = \alpha$

We have: $(3x^2 + a)\varphi(x) = x^3 + 3ax$

Differentiating above equation: $6x\varphi(x) + (3x^2 + a)\varphi'(x) = 3x^2 + 3a$

At $x = \alpha$: $6\alpha\varphi(\alpha) + (3\alpha^2 + a)\varphi'(\alpha) = 3\alpha^2 + 3a \Leftrightarrow 6\alpha\varphi(\alpha) + 6a\varphi'(\alpha) = 6a$

$$\Leftrightarrow 6\alpha\alpha + 4a\varphi'(\alpha) = 6a$$
$$\Leftrightarrow \varphi'(\alpha) = 0$$

Second order differentiation equation: $6\varphi(x) + 6x\varphi'(x) + 6x\varphi'(x) + (3x^2 + a)\varphi''(x) = 6x$

$$\Leftrightarrow 6\varphi(x) + 12x\varphi'(x) + (3x^2 + a)\varphi''(x) = 6x$$

At $x = \alpha$: $6\varphi(\alpha) + 12\alpha\varphi'(\alpha) + (3\alpha^2 + a)\varphi''(\alpha) = 6\alpha \Leftrightarrow 4a\varphi''(\alpha) = 0 \Leftrightarrow \varphi''(\alpha) = 0$

Third order differentiation equation:

$$6\varphi'(x) + 12\varphi'(x) + 12x\varphi''(x) + 6x\varphi''(x) + (3x^2 + a)\varphi'''(x) = 6$$
$$\Leftrightarrow 18\varphi'(x) + 18x\varphi''(x) + (3x^2 + a)\varphi'''(x) = 6$$

At $x = \alpha$: $18\varphi'(\alpha)+18\alpha\varphi''(\alpha)+(3\alpha^2+a)\varphi'''(\alpha) = 6 \Leftrightarrow 4a\varphi'''(\alpha) = 6 \Leftrightarrow \varphi'''(\alpha) = \dfrac{6}{4a} \neq 0$

$\Rightarrow$ the iteration converges at order $p = 3$ or *cubically*

The *asymptotic error constant* is determined by:

$$c = \frac{1}{3!}\varphi'''(\alpha) = \frac{1}{6}\frac{6}{4a} = \frac{1}{4a}$$

Applied Taylor expansion for $\varphi(x_0)$ at α we have:

$\varphi(x_0) = \varphi(\alpha)+\varphi'(\alpha)(x_0-\alpha)+\dfrac{\varphi''(\alpha)}{2!}(x_0-\alpha)^2+\dfrac{\varphi'''(\xi)}{3!}(x_0-\alpha)^3$ for some $\xi \in [x_0, \alpha]$

$\Leftrightarrow x_1 = \alpha - \dfrac{\varphi'''(\xi)}{6}(x_0 - \alpha)^3$ because $\alpha = \varphi(\alpha)$ and $\varphi'(\alpha) = \varphi''(\alpha) = 0$

$\Leftrightarrow e_1 = -\dfrac{\varphi'''(\xi)}{6}e_0^3$

In general, the convergence of this method is: $e_{k+1} = -\dfrac{\varphi'''(\xi)}{6}e_k^3$ for some $\xi \in [x_k, \alpha]$

Problem 4. Recall the derivation of Newton's method: A function $f \in C^2(\mathbb{R})$ was expanded in a Taylor series up to first order. Here we investigate the algorithm obtained by taking one more term in the Taylor series. Assume $f \in C^3(\mathbb{R})$.

1. Derive an algorithm analogous to Newton's method but based on the first three terms of the Taylor series for $f(x_*)$ expanded at $f(x_k)$ rather than just the first two. This method should include evaluations of both $f'(x_k)$ and $f''(x_k)$

   Taylor expansion of $f(x_*)$ at $x_k \approx x_*$ for first three term:

   $f(x_*) = f(x_k)+f'(x_k)(x_*-x_k)+\dfrac{f''(x_k)}{2!}(x_*-x_k)^2+\dfrac{f'''(\xi)}{3!}(x_*-x_k)^3$ for some $\xi \in [x_k, x_*]$

   Neglect the third order term:

$$0 = f(x_k) + f'(x_k)(x_{k+1} - x_k) + \frac{f''(x_k)}{2!}(x_{k+1} - x_k)^2$$
$$\Leftrightarrow f''(x_k)x_{k+1}^2 - 2f''(x_k)x_k x_{k+1} + 2f'(x_k)x_{k+1} + 2f(x_k) - 2f'(x_k)x_k = 0$$
$$\Leftrightarrow f''(x_k)x_{k+1}^2 - 2\left[f''(x_k)x_k - f'(x_k)\right]x_{k+1} + 2f(x_k) - 2f'(x_k)x_k = 0$$
$$\Delta = 4\left[(f''(x_k)x_k - f'(x_k)\right]^2 - 8f''(x_k)\left[f(x_k) - f'(x_k)x_k\right]$$
$$\Rightarrow x_{k+1} = \frac{2\left[(f''(x_k)x_k - f'(x_k)\right] - \sqrt{\Delta}}{2f''(x_k)} \quad \text{or} \quad x_{k+1} = \frac{2\left[(f''(x_k)x_k - f'(x_k)\right] + \sqrt{\Delta}}{2f''(x_k)}$$

2. At each iteration, Newton's method approximates the root of f by the root of the line tangent to f at x_k. Your new algorithm should approximate f by a parabola. Draw this parabola, together with the tangent line used by Newton's method, for the function $f(x) = \sin(x)e^x$ at the point $x_0 = 1.25$. (Please produce a careful MATLAB plot, not a hand-drawn sketch).

```
x = -1:0.05:2;
f = sin(x).*exp(x);
fx = sin(1.25).*exp(1.25);
f1x = cos(x).*exp(x) + sin(x).*exp(x);
f1 = cos(1.25).*exp(1.25) + sin(1.25).*exp(1.25);
f2x = 2*cos(x).*exp(x);
f2 = 2*cos(1.25).*exp(1.25);
N2 = f1*(x-1.25) + fx;
N3 = f2*x.^2 -2*(f2*1.25-f1)*x + 2*(fx-f1*1.25);
figure; plot(x,f,'k','linewidth',1); grid on; hold on;
plot(x,N2,'r--'); plot(x,N3,'b.');
xlabel('x'); ylabel('f(x) = sin(x).e^x');
title('Newton method approximation of root at x_0 = 1.25');
legend('fx','tangent line','parabola');
print('hw5_4b','-dpng');
```

3. The parabola in question will often have two roots. Describe which of these roots should be selected at each iteration.

   The parabola equation have 2 roots: $x_{k1} < x_{k2}$ with values computed in part (a).

   The subinterval is defined by $[a, b]$, and current iteration point is $x_k : a \leq x_k \leq b$

   $\Rightarrow$ the suitable root will be chosen by following steps:

   + Check: $x_{k1}, x_{k2} \in [a, b]$, which is not meet this requirement should be neglected.

   + If $a \leq x_{k1} < x_{k2} \leq x_k < b$, check sign of: $f(x_k) * f(x_{k1}), f(x_k) * f(x_{k2})$

   - If $\begin{cases} f(x_k) * f(x_{k1}) < 0 \\ f(x_k) * f(x_{k2}) > 0 \end{cases}$ $\Rightarrow$ new iteration point is x_{k2}

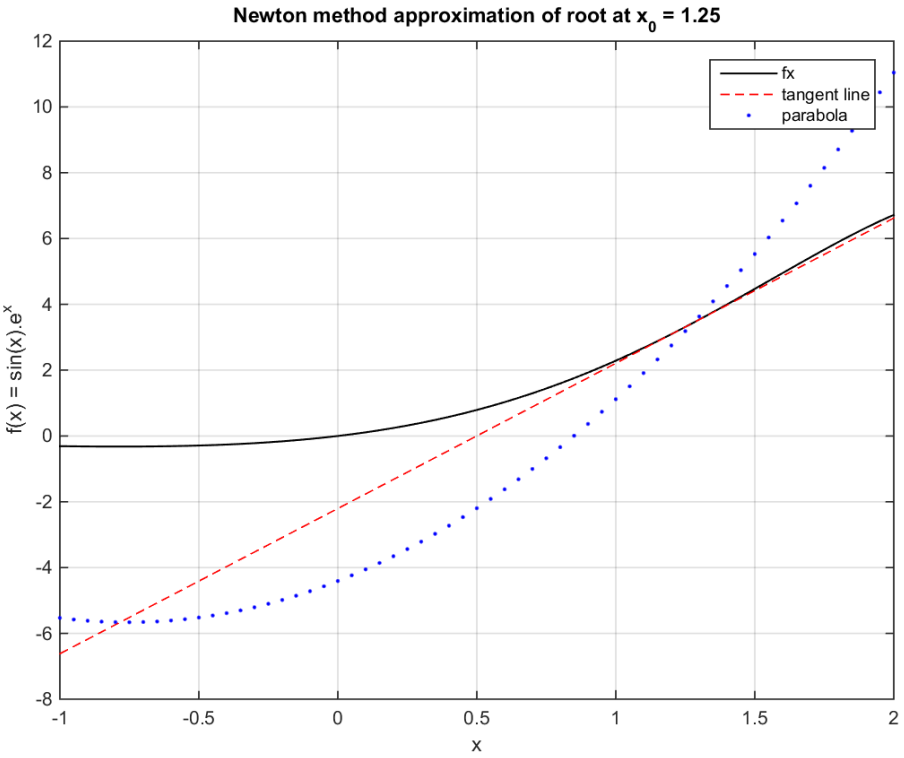

Figure 2: Newton's method approximation of root of $f(x) = \sin(x)e^x$

$$\bullet \text{ If } \begin{cases} f(x_k) * f(x_{k1}) > 0 \\ f(x_k) * f(x_{k2}) > 0 \end{cases} \Rightarrow \text{new iteration point is } x_{k1}$$

+ If $a < x_k \leq x_{k1} < x_{k2} \leq b$, check sign of: $f(x_k) * f(x_{k1}), f(x_k) * f(x_{k2})$

$$\bullet \text{ If } \begin{cases} f(x_k) * f(x_{k1}) > 0 \\ f(x_k) * f(x_{k2}) > 0 \end{cases} \Rightarrow \text{new iteration point is } x_{k2}$$

$$\bullet \text{ If } \begin{cases} f(x_k) * f(x_{k1}) > 0 \\ f(x_k) * f(x_{k2}) < 0 \end{cases} \Rightarrow \text{new iteration point is } x_{k1}$$

4. Under what circumstances will this parabola have no roots?

   As result from part (a), parabola have no root when:

$$\begin{aligned} \Delta &= 4\left[(f''(x_k)x_k - f'(x_k)\right]^2 - 8f''(x_k)\left[f(x_k) - f'(x_k)x_k\right] < 0 \\ &\Leftrightarrow 4(f''(x_k)x_k)^2 + 4(f'(x_k))^2 - 8f''(x_k)f(x_k) < 0 \\ &\Leftrightarrow (f''(x_k)x_k)^2 + (f'(x_k))^2 - 2f''(x_k)f(x_k) < 0 \end{aligned}$$

5. Implement this algorithm in MATLAB, in a form similar to the code `newton.m` discussed in class

```
function xstar = newton2(f,fprime,f2prime, x0)
maxit = 60;
fx = feval(f,x0); x=x0; k=0;        % initialize
fprintf(' %3d  %20.14f  %10.7e\n', k, x, fx);
while (abs(fx) > 1e-15) && (k < maxit)
a = feval(f2prime,x);
b = 2*feval(f2prime,x)*x - 2*feval(fprime,x);
c = 2*fx - 2*feval(fprime,x)*x;
delta = b.^2 - 4*a*c;
x1 = (b-sqrt(delta))/(2*a);
x2 = (b+sqrt(delta))/(2*a);
if (x2 < x) && (feval(f,x1)*feval(f,x) < 0)
    && (feval(f,x2)*feval(f,x) > 0)
        x = x2;
elseif (x2 < x) && (feval(f,x1)*feval(f,x) > 0)
    && (feval(f,x2)*feval(f,x) > 0)
        x = x1;
elseif (x < x1) && (feval(f,x1)*feval(f,x) > 0)
   && (feval(f,x2)*feval(f,x) > 0)
        x = x2;
elseif (x < x1) && (feval(f,x1)*feval(f,x) > 0)
    && (feval(f,x2)*feval(f,x) < 0)
        x = x1;
end
k = k+1;
fx = feval(f,x);
fprintf(' %3d  %20.14f  %10.7e\n', k, x, fx);
end
xstar = x;
```

6. Compare the results of your method with those for `newton.m` on $f(x) = \sin(x)e^x$ with

$x_0 = 1.25$.

```
fx = @(x) sin(x)*exp(x);
fprime = @(x) cos(x)*exp(x) + sin(x)*exp(x);
f2prime = @(x) 2*cos(x)*exp(x);

xstar1 = newton(fx,fprime,1.25);
xstar2 = newton2(fx,fprime,f2prime,1.25);

>> xstar1
            xstar1 = 9.1508e-29
>> xstar2
            xstar2 = 9.8325e-12
```

7. Conjecture about the convergence rate of this method when $f''(x) \neq 0$ and x_0 is sufficiently close to x. Why is not this algorithm more famous than Newtons method?

   Reconsider Taylor expansion in part (4.1) for first 3 term:

$$0 = f(x_k) + f'(x_k)(x_{k+1} - x_k) + \frac{f''(x_k)}{2!}(x_{k+1} - x_k)^2$$
$$\Leftrightarrow 0 = f(x_k) + \left[f'(x_k) + \frac{f''(x_k)(x_{k+1} - x_k)}{2}\right](x_{k+1} - x_k) \qquad (*)$$

   Utilizing Newton's method: $x_{k+1} - x_k = -\dfrac{f(x_k)}{f'(x_k)}$ replaced into bracketed term in (*):

$$0 = f(x_k) + \left[f'(x_k) - \frac{f''(x_k)}{2}\frac{f(x_k)}{f'(x_k)}\right](x_{k+1} - x_k)$$
$$\Leftrightarrow 0 = f(x_k) + \frac{2[f'(x_k)]^2 - f''(x_k)f(x_k)}{2f'(x_k)}(x_{k+1} - x_k)$$
$$\Leftrightarrow x_{k+1} = x_k - \frac{2f(x_k)f'(x_k)}{2[f'(x_k)]^2 - f(x_k)f''(x_k)} \qquad (**)$$

   We also can write Taylor expansion:

   - to the second order: $0 = f(x_k) + f'(x_k)e_k + \dfrac{f''(\eta)}{2}e_k^2$ with $\eta \in [x_*, x_k]$

$$\Rightarrow 0 = f(x_k)f''(x_k)e_k + f'(x_k)f''(x_k)e_k^2 + \frac{f''(x_k)f''(\eta)}{2}e_k^3 \qquad (***)$$

   - to the third order: $0 = f(x_k) + f'(x_k)e_k + \dfrac{f''(x_k)}{2}e_k^2 + \dfrac{f'''(\xi)}{6}e_k^3$ with $\xi \in [x_*, x_k]$

$$\Rightarrow 0 = 2f(x_k)f'(x_k) + 2[f'(x_k)]^2 e_k + f'(x_k)f''(x_k)e_k^2 + \frac{f'(x_k)f'''(\xi)}{3}e_k^3 \qquad (****)$$

   Subtract (****) to (***) we have:

$$0 = 2f(x_k)f'(x_k) + 2[f'(x_k)]^2 e_k - f(x_k)f''(x_k)e_k + \frac{f'(x_k)f'''(\xi)}{3}e_k^3 - \frac{f''(x_k)f''(\eta)}{2}e_k^3$$

$$\Leftrightarrow 0 = 2f(x_k)f'(x_k) + [2[f'(x_k)]^2 - f(x_k)f''(x_k)]\, e_k + \frac{2f'(x_k)f'''(\xi) - 3f''(x_k)f''(\eta)}{6} e_k^3$$

$$\Leftrightarrow e_k = -\frac{2f(x_k)f'(x_k)}{2[f'(x_k)]^2 - f(x_k)f''(x_k)} - \frac{2f'(x_k)f'''(\xi) - 3f''(x_k)f''(\eta)}{12[f'(x_k)]^2 - 6f(x_k)f''(x_k)} e_k^3 \qquad (*****)$$

From (**) and (*****): $e_{k+1} = -\frac{2f'(x_k)f'''(\xi) - 3f''(x_k)f''(\eta)}{12[f'(x_k)]^2 - 6f(x_k)f''(x_k)} e_k^3$

$\Rightarrow$ the converge rate is cubic.

This method may have better converge rate than Newton's method but the computation is more expensive, and when x_0 is close to x mean that e_k become smaller, the different between this 2 method is not significant.

8. How do you expect this method to perform near a double root? Why? Compare your iteration to Newtons method for $f(x) = x^2e^x$ with $x_0 = 1$.

   Halley method will provide 2 values of x_{k+1} (approximate by a parabola) so we will have 1 more value to get higher and faster estimation.

```
f4h = @(x) x^2*exp(x);
f4h_prime = @(x) 2*x*exp(x) + x^2*exp(x);
f4h_2prime = @(x) 2*exp(x) + 4*x*exp(x) + x^2*exp(x);
x4_star1 = newton(f4h,f4h_prime,1);
x4_star2 = halley(f4h,f4h_prime,f4h_2prime,1);

Result
>> x4_star1
            x4_star1 =  1.8433e-08
>> x4_star2
            x4_star2 =  1.3478e-08
```

We can easily the one root of provided function is 0 $\Rightarrow$ Value estimated base on Halley method get higher accuracy in comparison with Newton's method result.

```
% Halley function:
function xstar = halley(f,fprime,f2prime, x0)
maxit = 60;
fx = feval(f,x0); x=x0; k=0;        % initialize
```

```
while (abs(fx) > 1e-15) && (k < maxit)
x = x - (2*fx*feval(fprime,x))/(2*feval(fprime,x)*feval(fprime,x) ...
                                - fx*feval(f2prime,x));
k = k+1;
fx = feval(f,x);
end
xstar = x;
```

Problem 5. Polynomial root-finding is an important special case of the general root-finding problem. Thus it is no surprise that innumerable specialized algorithms have been proposed for computing the roots of a polynomial. The method outlined in this problem is particularly appealing, for it is guaranteed to find n roots of a degree-n polynomial. Indeed, it is the basis for `MATLABs roots` command.

Suppose we seek the zeros of the degree-n polynomial: $p(x) = c_0 + c_1 x + ... + c_n x^n$

1. Show that $p(x) = 0$ if and only if x is an eigenvalue of the *companion* matrix

$$\mathbf{C}_n = \begin{bmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \\ -\frac{c_0}{c_n} & -\frac{c_1}{c_n} & \dots & -\frac{c_{n-2}}{c_n} & -\frac{c_{n-1}}{c_n} \end{bmatrix}$$

with eigenvector $\mathbf{v}(x) = [1, x, ..., x^{n-1}]^T$

For x is an eigenvalue of $\mathbf{C}_n$ we have: $\det[x\mathbf{I}_n - \mathbf{C}_n] = 0$

$$\det[x\mathbf{I}_n - \mathbf{C}_n] = \det \begin{bmatrix} x & -1 & & & \\ & x & -1 & & \\ & & \ddots & \ddots & \\ & & & x & -1 \\ \frac{c_0}{c_n} & \frac{c_1}{c_n} & \dots & \frac{c_{n-2}}{c_n} & x + \frac{c_{n-1}}{c_n} \end{bmatrix}$$

$$= x \begin{bmatrix} x & -1 & & \\ & \ddots & \ddots & \\ & & x & -1 \\ \frac{c_1}{c_n} & \dots & \frac{c_{n-2}}{c_n} & x + \frac{c_{n-1}}{c_n} \end{bmatrix} + (-1)^{n-1} \frac{c_0}{c_n} \begin{bmatrix} -1 & & & \\ x & -1 & & \\ & \ddots & \ddots & \\ & & x & -1 \end{bmatrix}$$

$$= x^2 \begin{bmatrix} x & -1 & & \\ & \ddots & \ddots & \\ & & x & -1 \\ \frac{c_2}{c_n} & \dots & \frac{c_{n-2}}{c_n} & x + \frac{c_{n-1}}{c_n} \end{bmatrix} + (-1)^{n-2} x \frac{c_1}{c_n} \begin{bmatrix} -1 & & & \\ x & -1 & & \\ & \ddots & \ddots & \\ & & x & -1 \end{bmatrix} + (-1)^{n-1}(-1)^{n-1} \frac{c_0}{c_n}$$

$$= x^3 \begin{bmatrix} x & -1 & & \\ & \ddots & \ddots & \\ & & x & -1 \\ \frac{c_3}{c_n} & \dots & \frac{c_{n-2}}{c_n} & x + \frac{c_{n-1}}{c_n} \end{bmatrix} + x^2 (-1)^{n-3} \frac{c_2}{c_n} \begin{bmatrix} -1 & & & \\ x & -1 & & \\ & \ddots & \ddots & \\ & & x & -1 \end{bmatrix} + \frac{c_1}{c_n} x + \frac{c_0}{c_n}$$

Keep going in computing determinant of matrix $[x\mathbf{I}_n - \mathbf{C}_n]$ we have the final value:

$\det[x\mathbf{I}_n - \mathbf{C}_n] = x^n + \frac{c_{n-1}}{c_n} x^{n-1} + ... + \frac{c_2}{c_n} x^2 + \frac{c_1}{c_n} x + \frac{c_0}{c_n} = 0$

$\Leftrightarrow c_n x^n + c_{n-1} x^{n-1} + ... + c_2 x^2 + c_1 x + c_0 = 0$

$\Rightarrow$ x is a root of polynomial $p(x)$

2. Verify your results numerically with the polynomial: $p(x) = (x-1)(x-2)(x-3)$

```
c = poly(1:3);  n = length(c);  C = zeros(n-1);
for i = 1:n-1
    for j=1:n-1
        if j-i == 1
            C(i,j) = 1;
        end
    end
end
for k = 1:n-1
    C(n-1,k) = -c(n-k+1)/c(1);
end
x = eig(C);

Result:     >> x
                        x = 1.0000      2.0000      3.0000
```

3. Repeat this experiment for the degree-24 *Wilkinson's polynomial*:

$$p(x) = (x-1)(x-2)...(x-24)$$

```
c = poly(1:24);      n = length(c);  C = zeros(n-1);
for i = 1:n-1
    for j=1:n-1
        if j-i == 1
            C(i,j) = 1;
        end
    end
end
for k = 1:n-1
    C(n-1,k) = -c(n-k+1)/c(1);
end
x = eig(C);
```

```
Result:     >> x
                    x =
                    24.0924 + 0.0000i
                    22.9443 + 0.7623i
                    22.9443 - 0.7623i
                    20.9662 + 1.6961i
                    20.9662 - 1.6961i
                    18.6006 + 2.1099i
                    18.6006 - 2.1099i
                    16.1990 + 1.9639i
                    16.1990 - 1.9639i
                    13.9339 + 1.3150i
                    13.9339 - 1.3150i
                    12.9360 + 0.0000i
                    11.4054 + 0.4976i
                    11.4054 - 0.4976i
                    9.8218 + 0.0000i
                    9.0595 + 0.0000i
```

```
7.9903 + 0.0000i
7.0013 + 0.0000i
5.9999 + 0.0000i
5.0000 + 0.0000i
4.0000 + 0.0000i
3.0000 + 0.0000i
2.0000 + 0.0000i
1.0000 + 0.0000i
```

Result get back from method is not correct as it should be: $1, 2, ..., 24$.

4. Suppose that we have a polynomial represented in basis of Chebyshev polynomials:

$$p(x) = a_0 T_0(x) + a_1 T_1(x) + ... + a_n T_n(x),$$

where, as usual: $T_0(x) = 1; T_1(x) = x; ...; T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x); ...k = 1, 2, ...$ Generalize the notion of a companion matrix to this setting, showing how the roots of p can be found as eigenvalue of some matrix $\mathbf{G}_n$ that encodes the coefficients $a_0, ..., a_n$. What are the corresponding eigenvectors?

$$p(x) = \sum_{j=0}^{n} a_n T_n(x) = \sum_{j=0}^{n} b_n x^n$$

With respect to Chebyshev polynomial, it's companion matrix have form:

$$\mathbf{G}_n = \begin{bmatrix} 0 & 1 & 0 & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & & \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & & \\ & & \ddots & \ddots & \ddots & \\ -\frac{a_0}{2a_n} & -\frac{a_1}{2a_n} & ... & ... & ... & -\frac{a_{n-1}}{2a_n} \end{bmatrix}$$

Determinant of matrix $\det[x\mathbf{I}_n - \mathbf{G}_n]$ is defined by:

$$\det[x\mathbf{I}_n - \mathbf{G}_n] = \begin{bmatrix} x & -1 & 0 & & & \\ -\frac{1}{2} & x & -\frac{1}{2} & & & \\ 0 & -\frac{1}{2} & x & -\frac{1}{2} & & \\ & & \ddots & \ddots & \ddots & \\ \frac{a_0}{2a_n} & \frac{a_1}{2a_n} & \dots & \dots & \dots & x + \frac{a_{n-1}}{2a_n} \end{bmatrix}$$

$$= x \begin{bmatrix} -\frac{1}{2} & x & -\frac{1}{2} & & & \\ 0 & -\frac{1}{2} & x & -\frac{1}{2} & & \\ & & \ddots & \ddots & \ddots & \\ \frac{a_1}{2a_n} & \frac{a_2}{2a_n} & \dots & \dots & x + \frac{a_{n-1}}{2a_n} \end{bmatrix} + \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & & & & \\ 0 & x & -\frac{1}{2} & & & \\ & & \ddots & \ddots & & \\ \frac{a_0}{2a_n} & \frac{a_2}{2a_n} & \dots & \dots & x + \frac{a_{n-1}}{2a_n} \end{bmatrix}$$

Keep going we end-up with result: $\det[x\mathbf{I}_n - \mathbf{G}_n] = \frac{1}{a_n} p(x)$

The eigenvector of matrix $\mathbf{G}_n$ is

5. Verify your algorithm for the polynomial: $p(x) = -\frac{1}{8}T_0(x) + \frac{1}{2}T_1(x) - \frac{1}{2}T_3(x) + \frac{1}{8}T_4(x)$ which has root $-1, 0, 1, 2$

```
a = [-1/8 1/2 0 -1/2 1/8];
n = length(a); G = zeros(n-1);  G(1,2) = 1;
for i = 2:n-2
    for j = 1:n-1
        if (i-j ==1)
            G(i,j) = 1/2;
        elseif (j-i==1)
            G(i,j) = 1/2;
        end
    end
end
for k = 1:n-1
    G(n-1,k) = -a(k)/(2*a(n));
```

```
end
x = eig(G);

Result:      >> x
                          x =   -0.9801     0.0643   1.0808   1.8350
```

6. How is this result connected to your answer from Question 3 of Problem Set 4?

   Recall question 3 - PS4 we have: $\phi_{n+1}(x) = x\phi(x) - \alpha_n\phi_n(x) - \beta_n\phi_{n-1}(x)$

   for $\alpha = 0 : \phi_{n+1}(x) = x\phi(x) - \beta_n\phi_{n-1}(x)$

   For Chebyshev: $T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x)$ for $k = 1, 2, ...$