Let’s try to improve the way we make and present plots. While for this assignment, this counts as extra credit, it will be required for the next assignment and the take-home exam.

- Plots need titles or captions, and axes need labels. If you draw several curves in one plot, use a legend to label them and use either different colors for each curve and/or different line styles (dotted, dashed etc). If you know that you will print your document in black and white, make sure that lines and dots can be distinguished in gray scale. If you email me your homework, I will print it on a black and white printer.
- Export plots as vector graphics (PDF or EPS) and include those graphics in your homework (either by printing them out directly or by including them in your write-up. Compressed formats such as JPEG or PNG lead to compression artifacts that look particularly ugly for line plots. Please no more screen shots or similar crimes!

1. **[Finding all roots of a polynomial, 2+1+1pt]** An efficient way to find individual roots of a polynomial is to use Newton’s method. However, as we have seen, Newton’s method requires an initialization close to the root one wants to find, and it can be difficult to find all roots of a polynomial. Luckily, one can use the relation between eigenvalues and polynomial roots to find all roots of a give polynomial. Let us consider a polynomial of degree \( n \) with leading coefficient 1:

\[
p(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} + x^n \quad \text{with } a_i \in \mathbb{R}.
\]

(a) Show that \( p(x) \) is the characteristic polynomial of the matrix (sometimes called a companion matrix for \( p \))

\[
A_p := \begin{bmatrix} 0 & -a_0 \\ 1 & -a_1 \\ \vdots \\ \vdots \\ 1 & -a_{n-1} \end{bmatrix} \in \mathbb{R}^{n \times n}.
\]

Thus, the roots of \( p(x) \) can be computed as the eigenvalues of \( A_p \) using the QR algorithm (as implemented, e.g., in MATLAB’s \texttt{eig} function).

(b) The matrix \( A_p \) is in Hessenberg form, and it is sparse and thus only requires \( O(n) \) storage. Use a simple numerical example to show that after one step of the QR-algorithm, the sparse structure of \( A_p \) is lost and \( O(n^2) \) storage is required.\(^1\)

\(^1\)This memory requirement as well as the required flops has only recently been improved. We had a speaker present these new results in a recent talk in our Numerical Analysis & Scientific Computing seminar (which is every Friday at 10am in WWH 1302. See \url{https://cs.nyu.edu/webapps/nasc_seminars} for upcoming talks.). The corresponding paper is \textit{Fast and backward stable computation of roots of polynomials} by J. Auretz, T. Mach, R. Vandebril and D. Watkins, which appeared in SIMAX 36(3), 2015.
(c) Let us consider Wilkinson’s polynomial $p_w(x)$ of order 15, i.e., a polynomial with the roots 1, 2, ..., 15:

$$p_w(x) = (x - 1) \cdot (x - 2) \cdot \ldots \cdot (x - 15).$$

The corresponding coefficients can be found using the `poly()` function. Use these coefficients in the matrix $A_p$ to find the original roots again, and compute their error. Compare with the build-in method (called `roots()`) for finding the roots of a polynomial. What’s more accurate, your or the build-in method?

2. [3-term recursion and orthogonal polynomials, 2+2pt] We consider the polynomial recursion $l_0(x) = 1$, $l_1(x) = 1 - x$, and

$$l_{k+1}(x) = \frac{2k + 1 - x}{k + 1}l_k(x) - \frac{k}{k + 1}l_{k-1}(x) \quad \text{for } k = 1, 2, \ldots$$

(a) Derive the polynomials for $l_2(x)$ and $l_3(x)$ from the recursion. Verify that

$$\int_0^\infty \exp(-t)l_i(t)l_j(t) \, dt = 0 \quad \text{for } 0 \leq i < j \leq 3.$$

Since this orthogonality relation can be shown to hold for all $i \neq j$, the polynomials $l_i()$ are orthogonal on $[0, \infty)$ with weight function $\omega(t) = \exp(-t)$.

(b) Write a function `my_l(k, x)`, which returns $l_k(x)$. The function should also allow vector inputs $x = (x_1, \ldots, x_n)$ and return $(l_k(x_1), \ldots, l_k(x_n))$. Your function should not derive the polynomials analytically, but just compute their value at the points $x$ using the recursion. Hand in a code listing, and plot graphs of the first several polynomials $l_i$ (these polynomials are called Laguerre polynomials).

3. [Interpolation with trigonometric functions, 4pt] For $N \geq 1$, we define the set of complex trigonometric polynomials of degree $\leq N - 1$ as

$$T_{N-1} := \left\{ \sum_{j=0}^{N-1} c_je^{ijt}, c_j \in \mathbb{C} \right\},$$

where $i$ denotes the imaginary unit. The corresponding (complex) interpolation problem is: Given pairwise distinct nodes $t_0, \ldots, t_{N-1} \in [0, 2\pi)$ and corresponding nodal values $f_0, \ldots, f_{N-1} \in \mathbb{C}$, find a trigonometric polynomial $p \in T_{N-1}$ such that $p(t_i) = f_i$, for $i = 0, \ldots, N - 1$.

\textsuperscript{2}For many MATLAB functions that do not use external libraries, you can see how they are implemented by typing `edit name_of_function`. Doing that for the `roots` function might help to answer this question.

\textsuperscript{3}Note that in this recursion the polynomials are not normalized as in the problems we discussed in class, where the leading coefficient was assumed to be 1. Such a (or another) normalization can easily be achieved by multiplication of $L_i(x)$ with an appropriate scaling factor.

\textsuperscript{4}Feel free to look up the values for the indefinite integrals $\int_0^\infty \exp(-t)t^k \, dx \ (k = 0, 1, 2, 3)$—I use Wolfram Alpha for looking up things like that: \url{http://www.wolframalpha.com/}.

\textsuperscript{5}Note that $e^{jt} = \cos(jt) + i \sin(jt)$.
(a) Show that there exists exactly one 
\( p \in T_{N-1} \), which solves this interpolation problem.

(b) Define \( W_N := e^{-\frac{2\pi i}{N}} \). Show that
\[
\frac{1}{N} \sum_{j=0}^{N-1} (W_N^{l-k})^j = \delta_{kl} := \begin{cases} 
1 & \text{if } k = l, \\
0 & \text{if } k \neq l,
\end{cases}
\]
for \( 0 \leq l, k \leq N - 1 \).

(c) Let us now choose the equidistant nodes \( t_k := \frac{2\pi k}{N} \) for \( k = 0, \ldots, N - 1 \). Show that the trigonometric polynomial that satisfies
\[ p(t_i) = f_i \] for \( i = 0, \ldots, N - 1 \) has the coefficients
\[
c_j = \frac{1}{N} \sum_{k=0}^{N-1} W_N^{jk} f_k.
\]

(d) For equidistant nodes, the linear map from \( \mathbb{C}^N \rightarrow \mathbb{C}^N \) defined by \( (f_0, \ldots, f_{N-1}) \mapsto (c_0, \ldots, c_{N-1}) \) is called the discrete Fourier transformation. How many (complex) floating point operations are required to compute this map?

4. **[Chebyshev polynomials, 1+1+1+1+1+2+2pt]** The recurrence relation for Chebyshev polynomials \( T_k \) is \( T_0(x) = 1, T_1(x) = x \) and \( T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \). Show from this relation that:

(a) For even \( k \), \( T_k \) is symmetric, i.e., \( T_k(-x) = T_k(x) \). For odd \( k \), \( T_k \) satisfies \( T_k(-x) = -T_k(x) \).

(b) By showing that \( T'_k(x) = \cos(k \arccos(x)) \) satisfies this 3-term recurrence relation, argue that that \( T(x) = T'(x) \). Note that this implies that \( |T_k(x)| \leq 1 \) for all \( x \).

(c) By showing that
\[
T''_k(x) = \frac{1}{2} \left( (x + \sqrt{x^2 - 1})^k + (x - \sqrt{x^2 - 1})^k \right)
\]
satisfies the same 3-term recurrence relation, argue that \( T''(x) = T(x) \). Note that differently from \( T'(x) \), the form \( T''(x) \) is defined for all \( x \in \mathbb{R} \).

(d) Show that the zeros of \( T_k(x) \) are given by
\[
x_j = \cos \left( \frac{(2j + 1)\pi}{2k} \right) \quad \text{for } j = 0, \ldots, k - 1.
\]

These \( x_j \) are called Chebyshev nodes (corresponding to the \( k \)-th Chebyshev polynomial).

(e) The Newton polynomial \( \omega_{n+1} \) that is based on Chebyshev nodes (i.e., the roots of \( T_{n+1} \)) satisfies
\[
|\omega_{n+1}(x)| \leq \frac{1}{2^n} \quad \text{for all } x \in [-1, 1].
\]

**Hint:** Think about what the difference is between the Newton polynomial with node points given by the roots of the Chebyshev polynomial \( T_{n+1} \), and the Chebyshev polynomial \( T_{n+1} \) itself. Note that it can be shown that the Chebyshev nodes minimize the pointwise bound given by (1), which is referred to as the min-max property of the Chebyshev nodes and explains their usefulness for interpolation.
(f) Show that this implies the following error estimate for the interpolation error $E_f(x) := f(x) - p_n(x)$, where $p_n \in P_n$ is a polynomial that interpolates a function $f \in C^{n+1}$ at the Chebyshev nodes $x_0, \ldots, x_n$:

$$|E_f(x)| \leq \frac{1}{2^n(n+1)!} \|f^{(n+1)}\|_\infty,$$

where for continuous functions $g$, $\|g\|_\infty = \|g\|_{C^n([-1,1])} := \max_{-1 \leq t \leq 1} |g(t)|$ is the supremum norm on the interval $[-1, 1]$.

(g) Compute and plot the approximation of the function $f(x) = 1/(1 + 12x^2)$ with a polynomial of order $N = 14$ on the interval $[-1, 1]$. Choose $N + 1$ equidistant nodes (including the points $\pm 1$) and the Chebyshev nodes given by the roots of $T_{N+1}(x)$ as interpolation points, and compare the results.$^6$ What do you observe?

(h) Repeat the Chebyshev point-based interpolation with at least two larger numbers $N$ of Chebyshev interpolation points, and approximate the maximal interpolation error.$^7$ Plot the maximal error as a function of $N$ and compare with the estimate (2) (without evaluating (2) explicitly, just in terms of how it changes as you change $N$). Use a logarithmic scale for plotting the error (in MATLAB, you can use semilogy). Can you spot a trend for the error?

5. **[Polynomial interpolation and error estimation, 1+1+1+2+2pt]** Let us interpolate the function $f : [0, 1] \to \mathbb{R}$ defined by $f(t) = \exp(3t)$ using the nodes $t_i = i/2$, $i = 0, 1, 2$ by a quadratic polynomial $p_2 \in P_2$.

(a) Use the monomial basis $1, t, t^2$ and compute (numerically) the coefficients $c_j \in \mathbb{R}$ such that $p_2(t) = \sum_{j=0}^{2} c_j t^j$.

(b) Give an alternative form for $p_2$ using Lagrange interpolation polynomials.

(c) Give yet another alternative form of $p_2$ using the Newton polynomial basis $\omega_0(t) = 1$, and $\omega_j(t) = \prod_{i=0}^{j-1} (t - t_i)$ for $j = 1, 2$. Compute the coefficients of $p_2$ in this basis using divided differences.

(d) Compare the exact interpolation error $E_f(t) := f(t) - p_2(t)$ at $t = 3/4$ with the estimate

$$|E_f(t)| \leq \frac{\|\omega_{n+1}\|_\infty}{(n + 1)!} \|f^{(n+1)}\|_\infty,$$

where $f^{(n+1)}$ is the $(n + 1)$st derivative of $f$, and $\| \cdot \|$ is the supremum norm for the interval $[0, 1]$.

(e) Find a (Hermite) polynomial $p_3 \in P_3$ that interpolates $f$ in $t_0, t_1, t_2$ and additionally satisfies $p_3'(t_3) = f'(t_3)$, where $t_2 = t_3 = 1$. Give the polynomial $p_3$ using the Newton basis.$^8$

---

$^6$You can use MATLAB’s polynomial interpolation functions *polyfit*.

$^7$You can do that by choosing a very fine mesh with, say, 10,000 uniformly distributed points $s_i$ in $[-1,1]$, and compute the maximum of $|E_f(s_i)|$ for all $i$.

$^8$You should only have to add a row to the derivation you did when you computed $p_2$ using the Newton basis. Note that since $t_2 = t_3$, you can use that the divided difference $[t_2 t_3] f = f'(t_2)$. Moreover, note that $[t_3] f = f(t_3) = f(t_2)$. 
6. **[Cubic spline construction—brute force version, 3pt]** Let us construct a cubic spline\(^9\) for the nodes \(t_0 = 0, t_1 = 1, t_2 = 2, t_3 = 3\). A cubic spline follows a cubic polynomial in each interval \(I^{(j)} := [t_j, t_{j+1}], \ j = 0, 1, 2\), and is twice continuously differentiable everywhere in \([t_0, t_3]\). To find the cubic spline, let us use a monomial basis in each of the intervals \(I^{(j)}\):

\[
s^{(j)}(t) = a_0^{(j)} + a_1^{(j)} t + a_2^{(j)} t^2 + a_3^{(j)} t^3, \quad \text{for} \ j = 0, 1, 2.
\]

Express the conditions the spline satisfies at the nodes \(t_i\) in terms of conditions for \(s^{(j)}\) and its derivatives, and derive the resulting 10 linear conditions for the 12 coefficients \(a_i^{(j)}\). To make this under-determined system uniquely solvable, either add zero conditions for the first or the second derivatives at \(t_0\) and \(t_3\). Let us now interpolate the function \(f(t) = \sqrt{t}\). Compute the spline coefficients by solving the resulting linear system\(^{10}\) for both choices of boundary conditions at the first and last node. Plot your results and compare with the build-in cubic spline interpolation (in MATLAB, you can use the \texttt{interp1} function, which has a ‘spline’ option—see the description of the function.) What conditions at the first and last node does the build-in function use?

---

\(^9\)Note that there are more efficient and stable ways to construct cubic splines, namely de Boor’s algorithm and its variants, which we haven’t had time to cover in class. The purpose of this problem is to use a straightforward algorithm that allows us basic experiments with spline interpolation.

\(^{10}\)Note that in this approach, one has to solve a linear system of size \(4(n+1)\) to find a spline with \(n+1\) nodes. De Boor’s algorithm avoids that global solve by using a better basis that again satisfies a recursion.