Numerical Analysis Projects : Comprehensive Report

Jyotishka Ray Choudhury (BS - 1903)

Course : Numerical Analysis

Projects done : 3 , 4 , 6 , 7 , 9

Date of Submission : 31st May, 2020

Contents :

Notes :

All the vectors and matrices in the report are denoted by bold letters (e.g. $\mathbf{A}, \mathbf{x}$ etc.). The codes or presentation corresponding to each project is highlighted in green and the file names are mentioned at the end of each project. This document is written using *LaTeX* in Overleaf.

✻ Project #3

❖ **Answer :** The maximum likelihood estimator of p is 0.4795832 .

❖ **Solution :** We need to find a point p where the function

$$L(p) \;=\; p^{46}(q^2+2pq)^{77} = \; p^{46}q^{77}(1+p)^{77} = \; p^{46}(1-p^2)^{77}$$

is maximized, provided that the probability that a parent gives an "**a**" allele to an offspring is $p \in [0,1]$.

Now, $L(p)$ is a polynomial in p, thus continuous and differentiable over $(0,1)$. Also, whenever $0 < p < 1$, clearly we have $0 < L(p) < 1$.

Now, $L(0) = L(1) = 0$, i.e. L attains its minimum value at both of the end-points of its domain. So, if any point in $[0,1]$, where L attains its maximum value, must be strictly inside the open interval $(0,1)$. In fact, we can guarantee the existence of at least one such point because L is continuous over the closed interval $[0,1]$, henceforth bounded and it must attain its maximum value somewhere in $(0,1)$.

Differentiating L, we obtain

$$\begin{aligned} L'(p) &= 46p^{45}\cdot(1-p^2)^{77} - p^{46}\cdot 77(1-p^2)^{76}\cdot 2p \\ &= p^{45}(1-p^2)^{76}\big[46-46p^2-154p^2\big] \\ &= 2p^{45}(1-p^2)^{76}\big(23-100p^2\big) \end{aligned}$$

In order for $L(p)$ to maximize, we need $L'(p)=0$ at some point. So, we need to solve for p such that $f(p) = L'(p) = 0$. To do this, we employ **Bisection Method**. At first, we find two points a_0 and b_0 such that $\text{sgn}(f(a_0)) \neq \text{sgn}(f(b_0))$.

Now, for any $k = 0,1,2,\dots,$ in general, consider $m_k = \dfrac{a_k+b_k}{2}$. If $f(m_k) = 0$ for some k, we are done. If not, then we check $\text{sgn}(f(m_k))$ and find out if it matches with $\text{sgn}(f(a_k))$ or $\text{sgn}(f(b_k))$.

In general, for some $k \in \mathbb{N}$,

- If $\operatorname{sgn}(f(m_k)) = \operatorname{sgn}(f(a_k))$, then we let $a_{k+1} = m_k$ and $b_{k+1} = b_k$.
- If $\operatorname{sgn}(f(m_k)) = \operatorname{sgn}(f(b_k))$, then we let $a_{k+1} = a_k$ and $b_{k+1} = m_k$.

We iterate this over $\mathbb{N}$ until $\{f(m_k)\}_{k\geqslant 1}$ converges to 0.

Now, executing this iteration using an R program, we get that the sequence $\{f(m_k)\} \to 0$ as $\{m_k\} \to 0.4795832$. Let us call this number p_e. Therefore, $L'(p_e) \approx 0$ and so $L(p)$ attains either a maxima or a minima at p_e.

Now, observe that

$$L'(p) \;=\; 2p^{45}(1-p^2)^{76}\big(23-100p^2\big) = \left[2 \cdot \frac{L(p)}{p(1-p^2)}\right] \cdot \big(23-100p^2\big)$$

The number inside square brackets is strictly positive when $p \in (0,1)$. So, $L'(p) \lesseqgtr 0$ depends on whether $23 - 100p^2 \lesseqgtr 0$. Computation shows that if $L'(p_e) = 0$ and for $p \in (0,1)$, when $p < p_e$, $L'(p) > 0$ so L is increasing. Again, when $p > p_e$, $L'(p) < 0$ so L is decreasing. This proves that $L(p)$ attains its global minima in $(0,1)$ at $p = p_e = 0.4795832$. Therefore, in the interval $[0,1]$, the L attains its global maxima at p_e. So, maximum likelihood estimator of p is p_e i.e. 0.4795832 .

❖ **Code :** In the R - Script file "**proj-03.r**", we follow the above-mentioned algorithm. We begin the iterations by setting $a_0 = 0.2$ and $b_0 = 0.6$ and stop whenever for some $k \in \mathbb{N}$, we get $f(m_k) < \varepsilon$, where ε is a pre-defined tolerance value. In the program, ε has been denoted by "tol" and it's value is 10^{-30}. Running this program, we get that $\{m_k\}$ converges to 0.4795832 .

❖ **Remark :** This problem can also be solved without any computational numerical methods like Bisection or Newton-Raphson. Since,

$$\begin{aligned} L'(p) = 2p^{45}(1-p^2)^{76}\big(23-100p^2\big) = 0 \quad &\iff \quad 23 - 100p^2 = 0 \\ &\iff \quad p \;=\; \pm\frac{\sqrt{23}}{10} \;\approx\; \pm 0.4795832 \end{aligned}$$

and only $0.4795832 \;\in\; (0,1)$, so we can directly say that $p_e = 0.4795832$ is the maximum likelihood estimator of p.

✻ Project #4

❖ **Answer :** $L(p, a)$ is maximized when $p = 1.946419$ and $a = 2.87889$.

❖ **Solution :** We are required to maximize the function $L(p, a)$ w.r.t. both p and a, where

$$L(p,a) = \prod_{i=1}^{n} f\big(x_i \mid p, a\big) = \prod_{i=1}^{n} \frac{a^p}{\Gamma(p)} \cdot x_i^{p-1} e^{ax_i} = \frac{a^{np}}{\big[\Gamma(p)\big]^n} \cdot M^{p-1} e^{aS}$$

where $n = 996$, $M = \prod_{i=1}^{996} x_i$ and $S = \sum_{i=1}^{996} x_i$.

Now, $L(p, a) > 0$ as $p > 0$, $a > 0$ and we know that $\Gamma(p) > 0$ for all $p > 0$. So, we can take $\log$ on both sides of the preceding expression to obtain

$$\log L(p,a) = np \cdot \log a + (p-1)\log M - n \cdot \log \Gamma(p) - aS$$

In order for this to attain an extreme value, we need both the partial derivatives of the the function $L(p, a)$ w.r.t. p and a to be equal to 0. Differentiating the last equation accordingly, we obtain

$$g_1(p,a) = \frac{\partial L}{\partial p} = L(p,a) \cdot \big[n \cdot \log a + \log M - n \cdot \psi(p)\big]$$

$$g_2(p,a) = \frac{\partial L}{\partial p} = L(p,a) \cdot \left[\frac{np}{a} - S\right]$$

where $\psi(p) = \dfrac{d}{dp}\Big[\log \Gamma(p)\Big] = \dfrac{\Gamma'(p)}{\Gamma(p)}$ is known to be the Digamma Function.

Now, $L(p, a)$ attains an extrema when $g_1(p, a) = g_2(p, a) = 0$. Since, $L(p, a) > 0$ for all $p, a > 0$, so we have

$$f_1(p,a) = n \cdot \log a + \log M - n \cdot \psi(p) \ = \ 0 \quad \& \quad f_2(p,a) = \frac{np}{a} - S = 0$$

From the second-last equation, we get $\log a = \psi(p) - \dfrac{1}{n}\log M$, which implies

$$\begin{aligned} a &= \exp\left[\psi(p) - \frac{1}{n}\log M\right] \\ &= \exp\left[\psi(p) + \log M^{-\frac{1}{n}}\right] \\ &= M^{-\frac{1}{n}}\, e^{\psi(p)} \end{aligned}$$

Substituting this value of a in the equation $f_2(p, a) = 0$, we obtain

$$\frac{S}{n} \cdot M^{-\frac{1}{n}}\, e^{\psi(p)} = p$$

Now, consider the equation $\mathcal{H}(p) = \dfrac{S}{n} \cdot M^{-\frac{1}{n}}\, e^{\psi(p)} - p$

So, we need to solve for p such that $\mathcal{H}(p) = 0$. In order to do this, we employ **Newton-Raphson iterations** on the sequence $\{p_k\}_{k \geqslant 0}$ starting from $p_0 = 2$. So, the iteration will look like

$$p_{k+1} \; = \; p_k - \frac{\mathcal{H}(p)}{\mathcal{H}'(p)}$$

where

$$\begin{aligned} \mathcal{H}'(p) = \frac{d\mathcal{H}}{dp} &= \frac{d}{dp}\left[\frac{S}{n} \cdot M^{-\frac{1}{n}} e^{\psi(p)} - p\right] \\ &= \frac{S}{n} \cdot M^{-\frac{1}{n}} e^{\psi(p)} \cdot \psi'(p) - 1 \\ &= \left[\frac{S}{n} \cdot M^{-\frac{1}{n}} e^{\psi(p)} \cdot \psi'(p) - p + p\right] \cdot \psi'(p) - 1 \\ &= \Big[p + \mathcal{H}(p)\Big] \cdot \psi'(p) - 1 \end{aligned}$$

Here, $\psi'(p)$ is the first derivative of $\psi(p)$ and it is called the Trigamma Function.

Executing this iteration using an R program, we get that $\{p_k\}$ converges to $p_e = 1.946419$ and substituting, we obtain

$$\frac{np_e}{a_e} - S \;=\; 0 \quad \iff \quad a_e \;=\; \frac{np_e}{S} \;=\; 2.87889$$

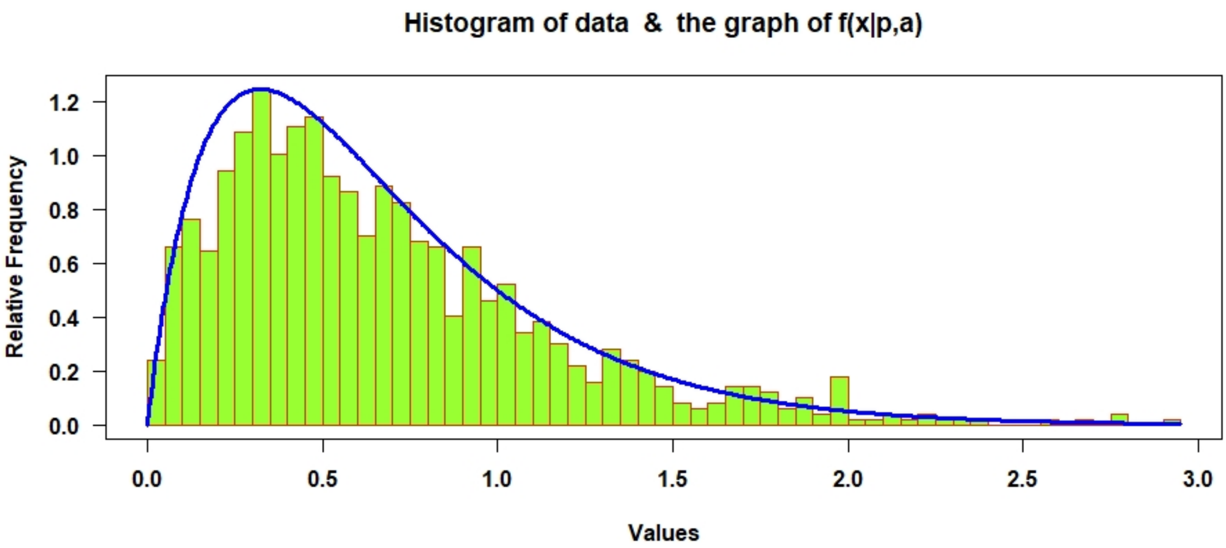
approximately. Let's represent (p_e, a_e) by T. Now, observe that

$$\left[\frac{\partial f_1}{dp}\right]_T = -n \cdot \psi'(p_e) = -\big(0.66727 \times n\big) \;<\; 0 \quad \& \quad \left[\frac{\partial f_2}{da}\right]_T = -\frac{np_e}{a_e^2} \;<\; 0$$

so $L(p, a)$ is maximized at (p_e, a_e). We'll verify this result using a histogram in the last paragraph titled "**Plots**".

❖ **Code :** In the R - Script file "**proj-04.r**", we follow the above-mentioned technique and stop the process whenever $|p_{k+1} - p_k| < \varepsilon$, where ε is a pre-defined tolerance value, very close to 0. In the code, it is represented as "tol" and its value is assigned to be $1e-8$ or 10^{-8}.Using this method, the Newton-Raphson iterations on $\mathcal{H}$ produces p_e and henceforth, a_e.

❖ **Plots :** Using the observations in the *data.txt* file, we have constructed a histogram below. The contents X-axis and Y-axis are written on the sides of the axes. Now, we superimpose the graph of the function $f(x \mid p_e, a_e)$ on the drawn histogram in order to compare their shapes visually. Here is the final diagram :



❖ **Conclusion :** From the above diagram, we can clearly see that the shape of the function f , when plotted using the numerically computed maximum likelihood estimates of p and a, is very similar to the approximate shape described by histogram. This indicates that the fit is good and thus, the computed values of p_e and a_e are indeed good approximates. Hence, visually, we may conclude that the maximum likelihood estimates, which we've computed numerically, are correct.

❖ **Remark :** Other than Newton-Raphson iterations, this problem can also be solved using Bisection method. Anyway, even in that procedure, we'll obtain the same (p_e, a_e) as the maximum likelihood estimators of (p, a).

✻ Project #6

❖ **Solution :** For given matrices $\mathbf{A}$ and **b** of orders $m \times n$ and $m \times 1$, we are required to find a least square solution to the system $\mathbf{Ax} = \mathbf{b}$, where x is an n-dimensional column vector. Mathematically speaking, we need to find an $n \times 1$ vector **x** among all possible vectors in $\mathbb{R}^n$ such that $\|\mathbf{Ax} - \mathbf{b}\|^2$ is minimum.

In order to solve this problem, we employ the **QR Decomposition** of the matrix $\mathbf{A}$ (If possible), i.e. we write

$$\mathbf{A}_{m\times n} = \mathbf{Q}_{m\times m} \cdot \mathbf{R}_{m\times n}$$

where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an upper triangular matrix. Now, in order to do this, we use the *Householder Transformation*. For each $i = 1, 2, \cdots, n$, we consider the submatrix $\mathbf{A}_i = \mathbf{A}\left[i : m, i : n\right]$ of $\mathbf{A}$ and let $\mathbf{x}_i = \left(\mathbf{A}_i\right)_{*1}$ and

$$\mathbf{y}_i = \begin{bmatrix} \|\mathbf{x}_i\| \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Now, let's define $\mathbf{u}_i = \text{unit}(\mathbf{x}_i - \mathbf{y}_i)$ and $\mathbf{H}_i^* = \mathbf{I} - 2\mathbf{u}_i{\mathbf{u}_i}'$. For each i, let

$$\mathbf{H}_i = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_i^* \end{bmatrix}$$

In fact, for finding out a least square solution to a system when $\mathbf{A}$ is not of full column rank, we might use a technique in which we delete those columns completely which appear to be some linear combination of its preceding columns at some point of pre-multiplication by a Householder matrix. Anyway, in this problem, we haven't used that algorithm.

Instead, in this problem, if at some k-th step, $\|\mathbf{u}\| = 0$, then we let $\mathbf{H}_k = \mathbf{I}$ and proceed. So, pre-multiplying $\mathbf{A}$ by $\mathbf{H}_1, \mathbf{H}_2, \cdots, \mathbf{H}_n$ one by one (Shaving method), we get an upper triangular matrix, say $\mathbf{R}$ with dimension $m \times n$. In mathematical notation,

$$\left(\mathbf{H}_n\mathbf{H}_{n-1}\cdots\mathbf{H}_2\mathbf{H}_1\right)\mathbf{A} = \mathbf{R} \quad \iff \quad \mathbf{Q}'\mathbf{A} = \mathbf{R}$$

where $\mathbf{Q}' = \mathbf{H}_n\mathbf{H}_{n-1}\cdots\mathbf{H}_2\mathbf{H}_1$ is an orthogonal matrix of dimension $n \times n$, since it's the product of n many orthogonal matrices. Pre-multiplying both sides by $\mathbf{Q}$, we get the QR Decomposition of $\mathbf{A}$.

- **Fact #1** : Orthogonal transformation (Pre-multiplication by an orthogonal matrix) preserves norm or length, i.e. if $\mathbf{G}$ is an orthogonal matrix and $\mathbf{x}$ is a column vector, then $\|\mathbf{Gx}\| = \|\mathbf{x}\|$. The proof is following :

$$\|\mathbf{Gx}\| = \sqrt{(\mathbf{Gx})' \cdot \mathbf{Gx}} = \sqrt{\mathbf{x}'\left(\mathbf{G}'\mathbf{G}\right)\mathbf{x}} = \sqrt{\mathbf{x}'\mathbf{x}} = \|\mathbf{x}\|$$

- **Fact #2** : Since $\mathbf{Q}$ is orthogonal, so using fact #1, it can be shown that

$$\begin{aligned}
\min_{\mathbf{x}\in\mathbb{R}^n} \left\|\mathbf{Ax} - \mathbf{b}\right\|^2 &= \min_{\mathbf{x}\in\mathbb{R}^n} \left\|\mathbf{QRx} - \mathbf{b}\right\|^2 \\
&= \min_{\mathbf{x}\in\mathbb{R}^n} \left\|\mathbf{Q}\left(\mathbf{Rx} - \mathbf{Q}'\mathbf{b}\right)\right\|^2 \\
&= \min_{\mathbf{x}\in\mathbb{R}^n} \left\|\mathbf{Rx} - \mathbf{Q}'\mathbf{b}\right\|^2 \\
&= \min_{\mathbf{x}\in\mathbb{R}^n} \left[\left\|\mathbf{R}_1\mathbf{x} - \mathbf{c_1}\right\|^2 + \left\|\mathbf{c_2}\right\|^2\right] \\
&= \min_{\mathbf{x}\in\mathbb{R}^n} \left\|\mathbf{R}_1\mathbf{x} - \mathbf{c_1}\right\|^2 + \left\|\mathbf{c_2}\right\|^2
\end{aligned}$$

where $\mathbf{R} = \begin{bmatrix}\mathbf{R_1} \\ \mathbf{R_2}\end{bmatrix}$, where $\mathbf{R_1}$ is an $n \times n$ upper triangular matrix.
Similarly, we have $\mathbf{Q}'\mathbf{b} = \begin{bmatrix}\mathbf{c_1} \\ \mathbf{c_2}\end{bmatrix}$, where $\mathbf{c_1}$ is an $n \times 1$ column vector.

- **Fact #3** : If the matrix $\mathbf{A}$ is of full column rank, then it has a unique QR Decomposition. Moreover, in that case, the least square solution will be unique too. Also, we have,

$$\rho(\mathbf{R_1}) = \rho \begin{bmatrix} \mathbf{R_1} \\ \mathbf{0} \end{bmatrix} = \rho \begin{bmatrix} \mathbf{R_1} \\ \mathbf{R_2} \end{bmatrix} = \rho(\mathbf{R}) = \rho(\mathbf{Q}'\mathbf{A}) = \rho(\mathbf{A}) = n$$

implying that $\mathbf{R_1}$ is of full column rank and hence, non-singular. In that case, $\mathbf{R_1}\mathbf{x} = \mathbf{c_1}$ is a consistent system, i.e. $\exists\, \mathbf{x}$ such that $\left\|\mathbf{R_1}\mathbf{x} - \mathbf{c_1}\right\|^2 = 0$. This $\mathbf{x}$ is the unique least square solution to our original system $\mathbf{A}\mathbf{x} = \mathbf{b}$. In case $\rho(\mathbf{A}) < n$, then $\mathbf{R}_1\mathbf{x} = \mathbf{c_1}$ is either inconsistent, or it has infinitely many solutions. Even there, we can find a least square solution, but in our problem, those cases are mostly ignored.

❖ **Code :** In the R - Script file "**proj-06.r**", we follow the above-mentioned technique, but in a more efficient way. In each step of Householder Transformation, we compute $\mathbf{u}$ and carry out the pre-multiplication of $\mathbf{H}_i$ with each columns of the latest updated $\mathbf{A}$ matrix in the following way :

$$\begin{aligned} \mathbf{H}_i\mathbf{x} &= \left(\mathbf{I} - 2\mathbf{u}_i\mathbf{u}_i{}'\right) \cdot \mathbf{x} \\ &= \mathbf{x} - 2\mathbf{u}_i \cdot \left(\mathbf{u}_i'\mathbf{x}\right) \\ &= \mathbf{x} - \underbrace{\left(2\mathbf{u}_i'\mathbf{x}\right) \cdot \mathbf{u}_i}_{(2n+1)\text{ multiplications}} \end{aligned}$$

This is possible since $\mathbf{u}_i'\mathbf{x}$ is a scalar. In the above process, the number of multiplications for each column reduces to $(2n+1)$ instead of n^2, thus computationally efficient.

Now, after shaving a column, we overwrite $\mathbf{u}_i$ it on the shaved portion of the corresponding column (Lower triangular portion of $\mathbf{A}$). The topmost numbers of the corresponding columns in all the steps are stored separately in a vector D. So, the final content of D is basically the principal diagonal elements of $\mathbf{R_1}$. Rest of the $\mathbf{R_1}$ matrix is overwritten on the upper triangular portion of $\mathbf{A}$. So, the final product of this operation is an updated (overwritten) matrix $\mathbf{A}$, hence a very little extra memory is required to store the $\mathbf{R_1}$ and the $\mathbf{u}_i$'s. So, on a whole, **this implementation is very efficient**.

After updating $\mathbf{A}$, we use the efficiently stored $\mathbf{R_1}$ matrix, we compute the required least square solution by backward substitution. If $\mathbf{A}$ has rank $\leqslant\ n$, we check if there are infinitely many solutions to the system $\mathbf{R_1x} = \mathbf{c_1}$ and if so, we show any one possible solution as the least square solution. If $\mathbf{R_1x} = \mathbf{c_1}$ is inconsistent, we print "ERROR: Given matrix is not of full column rank!". Our program gives the efficiently computed QR matrix, the principal diagonal elements of $\mathbf{R}$ and a least square solution as its output.

✻ Project #7

❖ **Notations :** All the notations used in our solution are either standard, or described in our class web-page. For example, the kth order divided difference of f is denoted by $f[x_k, \ldots, x_0]$.

❖ **Solution :** Suppose, we are given $(n+1)$ many points on the real plane, viz. $A_0 \equiv (x_0, y_0), A_1 \equiv (x_1, y_1), \ldots, A_n \equiv (x_n, y_n)$. We are required to show that if we traverse through the divided difference table made on the basis of these points and construct a polynomial accordingly, we'll end up getting the same polynomial as the forward interpolating polynomial. We'll use Newton divided difference formula, two observations and a lemma to prove this.

• **Newton's divided difference formula** : For any $(n+1)$ many given points say $(p_1, q_1), (p_2, q_2), \ldots, (p_n, q_n)$, there exists a unique interpolating polynomial h (depending on those points) with $\deg(h) \leqslant n$, such that $h(p_i) = q_i$ for $i = 0(1)n$. The expression of h looks like

$$\begin{aligned} h(p) \ &= \ h[p_0] + \\ &\quad (p-p_0)\cdot h[p_1,p_0] + \\ &\quad (p-p_0)(p-p_1)\cdot h[p_2,p_1,p_0] + \\ &\quad (p-p_0)(p-p_1)(p-p_2)\cdot h[p_3,p_2,p_1,p_0] + \cdots + \\ &\quad (p-p_0)\cdots(p-p_{n-1})\cdot h[p_n,\ldots,p_0] \end{aligned}$$

By this formula, for our $A_0, A_1, \ldots, A_n$, there is a polynomial, say f, with $\deg(f) \leqslant n$ which interpolates these points.

• **Lemma (♏)** : The interpolating polynomial of degree $\leqslant n$, mentioned above, is unique for the given set of $(n+1)$ points.

Now, let us construct the polynomial as instructed in the problem statement. We define z_i's in the following way :

❀ Suppose, we begin our traversal from y_k for some integer $k \in [0, n]$. Then, we define its corresponding x_k to be z_0. Obverse that the shadow of $f[x_k]$ is $\{x_k\}$ or $\{z_0\}$.

❀ In the 1st step, we may move to $f[x_k, x_{k-1}]$ or $f[x_{k+1}, x_k]$. Suppose, without loss of generality, we move to $f[x_{k+1}, x_k]$. Then, we define z_1 to be the newly included point, i.e. x_{k+1}. Observe that the shadow of $f[x_{k+1}, x_k]$ is $\{x_{k+1}, x_k\}$ or $\{z_1, z_0\}$.

❀ In the 2nd step, we may move to $f[x_{k+1}, x_k, x_{k-1}]$ or $f[x_{k+2}, x_{k+1}, x_k]$. Suppose, w.l.o.g., we move to $f[x_{k+1}, x_k, x_{k-1}]$. Then, we define z_2 to be the newly included point, i.e. x_{k-1}. Observe that the shadow of $f[x_{k+1}, x_k, x_{k-1}]$ is $\{x_{k+1}, x_k, x_{k-1}\}$ or $\{z_2, z_1, z_0\}$.

❀ In general, we define z_i to be the new x_j which is reached after the ith step in the traversal. So, the shadow of $f[z_i, \dots, z_0]$ is basically $\{z_i, \dots, z_0\}$.

❀ Proceeding in this similar manner, after n steps, we'll reach $f[x_n, \dots, x_1, x_0]$ eventually. So, clearly the shadow of $f[x_n, \dots, x_1, x_0]$ will be $\{z_n, \dots, z_1, z_0\}$.

Summing up this whole process, we get that by this traversal via an arbitrary path as per the instructions of the problem, we obtain a polynomial (Say g), which looks like

$$\begin{aligned} \mathrm{g}(x) \;=\; & f[z_0] + \\ & (x - z_0) \cdot h[z_1, z_0] + \\ & (x - z_0)(x - z_1) \cdot h[z_2, z_1, z_0] + \\ & (x - z_0)(x - z_1)(x - z_2) \cdot h[z_3, z_2, z_1, z_0] + \cdots + \\ & (x - z_0) \cdots (x - z_{n-1}) \cdot h[z_n, \dots, z_0] \end{aligned}$$

• **Observation (✯)** : In the traversal process, we start from some $f[x_k]$ and name it as z_0 and after n steps, finally we reach $f[x_n, \dots, x_1, x_0]$. So, right after beginning, we cover one x_i and at each of the next n steps, a new x_i is included. Since the total number of possible x_i is exactly $(n+1)$, so this proves that all x_i's are exhausted in the last step. **Therefore, $\{\mathbf{z_0}, \mathbf{z_1}, \ldots, \mathbf{z_n}\}$ is actually a permutation of $\{\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_n}\}$.**

We consider the points $B_0 \equiv (z_0, f(z_0)),\ B_1 \equiv (z_1, f(z_1)), \ldots, B_n \equiv (z_n, f(z_n))$ which, by **Observation (✯)**, is just a permutation of the points $A_0, A_1, A_2, \ldots, A_n$.

We construct a divided difference table on the basis of $B_0, B_1, \ldots, B_n$ and traverse through the forward path.

z_0	$f[z_0]$					
		$f[z_1,z_0]$				
z_1	$f[z_1]$		$f[z_2,z_1,z_0]$			
		$f[z_2,z_1]$		$\ddots$		
					$f[z_{n-1},z_{n-2},\ldots,z_0]$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$		$f[z_n,z_{n-1},\ldots,z_1,z_0]$
					$f[z_n,z_{n-1},\ldots,z_1]$	
		$f[z_{n-1},z_{n-2}]$		$\iddots$		
z_{n-1}	$f[z_{n-1}]$		$f[z_n,z_{n-1},z_{n-2}]$			
		$f[z_n,z_{n-1}]$				
z_n	$f[z_n]$					

- **Observation (✤)** : From the above divided difference table and applying Newton's Divided Difference Formula, we obtain that the polynomial of degree $\leqslant n$ which interpolates the points $B_0, B_1, \ldots, B_n$ will be

$$
\begin{aligned}
&f[z_0]\ + \\
&(z-z_0)\cdot h[z_1,z_0]\ + \\
&(z-z_0)(z-z_1)\cdot h[z_2,z_1,z_0]\ + \\
&(z-z_0)(z-z_1)(z-z_2)\cdot h[z_3,z_2,z_1,z_0] + \cdots + \\
&(z-z_0)\cdots(z-z_{n-1})\cdot h[z_n,\ldots,z_0]
\end{aligned}
$$

which is precisely the expression of g(z). **This proves that** g **is the unique interpolating polynomial which interpolates $B_0, B_1, \ldots, B_n$.**

Now, we summarize what we have got. Here's the comparison between the interpolating polynomials f and g :

- Both f and g are real polynomials with $\deg(f) \leqslant n$ and $\deg(\mathrm{g}) \leqslant n$.
- f interpolates the set of points $\{A_0, \ldots, A_n\}$ and g interpolates the set of points $\{B_0, \ldots, B_n\}$ which is a permutation of $\{A_0, \ldots, A_n\}$. So, both f and g basically interpolates the same set of $(n+1)$ points.

But, by **Lemma (♏)**, we know that such an interpolating polynomial is unique !

Therefore, $f \equiv \mathrm{g}$

Since our chosen path in the beginning of the proof was completely arbitrary, so the result that $f \equiv \mathrm{g}$ holds for all such permitted paths. Therefore, whatever valid path we choose, we'll always end up getting the unique interpolating polynomial. **[Proved]** □

❖ **Visual Representation :** In the file "**proj-07.pptx**", there is a rough demonstration of the above-mentioned procedure for the case $n = 5$. Since the writing of the solution is somewhat tedious, so a visual representation may help it being a little more clear.

✻ Project #9

❖ **Solution :** Crout's decomposition is a special case of LU decomposition of a non-singular matrix, where all the elements in the principal diagonal of $\mathbf{U}$ is taken to be equal to 1. In general, LU decomposition is not unique, but it can be shown that Crout's decomposition for a matrix is always unique (if exists).

At first, we may assume that $\mathbf{A}$ is non-singular. In that case, if we are interested in finding out the solution to the equation $\mathbf{Ax} = \mathbf{b}$, then we proceed like:

$$\mathbf{Ax} = \mathbf{b} \quad \Longrightarrow \quad \mathbf{LUx} = \mathbf{b} \quad \Longrightarrow \quad \mathbf{Ly} = \mathbf{b}$$

This is system is easily solvable using forward substitution. So, if $\mathbf{y}_0$ is the solution, then we get $\mathbf{Ux} = \mathbf{y}_0$ which can be solved by backward substitution.

Now, for a square matrix $\mathbf{A}$ of order n, if its LU decomposition exists, then we consider its unique Crout's decomposition. For the matrix $\mathbf{U}$, we have $u_{ii} = 1$ for all $i = 1(1)n$. It can be easily shown that

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj} \qquad \text{when} \quad i \geqslant j$$

$$u_{ij} = \frac{1}{l_{ii}}\left[a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}\right] \qquad \text{when} \quad i < j$$

The above is true only if $l_{ii} \neq 0$ for each i. Otherwise, for some i, if both the numerator and denominator of the last equation are equal to 0, then Crout's decomposition exists, but it will not be unique. For $i < j$, the actual form of the second equation is

$$l_{ii}u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj} = D_{ij}$$

Now, there are 3 possible cases, which determine whether the system has a unique solution or infinitely many solutions or the system is inconsistent.

- **Case #1** : If $D_{ij} \neq 0$ and $l_{ii} \neq 0$, then u_{ij} may take any real value. Then, the decomposition is clearly not unique.

- **Case #2** : If $D_{ij} = 0$ and $l_{ii} \neq 0$, then u_{ij} must be 0. In that case, its value is unique.

- **Case #3** : If $D_{ij} = 0$ and $l_{ii} = 0$, then no real value of u_{ij} satisfies the equation. In that case, there won't exist any such decomposition.

Once we obtain a possible LU decomposition (not necessarily unique), at first we are required to solve the system $\mathbf{Ly} = \mathbf{b}$ using forward substitution. Exactly similarly as before, there will be 3 cases and clearly, any solution won't exist if for some i,

$$l_{ii} = 0 \qquad \text{and} \qquad b_i - \sum_{k=1}^{i-1} l_{ik}\mathbf{x}_k \neq 0$$

Otherwise, they're will be at least one solution (Say $\mathbf{y}_0$) to the system. Since, all the principal diagonal elements of $\mathbf{U}$ are 1, so its determinant is 1, so $\mathbf{U}$ is non-singular. Therefore, there will be a solution (Say $\mathbf{x}_0$) to the original system, whose uniqueness depends on the uniqueness of the LU decomposition of $\mathbf{A}$.

❖ **Code :** In the R - Script file "**proj-09.r**", we follow the above-mentioned technique in an efficient manner.

We already know that the principal diagonal elements of $\mathbf{U}$ are all 1. Also, interestingly, $\mathbf{L}$ and $\mathbf{U}$ have nonzero elements at different positions. This ensures that we can somehow manage to store the matrices $\mathbf{L}$ and $\mathbf{U}$ in place of $\mathbf{A}$. Observe that for $i < j$, the value of a_{ij} is required to compute l_{ij} only. Similarly, for $i \geqslant j$, the value of a_{ij} is required to compute u_{ij} only. Thus, after the end of computation of such l_{ij} and u_{ij} in each step, we can simply overwrite them in place of a_{ij}. Therefore, no extra memory (storage) will be used for storing the matrices $\mathbf{L}$ and $\mathbf{U}$. Therefore, **this method is computationally efficient**.

· • • ● **End of report** ● • • ·