Emory University
MATH 315 Numerical Analysis
Learning Notes

Jiuru Lyu

October 8, 2023

Contents

1 Floating Point Numbers

1.1 Binary Representation

Definition 1.1.1 (Binary). 0 and 1; on and off.

Example 1.1.2 Represent Numbers in Base-2

Consider $13 = 1(10) + 3(1) = 1(10) + 3(10^0)$ in base-10. It can be converted into base-2 by decomposing 13 as $1(2^3) + 1(2^2) + 0(2^1) + 1(2^0)$.

Example 1.1.3 Fractions in Base-2

$$\frac{7}{16} = \frac{1}{16}(7) = \left(2^{-4}\right)\left(2^2 + 2^1 + 2^0\right) = 2^{-2} + 2^{-3} + 2^{-4}.$$

Example 1.1.4 Repeating Fractions in Base-2

$$\begin{aligned} \frac{1}{5} &= \frac{1}{8} + \varepsilon_1 &&\Longrightarrow \quad \varepsilon_1 = \frac{1}{5} - \frac{1}{8} = \frac{8-5}{(5\times 8)} = \frac{3}{40} \\ \varepsilon_1 &= \frac{3}{3(16)} + \varepsilon_2 &&\Longrightarrow \quad \cdots \end{aligned}$$

Repeating the steps above, we would finally get

$$\frac{1}{5} = \frac{1}{8} + \frac{1}{16} + \frac{1}{128} + \frac{1}{256} + \cdots$$

Theorem 1.1.5

Let $n \in \mathbb{Z}$ and $n \geq 1$, then

$$\sum_{k=0}^{n-1} 2^k = 2^{n-1} + 2^{n-2} + \cdots + 2^0 = 2^n - 1.$$

1.2 Integers in Computers

Definition 1.2.1 (Storing Integers). `unit8` stands for unsigned integers and `int8` stands for signed integers.

Remark. 1.1 *The 8 here represents 8 bits. It is a measure of how much storage (how many 0s or 1s).*

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
unsigned:	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
signed:	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Example 1.2.2

$$\texttt{unit8}(13) = 00001101$$

Since $-13 = 1(-2^7) + 1(2^6) + 1(2^5) + 1(2^4) + 0(2^3) + 0(2^2) + 1(2^1) + 1(2^0)$, we have

$$\texttt{int8}(-13) = 11110011$$

Remark. 1.2 *Largest and Smallest Integers:*

$$\begin{aligned}
\texttt{uint8}(x_L) &= 11111111 && \Longrightarrow x_L = 2^7 + 2^6 + \cdots + 2^0 = 2^8 - 1 = 255 \\
\texttt{uint8}(x_S) &= 00000000 && \Longrightarrow x_S = 0(2^7) + 0(2^6) + \cdots + 0(2^0) = 0 \\
\texttt{int8}(x_L) &= 01111111 && \Longrightarrow x_L = 0(-2^7) + 2^6 + \cdots + 2^0 = 2^7 - 1 = 127 \\
\texttt{int8}(x_S) &= 100000000 && \Longrightarrow x_S = 1(-2^7) + 0(2^6) + \cdots + 0(2^0) = -128
\end{aligned}$$

1.3 Representation of Floating Point Numbers

Definition 1.3.1 (Normalized Scientific Notation). Only 1 digit (non-zero) to the left of the decimal point.

Example 1.3.2

$$\begin{aligned}
&123.456 \times 10^7 \\
&12.3456 \times 10^8 \\
&1.23456 \times 10^9 \to \text{normalized}
\end{aligned}$$

Definition 1.3.3 (Anatomy of Floating Point Numbers). A floating point number, $\texttt{float}(x)$, consists of three parts: $s(x)$ (sign bit), $e(x)$ (exponent bits), and $f(x)$ (fraction bits).
Definition 1.3.4 (Precision). Precision is defined by the number of bits per part:

	$s(x)$	$e(x)$	$f(x)$	total
double precision (DP)	1	11	52	64
single precision (SP)	1	8	23	32
half precision (HP)	1	5	10	16

Remark. 1.3 *The less bits the float point number has, the less storage it requires and faster computation it performs, but more error introduces.*

Definition 1.3.5 (Floating Point Number).

$$\texttt{float}(x) = (-1)^{s(x)}\left(1 + \frac{f(x)}{2^{\text{\# of fraction bits}}}\right)2^{E(x)}, \tag{1}$$

where $E(x)$ is called the *unbiased exponent* because it is centered about 0 and is calculated through the $e(x)$, the *biased exponent* because it can only be non-negative integers, by the following formula:

$$E(x) = e(x) - \left(2^{\text{\# of exponent bits}-1} - 1\right).$$

Remark. 1.4 *Eq. (1) is in normalized scientific notation because the largest number $f(x)$ can represent is $2^{\text{\# of fraction bits}} - 1$. Hence,*

$$1 + \frac{f(x)}{2^{\text{\# of fraction bits}}} < 2,$$

and thus there will be only 1 digit in front of the decimal point.

Example 1.3.6 Formula for a Floating Point Number in Double Precision (DP)

$$\texttt{float}_{\text{DP}}(x) = (-1)^{s(x)}\left(1 + \frac{f(x)}{2^{52}}\right)2^{e(x)-1023}.$$

Example 1.3.7 Converting DP into Decimal

Suppose a DP floating number is stored as $s(x) = 0$, $e(x) = 10000000011$, and $f(x) = 0100100\cdots0$. Find its representation in decimal base-10.

Solution 1.

$e(x) = 10000000011 = 2^{10} + 2^1 + 2^0$ and $f(x) = 0100100\cdots0 = 2^{50} + 2^{47}$. Then, the unbiased exponent $E(x) = e(x) - 1023 = 2^{10} + 2^1 + 2^0 - (2^{10} - 1) = 4$. So,

$$\begin{aligned}
\texttt{float}_{\text{DP}}(x) &= (-1)^{s(x)} + \left(1 + \frac{f(x)}{2^{52}}\right)2^{E(x)} \\
&= (-1)^0\left(1 + \frac{2^{50} + 2^{47}}{2^{52}}\right)2^4 \\
&= \left(1 + 2^{-2} + 2^{-5}\right)2^4 \\
&= 2^4 + 2^2 + 2^{-1} \\
&= 16 + 4 + 0.5 = 20.5
\end{aligned}$$

□

Example 1.3.8 Converting Value to DP

Suppose a number in base-10 is -10.75. Find its representation of floating point number under DP.

Solution 2.

We have

$$\begin{aligned}\texttt{value}(x) = -10.75 &= (-1)(10+0.75)\\ &= (-1)\left(2^3+2^1+2^{-1}+2^{-2}\right)\\ &= (-1)\left(1+2^{-2}+2^{-4}+2^{-5}\right)2^3 \quad \left[\text{In normalized scientific notation}\right]\\ &= (-1)^1\left(1+\frac{2^{50}+2^{48}+2^{47}}{2^{52}}\right)2^{1026-1023}\\ &= (-1)^1\left(1+\frac{2^{50}+2^{48}+2^{47}}{2^{52}}\right)2^{2^{10}+2^1-1023}\end{aligned}$$

So, we have $s(x)=1$, $e(X)=10000000010$, and $f(x)=010110\cdots0$. □

Theorem 1.3.9 Some Special Rules

1. The formula
$$\texttt{value}(x) = (-1)^{s(x)} + \left(1+\frac{f(x)}{2^{52}}\right)2^{e(x)-1023}$$
only holds when $0 < e(x) < 2^{11}-1$ or $00\cdots01 < e(x) < 11\cdots10$.

2. If $e(x)=11\cdots1$, then it encodes special numbers.

3. If $e(x)=00\cdots0$:
   - If $f(x)=00\cdots0$, then $\texttt{value}(x)=0$.
   - If $f(x)>0$, it encodes a *denormalized floating point number*:
   $$\texttt{value}(x) = (-1)^{s(x)}\left(0+\frac{f(x)}{2^{52}}\right)2^{-1022}.$$
   This denormalized floating point number is more precise when describing really small things.

Definition 1.3.10 (Machine Epsilon/ε_{WP}). Let "WP" stands for the working precision (DP/SP/HP/etc.). The *machine epsilon,* denoted as ε_{WP}, is the gap between 1 and the next largest floating point number. Equivalently, it can be viewed as the smallest possible non-zero value of $\frac{f(x)}{2^{\text{number of fraction bits}}}$. So, $\varepsilon_{\text{DP}}=2^{-52}$, $\varepsilon_{\text{SP}}=2^{-23}$, and $\varepsilon_{\text{HP}}=2^{-10}$.

Definition 1.3.11 (Special Numbers).

1. ± 0: when $s(x)=\pm 1$ and $e(x)=f(x)=0$.

2. $\pm$`Inf`

3. `NaN`: not-a-number

Definition 1.3.12 (Floating Point Arithmetic).

1. The set of real numbers, $\mathbb{R}$, is closed under arithmetic operations.

2. The set of all WP floating point numbers, however, is not closed under arithmetic operations. For example, $\texttt{float}_{\text{DP}}(x) = \texttt{float}_{\text{DP}}(y) = 2^{52}+1$, but $xy = 2^{104}+\varepsilon$ cannot be represented using DP.

3. Suppose x and y are floating point numbers, then $x \oplus y = \texttt{float}(x+y)$ and $x \otimes y = \texttt{float}(xy)$. Consider `float` as a rounding process, we can also define subtraction and division of floating point numbers.

Example 1.3.13

Assume we are only allowed three significant digits (in Base-10) in a computer. Suppose $x = 1.23 \times 10^4$ and $y = 6.54 \times 10^3$. Find $x \oplus y = \texttt{float}(x+y)$.

Solution 3.

$$\begin{aligned}
x \oplus y &= \texttt{float}(x+y) \\
&= \texttt{float}(1.23 \times 10^4 + 6.54 \times 10^3) \\
&= \texttt{float}(1.23 \times 10^4 + 0.654 \times 10^3) \\
&= \texttt{float}(1.884 \times 10^4) \\
&= 1.88 \times 10^4.
\end{aligned}$$

□

1.4 Errors

Definition 1.4.1 (Errors We May See).

1. *Overflow*: The exponent is too large. This means $|x|$ is large and the computer will represent it as $\pm$`Inf`. Note: In DP, $x_{\text{large}} = (2-2^{-52}) \times 2^{1023} \approx 1.798 \times 10^{308}$. This number is referred as `realmax` in MATLAB.

2. *Underflow*: Large negative exponent. This means $|x|$ is tiny and the computer will represent it as ± 0. Note: In SP, $x_{\text{small}} \approx 2.225 \times 10^{-53}$ and is referred as `realmin` in MATLAB.

3. *Roundoff error*: cutoff or round at some point.

Note that sometimes we are encounter the catastrophic cancellation, meaning the subtraction leads to our loss of significance or information. In this case, it is different from underflow error or roundoff error.

Example 1.4.2 Catastrophic Cancellation/Loss of Significance Due to Subtraction

$$\begin{aligned} x &= 3.141592920353983 \approx \frac{355}{113} && \text{16 digits} \\ y &= 3.141592653589794 \approx \pi && \text{16 digits} \\ x - y &= 0.000000266764189 && \text{9 digits} \end{aligned}$$

Definition 1.4.3 (Relative Error). Let $z \in \mathbb{R}$. The relative error between $\texttt{float}(z)$ and z is denoted as μ and

$$\begin{aligned} \mu &= \frac{\texttt{float}(z) - z}{z} \\ \texttt{float}(z) &= z(1+\mu), \end{aligned}$$

where we know

$$|\mu| \leq \frac{\varepsilon_{\text{WP}}}{2}.$$

Example 1.4.4 Propagation of Errors

There are two major sources of errors: storing number and arithmetics.

Consider a computer only allow 3 significant figures. Then $\varepsilon_{\text{WP}} = 0.01$.

Consider $x = \frac{1}{3}$, $y = \frac{8}{7}$, and $x + y = \frac{31}{21}$. Then,

$$\texttt{float}(x) = 0.333 = 3.33 \times 10^{-1} = x(1+\mu_x).$$

Solving for μ_x:

$$\begin{aligned} \frac{333}{1000} &= \frac{1}{3}(1+\mu_x) \\ \mu_x &= \frac{999}{1000} - 1 = \frac{-1}{1000} = -0.001 \end{aligned}$$

Note that $|\mu_x| = 0.01 < \frac{\varepsilon_{\text{WP}}}{2}$. Similarly, we can solve $\texttt{float}(y) = 1.14 \times 10^0 = y(1+\mu_y)$ for $|\mu_y| = 0.0025$. Now, consider the floating point addition

$$\begin{aligned} x \oplus y &= \texttt{float}(\texttt{float}(x) + \texttt{float}(y)) \\ &= \texttt{float}(3.33 \times 10^{-1} + 1.14 \times 10^0) \\ &= \texttt{float}(1.473 \times 10^0) \\ &= 1.47 \times 10^0. \end{aligned}$$

Also, solve $x \oplus y = (x+y)(1+\mu_a)$ for $|\mu_a| = 0.0042$. Note that

$$|\mu_x| + |\mu_y| = 0.0035 < 0.0042 = |\mu_a|.$$

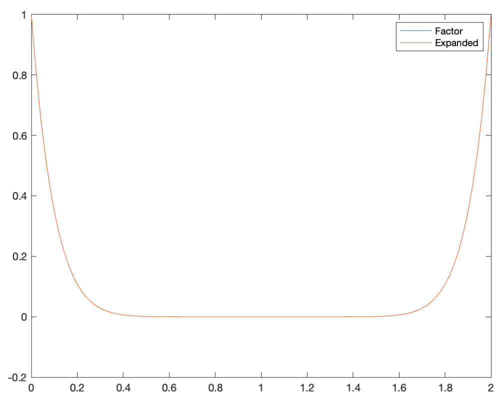
This is called the propagation of error.

Example 1.4.5 Plotting Exponentials Using Factored and Expanded Forms

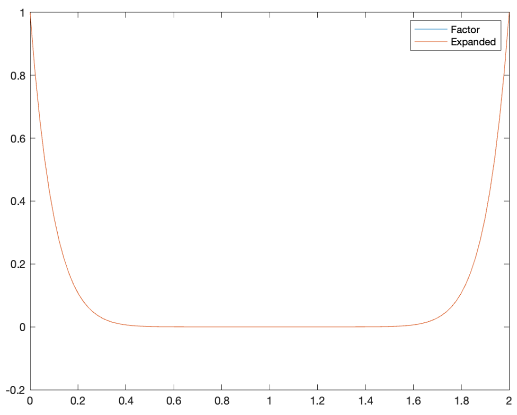
Consider $p(x) = (1-x)^{10}$ and its expanded form. Plot them to see which is better.

Example 1.4.5

```
%% Defining the Functions
p_1 = @(x) (1-x).^10;
p_2 = @(x) x.^10-10*x.^9+45*x.^8-120*x.^7+210*x.^6-252*x.^5+...
       210*x.^4-120*x.^3+45*x.^2-10*x+1;
%% Ploting the Functions
x = linspace(0, 2, 100);
plot(x, p_1(x))
hold on
plot(x, p_2(x))
legend("Factor", "Expanded")
%% Zooming In
y = linspace(0.99, 1.01, 100);
hold off
plot(y, p_1(y))
hold on
plot(y, p_2(y))
legend("Factor", "Expanded")
```



(a) Plotting Functions



(b) Zooming In

It seems that the two functions are the same (Fig 1(a)); however if we zooming in (Fig 1(b)), the expanded version introduces more error than the factored version because the expanded version requires more arithmetical operations in it.

2 Solutions of Linear Systems

Remark. 2.1 *Assumption throughout this chapter:* $\mathbf{A}$ *is a square* $n \times n$ *matrix.*

2.1 Simply Solved Linear Systems

Definition 2.1.1 (Linear System).

- Equation form: x_i are variables (what we solve for) and a_{ij} are coefficients:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

  This system has n equations and n variables.

- Matrix form:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \implies \mathbf{A}x = b,$$

  where $\mathbf{A}$ is the coefficient matrix, a $n \times n$ matrix, x is the unknown, the solution vector with length n, and b is the right hand side, vector with length n.

Theorem 2.1.2 Number of Solutions to a Linear System
A linear system $\mathbf{A}x = b$ could have the following numbers of solutions:

- One unique solution: $\mathbf{A}x = b$ is nonsingular; $\mathbf{A}$ is invertible.
- No solutions: $\mathbf{A}x = b$ is singular.
- Infinite many solutions: $\mathbf{A}x = b$ is singular.

Theorem 2.1.3 Matrix-Vector Multiplication
Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $x \in \mathbb{R}^n$.

- View 1: Row-wise. Let $y = \mathbf{A}x$, then $y_i = \sum_{j=1}^{n} a_{ij}x_j$ as the i^{th} row of y.
- View 2: Column-wise. $\mathbf{A}x$ is a linear combination of columns of $\mathbf{A}$. So, $y = \sum_{j=1}^{n} x_j\vec{\mathbf{a_j}}$, where we regard $\mathbf{A}$ as $\begin{bmatrix} \vec{\mathbf{a_1}} & \vec{\mathbf{a_2}} & \cdots & \vec{\mathbf{a_n}} \end{bmatrix}$

Row-Wise Vector Multiplication

```
y = zeros(n, 1);
for i = 1:n % loop over rows
    for j = 1:n % loop over sum
        y(i) = y(i) + A(i,j) + x(j);
    end
end
```

Row-Wise Vector Multiplication (Vectorization)

```
y = zeros(n, 1);
for i = 1:n % loop over rows
    y(i) = A(i,:) * x(i); % vectorization
end
```

Column-Wise Vector Multiplication

```
y = zeros(n, 1);
for j = 1:n % loop over columns
    y = y + x(j) * A(:, j);
end
```

Definition 2.1.4 (Important Part of a Matrix).

- Diagonal Part
- Strictly Upper Triangular Part
- Strictly Lower Triangular Part

Theorem 2.1.5 Solving Diagonal Matrix
Given

$$\begin{bmatrix} a_{11} & & \\ & \ddots & \\ & & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix},$$

we have

$$a_{11}x_1 = b_1; \quad a_{22}x_2 = b_2; \quad \cdots \quad a_{nn}x_n = b_n$$

So we have

$$x_i = \frac{b_i}{a_{ii}},$$

only if $a_{ii} \neq 0$.

Remark. 2.2 $a_{ii} \neq 0$ *holes if* $\mathbf{A}$ *is invertible.*

Remark. 2.3 *A Diagonal matrix is also a lower triangular matrix or an upper triangular matrix.*

Solving Diagonal Matrix

```
x = zeros(n, 1);
for i = 1:n
    x(i) = b(i) / A(i,i); % overflow and underflow
end
```

Theorem 2.1.6 Solving Lower Triangular Systems

Given

$$\begin{bmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ \vdots & & \ddots & \\ a_{n1} & \cdots & & a_{nn} \end{bmatrix} \begin{bmatrix} x_2 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ v_n \end{bmatrix},$$

we have

$$\begin{aligned} a_{11}x_1 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

We can use the Forward Substitution to solve:

$$x_i = \frac{b_1 - a_{i1}x_1 - a_{i2}x_2 - \cdots - a_{i(i-1)}x_{i-1}}{a_{ii}.}$$

Algorithm 1: Row-Oriented Forward Substitution

Input: matrix A $= [a_{ij}]$; vector $b = [b_i]$
Output: solution vector $x = [x_i]$

```
1 begin
2     for i = 1 to n do // loop over rows
3
4         for j = 1 to i-1 do // loop over columns
5
6             $b_i := b_i - a_{ij}x_j$;
7         $x_i := b_i/a_{ii}$;
```

Example 2.1.7

Given

$$\begin{bmatrix} -5 & & \\ 3 & 3 & \\ 2 & -5 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -10 \\ 3 \\ 21 \end{bmatrix}.$$

Use column-wise forward substitution to solve this system.

Solution 1.

In column-wise:

$$x_1 \begin{bmatrix} -5 \\ 3 \\ 2 \end{bmatrix} + x_2 \begin{bmatrix} 0 \\ 3 \\ -5 \end{bmatrix} + x_3 \begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix} = \begin{bmatrix} -10 \\ 3 \\ 21 \end{bmatrix}.$$

1. Step 1: Solve for $x_1 = -10/-5 = 2$.
2. Step 2: Plug $x_1 = 2$ into the equation:

$$x_2 \begin{bmatrix} 0 \\ 3 \\ -5 \end{bmatrix} + x_3 \begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix} = \begin{bmatrix} -10 \\ 3 \\ 21 \end{bmatrix} - (2) \begin{bmatrix} -5 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ -3 \\ 17 \end{bmatrix}.$$

3. Step 3: Solve for $x_2 = -3/3 = -1$.
4. Step 4: Plug $x_2 = -1$ into the equation:

$$x_3 \begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ -3 \\ 17 \end{bmatrix} - (-1) \begin{bmatrix} 0 \\ 3 \\ -5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 12 \end{bmatrix}.$$

5. Step 5: Solve for $x_3 = 12/4 = 3$.

□

Algorithm 2: Column-Oriented Forward Substitution

```
    Input: matrix A = $[a_{ij}]$; vector $b = [b_i]$
    Output: solution vector $x = [x_i]$
1 begin
2     for j = 1 to n do
3         $x_j := b_j/a_{jj}$;
4         for i = j+1 to n do
5             $b_i := b_i - a_{ij}x_j$;
```

Theorem 2.1.8 Computational Cost of Forward Substitution
Number of floating point operations $(+, -, \times, /)$ in row i is 1 division, $(i-1)$ multiplications, and $(i-1)$ subtractions. So, Number of floating points operations, or flops, of the algorithm is

$$\begin{aligned}\text{flops} = \sum_{i=1}^{n}(1+i-1+i-1) &= \sum_{i=1}^{n}(2i-1)\\ &= 2\sum_{i=1}^{n} i - \sum_{i=1}^{n} 1\\ &= 2\left[\frac{(n+1)(n)}{2}\right] - n\\ &= n^2\end{aligned}$$

Remark. 2.4 *It should be the same number of flops if we do column-oriented forward substitution.*

Remark. 2.5 *Solving upper triangular system using backward substitution.*

2.2 GEPP and Matrix Factorization

Theorem 2.2.1 Gaussian Elimination
In Gaussian Elimination, we are allowed to

1. Swap rows (exchange, pivot)
2. Add multiple of one row to another
3. Multiply row by non-zero scalar.

Remark. 2.6 *We require the equation with the largest coefficient in magnitude at the top when doing the Gaussian Elimination. This is because we want to divide by large numbers instead of smaller ones (which will cause errors).*

Algorithm 3: General Structure of GEPP

```
begin
    for all stages do
        pivot;
        eliminate;
```

At stage k, eliminate x_k, from rows $k+1$ to n

```
for i = k+1:n
    m(i,k) = a(i,k) / a(k,k); % find the multiplier
    a(i,k) = 0;
    for j = k+1:n
        a(i,j) = a(i,j) - m(i,k) * a(k,j); % could use vectorization
    end
    b(i) = b(i) - m(i,k) * b(k);
end
```

Pivoting at stage k: find the coefficient with the largest magnitude

```
%% The code tells us which row has the pivot.
p = k;
for i = k+1:n
    if abs(a(p,k)) < abs(a(i,k))
        p = i;
    end
end
%% Swap rows in A and b
A([p,k],:) = A([k,p],:);
b(p) = b(k);
```

Theorem 2.2.2 Cost of GEPP
At stage k, we only focus on rows k through n and columns k through n. We have $(n-k)$ divisions for multipliers. For every multiplier we have $(n-k)$ multiplications, which are then used $(n-k)$ times to change each row. So, we have $(n-k)(n-k)$ multiplications in total. Subtractions come with multiplications, so we also have $(n-k)(n-k)$ subtractions.

Theorem 2.2.3 Another Perspective on GEPP
The process of GEPP can be written as matrix multiplication $\mathbf{EA}$, where $\mathbf{E}$ is an elementary matrix and act on the rows of $\mathbf{A}$.

Theorem 2.2.4 Matrix Factorizations: $\mathbf{PA} = \mathbf{LU}$

$$\mathbf{PA} = \mathbf{LU},$$

where $\mathbf{U}$ is upper-triangular, $\mathbf{L}$ is lower-triangular, and $\mathbf{P}$ is the pivot or permutation matrix.
This factorization comes from GEPP. Almost all matrices $\mathbf{A}$ have $\mathbf{PA} = \mathbf{LU}$ unless we have a column of all zeros in $\mathbf{A}$.

Theorem 2.2.5 Solving $\mathbf{A}x = b$ with $\mathbf{PA} = \mathbf{LU}$
Given $\mathbf{A}x + b$ and pre-computed $\mathbf{PA} = \mathbf{LU}$:

$$\begin{aligned}
\mathbf{PA} &= \mathbf{P}b \\
\mathbf{LU} &= \mathbf{P}b \\
\mathbf{U}x &= \mathbf{L}^{-1}\mathbf{P}b && \text{using forward substitution} \\
x &= \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{P}b && \text{using backward substitution}
\end{aligned}$$

This process need around $\mathcal{O}(n^2)$ operations. $\mathcal{O}(\cdot)$ is the big-O notation, meaning the number is dominated by n^2.

$\mathbf{PA} = \mathbf{LU}$ in MATLAB

```
[L, U, P] = lu(A);
% P could be omitted sometimes.
```

Theorem 2.2.6 Cholesky Factorization
If $\mathbf{A}$ is symmetric ($\mathbf{A} = \mathbf{A}^T$) and positive definite (all eigenvalues are positive), then $\mathbf{A} = \mathbf{R}^T\mathbf{R}$, where $\mathbf{R}$ is upper-triangular and $\mathbf{R}^T$ is lower-triangular. This factorization is 2 time less expensive than GEPP.

Remark. 2.7 *Choleksy Factorization is just the* $\mathbf{PA} = \mathbf{LU}$ *factorization for SPDs (symmetric positive definite matrices).*

Theorem 2.2.7 Other Matrix Factorization

1. QR decomposition:
$$\mathbf{A} = \mathbf{QR},$$
where $\mathbf{R}$ is upper-triangular and $\mathbf{Q}$ is orthogonal such that $\mathbf{Q}^T\mathbf{Q} = \mathbf{Q}^T\mathbf{Q} = \mathbf{I}$.
$\mathbf{Q}$ is really easy to be inverted and comes from the Gram-Schmidt process.
This composition is a bit more expensive than $\mathbf{PA} = \mathbf{LU}$.

2. Singular Value Decomposition (SVD):
$$\mathbf{A} = \mathbf{U\Sigma V}^T,$$
where $\mathbf{U}$ and $\mathbf{V}^T$ are orthogonal and $\mathbf{\Sigma}$ is diagonal. This factorization is also more expensive than $\mathbf{PA} = \mathbf{LU}$ decomposition.

2.3 Measuring Accuracy of Solutions

Definition 2.3.1 (Vector Norms). A vector norm is a function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ that satisfies

- **Positive Definiteness**: $\|x\| \geq 0 \quad \forall x \in \mathbb{R}^n$ and $\|x\| = 0$ if and only if $x = 0$.
- **Positive Homogeneity**: $\|cx\| = |c|\|x\| \quad \forall x \in \mathbb{R}^n$ and $c \in \mathbb{R}$.
- **Triangular Inequality**: $\|x + y\| \leq \|x\| + \|y\| \quad \forall x, y \in \mathbb{R}^n$.

Definition 2.3.2 (Common Definitions of Norm).

- Pythagorean Distance: $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$.
- Taxicab/Manhattan Distance: $\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|$.
- Infinity Norm: $\|x\|_\infty = \max\limits_{i=1,\dots,n} |x_i|$.

Proof 1. In this prove, we want to show the $1-$norm is a proper norm.

- **Positive Definiteness**: Note that $\|x\|_1 = |x_1 + + \cdots + ()|x_n| \geq 0$ since each $|x_j| \geq 0$. If one $x_i \neq 0$, $|x_i| \geq 0$, then $\|x\|_1 > 0$. So, if $\|x\|_1 = 0$, it must be $x = 0$. □
- **Positive Homogeneity**:

$$\begin{aligned}
\|cx\|_1 &= |cx_1| + |cx_2| + \cdots + |cx_n| \\
&= |c||x_1| + |c||x_2| + \cdots + |c||x_n| \\
&= |c|(|x_1| + |x_2| + \cdots + |x_n|) \\
&= |c|\|x\|_1. \quad \square
\end{aligned}$$

- **Triangle Inequality**:

$$\begin{aligned}
\|x + y\|_1 &= |x_1 + y_1| + |x_2 + y_2| + \cdots + |x_n + y_n| \\
&\leq |x_1| + |y_1| + |x_2| + |y_2| + \cdots + |x_n| + |y_n| \\
&= (|x_1| + |x_2| + \cdots + |x_n|) + (|y_1| + |y_2| + \cdots + |y_n|) \\
&= \|x\|_1 + \|y\|_1.
\end{aligned}$$

■

Definition 2.3.3 (Matrix Norm). A matrix norm is a function $\|\cdot\| : \mathbb{R}^{n\times n} \to \mathbb{R}$ *s.t.*

- **Positive Definiteness**: $\|\mathbf{A}\| \geq 0$ and $\|\mathbf{A}\| = 0$ if and only if $\mathbf{A} = 0$.
- **Positive Homogeneity**: $\|c\mathbf{A}\| = |c|\|\mathbf{A}\|$.
- **Triangle Inequality**: $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$.

Definition 2.3.4 (Some Matrix Norms).

1. **Frobenius Norm**:

$$\|\mathbf{A}\|_F = \sqrt{\sum_{j=1}^{n}\sum_{i=1}^{n} a_{ij}^2}.$$

2. **Induced Matrix Norm**: Let A $\in \mathbb{R}^{n\times n}$, $x \in \mathbb{R}^{n\times 1}$, and $p = 1, 2, \infty, \cdots$, then

$$\|\mathbf{A}\|_p = \max_{x\neq 0} \frac{\|\mathbf{A}x\|_p}{\|x\|_p} = \max_{\|x\|_p=1} \|\mathbf{A}x\|_p.$$

Remark. 2.8 *Induced norm intuition: how much does $\|x\|_p$ change when we apply* $\mathbf{A}$*?*

Theorem 2.3.5 Induced Matrix Norms with different p's

- $\|\mathbf{A}\|_2 = \sigma_1$, the largest singular value. That is, if $\mathbf{A} = \mathbf{U\Sigma V}^T$, then $\|\mathbf{A}\|_2$ is the largest entry in $\mathbf{\Sigma}$.
- $\|\mathbf{A}\|_1 = \max_{j=1,\ldots,n} \sum_{i=1}^{n} |a_{ij}|$, the maximum column sum.
- $\|\mathbf{A}\|_\infty = \max_{i=1,\ldots,n} \sum_{j=1}^{n} |a_{ij}|$, the maximum row sum.

Proof 2. Let's show that $\|\mathbf{A}\|_1$ is the maximum column sum. We will (1) show $\|\mathbf{A}\|_1 \leq$ the maximum column sum, and (2) find one case when we attain the upper bound. Given $\|x\|_1 = 1$, then

$$\begin{aligned}
\|\mathbf{A}x\|_1 = \sum_{i=1}^{n} \left| \underbrace{\sum_{j=1}^{n} a_{ij}x_j}_{i-\text{th entry of}\mathbf{A}x} \right| &\leq \sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}||x_j| \\
&= \sum_{j=1}^{n} |x_j| \left(\sum_{i=1}^{n} |a_{ij}| \right) \\
&\leq \max_{j=1,\ldots,n} \sum_{i=1}^{n} |a_{ij}|, \text{ when } x \text{ has exactly 1 entry equal to } 1.
\end{aligned}$$

When $x = e_{j^*}$ be a standard basis vector with 1 in j^* position, where j^* is the column of A with maximum column sum, we have

$$\|\mathbf{A}e_{j^*}\|_1 = \|j - \text{th column of } \mathbf{A}\|_1 = \max_{j=1,\ldots,n} \sum_{i=1}^{n} |a_{ij}|.$$

■

Example 2.3.6

Given that $\mathbf{A} = \begin{bmatrix} -1 & 2 \\ -12 & 9 \end{bmatrix}$, find $\|\mathbf{A}\|_1$ and $\|\mathbf{A}\|_\infty$.

$$\|\mathbf{A}\|_1 = \text{maximum column sum} = \max_{j=1,\dots,n} \|\mathbf{A}(: .j)\|_1 = 13.$$

$$\|\mathbf{A}\|_\infty = \text{maximum row sum} = 21.$$

Theorem 2.3.7 Submultiplicativity of Induced Norm

$$\|\mathbf{A}x\|_p \leq \|\mathbf{A}\|_p \|x\|_p.$$

Proof 3. By definition, we know $\|\mathbf{A}\|_p = \max_{x \neq 0} \dfrac{\|\mathbf{A}x\|_p}{\|x\|_p}$. Then,

$$\|\mathbf{A}\|_p \geq \frac{\|\mathbf{A}x\|_p}{\|x\|_p}$$
$$\|\mathbf{A}x\|_p \leq \|\mathbf{A}\|_p \|x\|_p.$$

■

Corollary 2.3.8 $\|\mathbf{AB}\|_p \leq \|\mathbf{A}\|_p \|\mathbf{B}\|_p$.

Definition 2.3.9 (Measuring Erros). Suppose x is the true solution, and $\widehat{x}$ is the approximate solution. Then

$$\textit{Error} = \|\widehat{x} - x\|$$
$$\textit{Relative Error} = \frac{\|\widehat{x} - x\|}{\|x\|}, \quad x \neq 0.$$

Remark. 2.9 *In practice, we do not know x, the true solution. So this measurement cannot be used.*

Definition 2.3.10 (Residual). We know $\mathbf{A}$, b, $\widehat{x}$, and we want to solve $\mathbf{A}x = b$. So, the

$$\textit{Residual} = \mathbf{A}\widehat{x} - b$$
$$\textit{Residual Norm} = \|\mathbf{A}\widehat{x} - b\|$$
$$\textit{Relative Residual Norm} = \frac{\|\mathbf{A}\widehat{x} - b\|}{\|b\|}$$

Example 2.3.11

Let $\mathbf{A} = \begin{bmatrix} 0.835 & 0.667 \\ 0.333 & 0.266 \end{bmatrix}$, $b = \begin{bmatrix} 0.168 \\ 0.067 \end{bmatrix}$. Let $x = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ be the exact solution to the system $\mathbf{A}x = b$ and $\widehat{x} = \begin{bmatrix} 267 \\ -334 \end{bmatrix}$, a bad computation of the solution. Then,

$$\frac{\|b - \mathbf{A}\widehat{x}\|_2}{\|x\|_2} \approx 0.006$$
$$\frac{\|x - \widehat{x}\|_2}{\|x\|_2} \approx \mathcal{O}(10^2)$$

Remark. 2.10 *The residual norm is not always a good estimate of the relative error.*

Definition 2.3.12 (Ill-Conditioned, Well-Conditioned). If the system is linearly dependent, we call the system *ill-conditioned.* If the system is linearly independent, we call it *well-conditioned.*

Definition 2.3.13 (Condition Numbers). The condition number of $\mathbf{A}$ is $\kappa(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$. Note that $\kappa(\mathbf{A}) \geq 1$ and $\kappa(\mathbf{I}) = 1$. If $\kappa(\mathbf{A})$ is large, then $\mathbf{A}$ is ill-conditioned. If $\kappa(\mathbf{A})$ is close to 1, then $\mathbf{A}$ is well-conditioned.

Remark. 2.11 *Some intuition on $\kappa(\mathbf{A})$:*

- $\|\mathbf{A}\|$*: how much* $\mathbf{A}$ *moves* x*:* $\mathbf{A}x = b$.
- $\|\mathbf{A}^{-1}\|$*: how much* $\mathbf{A}^{-1}$ *moves* b*:* $x = \mathbf{A}^{-1}b$.

So, if $\kappa(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$ *is close to* 1*, the moves balance each other. If* $\kappa(\mathbf{A})$ *is large, then we move the vectors a lot.*

Theorem 2.3.14 Upper Bound for Relative Error

$$\underbrace{\frac{\|x - \widehat{x}\|}{\|x\|}}_{\text{Relative Error}} \leq \kappa(\mathbf{A}) \cdot \underbrace{\frac{\|b - \mathbf{A}\widehat{x}\|}{\|b\|}}_{\text{Relative Residual}} = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|\frac{\|b - \mathbf{A}\widehat{x}\|}{\|b\|}.$$

Proof 4. We want to use residual norm to compare $\|x - \widehat{x}\|$ and $\|x\|$. Suppose x is the true solution: $b = \mathbf{A}x$. Then,

$$\|b\| = \|\mathbf{A}x\| \leq \|\mathbf{A}\|\|x\|.$$

So,

$$\frac{1}{\|x\|} \leq \frac{\|\mathbf{A}\|}{\|b\|} \tag{2}$$

Consider the residual: $r = b - \mathbf{A}\widehat{x} = \mathbf{A}x - \mathbf{A}\widehat{x} = \mathbf{A}(x - \widehat{x})$. So, $x - \widehat{x} = \mathbf{A}^{-1}r$. Therefore,

$$\|x - \widehat{x}\| = \|\mathbf{A}^{-1}r\| \leq \|\mathbf{A}^{-1}\|\|r\| \tag{3}$$

Putting Eq. (2) and Eq. (3) together, we have

$$\|x - \widehat{x}\| \cdot \frac{1}{\|x\|} \leq \|\mathbf{A}^{-1}\|\|r\| \cdot \frac{\|\mathbf{A}\|}{\|b\|}$$

Re-arrange the inequality, we have

$$\frac{\|x - \widehat{x}\|}{\|x\|} \leq \|\mathbf{A}\|\|\mathbf{A}^{-1}\|\frac{\|b - \mathbf{A}\widehat{x}\|}{\|b\|}.$$

■

Remark. 2.12 *Since norms measure how far two things are apart from each other,* $\|x - \widehat{x}\| = \|\widehat{x} - x\|$ *and* $\|b - \mathbf{A}\widehat{x}\| = \|\mathbf{A}\widehat{x} - b\|$.

Corollary 2.3.15 If $\kappa(\mathbf{A}) \approx 1$, then a small residual implies that $\widehat{x}$ is a good approximation to the true solution x. If $\kappa(\mathbf{A})$ is large, then we still don't know if $\widehat{x}$ is a good approximation to the true solution.

Example 2.3.16

Given $\mathbf{A}_1 = \begin{bmatrix} 1 & 10 \\ 0 & 1 \end{bmatrix}$ and $\mathbf{A}_2 = \begin{bmatrix} 1 & 10^6 \\ 0 & 1 \end{bmatrix}$. Which matrix will have a better approximation to the true solution?

Solution 5.

$$\kappa_1(\mathbf{A}_1) = \|\mathbf{A}_1\|_1\|\mathbf{A}_1^{-1}\|_1.$$

Since $\det(\mathbf{A}_1) = 1 - 0 = 1$, we know $\mathbf{A}_1^{-1} = \dfrac{1}{\det(\mathbf{A}_1)}\begin{bmatrix} 1 & -10 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -10 \\ 0 & 1 \end{bmatrix}$. So,

$$\kappa_1(\mathbf{A}_1) = \|\mathbf{A}_1\|_1\|\mathbf{A}_1^{-1}\|_1 = (11)(11) = 121.$$

Similarly,

$$\kappa_2(\mathbf{A}_2) = \|\mathbf{A}_2\|_1\|\mathbf{A}_2^{-1}\|_1 = \left(1 + 10^6\right)\left(1 + 10^6\right) = \mathcal{O}(10^12).$$

Since $\kappa_2(\mathbf{A}_2) \gg \kappa_1(\mathbf{A}_1)$, $\mathbf{A}_1$ will yield a more accurate approximation. □

Remark. 2.13 *Think* $\kappa(\mathbf{A})$ *as an indicator for how much movement of* x *will there be if we apply* $\mathbf{A}$ *on* x.

Claim. 2.1 *Conditioning is inherent to the problem. So, no algorithms can improve conditioning.*

Definition 2.3.17 (Algorithm Stability/Backward Stability). When we solve $\mathbf{A}x = b$, we will have some algorithm $\widehat{x} = \text{algorithm}(\mathbf{A}, b)$. Imagine we run the algorithm in reverse (backwards). We should obtain $\widehat{\mathbf{A}}$ and $\widehat{b}$ *s.t.* $\widehat{\mathbf{A}}\widehat{x} = \widehat{b}$ in exact arithmetic. An algorithm is *backward stable* if $\|\mathbf{A} - \widehat{\mathbf{A}}\|$ and $\|b - \widehat{b}\|$ are small.

Remark. 2.14 *Algorithm stability has nothing to do with conditioning.*

Example 2.3.18

Given $\mathbf{A} = \begin{bmatrix} \alpha & 1 \\ 1 & 2 \end{bmatrix}$. We know that solving $\mathbf{A}x = b$ using Gaussian Elimination without pivoting, we could get a solution far from true. So, Gaussian Elimination without pivoting is not backward stable. In contrast, GEPP is a backward stable algorithm.

Example 2.3.19 Is Multiplication Backward Stable?

Define

$$x \otimes y = \texttt{float}(\texttt{float}(x) \times \texttt{float}(y))$$

on a computer with 3 significant digits. Suppose $x = \frac{1}{3}$ and $y = \frac{1}{2}$. Then,

$$x \otimes y = \texttt{float}((0.333)(0.500) = 0.167; \quad \varepsilon_{\text{WP}} = 1.01 - 1.00 = 10^{-2}$$

Take $\widehat{x} = 0.334$ and $\widehat{y} = 0.500$, we get $\widehat{x}\widehat{y} = 0.167$. Since $\|x - \widehat{x}\| \approx 0.001 \leq \frac{1}{2}\varepsilon_{\text{WP}}$, we say multiplication is backward stable. Similarly, we could show all floating point operations are backward stable, in fact.

Example 2.3.20

Prove that $\|\mathbf{Q}x\|_2 = \|x\|_2$ for any $x \in \mathbb{R}^n$ if $\mathbf{Q}$ is orthogonal.

Proof 6. Since $\mathbf{Q}$ is orthogonal, $\mathbf{Q}^T\mathbf{Q} = \mathbf{Q}\mathbf{Q}^T = \mathbf{I}$. Note that the $2-$norm:

$$\|x\|_2^2 = x_1^2 + x_2^2 + \cdots + x_n^2 = x^T x.$$

Then,

$$\|\mathbf{Q}x\|_2^2 = (\mathbf{Q}x)^T(\mathbf{Q}x) = x^T\mathbf{Q}^T\mathbf{Q}x = x^T x = \|x\|_2^2.$$

So,

$$\|\mathbf{Q}x\|_2 = \|x\|_2.$$

■

Extension. 2.1 *What is $\|\mathbf{Q}\|_2$? What is $\kappa(\mathbf{Q})$?*

Solution 7.

$$\|\mathbf{Q}\|_2 = \max_{\|x\|_2=1} \frac{\|\mathbf{Q}x\|_2}{\|x\|_2} = \max_{\|x\|_2=1} \frac{\|x\|_2}{\|x\|_2} = \max_{\|x\|_2=1} 1 = 1.$$

$$\kappa(\mathbf{Q}) = \|\mathbf{Q}\|_2\|\mathbf{Q}^{-1}\|_2 = \|\mathbf{Q}\|_2\|\mathbf{Q}^T\|_2 = 1 \cdot 1 = 1.$$

□

3 Curve Fitting

3.1 Polynomial Interpolation

Definition 3.1.1 (Interpolation). A function $p(x)$ interpolates data $\{(x_i, f_i)\}_{i=0}^N$ if $p(x_i) = f_i$ for $i = 0, \ldots, N$.

Remark. 3.1 *Uniqueness of the interpolating polynomial.*

Definition 3.1.2 (Polynomial). A polynomial $p_k(x)$ is of *degree* k if there are constants $c_0, \ldots, c_k$ *s.t.*

$$p_k(x) = c_0 + c_1 x + \cdots + c_k x^k.$$

A polynomial $p_k(x)$ is in *exact degree* k if $c_k \neq 0$.

Theorem 3.1.3 Steps for Polynomial Interpolation

1. Create a problem.
   - Find some data.
   - Design the problem
2. Choose a degree (based on the number of interpolation points)
3. Determine the coefficients. $\rightarrow$ Construct a polynomial
   - Choose a polynomial basis
   - Solve a linear system.
4. Draw the curve: evaluating at lots of points. $\rightarrow$ Evaluate a polynomial

Algorithm 4: Constructing a Polynomial Interpolant

```
Input: data, $\{(x_i, f_i)\}_{i=0}^N$; polynomial basis: $\{q_j(x)\}_{i=0}^N$
Output: $c_0, \ldots, c_M$ s.t. $\sum_{j=0}^{M} c_j q_j(x_i) = f_i$ for $i = 0, \ldots, N$
1 Solve for $c_0, \ldots, c_M$:
```

$$\begin{cases} c_0 q_0(x_0) + c_1 q_1(x_0) + \cdots + c_M q_M(x_0) = f_0 \\ \vdots \\ c_0 q_0(x_N) + c_1 q_1(x_N) + \cdots + c_M q_M(x_N) = f_0 \end{cases}$$

$$\begin{bmatrix} q_0(x_0) & \cdots & q_M(x_0) \\ \vdots & \ddots & \vdots \\ q_0(x_N) & \cdots & q_M(x_N) \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_M \end{bmatrix} = \begin{bmatrix} f_0 \\ \vdots \\ f_N \end{bmatrix}$$

Theorem 3.1.4 Polynomial Uniqueness
When the nodes $\{(x_i, f_i)\}_{i=0}^{N}$ are distinct, there is a unique polynomial, the interpolating polynomial $p_N(x)$ of degree N that interpolates the data. That is,

$$\begin{bmatrix} q_0(x_0) & \cdots & q_M(x_0) \\ \vdots & \ddots & \vdots \\ q_0(x_N) & \cdots & q_M(x_N) \end{bmatrix} \in \mathbb{R}^{n\times n}$$

Definition 3.1.5 (Power Series).

$$p_N(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_N x^N$$

The Vandermonde Matrix is defined as

$$\begin{bmatrix} 1 & x_0 & \cdots & x_0^N \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \cdots & x_N^N \end{bmatrix}.$$

- Pros: easy to understand and implement
- Cons: ill-conditioning, near singularity of the Vandermonde matrix, when $|x_i - x_j|$ is small.

Example 3.1.6 Power Series
Interpolate the points $\{(-1,0),(0,1),(1,3)\}$.
Solution 1.
Suppose $p_2(x) = c_0 + c_1 x + c_2 x^2$. Then,

$$\begin{aligned} (-1,0):\ p_2(-1) &= c_0 + c_1(-1) + c_2(-1)^2 = 0 \\ (0,1):\ p_2(0) &= c_0 + c_1(0) + c_2(0^2) = 1 \\ (1,3):\ p_2(1) &= c_0 + c_1(1) + c_2(1)^2 = 3. \end{aligned}$$

In matrix form:
$$\begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix} \implies \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3/2 \\ 1/2 \end{bmatrix}.$$

□

Definition 3.1.7 (Newton Form).

$$p_N(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \cdots + b_N(x - x_0)(x - x_1)\cdots(x - x_{N-1}).$$

- Pros: we are having a lower triangular system:

$$\begin{aligned}
p_N(x_0) &= b_0 \\
p_N(x_1) &= b_0 + b_1(x - x_0) \\
p_N(x_2) &= b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1).
\end{aligned}$$

Example 3.1.8 Newton Form

Interpolate the points $\{(-1,0),(0,1),(1,3)\}$.

Solution 2.

$$\begin{aligned}
p_N(x) &= b_0 + b_1(x - (-1)) + b_2(x - (-1))(x - 0) \\
p_N(-1) &= b_0 \\
p_N(0) &= b_0 + b_1 \\
p_N(1) &= b_1 + 2b_1 + 2b_2
\end{aligned}$$

In matrix form:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix} \xrightarrow[\text{Substitution}]{\text{Forward}} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1/2 \end{bmatrix}.$$

□

Definition 3.1.9 (Lagrange Polynomials).

- Coefficient = function values f_i. $\implies$ we don't need to solve anything
- $$\ell(x) : \begin{cases} 1 & \text{at } x = x_i \\ 0 & \text{at} x = x_j,\ j \neq i \end{cases}$$

  Now, we want to construct a polynomial that has roots at all the nodes:

  $$\omega(x) = (x - x_0)(x - x_1)\cdots(x - x_N).$$

  Then, if we assume we have distinct noes, we have

  $$\ell_0(x) = \frac{(x - x_1)(x - x_2)\cdots(x - x_N)}{(x_0 - x_1)(x_0 - x_2)\cdots(x_0 - x_N)}$$

  $$\ell_3(x) = \frac{(x - x_0)(x - x_1)(x - x_2)(x - x_4)\cdots(x - x_N)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)(x_3 - x_4)\cdots(x_3 - x_N)}$$

Generalizing, we have

$$\ell_k(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_N)}{\text{numerator evaluated at } x = x_k} = \prod_{j=0,\ j\neq k}^{N} = \frac{x-x_j}{x_k-x_j}.$$

- $p_N(x) = \sum_{i=0}^{N} f_i \cdot \ell_i(x)$.
- Pros: No solving. Great for theory.
- Cons: constructing $\ell_i(x)$ is tricky.

Example 3.1.10 Lagrange Polynomials

Interpolate the points $\{(-1,0),(0,1),(1,3)\}$.