Mario L. Gutierrez Abed
364009832
mlg3843@rit.edu

Problem Set 2
Numerical Analysis I

10-05-2020

Problem 1. *Use Lagrangian interpolation and Newton's divided differences to find interpolating polynomials of the points* $(-1,0)$, $(2,1)$, $(3,1)$, $(5,2)$.

Solution. We now present interpolating polynomials for these points using both methods:

- **Lagrange Interpolation:** Let

$$\ell_i(x) = \prod_{j\neq i} \frac{x - x_j}{x_i - x_j} \qquad \text{for } i = 0, \dots, n. \tag{1}$$

(Note the peculiar property $\ell_i(x_j) = \delta_{ij}$.) Then the ***Lagrange polynomial*** interpolating points (x_i, y_i), for $i = 0, \dots, n$, is given by

$$^{\mathcal{L}}p_n(x) = \sum_{i=0}^{n} y_i \ell_i(x). \tag{2}$$

Thus, in the case at hand,

$$\begin{aligned} ^{\mathcal{L}}p_3(x) &= \sum_{i=0}^{3} y_i \ell_i(x) \\ &= 0 \cdot \frac{(x-2)(x-3)(x-5)}{(-1-2)(-1-3)(-1-5)} + 1 \cdot \frac{(x-(-1))(x-3)(x-5)}{(2-(-1))(2-3)(2-5)} \\ &\quad + 1 \cdot \frac{(x-(-1))(x-2)(x-5)}{(3-(-1))(3-2)(3-5)} + 2 \cdot \frac{(x-(-1))(x-2)(x-3)}{(5-(-1))(5-2)(5-3)} \\ &= \frac{x^3 - 7x^2 + 7x + 15}{9} - \frac{x^3 - 6x^2 + 3x + 10}{8} + \frac{x^3 - 4x^2 + x + 6}{18} \\ &= \frac{1}{24}\left(x^3 - 6x^2 + 11x + 18\right). \end{aligned} \tag{3}$$

- **Divided Differences:** In what follows we use $f[x_0 \dots x_n]$ to denote the coefficient of the x^n term in the (unique) polynomial that interpolates the $n+1$ points $(x_0, f(x_0)), \dots, (x_n, f(x_n))$.[1] These coefficients are given by the recursive relation

$$\begin{aligned} f[x_k] &= f(x_k) \\ f[x_k\ x_{k+1}] &= \frac{f[x_{k+1}] - f[x_k]}{x_{k+1} - x_k} \\ f[x_k\ x_{k+1}\ x_{k+2}] &= \frac{f[x_{k+1}\ x_{k+2}] - f[x_k\ x_{k+1}]}{x_{k+2} - x_k} \\ f[x_k\ x_{k+1}\ x_{k+2}\ x_{k+3}] &= \frac{f[x_{k+1}\ x_{k+2}\ x_{k+3}] - f[x_k\ x_{k+1}\ x_{k+2}]}{x_{k+3} - x_k}, \end{aligned} \tag{4}$$

and so on ... The ***Newton's divided difference formula*** for an n-degree polynomial interpolating $n+1$ points (x_i, y_i), for $i = 0, \dots, n$ is then given by

$$\begin{aligned} ^{\mathcal{N}}p_n(x) &= f[x_0] + \sum_{i=1}^{n} \left\{ f[x_0 \cdots x_i] \prod_{j=0}^{i-1} (x - x_j) \right\} \\ &= f[x_0] + f[x_0\ x_1](x - x_0) \\ &\qquad + f[x_0\ x_1\ x_2](x - x_0)(x - x_1) \\ &\qquad + f[x_0\ x_1\ x_2\ x_3](x - x_0)(x - x_1)(x - x_2) \\ &\qquad + \dots \\ &\qquad + f[x_0 \cdots x_n](x - x_0) \cdots (x - x_{n-1}). \end{aligned} \tag{5}$$

[1] Please note that my notation deviates from the one we are following in class, because my main programming languages are C++ and PYTHON, where the index count conventionally starts from 0.

The recursive nature of Eq. (4) allows arrangement into a convenient table form. For four points we have

$$\begin{array}{c|ccccccc} x_0 & f[x_0] & & & & & & \\ & & f[x_0\ x_1] & & & & & \\ x_1 & f[x_1] & & f[x_0\ x_1\ x_2] & & & & \\ & & f[x_1\ x_2] & & f[x_0\ x_1\ x_2\ x_3] & & & \\ x_2 & f[x_2] & & f[x_1\ x_2\ x_3] & & & & \\ & & f[x_2\ x_3] & & & & & \\ x_3 & f[x_3] & & & & & & \end{array} \tag{6}$$

The coefficients of the Newton polynomial Eq. (5) can then be read from the top edge of the triangle of this table.

Let us put all this machinery to good use for the case at hand. We can compute the coefficients using Eq. (4):

$$\begin{aligned} f[x_0\ x_1] &= \frac{f[x_1]-f[x_0]}{x_1-x_0} = \frac{1-0}{2-(-1)} = \frac{1}{3} \\ f[x_1\ x_2] &= \frac{f[x_2]-f[x_1]}{x_2-x_1} = \frac{1-1}{3-2} = 0 \\ f[x_2\ x_3] &= \frac{f[x_3]-f[x_2]}{x_3-x_2} = \frac{2-1}{5-3} = \frac{1}{2} \\ f[x_0\ x_1\ x_2] &= \frac{f[x_1\ x_2]-f[x_0\ x_1]}{x_2-x_0} = \frac{0-\frac{1}{3}}{3-(-1)} = -\frac{1}{12} \\ f[x_1\ x_2\ x_3] &= \frac{f[x_2\ x_3]-f[x_1\ x_2]}{x_3-x_1} = \frac{\frac{1}{2}-0}{5-2} = \frac{1}{6} \\ f[x_0\ x_1\ x_2\ x_3] &= \frac{f[x_1\ x_2\ x_3]-f[x_0\ x_1\ x_2]}{x_3-x_0} = \frac{\frac{1}{6}-\left(-\frac{1}{12}\right)}{5-(-1)} = \frac{1}{24}. \end{aligned}$$

Or we could have easily used the table form, in which calculations are much faster:

$$\begin{array}{c|cccc} -1 & 0 & & & \\ & & \frac{1}{3} & & \\ 2 & 1 & & -\frac{1}{12} & \\ & & 0 & & \frac{1}{24} \\ 3 & 1 & & \frac{1}{6} & \\ & & \frac{1}{2} & & \\ 5 & 2 & & & \end{array} \tag{7}$$

Reading the coefficients off the top of the triangle and plugging them back into Eq. (5), we get

$$\begin{aligned} {}^{\mathcal{N}}p_3(x) &= f[x_0] + f[x_0\ x_1](x-x_0) \\ &\qquad + f[x_0\ x_1\ x_2](x-x_0)(x-x_1) \\ &\qquad + f[x_0\ x_1\ x_2\ x_3](x-x_0)(x-x_1)(x-x_2) \\ &= 0 + \frac{1}{3}(x-(-1)) \\ &\quad + \left(-\frac{1}{12}\right)(x-(-1))(x-2) \\ &\quad + \frac{1}{24}(x-(-1))(x-2)(x-3) \\ &= \frac{1}{3}(x+1) - \frac{1}{12}(x+1)(x-2) + \frac{1}{24}(x+1)(x-2)(x-3) \\ &= \frac{1}{24}\left(x^3 - 6x^2 + 11x + 18\right). \end{aligned} \tag{8}$$

We see that the polynomials ${}^{\mathcal{L}}p_3$ and ${}^{\mathcal{N}}p_3$ are identical, which must be true since, according to the ***Main Theorem of Polynomial Interpolation***, if $(x_0, y_0), \ldots, (x_n, y_n)$ are $n+1$ points in the plane with distinct x_i, then there exists *one and only one* polynomial p of degree n or less that satisfies $p(x_i) = y_i$, for $i = 0, \ldots, n$. □

Problem 2. *Let $P(x)$ be the degree 9 polynomial that takes the value 112 at $x = 1$, takes the value 2 at $x = 10$ and equals 0 for $x = 2, \ldots, 9$. Calculate $P(0)$.*

Solution. Let us find the polynomial using Lagrange interpolation; using Eq. (2), we have

$$^{\mathcal{L}}p_9(x) = \sum_{i=0}^{9} y_i \ell_i(x) = y_0 \ell_0(x) + y_9 \ell_9(x). \tag{9}$$

Note that we are only left with two terms in the summation, since $y_i = 0$ for $i = 1, \ldots, 8$, by construction. Thus we are left with $y_0 = 112$ and $y_9 = 2$, and

$$\ell_0(x) = \prod_{j \neq 0} \frac{x - x_j}{x_0 - x_j} = \frac{x-2}{1-2} \cdot \frac{x-3}{1-3} \cdots \frac{x-10}{1-10} = -\frac{1}{9!} \prod_{k=2}^{10} (x-k)$$

$$\ell_9(x) = \prod_{j \neq 9} \frac{x - x_j}{x_9 - x_j} = \frac{x-1}{10-1} \cdot \frac{x-2}{10-2} \cdots \frac{x-9}{10-9} = \frac{1}{9!} \prod_{k=1}^{9} (x-k).$$

Plugging this back into Eq. (9), we get

$$^{\mathcal{L}}p_9(x) = -112 \frac{1}{9!} \prod_{k=2}^{10} (x-k) + 2 \frac{1}{9!} \prod_{k=1}^{9} (x-k). \tag{10}$$

Evaluating this polynomial at $x = 0$ yields

$$\begin{aligned} ^{\mathcal{L}}p_9(0) &= -112 \frac{1}{9!} \prod_{k=2}^{10} (-k) + 2 \frac{1}{9!} \prod_{k=1}^{9} (-k) \\ &= -112 \left(-\frac{10!}{9!} \right) + 2 \left(-\frac{9!}{9!} \right) \\ &= 1120 - 2 \\ &= 1118. \end{aligned}$$

□

Problem 3. *Given the data points $(1, 0)$, $(2, \log 2)$, $(4, \log 4)$, find the degree-2 interpolating polynomial and use it to approximate $\log 3$. Give an error bound and compare the actual error to your error bound.*

Solution. We use divided differences in its simple and elegant form:

$$\begin{array}{c|ccc} 1 & 0 & & \\ & & \log 2 & \\ 2 & \log 2 & & -\frac{\log 2}{6} \\ & & \frac{\log 2}{2} & \\ 4 & \log 4 & & \end{array} \tag{11}$$

Reading the coefficients off the top of the triangle and plugging them back into Eq. (5), we get

$$\begin{aligned} ^{\mathcal{N}}p_2(x) &= f[x_0] + f[x_0\ x_1](x - x_0) + f[x_0\ x_1\ x_2](x - x_0)(x - x_1) \\ &= 0 + \log 2(x-1) - \frac{\log 2}{6}(x-1)(x-2) \\ &= -\frac{\log 2}{6} \left(x^2 - 9x + 8 \right). \end{aligned} \tag{12}$$

In order to approximate $\log 3$, we now evaluate this polynomial at $x = 3$:

$$^{\mathcal{N}}p_2(3) = -\frac{\log 2}{6} \left(3^2 - 9(3) + 8 \right) = \frac{5 \log 2}{3}. \tag{13}$$

To find the error bound we use the following interpolation error theorem:

Theorem 3.4 (Sauer's)

Assume that $p(x)$ is the (degree n or less) interpolating polynomial fitting the $n+1$ points $(x_0, y_0), \dots, (x_n, y_n)$. The interpolation error for some function f is given by

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i), \tag{14}$$

where ξ lies between the smallest and largest of the numbers $x, x_0, \dots, x_n$.

Now, using Eq. (14), with $f(x) = \log x$, we have

$$\begin{aligned} \log x - {}^{\mathcal{N}}p_2(x) &= \frac{\log^{(3)}(\xi)}{3!} \prod_{i=0}^{2} (x - x_i) \\ &= \frac{2}{3 \cdot 2\xi^3}(x-1)(x-2)(x-4) \\ &= (x-1)(x-2)(x-4)\frac{1}{3\xi^3}, \end{aligned}$$

where ξ lies between the smallest and largest of the numbers $\{x, 1, 4\}$. Since we are interested in the error at $x = 3$, we have $\xi \in (1, 4)$. Thus,

$$\left| \log 3 - {}^{\mathcal{N}}p_2(3) \right| = \left| -2\frac{1}{3\xi^3} \right|.$$

The largest possible error we can get is when ξ is smallest, i.e., when $\xi \to 1$. Hence,

$$\left| \log 3 - {}^{\mathcal{N}}p_2(3) \right| \leq \frac{2}{3} \tag{15}$$

is an upper bound for the error. Now, the actual error (to 10 decimals precision) is

$$\left| \log 3 - {}^{\mathcal{N}}p_2(3) \right| = \left| \log 3 - \frac{5 \log 2}{3} \right| \approx |-0.05663301227| = 0.05663301227 \ll \frac{2}{3}. \tag{16}$$

This shows that our upper bound (15) is way too generous. Sauer does warn us that in order to have smaller errors we're better off evaluating far from one of the endpoints. For instance, if we take $\xi = 2$, then the error upper bound we get would be

$$\frac{2}{3 \cdot 2^3} \approx 0.083,$$

which is much closer to the actual error given by (16). □

Problem 4. *Find c in the following cubic splines. Which of the three end conditions (natural, parabolically-terminated, or not-a-knot), if any, are satisfied?*

$$S(x) = \begin{cases} 4 - \frac{11}{4}x + \frac{3}{4}x^3 & \text{if } x \in [0, 1]; \\ 2 - \frac{1}{2}(x-1) + c(x-1)^2 - \frac{3}{4}(x-1)^3 & \text{if } x \in [1, 2]. \end{cases} \tag{17a}$$

$$\widehat{S}(x) = \begin{cases} 3 - 9x + 4x^2 & \text{if } x \in [0, 1]; \\ -2 - (x-1) + c(x-1)^2 & \text{if } x \in [1, 2]. \end{cases} \tag{17b}$$

$$\widetilde{S}(x) = \begin{cases} -2 - \frac{3}{2}x + \frac{7}{2}x^2 - x^3 & \text{if } x \in [0, 1]; \\ -1 + c(x-1) + \frac{1}{2}(x-1)^2 - (x-1)^3 & \text{if } x \in [1, 2]; \\ 1 + \frac{1}{2}(x-2) - \frac{5}{2}(x-2)^2 - (x-2)^3 & \text{if } x \in [2, 3]. \end{cases} \tag{17c}$$

Solution. We start with Eq. (17a). Using Eq. (22) and notation from the next problem, we have

$$\begin{aligned} d_0 &= \frac{c_1 - c_0}{3\,{}^x\Delta_0} \\ \frac{3}{4} &= \frac{c - 0}{3 \cdot 1} \\ c &= \frac{9}{4}. \end{aligned}$$

Now let's test for the endpoint conditions:

$$\begin{aligned} S_0(0)'' &= \frac{9}{2} \cdot 0 = 0; \\ S_1(2)'' &= 2 \cdot \frac{9}{4} - \frac{9}{2} \cdot (2 - 1) = 0. \end{aligned}$$

Hence, we conclude that the natural spline condition is satisfied. $\surd$

Moving on to Eq. (17b),

$$\begin{aligned} d_0 &= \frac{c_1 - c_0}{3\,{}^x\Delta_0} \\ 0 &= \frac{c - 4}{3 \cdot 1} \\ c &= 4. \end{aligned}$$

Since the degree of both $\widehat{S}_0$ and $\widehat{S}_1$ is at most 2, we conclude that the spline is parabolically-terminated. However, it is curious to note that, if we evaluate the second derivatives at the endpoints, we get the curvature-adjusted condition:

$$\begin{aligned} \widehat{S}_0(0)'' &= 8; \\ \widehat{S}_1(2)'' &= 2 \cdot 4 = 8. \end{aligned}$$

(Question for Dr. Khan: What's going on? Is it possible for a spline to satisfy more than one endpoint condition?) $\surd$

Lastly, we tackle Eq. (17c), this time using Eq. (23)

$$\begin{aligned} b_1 &= \frac{{}^y\Delta_1}{{}^x\Delta_1} - \frac{{}^x\Delta_1}{3}\left(c_2 + 2c_1\right) \\ c &= \frac{3 - 2}{2 - 1} - \frac{2 - 1}{3}\left((-1) + 2(-1)\right) \\ c &= 1. \end{aligned}$$

From just eyeballing this spline, it is not hard to guess that it'll be a not-a-knot spline; let us test that:

$$\begin{aligned} \widetilde{S}_0(1)''' &= -6 = \widetilde{S}_1(1)'''; \\ \widetilde{S}_1(2)''' &= -6 = \widetilde{S}_2(2)'''. \end{aligned}$$

Hence, the not-a-knot condition is indeed satisfied. $\surd$ $\square$

Problem 5. *Take four data points* (x_i, y_i), *with* $i = 0, \ldots, 3$. *Give detailed procedure for computing cubic splines with natural, curvature-adjusted, clamped, parabolically-terminated and not-a-knot endpoint conditions. Check your results for the data points* $(0, 3)$, $(1, 5)$, $(2, 4)$, *and* $(3, 1)$.

Solution. A ***cubic spline*** $S(x)$ through $n + 1$ data points $(x_0, y_0), \ldots, (x_n, y_n)$ is a set of cubic polynomials S_i, with $0 \le i \le n - 1$, given by [2]

$$\begin{aligned} S_0(x) &= y_0 + b_0(x - x_0) + c_0(x - x_0)^2 + d_0(x - x_0)^3 \qquad \text{for } x \in [x_0, x_1] \\ &\ \ \vdots \\ S_{n-1}(x) &= y_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + d_{n-1}(x - x_{n-1})^3 \qquad \text{for } x \in [x_{n-1}, x_n], \end{aligned} \tag{18}$$

where $\{b_i, c_i, d_i\}$ are coefficients that need to be determined. These polynomials need to satisfy the following four properties:

[2] I decided to explain the procedure for the more general case of $n + 1$ data points; we get the desired result for this particular exercise by setting $n = 3$.

- **Property I:** $S_i(x_i) = y_i$ for $i = 0, \ldots, n-1$.
- **Property II:** $S_i(x_{i+1}) = y_{i+1}$ for $i = 0, \ldots, n-1$.
- **Property III:** $S'_{i-1}(x_i) = S'_i(x_i)$ for $i = 1, \ldots, n-1$.
- **Property IV:** $S''_{i-1}(x_i) = S''_i(x_i)$ for $i = 1, \ldots, n-1$.

Hence, constructing a spline from a set of data points means finding the coefficients $\{b_i, c_i, d_i\}$ that satisfy these four properties stated above.

Property I follows easily from just plugging $x = x_i$ into Eq. (18). Property II, on the other hand, yields n independent equations, one for each i, that need to be satisfied by the coefficients:

$$\begin{aligned} y_1 &= S_0(x_1) = y_0 + b_0(x_1 - x_0) + c_0(x_1 - x_0)^2 + d_0(x_1 - x_0)^3 \\ &\vdots \\ y_n &= S_{n-1}(x_n) = y_{n-1} + b_{n-1}(x_n - x_{n-1}) + c_{n-1}(x_n - x_{n-1})^2 + d_{n-1}(x_n - x_{n-1})^3. \end{aligned} \tag{19}$$

Similarly, Property III yields $n-1$ additional equations:

$$\begin{aligned} 0 &= S'_0(x_1) - S'_1(x_1) = b_0 - b_1 + 2c_0(x_1 - x_0) + 3d_0(x_1 - x_0)^2 \\ &\vdots \\ 0 &= S'_{n-2}(x_{n-1}) - S'_{n-1}(x_{n-1}) = b_{n-2} - b_{n-1} + 2c_{n-2}(x_{n-1} - x_{n-2}) + 3d_{n-2}(x_{n-1} - x_{n-2})^2. \end{aligned} \tag{20}$$

And a further $n-1$ equations from Property IV:

$$\begin{aligned} 0 &= S''_0(x_1) - S''_1(x_1) = 2(c_0 - c_1) + 6d_0(x_1 - x_0) \\ &\vdots \\ 0 &= S''_{n-2}(x_{n-1}) - S''_{n-1}(x_{n-1}) = 2(c_{n-2} - c_{n-1}) + 6d_{n-2}(x_{n-1} - x_{n-2}). \end{aligned} \tag{21}$$

Hence, in total, we have

$$\overbrace{n}^{\text{From Prop. II}} + \overbrace{n-1}^{\text{From Prop. III}} + \overbrace{n-1}^{\text{From Prop. IV}} = 3n - 2.$$

Moreover, there are 3 coefficients $\{b_i, c_i, d_i\}$ on each polynomial S_i, and there are n of the latter; thus we have a total of $3n$ coefficients. In other words, we have an underdetermined system of equations, since we have

$$3n \text{ unknowns} \quad \& \quad 3n - 2 \text{ equations}.$$

Hence we have infinitely many solutions; i.e., infinitely many cubic splines passing through the arbitrary set of $n+1$ data points $(x_0, y_0), \ldots, (x_n, y_n)$. If, however, we were to impose two further constraints (i.e., equations) we would get a fully determined system (same number of unknowns and equations). This is precisely how we classify splines, depending on which two constraints we add to the system (these extra couple of constraints are usually applied to the left and right ends of the spline, so they are called ***endpoint conditions***). The classification goes as follows: A spline is said to be

- ***natural*** if $S''_0(x_0) = 0$ and $S''_{n-1}(x_n) = 0$;
- ***curvature-adjusted*** if $S''_0(x_0) = \kappa_0$ and $S''_{n-1}(x_n) = \kappa_n$, where κ_0 and κ_n are user-defined, nonzero values;
- ***clamped*** if $S'_0(x_0) = \nu_0$ and $S'_{n-1}(x_n) = \nu_n$, where ν_0 and ν_n are user-defined, nonzero values;
- ***parabolically-terminated*** if $\deg S_0 \leq 2$ and $\deg S_{n-1} \leq 2$. That is, the first and last polynomials of the spline –S_0 and S_{n-1}, respectively– are forced to have degree at most 2. This can be enforced by setting $d_0 = d_{n-1} = 0$.
- ***not-a-knot*** if $S'''_0(x_1) = S'''_1(x_1)$ and $S'''_{n-2}(x_{n-1}) = S'''_{n-1}(x_{n-1})$. Equivalently, we may set $d_0 = d_1$ and $d_{n-2} = d_{n-1}$, Since S_0 and S_1 are polynomials of degree ≤ 3, requiring their third derivatives to agree at x_1, while their zeroth, first, and second derivatives already agree there, causes S_0 and S_1 to be identical cubic polynomials. Thus, x_1 is not needed as a base point: the spline is given by the same formula $S_0 = S_1$ on the entire interval $[x_0, x_2]$. The same reasoning shows that $S_{n-2} = S_{n-1}$, so both x_1 and x_{n-1} are "no longer knots."

Let us now put together all of this machinery and write down the general solution for each type of spline:

★ **`Natural Spline:`** Now that we have $3n$ equations to solve $3n$ unkowns, we could use some linear algebra solver in C++ (or whatever language of preference). However, it turns out that we can drastically simplify the system by decoupling the equations first. Let us introduce the notation

$$\begin{aligned} {}^x\Delta_i &= x_{i+1} - x_i \\ {}^y\Delta_i &= y_{i+1} - y_i. \end{aligned}$$

Now consider (21), for any $i = 1, \ldots, n-1$:

$$0 = 2(c_i - c_{i+1}) + 6d_i\,{}^x\Delta_i.$$

Isolating d_i, we get

$$d_i = \frac{c_{i+1} - c_i}{3\,{}^x\Delta_i}. \tag{22}$$

Substituting this expression into (19) and solving for b_i, we get

$$b_i = \frac{{}^y\Delta_i}{{}^x\Delta_i} - \frac{{}^x\Delta_i}{3}\left(c_{i+1} + 2c_i\right). \tag{23}$$

Plugging both of these expressions, (22)–(23), into (20), we get $n-1$ equations in $c_0, \ldots, c_n$:

$$\begin{gathered} {}^x\Delta_0 c_0 + 2\left({}^x\Delta_0 + {}^x\Delta_1\right) c_1 + {}^x\Delta_1 c_2 = 3\left(\frac{{}^y\Delta_1}{{}^x\Delta_1} - \frac{{}^y\Delta_0}{{}^x\Delta_0}\right) \\ \vdots \\ {}^x\Delta_{n-2} c_{n-2} + 2\left({}^x\Delta_{n-2} + {}^x\Delta_{n-1}\right) c_{n-1} + {}^x\Delta_{n-1} c_n = 3\left(\frac{{}^y\Delta_{n-1}}{{}^x\Delta_{n-1}} - \frac{{}^y\Delta_{n-2}}{{}^x\Delta_{n-2}}\right). \end{gathered} \tag{24}$$

Adding the two additional constraints that pertain to natural splines, namely

$$S_0''(x_0) = 0 \quad\Longrightarrow\quad 2c_0 = 0; \tag{25a}$$

$$S_{n-1}''(x_n) = 0 \quad\Longrightarrow\quad 2c_n = 0, \tag{25b}$$

we end up with $n+1$ equations for the $n+1$ unknowns $c_0, \ldots, c_n$. In matrix form, this looks like

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & & \\ {}^x\Delta_0 & 2({}^x\Delta_0 + {}^x\Delta_1) & {}^x\Delta_1 & \ddots & & \\ 0 & {}^x\Delta_1 & 2({}^x\Delta_1 + {}^x\Delta_2) & {}^x\Delta_2 & & \\ \vdots & \ddots & \ddots & \ddots & & \\ & & & {}^x\Delta_{n-2} & 2({}^x\Delta_{n-2} + {}^x\Delta_{n-1}) & {}^x\Delta_{n-1} \\ & & & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ \\ \vdots \\ \\ c_n \end{pmatrix} = \begin{pmatrix} 0 \\ 3\left(\frac{{}^y\Delta_1}{{}^x\Delta_1} - \frac{{}^y\Delta_0}{{}^x\Delta_0}\right) \\ \vdots \\ 3\left(\frac{{}^y\Delta_{n-1}}{{}^x\Delta_{n-1}} - \frac{{}^y\Delta_{n-2}}{{}^x\Delta_{n-2}}\right) \\ 0 \end{pmatrix} \tag{26}$$

Once we obtain the c_i from (26), the d_i and b_i follow from (22) and (23), respectively.

Now, FINALLY, we construct a natural spline for the provided data points $(0,3)$, $(1,5)$, $(2,4)$, and $(3,1)$. For four data points ($n = 3$), the matrix equation (26) reduces to

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ {}^x\Delta_0 & 2({}^x\Delta_0 + {}^x\Delta_1) & {}^x\Delta_1 & 0 \\ 0 & {}^x\Delta_1 & 2({}^x\Delta_1 + {}^x\Delta_2) & {}^x\Delta_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 3\left(\frac{{}^y\Delta_1}{{}^x\Delta_1} - \frac{{}^y\Delta_0}{{}^x\Delta_0}\right) \\ 3\left(\frac{{}^y\Delta_2}{{}^x\Delta_2} - \frac{{}^y\Delta_1}{{}^x\Delta_1}\right) \\ 0 \end{pmatrix}. \tag{27}$$

In the case at hand, we have

$$
\begin{aligned}
{}^x\Delta_0 &= x_1 - x_0 = 1 - 0 = 1 \\
{}^x\Delta_1 &= x_2 - x_1 = 2 - 1 = 1 \\
{}^x\Delta_2 &= x_3 - x_2 = 3 - 2 = 1 \\
{}^y\Delta_0 &= y_1 - y_0 = 5 - 3 = 2 \\
{}^y\Delta_1 &= y_2 - y_1 = 4 - 5 = -1 \\
{}^y\Delta_2 &= y_3 - y_2 = 1 - 4 = -3.
\end{aligned}
$$

Plugging this into (27), we have

$$
\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ -9 \\ -6 \\ 0 \end{pmatrix}, \tag{28}
$$

which has solution

$$
\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ -2 \\ -1 \\ 0 \end{pmatrix}. \tag{29}
$$

As for the remaining coefficients, we use Eqs. (22)–(23) and plug in the values just acquired for the c_i:

$$
\begin{aligned}
d_0 &= \frac{c_1 - c_0}{3\,{}^x\Delta_0} = \frac{-2-0}{3\cdot 1} = -\frac{2}{3} \\
d_1 &= \frac{c_2 - c_1}{3\,{}^x\Delta_1} = \frac{-1-(-2)}{3\cdot 1} = \frac{1}{3} \\
d_2 &= \frac{c_3 - c_2}{3\,{}^x\Delta_2} = \frac{0-(-1)}{3\cdot 1} = \frac{1}{3} \\
b_0 &= \frac{{}^y\Delta_0}{{}^x\Delta_0} - \frac{{}^x\Delta_0}{3}\left(c_1 + 2c_0\right) = \frac{2}{1} - \frac{1}{3}(-2+2\cdot 0) = \frac{8}{3} \\
b_1 &= \frac{{}^y\Delta_1}{{}^x\Delta_1} - \frac{{}^x\Delta_1}{3}\left(c_2 + 2c_1\right) = \frac{-1}{1} - \frac{1}{3}(-1+2\cdot(-2)) = \frac{2}{3} \\
b_2 &= \frac{{}^y\Delta_2}{{}^x\Delta_2} - \frac{{}^x\Delta_2}{3}\left(c_3 + 2c_2\right) = \frac{-3}{1} - \frac{1}{3}(0+2\cdot(-1)) = -\frac{7}{3}.
\end{aligned}
$$

Hence, our natural spline is given by

$$
\boxed{S(x) = \begin{cases} 3 + \frac{8}{3}x - \frac{2}{3}x^3 & \text{if } x \in [0,1]; \\ 5 + \frac{2}{3}(x-1) - 2(x-1)^2 + \frac{1}{3}(x-1)^3 & \text{if } x \in [1,2]; \\ 4 - \frac{7}{3}(x-2) - (x-2)^2 + \frac{1}{3}(x-2)^3 & \text{if } x \in [2,3]. \end{cases}} \tag{30}
$$

(This is the only spline I will fully derive; for the remaining cases I will simply mention the difference in procedure and will skip all the messy algebra. It's obvious that if you know how to derive one, you know how to derive the rest. . . it's just tedious algebra.)

★ **`Curvature-Adjusted Spline:`** The only change here is that now we have nonzero values at c_0 and c_n, namely

$$
S_0''(x_0) = \kappa_0 \quad \Longrightarrow \quad 2c_0 = \kappa_0; \tag{31a}
$$

$$
S_{n-1}''(x_n) = \kappa_n \quad \Longrightarrow \quad 2c_n = \kappa_n. \tag{31b}
$$

In turn, the matrix equation (27) changes to

$$
\begin{pmatrix} 2 & 0 & 0 & 0 \\ {}^x\Delta_0 & 2({}^x\Delta_0 + {}^x\Delta_1) & {}^x\Delta_1 & 0 \\ 0 & {}^x\Delta_1 & 2({}^x\Delta_1 + {}^x\Delta_2) & {}^x\Delta_2 \\ 0 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} \kappa_0 \\ 3\left(\frac{{}^y\Delta_1}{{}^x\Delta_1} - \frac{{}^y\Delta_0}{{}^x\Delta_0}\right) \\ 3\left(\frac{{}^y\Delta_2}{{}^x\Delta_2} - \frac{{}^y\Delta_1}{{}^x\Delta_1}\right) \\ \kappa_3 \end{pmatrix}. \tag{32}
$$

Hence, our system is

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} \kappa_0 \\ -9 \\ -6 \\ \kappa_3 \end{pmatrix}, \tag{33}$$

which has solution

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\kappa_0 \\ \frac{1}{30}\kappa_3 - \frac{2}{15}\kappa_0 - 2 \\ -\frac{2}{15}\kappa_3 + \frac{1}{30}\kappa_0 - 1 \\ \frac{1}{2}\kappa_3 \end{pmatrix}. \tag{34}$$

Determining the b_i and d_i then follow as we did above.

★ **`Clamped Spline:`** This time it is the first derivatives that have user-defined values. Using (22)–(23), we can write the two extra constraints as

$$S_0'(x_0) = \nu_0 \quad \Longrightarrow \quad 2\,{}^x\Delta_0\, c_0 + {}^x\Delta_0\, c_1 = 3\left(\frac{{}^y\Delta_0}{{}^x\Delta_0} - \nu_0\right); \tag{35a}$$

$$S_{n-1}'(x_n) = \nu_n \quad \Longrightarrow \quad {}^x\Delta_{n-1}\, c_{n-1} + 2\,{}^x\Delta_{n-1}\, c_n = 3\left(\nu_n - \frac{{}^y\Delta_{n-1}}{{}^x\Delta_{n-1}}\right). \tag{35b}$$

In turn, the matrix equation (27) changes to

$$\begin{pmatrix} 2\,{}^x\Delta_0 & {}^x\Delta_0 & 0 & 0 \\ {}^x\Delta_0 & 2({}^x\Delta_0 + {}^x\Delta_1) & {}^x\Delta_1 & 0 \\ 0 & {}^x\Delta_1 & 2({}^x\Delta_1 + {}^x\Delta_2) & {}^x\Delta_2 \\ 0 & 0 & {}^x\Delta_2 & 2{}^x\Delta_2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 3\left(\frac{{}^y\Delta_0}{{}^x\Delta_0} - \nu_0\right) \\ 3\left(\frac{{}^y\Delta_1}{{}^x\Delta_1} - \frac{{}^y\Delta_0}{{}^x\Delta_0}\right) \\ 3\left(\frac{{}^y\Delta_2}{{}^x\Delta_2} - \frac{{}^y\Delta_1}{{}^x\Delta_1}\right) \\ 3\left(\nu_3 - \frac{{}^y\Delta_2}{{}^x\Delta_2}\right) \end{pmatrix}. \tag{36}$$

Solving for $\{c_i, b_i, d_i\}$ follows as above. (We may pick some arbitrary values for the user-defined variables to simplify the algebra).

★ **`Parabolically-Terminated Spline:`** Using (22), we can write the two extra constraints as

$$d_0 = 0 \quad \Longrightarrow \quad c_0 = c_1 \tag{37a}$$

$$d_{n-1} = 0 \quad \Longrightarrow \quad c_{n-1} = c_n. \tag{37b}$$

This turns the matrix equation (26) into

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ {}^x\Delta_0 & 2({}^x\Delta_0 + {}^x\Delta_1) & {}^x\Delta_1 & 0 \\ 0 & {}^x\Delta_1 & 2({}^x\Delta_1 + {}^x\Delta_2) & {}^x\Delta_2 \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 3\left(\frac{{}^y\Delta_1}{{}^x\Delta_1} - \frac{{}^y\Delta_0}{{}^x\Delta_0}\right) \\ 3\left(\frac{{}^y\Delta_2}{{}^x\Delta_2} - \frac{{}^y\Delta_1}{{}^x\Delta_1}\right) \\ 0 \end{pmatrix}. \tag{38}$$

Solving for $\{c_i, b_i, d_i\}$ follows as above.

★ **`Not-a-Knot Spline:`** The two extra constraints are now

$$S_0'''(x_1) = S_1'''(x_1) \implies d_0 = d_1 \implies {}^x\Delta_1\, c_0 - ({}^x\Delta_0 + {}^x\Delta_1)\, c_1 + {}^x\Delta_0\, c_2 = 0 \tag{39a}$$

$$S_{n-2}'''(x_{n-1}) = S_{n-1}'''(x_{n-1}) \implies d_{n-2} = d_{n-1} \implies {}^x\Delta_{n-1}\, c_{n-2} - \left({}^x\Delta_{n-2} + {}^x\Delta_{n-1}\right) c_{n-1} + {}^x\Delta_{n-2}\, c_n = 0. \tag{39b}$$

(The right-most implications come from using Eq. (22).) This turns the matrix equation (26) into

$$\begin{pmatrix} {}^x\Delta_1 & -\left({}^x\Delta_0 + {}^x\Delta_1\right) & {}^x\Delta_0 & 0 \\ {}^x\Delta_0 & 2({}^x\Delta_0 + {}^x\Delta_1) & {}^x\Delta_1 & 0 \\ 0 & {}^x\Delta_1 & 2({}^x\Delta_1 + {}^x\Delta_2) & {}^x\Delta_2 \\ 0 & {}^x\Delta_2 & -\left({}^x\Delta_1 + {}^x\Delta_2\right) & {}^x\Delta_1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 3\left(\frac{{}^y\Delta_1}{{}^x\Delta_1} - \frac{{}^y\Delta_0}{{}^x\Delta_0}\right) \\ 3\left(\frac{{}^y\Delta_2}{{}^x\Delta_2} - \frac{{}^y\Delta_1}{{}^x\Delta_1}\right) \\ 0 \end{pmatrix}. \tag{40}$$

Solving for $\{c_i, b_i, d_i\}$ follows as above.

$\square$

Problem 6. *For the* Runge function $R(x)$, *use the given conditions to interpolate. Then compare the interpolation with the actual function* $R(x)$*:*

a) *Use uniform nodes* $x_i = -5 + i$, *with* $i = 0, \ldots, 10$, *and plot its Newton interpolation of degree* 10.

b) *Use uniform nodes* $x_i = 5\cos\left(\frac{(2i+1)\pi}{42}\right)$, *with* $i = 0, \ldots, 20$, *and plot its Lagrange polynomial interpolation of degree* 20.

c) *Use uniform nodes* $x_i = -5 + i$, *with* $i = 0, \ldots, 10$, *and plot its piecewise linear function interpolation.*

d) *Use uniform nodes* $x_i = -5 + i$, *with* $i = 0, \ldots, 10$, *and plot its piecewise cubic function interpolation.*

Solution. The Runge function is given by

$$R(x) = \frac{1}{1 + 25x^2} \qquad \text{for } x \in [-1, 1]. \tag{41}$$

I wrote the following C++ code for both the Newton and Lagrangian interpolations:

```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <algorithm>
#include <iterator>
#include <numeric>
#include <array>

using namespace std;



// Represent a data point corresponding to x and y = f(x)
struct Data
{
    double x, y;
};

//Define the Runge function
double Runge (double &x){
    return 1.0/(1.0 + 25.0 * pow(x,2));
}

//Define the Lagrange interpolation, to be evaluated at some point xi
double LagrangeInt(Data f[], double xi, const int n){
    double result {};
    double yval {};

    for (int i {0}; i <= n; i++){
        yval = f[i].y;      //set yval equal to y value of the ith data point
        for (int j {0}; j <= n; j++)
        {
            if (j!=i)
                yval = yval * (xi - f[j].x)/(f[i].x - f[j].x);
        }
        result += yval;
    }

    return result;
}
```

```
//Function to find the coefficients at the top of the triangle in Newton's code
void f_coeff(Data f[], int n, vector <vector <double>> &coeff){
   for (int j {0}; j <= n; j++) {
       for (int i {0}; i <= n-j; i++){
           if (j == 0){
               coeff.at(i).at(j) = f[i].y;
               coeff.at(i).push_back(coeff.at(i).at(j));
           }
           else{
               coeff.at(i).at(j) = ( coeff.at(i+1).at(j-1) - coeff.at(i).at(j-1) )/
               ( f[i+j].x - f[i].x );
               coeff.at(i).push_back(coeff.at(i).at(j));
           }
       }
   }
}



// Function to find the product term to be used in Newton's code
double product(Data f[], double x, int i){
    double prod {1.0};
    for (int j {0}; j <= i-1; j++) {
        prod = prod * (x - f[j].x);
    }
    return prod;
}


//Define the Newton interpolation, to be evaluated at some point xi
double NewtonInt(Data f[], double x, int n){
    vector <vector <double>> coeff {};
    vector <double> coeff_vec {};

    for (int j {0}; j <= n; j++) {          /*initialization of vector to size it as
                                             an (n-j)xj matrix  */
        for (int i {0}; i <= n-j; i++){
            coeff_vec.push_back(0.0);
        }
        coeff.push_back(coeff_vec);
    }


    f_coeff(f, n, coeff);

    double result {};

    for (int j {0}; j <= n; j++){
        if (j == 0)
            result = coeff.at(0).at(0);
        else
            result +=  coeff.at(0).at(j) * product(f, x, j);
    }

    return  result;
}


int main(int argc, const char * argv[]) {

    Data L[21] = {};    //Lagrange data array
    Data N[11] = {};    //Newton data array

        double xL{};
        double yL {};
        double dxL {0.01};

        for (int i {0}; i <= 20; i++){
            xL = 5.0 * cos( ( (2.0 * i + 1.0 ) * M_PI)/42.0   );
            /*These nodes go from x= -5 to x =  5;
                   thus the point xi in LagrangeInt must also be between -5 and 5 */
            yL = Runge(xL);
            L[i] = {xL, yL};
        }

    vector <double> Lagrange_vec {};
```

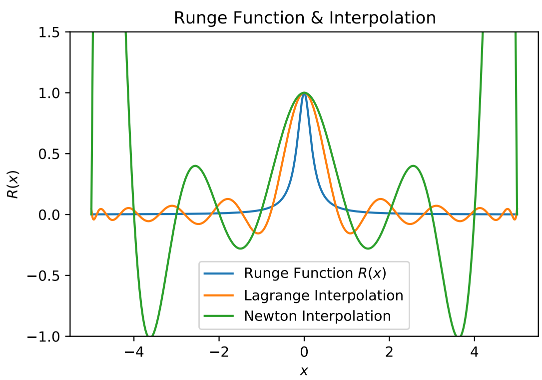
```
    for (int i {-500}; i <= 500; i++) {
        Lagrange_vec.push_back(LagrangeInt(L, i * dxL, 20));
    }


    double xN{};
    double yN {};
    double dxN {0.01};


    for (int i {0}; i <= 10; i++){
        xN = - 5.0 + i;
        /*These nodes go from x= -5 to x =  5;
               thus the point x in NewtonInt must also be between -5 and 5 */
        yN = Runge(xN);
        N[i] = {xN, yN};
    }


    vector <double> Newton_vec {};
    for (int i {-500}; i <= 500; i++) {
        Newton_vec.push_back(NewtonInt(N, i * dxN, 10));
    }



    //OUTPUT LAGRANGE DATA TO FILE
    ofstream myyfileL ("lagrange_y_data.csv");
     for (int i{0}; i <= 1000; i++) {
         if (i != 1000) {
             myyfileL << Lagrange_vec.at(i) << ",";
         } else {
             myyfileL << Lagrange_vec.at(i) << endl;
         }
     }

    myyfileL.close();



    //OUTPUT NEWTON DATA TO FILE
    ofstream myyfileN ("newton_y_data.csv");
     for (int i{0}; i <= 1000; i++) {
         if (i != 1000) {
             myyfileN << Newton_vec.at(i) << ",";
         } else {
             myyfileN << Newton_vec.at(i) << endl;
         }
     }

    myyfileN.close();

    return 0;
}
```

We then plot the results using MATPLOTLIB:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd

font = {'family' : 'serif',
        'weight' : 'normal',
        'size'   : 44}


fig = plt.figure()  # an empty figure with no axes

#data
Lagrange = pd.read_csv("~/MyXCodeProjects/Numerical_AnalysisI/PolyInterp/lagrange_y_data.csv
    ", header = None)
Lagrange = Lagrange.transpose()

Newton = pd.read_csv("~/MyXCodeProjects/Numerical_AnalysisI/PolyInterp/newton_y_data.csv",
    header = None)
Newton = Newton.transpose()
```

```


def Runge(x):
    return 1.0/(1.0 + (25.0 * x**2) )

x = np.linspace(-5, 5, 1001)


#plot
plt.plot(x, Runge(x), label=r'Runge Function $R(x)$')
plt.plot(x, Lagrange[0], label='Lagrange Interpolation')
plt.plot(x, Newton[0], label='Newton Interpolation')

plt.ylim([-1, 1.5])

plt.xlabel(r'$x$')
plt.ylabel(r'$R(x)$')

plt.title("Runge Function & Interpolation")

plt.legend()

plt.savefig('Figures/Runge.pdf')
plt.close()
```

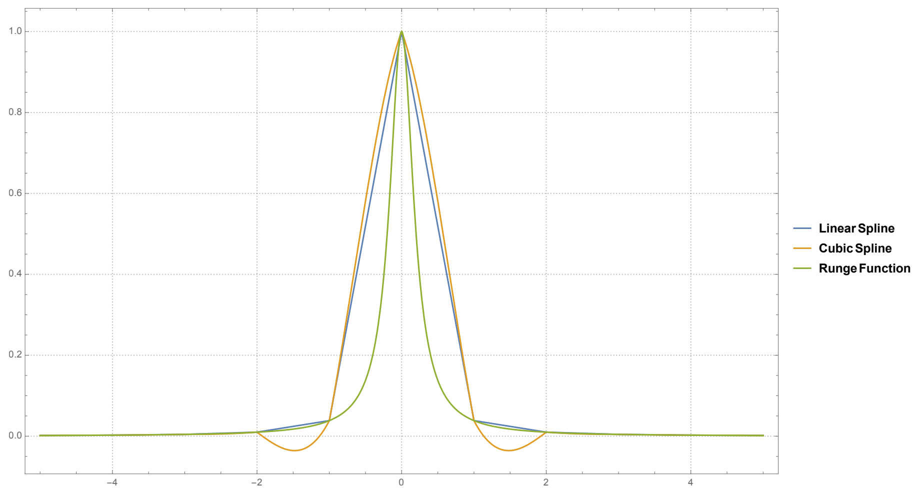
which yields the following plot



As for the splines, I made my life easier and used Mathematica, which just requires the very basic code:

```
Runge[x_] := 1/(1 + 25 x^2);
pts = Table[{i, Runge[i] }, {i, -5, 5, 1}];

f = Interpolation[pts, InterpolationOrder -> 1];
g = Interpolation[pts, InterpolationOrder -> 3];
Plot[{f[x], g[x], Runge[x]}, {x, -5, 5}, PlotTheme -> "Detailed", PlotRange -> All]
```

This yields the remaining plots:



□

Problem 7. *Show that if g is a function (not necessarily a polynomial) that interpolates a function f at nodes $x_0, \dots, x_{n-1}$, and h is a function such that $h(x_i) = \delta_{in}$ for $1 \le i \le n$, then for some constant c, the function $g + ch$ interpolates f at nodes $x_0, \dots, x_n$.*

Proof. Note that, for any $0 \le i < n$,

$$(g + ch)(x_i) = g(x_i) + ch(x_i) = g(x_i) = f(x_i), \tag{42}$$

since the function h vanishes everywhere except at x_n (where it equals 1), and g interpolates f at all nodes x_i, $i = 0, \dots, x_{n-1}$. On the other hand, at the node x_n,

$$(g + ch)(x_n) = g(x_n) + ch(x_n) = g(x_n) + c.$$

Now, g does not interpolate f at the node x_n; therefore the difference $f(x_n) - g(x_n)$ is nontrivial (i.e., $f(x_n) - g(x_n) = \alpha \neq 0$). If we now set $\alpha = c$, we get

$$(g + ch)(x_n) = g(x_n) + (f(x_n) - g(x_n)) = f(x_n).$$

Hence we have shown that $g + ch$ interpolates f at all nodes x_i, $i = 0, \dots, n$. ☐

Problem 8. *Show that if g interpolates the function f at nodes $x_0, \dots, x_{n-1}$, and if h interpolates f at nodes $x_1, \dots, x_n$, then the function*

$$\Psi(x) := g(x) + \frac{x_0 - x}{x_n - x_0}\left[g(x) - h(x)\right]$$

interpolates f at nodes $x_0, \dots, x_n$. Notice that h and g need not be polynomials.

Solution. The only node that g is missing is x_n, while h misses x_0; to summarize:

$$g(x_0) = f(x_0); \qquad h(x_n) = f(x_n); \qquad g(x_i) = h(x_i) = f(x_i) \quad \text{for } 1 \le i \le n-1.$$

Thus,

$$\begin{aligned}
\Psi(x_0) &= g(x_0) + \frac{\overbrace{x_0 - x_0}^{=0}}{x_n - x_0}\left[g(x_0) - h(x_0)\right] \\
&= g(x_0) = f(x_0);
\end{aligned}$$

$$\begin{aligned}
\Psi(x_n) &= g(x_n) + \frac{x_0 - x_n}{x_n - x_0}\left[g(x_n) - h(x_n)\right] \\
&= g(x_n) - \frac{x_0 - x_n}{x_0 - x_n}\left[g(x_n) - h(x_n)\right] \\
&= g(x_n) - g(x_n) + h(x_n) \\
&= h(x_n) = f(x_n);
\end{aligned}$$

$$\begin{aligned}
\Psi(x_i) &= g(x_i) + \frac{x_0 - x_i}{x_n - x_0}\overbrace{[\underbrace{g(x_i)}_{=h(x_i)} - h(x_i)]}^{=0} \\
&= g(x_i) = f(x_i).
\end{aligned}$$

Hence we have shown that $\Psi(x_k) = f(x_k)$ for all $0 \le k \le n$, and thus we conclude that Ψ interpolates the function f. ☐

Problem 9. *Show that divided differences are linear maps on functions. That is, for $\alpha, \beta \in \mathbb{R}$, prove*

$$(\alpha f + \beta g)[x_0, \dots x_n] = \alpha f[x_0, \dots x_n] + \beta g[x_0, \dots x_n]. \tag{43}$$

Proof. Refer back to Eq. (4), and consider the $k = 0$ case:

$$f[x_0] = f(x_0) \quad \Longrightarrow \quad (\alpha f + \beta g)[x_0] = (\alpha f + \beta g)(x_0) = \alpha f(x_0) + \beta g(x_0). \tag{44}$$

Here, of course, we assume that f and g are both linear functions. Similarly, for $k = 1$,

$$\begin{aligned}
(\alpha f + \beta g)[x_0\ x_1] &= \frac{(\alpha f + \beta g)[x_1] - (\alpha f + \beta g)[x_0]}{x_1 - x_0} \\
&= \frac{(\alpha f + \beta g)(x_1) - (\alpha f + \beta g)(x_0)}{x_1 - x_0} \\
&= \frac{\alpha f(x_1) + \beta g(x_1) - \alpha f(x_0) - \beta g(x_0)}{x_1 - x_0} \\
&= \alpha \frac{f(x_1) - f(x_0)}{x_1 - x_0} + \beta \frac{g(x_1) - g(x_0)}{x_1 - x_0} \\
&= \alpha f[x_0\ x_1] + \beta g[x_0\ x_1].
\end{aligned} \tag{45}$$

Assume now that this property holds for any arbitrary k elements, say $k = n - 1$; then we will show that it must also hold for $k = n$. Hence, assuming it holds for $k = n - 1$ elements,

$$(\alpha f + \beta g)[x_0, \dots x_{n-1}] = \alpha f[x_0, \dots x_{n-1}] + \beta g[x_0, \dots x_{n-1}]; \tag{46a}$$
$$(\alpha f + \beta g)[x_1, \dots x_n] = \alpha f[x_1, \dots x_n] + \beta g[x_1, \dots x_n]. \tag{46b}$$

Then, for $k = n$ elements,

$$\begin{aligned}
(\alpha f + \beta g)[x_0, \dots x_n] &= \frac{(\alpha f + \beta g)[x_1, \dots x_n] - (\alpha f + \beta g)[x_0, \dots x_{n-1}]}{x_n - x_0} \\
&= \frac{\alpha f[x_1, \dots x_n] + \beta g[x_1, \dots x_n] - \alpha f[x_0, \dots x_{n-1}] - \beta g[x_0, \dots x_{n-1}]}{x_n - x_0} \\
&= \alpha \frac{f[x_1, \dots x_n] - f[x_0, \dots x_{n-1}]}{x_n - x_0} + \beta \frac{g[x_1, \dots x_n] - g[x_0, \dots x_{n-1}]}{x_n - x_0} \\
&= \alpha f[x_0, \dots x_n] + \beta g[x_0, \dots x_n].
\end{aligned}$$

□

Problem 10. *Use Cramer's rule in matrix theory to prove that*

$$f[x_0, \dots, x_n] = \begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} & f(x_0) \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} & f(x_1) \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} & f(x_n) \end{vmatrix} \div \begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} & x_1^n \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} & x_n^n \end{vmatrix}. \tag{47}$$

Deduce that for the particular function $f(x) = x^m$, where $m \in \mathbb{N}$, we have

$$f[x_0, \dots, x_n] = \begin{cases} 1 & \text{if } n = m; \\ 0 & \text{if } n > m. \end{cases} \tag{48}$$

Solution. From Eq. (5), we gather

$$\begin{aligned}
{}^{\mathcal{N}}p_{n+1}(x) &= f[x_0] + f[x_0\ x_1](x - x_0) + \cdots + f[x_0 \cdots x_n](x - x_0) \cdots (x - x_{n-1}) \\
&= f[x_0] + f[x_0\ x_1]x - f[x_0\ x_1]x_0 + \cdots + f[x_0 \cdots x_n]x^n.
\end{aligned}$$

Whence, we can see that

$${}^{\mathcal{N}}p_{n+1}(x_i) = f[x_i] = f(x_i).$$

This can be written in the form

$$\underbrace{\begin{pmatrix} f(x_0) \\ \vdots \\ f(x_n) \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} & x_1^n \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} & x_n^n \end{pmatrix}}_{\mathbf{V}} \underbrace{\begin{pmatrix} f[x_0] \\ \vdots \\ f[x_0 \cdots x_n] \end{pmatrix}}_{\mathbf{x}}, \tag{49}$$

where $\mathbf{V}$ is the Vandermonde matrix. According to Cramer's rule,

$$\underbrace{x_k}_{k^{\text{th}} \text{ element in } \mathbf{x}} = \frac{\overbrace{\det V_k}^{V_k \,=\, \text{insert } \mathbf{y} \text{ into } k^{\text{th}} \text{ column of } \mathbf{V}}}{\det V}. \tag{50}$$

But this is precisely what Eq. (47) says, for $k = n$.

Now, assume that $f(x) = x^m$ for some $m \in \mathbb{N}$. From Eq. (50) we get:

- If $m = n$,

$$f[x_0, \dots, x_n] = \frac{\begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} & x_1^n \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} & x_n^n \end{vmatrix}}{\begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} & x_1^n \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} & x_n^n \end{vmatrix}} = 1. \tag{51}$$

- If, on the other hand, $n > m$ (say, $m = k$, for some $k = 0, \dots, n-1$), then

$$f[x_0, \dots, x_n] = \frac{\begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^k & \cdots & x_0^k \\ 1 & x_1 & x_1^2 & \cdots & x_1^k & \cdots & x_1^k \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^k & \cdots & x_n^k \end{vmatrix}}{\begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} & x_1^n \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} & x_n^n \end{vmatrix}} = 0. \tag{52}$$

  This last expression is zero because there are two repeated columns in the matrix of the numerator. In other words, the matrix is singular and its determinant is zero. □

Problem 11. *Referring to the Lagrange interpolation process, we define*

$$w_i := \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{1}{x_i - x_j}. \tag{53}$$

Show that if x is not a node, then the interpolating polynomial can be evaluated by the formula:

$$^{\mathcal{L}}p_n(x) = \frac{\sum_{i=0}^{n} \frac{y_i w_i}{x - x_i}}{\sum_{i=0}^{n} \frac{w_i}{x - x_i}}. \tag{54}$$

(This is the **barycentric form** *of the Lagrange interpolation process.) Moreover, show that $^{\mathcal{L}}p_n$ is stable in the sense that if the w_i are incorrectly computed, we still have the interpolation property*

$$\lim_{x \to x_k} {}^{\mathcal{L}}p_n(x) = y_k \qquad \text{for } k \in [0, n]. \tag{55}$$

Solution. Recall from Eqs. (1)-(2) that the Lagrange polynomial is given by

$$ {}^{\mathcal{L}}p_n(x) = \sum_{i=0}^{n} y_i \ell_i(x), \qquad \text{where} \quad \ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}. \tag{56} $$

Using the definition of the weights w_i given by (53), we rewrite (56) as

$$ {}^{\mathcal{L}}p_n(x) = \sum_{i=0}^{n} y_i w_i \prod_{\substack{j=0 \\ j\neq i}}^{n} (x-x_j). \tag{57} $$

Now we simplify further this expression by making less terms dependent on i, meaning let

$$ \mathfrak{L}(x) = \prod_{j=0}^{n} (x-x_j), $$

so that

$$ \prod_{\substack{j=0 \\ j\neq i}}^{n} (x-x_j) = \frac{\mathfrak{L}(x)}{x-x_i}. $$

Thus, Eq. (57) takes the form

$$ {}^{\mathcal{L}}p_n(x) = \mathfrak{L}(x) \sum_{i=0}^{n} \frac{y_i w_i}{x-x_i}. \tag{58} $$

Note that this equation is interpolating a function f that satisfies $f(x_i) = y_i$. How about interpolating the constant function $f = 1$, so that $f(x_i) = y_i = 1$? This way we have, as desired,

$$ {}^{\mathcal{L}}p_n(x) = \frac{{}^{\mathcal{L}}p_n(x)}{1} = \frac{\mathfrak{L}(x) \sum_{i=0}^{n} \frac{y_i w_i}{x-x_i}}{\mathfrak{L}(x) \sum_{i=0}^{n} \frac{w_i}{x-x_i}} = \frac{\sum_{i=0}^{n} \frac{y_i w_i}{x-x_i}}{\sum_{i=0}^{n} \frac{w_i}{x-x_i}}. $$

Lastly, we show that Eq. (55) holds (the trick is to go backwards a bit and re-introduce the factor $\mathfrak{L}(x)$):

$$ \begin{aligned} \lim_{x\to x_k} {}^{\mathcal{L}}p_n(x) &= \lim_{x\to x_k} \frac{\sum_{i=0}^{n} \frac{y_i w_i}{x-x_i}}{\sum_{i=0}^{n} \frac{w_i}{x-x_i}} \\ &= \lim_{x\to x_k} \frac{\mathfrak{L}(x) \sum_{i=0}^{n} \frac{y_i w_i}{x-x_i}}{\mathfrak{L}(x) \sum_{i=0}^{n} \frac{w_i}{x-x_i}} \\ &= \lim_{x\to x_k} \frac{\sum_{i=0}^{n} \mathfrak{L}(x) \frac{y_i w_i}{x-x_i}}{\sum_{i=0}^{n} \mathfrak{L}(x) \frac{w_i}{x-x_i}} \\ &= \lim_{x\to x_k} \frac{\sum_{i=0}^{n} y_i w_i \prod_{\substack{j=0 \\ j\neq i}}^{n} (x-x_j)}{\sum_{i=0}^{n} w_i \prod_{\substack{j=0 \\ j\neq i}}^{n} (x-x_j)} \\ &= \lim_{x\to x_k} \frac{\sum_{i=0}^{n} y_i \ell_i(x)}{\sum_{i=0}^{n} \ell_i(x)} \\ &= \frac{\sum_{i=0}^{n} y_i \ell_i(x_k)}{\sum_{i=0}^{n} \ell_i(x_k)} \\ &= \frac{{}^{\mathcal{L}}_{f}p_n(x_k)}{{}^{\mathcal{L}}_{\mathbb{1}}p_n(x_k)} \\ &= \frac{y_k}{1} = y_k. \qquad \checkmark \end{aligned} $$

The notation introduced on the second-to-last equality was to denote the Lagrange interpolation ${}^{\mathcal{L}}_{f}p_n(x_k)$ of the function f that satisfies $f(x_k) = y_k$, and similarly, ${}^{\mathcal{L}}_{\mathbb{1}}p_n(x_k)$ interpolates the constant function $f = 1$. $\square$

Problem 12. *The* Chebyshev polynomials *of the second kind are defined by*

$$U_n(x) = \frac{1}{n+1} T'_{n+1}(x) \qquad \text{for } n \geq 0, \tag{59}$$

where $T_{n+1}(x)$ is the Chebyshev polynomial of the first kind.

a) *Using the form $T_n(x) = \cos(n\theta)$, $x = \cos\theta$, $x \in [-1, 1]$, derive a similar expression for $U_n(x)$.*

b) *Show that the Chebyshev polynomials of the second kind satisfy the recursion*

$$\begin{aligned} U_0(x) &= 1, \\ U_1(x) &= 2x, \\ U_{n+1}(x) &= 2xU_n(x) - U_{n-1}(x). \end{aligned}$$

c) *Show that the Chebyshev polynomials of the second kind are orthogonal with respect to the inner product*

$$\langle f, g \rangle = \int_{-1}^{1} f(x) g(x) \sqrt{1 - x^2}\, \mathrm{d}x. \tag{60}$$

Solution to a). Using $T_n(x) = \cos(n\theta)$ and $x = \cos\theta$, we have

$$\begin{aligned} T_{n+1}(x) &= \cos[(n+1)\theta] = \cos(n\theta + \theta) \\ &= \cos(n\theta)\cos\theta - \sin(n\theta)\overbrace{\sin\theta}^{=\sqrt{1-\cos^2\theta}} \\ &= \cos(n \arccos x) x - \sin(n \arccos x)\sqrt{1 - x^2}. \end{aligned}$$

Now, taking the derivative with respect to x of this expression, we end up with

$$T'_{n+1}(x) = \frac{(1+n)\left[\sqrt{1-x^2}\cos(n\arccos x) + x\sin(n\arccos x)\right]}{\sqrt{1-x^2}}.$$

Hence,

$$\begin{aligned} U_n(x) &= \frac{1}{n+1} T'_{n+1}(x) \\ &= \cos(n\arccos x) + \frac{x\sin(n\arccos x)}{\sqrt{1-x^2}} \\ &= \cos(n\theta) + \frac{\cos\theta\ \sin(n\theta)}{\sin\theta}. \end{aligned} \tag{61}$$

□

Solution to b). Using (61),

$$\begin{aligned} U_0(x) &= \cos(0\arccos x) + \frac{x\sin(0\arccos x)}{\sqrt{1-x^2}} \\ &= \cos 0 + \frac{x\sin 0}{\sqrt{1-x^2}} \\ &= 1. \qquad \checkmark \end{aligned}$$

$$\begin{aligned} U_1(x) &= \cos(\arccos x) + \frac{x\sin(\arccos x)}{\sqrt{1-x^2}} \\ &= x + \frac{x\ \overbrace{\sin\theta}^{=\sqrt{1-\cos^2\theta}}}{\sqrt{1-x^2}} \\ &= x + \frac{x\sqrt{1-x^2}}{\sqrt{1-x^2}} \\ &= x + x \\ &= 2x. \qquad \checkmark \end{aligned}$$

More generally, for any $n \geq 1$,

$$\begin{aligned}
U_{n+1}(x) &= \cos[(n+1)\theta] + \frac{x \sin[(n+1)\theta]}{\sqrt{1-x^2}} \\
&= \cos(n\theta)\cos\theta - \sin(n\theta)\sin(\theta) + \frac{x}{\sqrt{1-x^2}}\left[\sin(n\theta)\cos\theta + \sin\theta\cos(n\theta)\right];
\end{aligned}$$

$$\begin{aligned}
U_{n-1}(x) &= \cos[(n-1)\theta] + \frac{x \sin[(n-1)\theta]}{\sqrt{1-x^2}} \\
&= \cos(n\theta)\cos\theta + \sin(n\theta)\sin(\theta) + \frac{x}{\sqrt{1-x^2}}\left[\sin(n\theta)\cos\theta - \sin\theta\cos(n\theta)\right];
\end{aligned}$$

$$\begin{aligned}
U_{n+1}(x) + U_{n-1}(x) &= \cos(n\theta)\cos\theta - \sin(n\theta)\sin(\theta) + \frac{x}{\sqrt{1-x^2}}\left[\sin(n\theta)\cos\theta + \sin\theta\cos(n\theta)\right] \\
&\quad + \cos(n\theta)\cos\theta + \sin(n\theta)\sin(\theta) + \frac{x}{\sqrt{1-x^2}}\left[\sin(n\theta)\cos\theta - \sin\theta\cos(n\theta)\right] \\
&= 2\cos(n\theta)\cos\theta + 2\frac{x}{\sqrt{1-x^2}}\sin(n\theta)\cos\theta \\
&= 2\cos\theta\left[\cos(n\theta) + \frac{x}{\sqrt{1-x^2}}\sin(n\theta)\right] \\
&= 2xU_n(x).
\end{aligned}$$

Hence, we have shown that

$$U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x),$$

as desired. This result is very similar to the recursive relation for Chebyshev polynomials of the first kind, T_n, which also satisfy

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \qquad \text{for } n \geq 1,$$

the only difference being that $T_1(x) = x$, whereas $U_1(x) = 2x$. □

Solution to c). From Eq. (61), using $x = \cos\theta$, we have

$$\begin{aligned}
U_n(x) &= \cos(n \arccos x) + \frac{x \sin(n \arccos x)}{\sqrt{1-x^2}} \\
&= \frac{\cos(n\theta)\sin\theta + \cos\theta\sin(n\theta)}{\sin\theta} \\
&= \frac{\sin[\theta(n+1)]}{\sin\theta}.
\end{aligned}$$

Hence, applying Eq. (60) to two Chebyshev polynomials of the second kind, U_n and U_m, we get

$$\begin{aligned}
\langle U_n, U_m \rangle &= \int_{-1}^{1} U_n(x)U_m(x)\sqrt{1-x^2}\,\mathrm{d}x \\
&= \int_{\pi}^{0} \frac{\sin[\theta(n+1)]}{\sin\theta}\frac{\sin[\theta(m+1)]}{\sin\theta}\sin\theta\,\mathrm{d}[\cos\theta] \\
&= \int_{0}^{\pi} \frac{\sin[\theta(n+1)]}{\sin\theta}\frac{\sin[\theta(m+1)]}{\sin\theta}\sin^2\theta\,\mathrm{d}\theta \\
&= \int_{0}^{\pi} \sin[\theta(n+1)]\sin[\theta(m+1)]\,\mathrm{d}\theta \\
&= 0 \quad \text{if } n \neq m.
\end{aligned}$$

□