Numerical Analysis (Theory part)

Daniel Lin

Based on lectures by Dr Sheehan Olver

This note focuses on the definitions, theories and ideas. Some codes are included for reference. And this is not suitable for first time learning.
For exam cheat sheet(short version of this note), see another document.

1 Basics of Linear algebra

This section mainly summarises conclusions of non-square matrices, which is almost completely omitted in the course, but turns out to be very important in many places.

Matrix is closely related to linear transformations. Assume U, V are vector spaces over the same field $\boldsymbol{F}$ with dimension n, m respectively. Assume B_U, B_V are basis of U, V respectively, and $T : U \to V$ is a linear transformation. Matrix $A \in \mathbb{R}^{m\times n}$ represent the linear transformation T if $y = T(x) \Leftrightarrow [y]_{B_V} = A[x]_{B_U}$.

Conjugate transpose of A is defined as $A^* := \overline{A}^T$. Note as long as A, B has appropriate dimensions

$$(AB)^* = B^*A^* \quad (AB)^T = B^TA^T \quad \text{but } \overline{AB} = \overline{A}\,\overline{B}$$

many definitions of families of matrices use transpose and conjugate transpose

- *Symmetric* if $A^T = A$
- *Skew-symmetric* if $A^T = -A$
- *Orthogonal* If A is real-valued and $A^TA = I$
- *Hermitian* If $A^* = A$
- *Skew-Hermitian* if $A^* = -A$
- *Unitary* if $A^*A = I$
- *Essentially Hermitian* if $e^{i\theta}A$ is Hermitian for some $\theta \in \mathbb{R}$.
- *Normal* if $A^*A = AA^*$

Note if A is real valued, A is orthogonal $\Rightarrow$ A is unitary $\Rightarrow$ A is normal.

Trace of matrix is defined as $\operatorname{tr} A := \sum_{i=1}^{q} a_{ii}$ where $q := \min\{m, n\}$. If $A \in \mathbb{C}$, then $\operatorname{tr} A^*A = \operatorname{tr} AA^* = \sum_{i,j} |a_{ij}|^2$. So

$$\operatorname{tr} AA^* = 0 \Leftrightarrow A = 0$$

for real matrix, simply replace A^* by A^T.

There are different ways to understand matrix multiplication

- Ax is linear combination of columns of A whose coefficients are entries of x.
- $y^T A$ is linear combination of row of A whose coefficients are entries of y.
- If b_j is j'th column of B and a_i^T is the i'th row of A. Then j'th column of AB is Ab_j, and i'th row of AB is $a_i^T B$. i.e. Left multiplication by A is multiplying columns by A, right multiplication by B is multiplying rows by B.
- If $A \in F^{m\times p},\ B \in F^{m\times q}$ where F is arbitrary field, denote a_k, b_k as the k'th column of A, B, then $A^T B = [a_i^T b_j]$ i.e. i, j entry is scalar $a_i^T b_j$.
  If instead $A \in F^{m\times p},\ B \in F^{n\times p}$, then $AB^T = \sum_{k=1}^{p} a_k b_k^T$. i.e. the summation of outer product(a matrix) of a_k, b_k.

1.1 Rank-nullity theorem

Rank of a matrix $A \in F^{m\times n}$ is defined as $\dim \operatorname{ColSpace} A$, row rank is defined as $\operatorname{rank} A^T$. Remarkably, even for non-square matrices, $\operatorname{rank} A^T = \operatorname{rank} A$.

Rank-nullity theorem describes the relation between null spaces and column, row spaces. If $A \in \mathbb{R}^{m\times n}$

- $\dim \operatorname{ColSpace} A + \ker A = \operatorname{rank} A + \ker A = n$
- $\dim \operatorname{RowSpace} A + \ker A^T = \operatorname{rank} A + \ker A^T = m$

Here are some other properties of column space and kernel when we combine matrices

- If $A \in F^{m\times p},\ B \in F^{m\times q}$, then

  $$\operatorname{ColSpace} A + \operatorname{ColSpace} B = \operatorname{ColSpace} \begin{bmatrix} A & B \end{bmatrix}$$

  where we constructed a new matrix $[A\ B]$ using A, B as sub-blocks.
- If $A \in F^{m\times p},\ B \in F^{n\times p}$, then

  $$\ker A \cap \ker B = \ker \begin{bmatrix} A \\ B \end{bmatrix}$$

Rank of a matrix is closely related to solutions of linear systems. Given $A \in F^{m\times p}, b \in F^n$, the linear system $Ax = b$ have at least one solution iff rank $\begin{bmatrix} A & b \end{bmatrix} = \operatorname{rank} A$. $\begin{bmatrix} A & b \end{bmatrix}$ is called augmented matrix. The rank equality just means b is linear combination of columns of A, so the coefficients form a vector x that solves the linear system.

Note elementary operations(on rows or on columns) do not change rank of a matrix, so r.r.e.f. of A has the same rank as A.

Except of dimension of row space or column space, there are other interpretations of rank. Denote $r := \operatorname{rank} A$ where $A \in F^{m\times n}$

- Some $r \times r$ submatrix of A has non-zero determinant and every $(r+1) \times (r+1)$ submatrix has determinant 0.
- Exists list of LI vectors $b_1, ..., b_r$ s.t. $Ax = b_j$ has at least one solution for $j = 1, ..., r$. But any list of LI vectors of length longer than r fails.
- $r = \min\{p : A = XY^T \text{ for some } X \in F^{m\times p}, Y \in F^{n\times p}\} = \min\{p : A = x_1y_1^T + ... + x_py_p^T \text{ for some } x_1, ..., x_p \in F^m \ y_1, ...y_p \in F^n\}$

Properties of rank

- If $A \in F^{m\times n}$, $1 \le \operatorname{rank} A \le \min\{m, n\}$.
- $\operatorname{rank} \tilde{A} \le \operatorname{rank} A$ where $\tilde{A}$ is A with some rows and/or columns deleted.
- (Sylvester inequality)If $A \in F^{m\times p}$, $B \in F^{p\times n}$, then
$$(\operatorname{rank} A + \operatorname{rank} B) - p \le \operatorname{rank} AB \le \min\{\operatorname{rank} A, \operatorname{rank} B\}$$
- (rank-sum inequality) If $A, B \in F^{m\times n}$, then
$$|\operatorname{rank} A - \operatorname{rank} B| \le \operatorname{rank}(A + B) \le \operatorname{rank} A + \operatorname{rank} B$$
equality of second inequality holds iff $\operatorname{ColSpace} A \cap \operatorname{ColSpace} B = \{0\}$ and $\operatorname{RowSpace} A \cap \operatorname{RowSpace} B = \{0\}$. So if $\operatorname{rank} B = 1$,
$$|\operatorname{rank}(A + B) - \operatorname{rank} A| \le 1$$
so changing one entry of matrix can only change rank by at most 1.
- If $A \in \mathbb{C}^{m\times n}$, then $\operatorname{rank} A^* = \operatorname{rank} A^T = \operatorname{rank} \overline{A} = \operatorname{rank} A$.
- If $A \in F^{m\times m}, C \in F^{n\times n}$ are non-singular, $B \in F^{m,n}$, then
$$\operatorname{rank} AB = \operatorname{rank} B = \operatorname{rank} BC = \operatorname{rank} ABC$$
i.e. left and right multiplications by non-singular matrices leave rank unchanged.
- For $A, B \in F^{m\times n}$, $\operatorname{rank} A = \operatorname{rank} B$ iff exists non-singular matrices $X \in F^{m\times m}, Y \in F^{n\times n}$ s.t. $B = XAY$.
- If $A \in \mathbb{C}^{m\times n}$, $\operatorname{rank} A^*A = \operatorname{rank} A$.

1.2 Non-singularity

Left inverse of A is defined as matrix B s.t. $BA = I$, right inverse of A is defined as C s.t. $AC = I$. We say inverse of A iff $B = C$, we can see non-square A cannot have inverse as B, C do not even have the same dimensions.
But there is no guarantee left inverse = right inverse for non-square matrices. So $U^T U = I$ CANNOT imply $UU^T = I$.

The group of non-singular matrices in $F^{n\times n} \subset F^{n\times n}$ is called general linear group $GL(n, F)$. If $A \in GL(n, F)$, then inverse A^{-1} exists with $A^{-1}A = AA^{-1} = I$. And $(A^{-1})^T = (A^T)^{-1}$ so we can denote it as A^{-T}. If $A \in GL(n, \mathbb{C})$, then $(A^{-1})^* = (A^*)^{-1}$ so we can denote it as A^{-*}.

Non-square matrices can have left/right inverse. If $A \in F^{m\times n}$ and rank $A = n$, then left inverse exists. If rank $A = m$, then right inverse exists.

1.3 Spectral Theorem

Hermitian case: If A is Hermitian ($A = A^*$), then A is diagonalisable.
Normal case: A is normal ($AA^* = A^*A$) $\Leftrightarrow$ exists unitary U s.t. $A = UDU^*$ where D is diagonal.
Note every unitary matrix is automatically normal.

2 Numbers

2.1 Integers

We record binary numbers using brackets e.g. $(10101100)_2$. And if we need to specify a number is written in decimal, we write $(n)_{10}$
Decimals to binary Use division with remainder by 2 repeated. Each time divide the quotient further. For example,

$$11/2 = 5...\mathbf{1}$$

$$5/2 = 2...\mathbf{1}$$

$$2/2 = \mathbf{1}...\mathbf{0}$$

so binary expression is $(1011)_2$. It is written in REVERSE order.

One's complement of a binary integer

1. Positive: the same

2. Negative: revert every bit

Two's complement

1. Positive: the same

2. Negative: 1's complement +1

Signed integer is the same as 2's complement except the first bit is used to represent symbol: 0 - positive, 1 - negative.

Rational Number It is possible to create fractions (rational numbers) in Julia as below

```
a = 3 // 2
b = 1 // 4
a + b # returns 7 // 4

# to avoid overflow, use big for tricky fractions
big(13323) // 12421
```

2.2 Floating Point Numbers

1. $F^{\textbf{normal}}_{\sigma,Q,S}$ - set of normal FPN
where $\sigma \in \mathbb{N}$ means bias, $Q \in \mathbb{N}$ means number of bits used to represent exponent, $S \in \mathbb{N}$ means number of bits used to represent mantissa. For FPN $(\boldsymbol{sem})_2$ (where s means sign bit, $\boldsymbol{e}$ are the exponent bits and $\boldsymbol{m}$ are mantissa bits), it represents

$$(-1)^s 2^{((\boldsymbol{e})_{10}-\sigma)}(1.\boldsymbol{m})_{10}$$

Common values of σ, Q, S:

- Double-Precision(64-bits): $1023, 11, 52$
- Single-Precision(32-bits): $127, 8, 23$
- Half-Precision(16-bits): $15, 5, 10$

2. $F^{\textbf{sub}}_{\sigma,Q,S}$ - set of subnormal/denormalised FPN
note in this case, any number has bit string $(s\boldsymbol{0m})_2$ and it represents

$$(-1)^s 2^{-\sigma+1}(0.\boldsymbol{m})_{10}$$

Subnormal numbers enable us to represent numbers even smaller than the minimum value of normal FPN.
3. $F^{\textbf{special}}$ - the special FPNs (i.e. ± 0, $\pm\infty$, NaN)

1. $\boldsymbol{e} = \boldsymbol{0}, \boldsymbol{m} = \boldsymbol{0}$ – ± 0
2. $\boldsymbol{e} = \boldsymbol{0}, \boldsymbol{m} \neq \boldsymbol{0}$ – subnormal numbers
3. $\boldsymbol{e} = \boldsymbol{1}, \boldsymbol{m} = \boldsymbol{0}$ – $\pm\infty$
4. $\boldsymbol{e} = \boldsymbol{1}, \boldsymbol{m} \neq \boldsymbol{0}$ – NaN

1 means all bits are 1.

All of the above 3 categories are FPN.

Decimal system to binary for real numbers This only applies to numbers in $[0, 1)$. Multiply by 2, take the integer part, and repeat the process with the decimal part. e.g.

$$0.8125 \times 2 = 1.625 \to 1$$

$$0.625 \times 2 = 1.25 \to 1$$

$$0.25 \times 2 = 0.5 \to 0$$

$$0.5 \times 2 = 1.0 \to 1$$

so binary expression is $(0.1101)_2$

The rounding function
It is not possible to represent every real number exactly using finite number of digits. So we need rounding functions:

- $f|^{\text{up}}(x)$ - smallest binary decimal y s.t. $x \leq y$
- $f|^{\text{down}}(x)$ - largest binary decimal y s.t. $x \geq y$
- $f|^{\text{near}}(x)$ - pick the nearer one of above. (There may be a tie, pick the one with last bit 0)

If not specified, $f|^{\text{near}}(x)$ is used.
Here are the codes in Julia

```
Float32(1/5, RoundUp)
Float32(1/5, RoundDown)
Float32(1/5, RoundNearest)

# or you can directly set the environment
setrounding(Float32, RoundDown) do
    # some codes
end
```

Floating point operations actually needs rounding. (So FP additions, multiplications etc. are not associative)

The following code finds the maximum and minimum of a type

```
typemin(Int8)  # returns -128
typemax(Int8)   # returns 127

floatmin(Float32)
floatmax(Float32)
eps(Float32)   # returns the machine epsilon
```

Floating point arithmetic
Addition is divided into these steps:

1. shift to align bits. For example $(1.01001 \times 2^{10}) \times (1.111 \times 2^5)$, the second number must be shifted to 0.00001111×2^{10}
2. add the significant parts. $1.01001 + 0.00001111 = 1.010100111$
3. Perform necessary rounding
4. Normalise if necessary. You may end up with 11.1×2^2, so it must be normalised to 1.11×2^3

Multiplication are simply multiplying significant bits and adding the exponents, division works similarly (dividing significant bits and subtracting the exponents).

```
# Arithmetic of special numbers Inf and NaN
1 / 0.0   # Inf
1 / (-0.0)  # -Inf
0.0 / 0.0    # NaN

Inf*0        #  NaN
Inf+5        #  Inf
(-1)*Inf     # -Inf
1/Inf        #  0.0
1/(-Inf)     # -0.0
Inf - Inf    #  NaN
Inf ==  Inf  #  true

# NaN CANNOT be compared using ==, this is very important
NaN == NaN   #  false
isnan(NaN)   # true
```

2.3 Floating Point Errors

If $\tilde{x}$ is the approximated value of x, and $\tilde{x} = x + \delta_a$. $|\delta_a|$ is defined as absolute error. Rewrite as $\tilde{x} = x(1+\delta_r)$, where $\delta_r = \delta_a/x$. $|\delta_r|$ is defined as relative error.

Machine epsilon: (lowest error an algorithm can reach) $\epsilon_{m,S} = 2^{-S}$ where S is the number of significant bits.
Normalised range: Numbers with in this range N can be written as normal FPN. $N = [\min F^{\text{norm}}, \max F^{\text{norm}}]$.
For $f|^{\text{near}}(x)$ $(x \in N)$ relative error $\leq \frac{\epsilon_{m,S}}{2}$. And for $f|^{\text{up}}(x)$, $f|^{\text{down}}(x)$ relative error $< \epsilon_{m,S}$.

Bounding multiplication Under any precision and RoundNearest mode, $1.1 \otimes$

$1.2 \leq (1.1+\delta_1)\times(1.2+\delta_2) = 1.1\times 1.2 + 1.1\delta_2 + 1.2\delta_1 + \delta_1\delta_2$ so we can bound the absolute error by : $1.1\delta_2 + 1.2\delta_1 + \delta_1\delta_2 \leq \frac{\epsilon_m}{2}(1.1+1.2+1) = 3.3\epsilon_m$
Rules for bounding: keep calculation simple! Round 0.6 to 1, round 3.4 to 4. You are not asked to find sharpest bound.

Bounding division For $1 \oslash 1.1$, there is one error δ_1 for rounding 1.1 and δ_2 for division. So consider $\frac{1}{1.1(1+\delta_1)}(1+\delta_2)$. But

$$\left|\frac{1}{1+\delta_1} - 1\right| = \left|\frac{\delta_1}{1+\delta_1}\right| \leq \frac{\epsilon_m}{2}\frac{1}{1-1/2}(\text{ as } \delta_1 \leq 1/2) = \epsilon_m$$

So we can express $\frac{1}{1+\delta_1}$ as $1+\delta_3$ where $|\delta_3| \leq \epsilon_m$. Now $\frac{1}{1.1(1+\delta_1)}(1+\delta_2) = \frac{1}{1.1}(1+\delta_3)(1+\delta_2)$. Then work in the same way as bounding multiplication.

Useful inequalities:

- $e^x < 3$ if $x \leq 1$
- if $a > 0$, $\theta \in [0, \pi/2a]$, then $2a\theta/\pi \leq \sin(a\theta) \leq a\theta$

3 Differentiation

In exact arithmetic, derivative can be estimated by:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} =: r_f$$

where h is chosen to be as small as possible. We can bound the error of this estimation using Taylor series: $f(x+h) = f(x) + f'(x)h + \frac{f''(t)}{2}h^2$ where t is some number between x and $x+h$. So $r_f \approx f'(x) + \frac{f''(t)}{2}h$, then the error $\frac{f''(t)}{2}h$ is bounded by

$$\delta_{x,h} \leq \frac{M}{2}h, \quad \text{where } M := \sup_{x\leq t\leq x+h}(f''(t))$$

Floating Point estimation
Assume estimation of $f(x)$ is $f^{\text{FP}}(x)$ with absolute error δ_x^f s.t. $|\delta_x^f| \leq c\epsilon_m$(uniform accuracy). And assume $h = 2^{-n}$, $n \in \mathbb{N}$ to avoid error when dividing h. We have

$$\frac{f^{\text{FP}}(x+h) \ominus f^{\text{FP}}(x)}{h} = \frac{f(x+h) + \delta_{x+h}^f - f(x) - \delta_x^f}{h}(1+\delta_1) \quad \text{where } (1+\delta_1) \text{ is error due to } \ominus$$

$$= \frac{f(x+h) - f(x)}{h}(1+\delta_1) + \frac{\delta_{x+h}^f - \delta_x^f}{h}(1+\delta_1)$$

Then using Taylor expansion in the same way, (sometimes we need more terms for Taylor expansion)

$$= f'(x) + f'(x)\delta_1 + \frac{f''(t)}{2}h(1+\delta_1) + \frac{\delta^f_{x+h} - \delta^f_x}{h}(1+\delta_1)$$

So the error δ^D is the last three terms,

$$|\delta^D| \leq |f'(x)|\frac{\epsilon_m}{2} + Mh + \frac{4c}{h}\epsilon_m$$

where we used $1 + \delta_1 \leq 2$. As $h \to 0$, this error will decrease first and then increase. To get the best result, the last two terms should be balanced, i.e. $Mh \approx \frac{4c}{h}\epsilon_m$, so $h \approx \sqrt{\frac{4c}{M}\epsilon_m} \propto \sqrt{\epsilon_m}$.

Dual number estimation The method above is not the best one due to floating point error. We define dual numbers to be the system of numbers $\mathbb{D} := \{a + b\epsilon : a, b \in \mathbb{R}\}$ (just like complex number) where we define $\epsilon^2 = 0$(just like $i^2 = -1$). It is easy to show that this system is a commutative ring if we define addition as $(a + b\epsilon) + (c + d\epsilon) = (a + c) + (b + d)\epsilon$ and multiplication as $(a + b\epsilon)(c + d\epsilon) = ac + (bc + ad)\epsilon$. Conjugate of $a + b\epsilon$ is defined as $a - b\epsilon$ and note $(a+b\epsilon)(a-b\epsilon) = a^2$. This can be used to handle division. In fact, if $c \neq 0$,

$$\frac{a + b\epsilon}{c + d\epsilon} = \frac{a}{c} + \frac{bc - ad}{c^2}\epsilon$$

we can see that division by $d\epsilon$ fails, it can be proved $d\epsilon$ has no multiplicative inverse in $\mathbb{D}$, so $\mathbb{D}$ is not a field.
Matrix representation

$$a + b\epsilon \mapsto \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}, a + bi \mapsto \begin{pmatrix} a & b \\ -b & a \end{pmatrix}$$

multiplication of dual numbers are equivalent to multiplication of these matrices.

Note: $(a + b\epsilon)^n = a^n + bna^{n-1}\epsilon$ (easy proof by induction)
So as a consequence, for polynomial $p(x)$:

$$p(a + b\epsilon) = p(a) + bp'(a)\epsilon$$

And using Taylor series, the above formulae works for general function $f(x)$:

$$f(a + b\epsilon) = f(a) + bf'(a)\epsilon$$

So for example, $sin(1+\epsilon) = sin(1)+cos(1)\epsilon$. We may use these basic functions to estimate derivative of more complex ones. e.g. $f(x) = \exp(x^2 + e^x)$, find $f'(1)$:

$$f(1 + \epsilon) = \exp((1 + 2\epsilon + e + e\epsilon)) = \exp(1 + e) + \exp(1 + e)(2 + e)\epsilon$$

so $f'(1) = \exp(1 + e)(2 + e)$.

The following can be used to find the second derivative:

```
    f(Dual(Dual(a, 1), Dual(1, 0))).b.b
```

Note that here the outer Dual is a new dual of dual numbers. i.e. DualDual. So its ϵ will be different from the ϵ of Dual(a, 1) and Dual(1, 0). For this reason, we denote it $\tilde{\epsilon}$.

$$\begin{aligned} f((a+b\epsilon)+\tilde{\epsilon}(c+d\epsilon)) &= f(a+b\epsilon)+f'(a+b\epsilon)\tilde{\epsilon}(c+d\epsilon) \\ &= f(a)+b\epsilon f'(a)+(f'(a)+b\epsilon f''(a))(c+d\epsilon)\tilde{\epsilon} \end{aligned}$$

An application of numerical differentiation is Newton iteration for finding roots of function.

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

with appropriately chosen x_0 we can get approximation of a root.

4 Asymptotic costs

The same order as ϕ:$f(n) = O(\phi(n)), n \to \infty$ means $\left|\frac{f(n)}{\phi(n)}\right|$ is bounded for large n.
Negligible compared to ϕ: $f(n) = o(\phi(n)), n \to \infty$ means $\lim_{n\to\infty} \frac{f(n)}{\phi(n)} = 0$
$f(n) \sim o(\phi(n)), n \to \infty$ means $\lim_{n\to\infty} \frac{f(n)}{\phi(n)} = 1$

Same definitions apply at x_0, just by replacing $n \to \infty$ by $x \to x_0$. For example, $f(n) = O(\phi(n))$ as $x \to x_0$ if $\left|\frac{f(n)}{\phi(n)}\right|$ is bounded for neighbourhood of x_0.
Rules:

$$cO(\phi(n)) = O(\phi(n)), O(c\phi(n)) = O(\phi(n)) \quad \text{where } c \text{ is constant}$$

$$O(\phi(n)) + o(\phi(n)) = O(\phi(n))$$
$$O(\phi(n))O(\psi(n)) = O(\phi(n)\psi(n))$$
$$O(\phi(n)) + O(\psi(n)) = O(|\phi(n)| + |\psi(n)|)$$

If $g(n) = O(f(n)), f(n) = o(\phi(n))$, then $g(n) = o(\phi(n))$

Note $\log(n) = o(n)$ might be useful.

5 Matrix

5.1 Matrix multiplication

In Julia, matrices are stored as vectors. For example the matrix $A = \begin{pmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & ... & \boldsymbol{a}_n \end{pmatrix}$ where $\boldsymbol{a}_k$ represents a column, is actually stored as

$$\begin{pmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ ... \\ \boldsymbol{a}_n \end{pmatrix}$$

Due to this reason, it is faster to multiply the matrix in a special way:

$$A\boldsymbol{x} = \boldsymbol{a}_1 x_1 + ...\boldsymbol{a}_n x_n$$

this operation has cost $O(mn)$ if A has shape (m, n).

Solving matrix equation $A\boldsymbol{x} = \boldsymbol{b}$ of upper and lower triangular matrices is easier. Upper triangular case:

$$\begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ 0 & \ddots & & \vdots \\ 0 & 0 & \ddots & u_{n-1,n} \\ 0 & 0 & 0 & u_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ ... \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ ... \\ b_{n-1} \\ b_n \end{pmatrix}$$

begin from x_n,

$$x_k = \frac{b_k - \sum_{j=1}^{n-k} u_{k,k+j} x_{k+j}}{u_{k,k}}$$

Lower triangular case:

$$\begin{pmatrix} l_{1,1} & 0 & 0 & 0 \\ l_{2,1} & \ddots & 0 & 0 \\ \vdots & & \ddots & 0 \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ ... \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ ... \\ b_{n-1} \\ b_n \end{pmatrix}$$

begin from x_1,

$$x_k = \frac{b_k - \sum_{j=1}^{k-1} l_{k,j} x_j}{l_{k,k}}$$

5.2 Banded Matrix

It is a special type of sparse matrix where we only keep the main diagonal and u sub-diagonals above main diagonal, l sub-diagonals below main diagonal. It stores $n+\frac{(2n-u-1)u+(2n-l-1)l}{2}$ elements. When multiplying band matrix B with vector $\boldsymbol{x}$, the k'th entry is

$$\sum_{j=\max(1,k-l)}^{\min(n,k+1)} b_{k,j}x_j$$

to calculate each entry, at most $u+l+1$ operations are performed. So band matrix multiplication has cost $O(n)$

Special cases of banded matrix

- $l=u=0$ diagonal
- $l=1, u=0$ or $l=0, u=1$ bi-diagonal
- $l=u=1$ tri-diagonal

Codes for building banded matrices

```
Bidiagonal(dd, dl, :L)
# dd is the diagonal, dl is lower diagonal
# :L means the sub-diagonal is below the main diagonal.

Bidiagonal(dd, du, :U)
# :U means sub-diagonal above the main diagonal, du is the sub-diagonal

Tridiagonal(dl, dd, du)
```

5.3 Permutation

Permutations are usually written using Cauchy notation:

$$P_\sigma = \begin{pmatrix} 1 & 2 & \dots & n \\ \sigma_1 & \sigma_2 & \dots & \sigma_n \end{pmatrix}$$

this means $i \mapsto \sigma_i$. We can permute a vector

$$P\boldsymbol{v} = \begin{pmatrix} v_{\sigma_1} \\ \vdots \\ v_{\sigma_n} \end{pmatrix}$$

Warning: this is different from sending entry k to entry σ_k !!! It is sending σ_k to k. So $P_\sigma e_i = e_{\sigma^{-1}(i)}$
Permutation is a linear operation on vector space. So we may represent permutation using matrix:

$$P = \begin{pmatrix} e_{\sigma_1^{-1}} & e_{\sigma_2^{-1}} & \dots & e_{\sigma_n^{-1}} \end{pmatrix}$$

this matrix must be orthogonal i.e. $P^{-1} = P^T$. So there is another way to find P:

$$P = \begin{pmatrix} e_{\sigma_1}^T \\ e_{\sigma_2}^T \\ \vdots \\ e_{\sigma_n}^T \end{pmatrix}$$

Side note: P is orthogonal iff $e_i^T P^T P e_j = \delta_{i,j}$ for all i, j.

5.4 Reflection and Rotation

Rotation matrices Q_θ are all orthogonal.

$$Q_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Reflections: Suppose we want to send $\vec{x}$ to $\|\vec{x}\|\vec{e_1}$:

$$Q = \frac{1}{\|\vec{x}\|} \begin{pmatrix} x & y \\ -y & x \end{pmatrix}$$

To send $\vec{x}$ to $\|\vec{x}\|\vec{e_2}$:

$$Q = \frac{1}{\|\vec{x}\|} \begin{pmatrix} y & -x \\ x & y \end{pmatrix}$$

Fix some vector $\vec{x}$, the reflection that sends $\vec{x}$ to $-\vec{x}$ (i.e. reflection about the dotted line) is:

$$Q_{\vec{u}} = I - 2\vec{u}\vec{u}^T \quad \text{where } \vec{u} = \frac{\vec{x}}{\|\vec{x}\|}$$

Vector $\vec{u}$ is the unit normal vector.

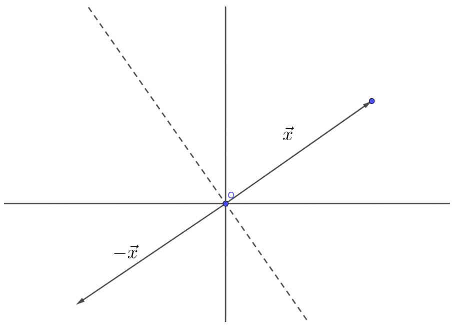

Figure 1: Reflection according to a vector

In general, reflection in direction $\mathbf{v}$ ($\|\mathbf{v}\| = 1$) is

$$Q_{\mathbf{v}} = I - 2\mathbf{v}\mathbf{v}^T$$

. This is because

$$(I - 2\mathbf{v}\mathbf{v}^T)\boldsymbol{x} = \boldsymbol{x} - 2(\mathbf{v}^T\boldsymbol{x})\mathbf{v} = \boldsymbol{x} - 2(\mathbf{v}\cdot\boldsymbol{x})\mathbf{v}$$

Note $Q_{\mathbf{v}}$ is symmetric, orthogonal. And $\mathbf{v}$ is Eigenvector of $Q_{\mathbf{v}}$ with Eigenvalue -1. For all vectors $\mathbf{w} \in \mathbf{v}^\perp$, $Q_{\mathbf{v}}\mathbf{w} = \mathbf{w}$. So 1 is also an Eigenvalue and with multiplicity $n-1 = \dim(\mathbf{v}^\perp)$. Therefore, $\det(Q_{\mathbf{v}}) = -1$. $\operatorname{rank}(\mathbf{v}\mathbf{v}^T) = 1$ for any vector $\mathbf{v}$.

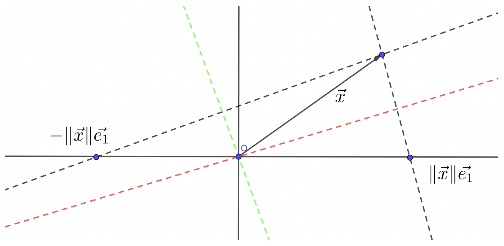

Figure 2: Householder reflection

Householder reflection is the linear map sending $\vec{x}$ to $\|\vec{x}\|\vec{e_1}$. It can be done by reflection about the red dotted line shown in figure 2. Let $\vec{y} = \vec{x} - \|\vec{x}\|\vec{e_1}$

$$Q_{\vec{w}} = I - 2\vec{w}\vec{w}^T \quad \text{where } \vec{w} = \frac{\vec{y}}{\|\vec{y}\|}$$

If $\vec{x}$ is too close to positive real axis, to reduce error, reflect around green line instead to obtain $-\|\vec{x}\|\vec{e_1}$. This is done by $Q_{\vec{w}}$ but replace $\vec{y}$ with $\vec{x} + \|\vec{x}\|\vec{e_1}$ instead.
By default, we choose to reflect to $-sign(x_1)\|\boldsymbol{x}\|\vec{e}_1$.

6 Decomposition of matrices and Applications

6.1 QR decomposition

Given matrix $A_{m\times n}$ $(m \geq n)$, $A = QR$ where $Q_{m\times m}$ is orthogonal $(Q^T = Q^{-1})$ and $R_{m\times n}$ is right triangular. If $m > n$, $R = \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix}$ where $\hat{R}$ is upper triangular.

reduced QR decomposition
$A = \hat{Q}\hat{R}$. $\hat{Q}_{m\times n}$ takes first n columns of Q and $\hat{R}$(upper-triangular) takes first n rows of R. Note $\hat{Q}^T\hat{Q} = I$ again but we cannot invert $\hat{Q}$.
Method 1
If $A = \begin{pmatrix}\boldsymbol{a}_1| & \boldsymbol{a}_2| & ... & |\boldsymbol{a}_n\end{pmatrix}$ and $\hat{Q} = \begin{pmatrix}\boldsymbol{q}_1| & \boldsymbol{q}_2| & ... & |\boldsymbol{q}_n\end{pmatrix}$. For any $1 \leq j \leq n$, we have

$$span(\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_j) = span(\boldsymbol{q}_1, \boldsymbol{q}_2, ..., \boldsymbol{q}_j)$$

So perform Gram-Schmidt on columns of A to get columns of $\hat{Q}$, and then columns of $\hat{R}$ can be deduced. For each j,

$$\boldsymbol{q}_j = \frac{\boldsymbol{v}_j}{\|\boldsymbol{v}_j\|}, \text{ where } \boldsymbol{v}_j = \boldsymbol{a}_j - \sum_{k=1}^{j-1}(\boldsymbol{q}_k^T\boldsymbol{a}_j)\boldsymbol{q}_k, \quad \boldsymbol{r}_j = \begin{pmatrix} \boldsymbol{q}_1^T\boldsymbol{a}_j \\ ... \\ \boldsymbol{q}_{j-1}^T\boldsymbol{a}_j \\ \|v_j\| \end{pmatrix}$$

This method lose some orthogonality of $\hat{Q}$ i.e. $\hat{Q}^T\hat{Q} - I$ is far from 0.
Code:

```
function gramschmidt(A)
    m,n = size(A)
    m >= n || error("Not supported")
    # Initialise R, Q
    R = zeros(n,n)
    Q = zeros(m,n)
    for j = 1:n
        # take inner products and store them into R
        for k = 1:j-1
            R[k,j] = Q[:,k]'*A[:,j]
        end
        # find v_j
        v = A[:,j] - Q[:,1:j-1]*R[1:j-1,j]
        #normalise
        R[j,j] = norm(v)
        Q[:,j] = v/R[j,j]
    end
    Q,R
end
```

Computation cost: $O(mn^2)$

Method 2
Aim to find a sequence of $m \times m$ symmetric orthogonal matrices Q_i s.t. $Q_n ... Q_1 A = R$. So then $A = QR$ where $Q = Q_1^T ... Q_n^T$. (This method gives the irreduced QR)
Take Q_i to be the householder reflection on $a_j[j : end]$ will work. That makes all entries below entry j, j to be 0.
Code

```
function householderreflection(x)
    # make a copy to avoid changing x
    y = copy(x)
    # sign(0.0) = 0.0 so use ? : instead of sign(x[1]).
    y[1] += (x[1] >= 0 ? 1 : -1)*norm(x)
    w = y/norm(y)
    I - 2*w*w'
end

function householderqr(A)
    m,n = size(A)
    # We begin with A and apply consecutive household reflections to change it to R
    R = copy(A)
    # create m by m identity matrix
    Q = Matrix(1.0I, m, m)
    for j = 1:n
        Q_j = householderreflection(R[j:end,j])
        R[j:end,:] = Q_j*R[j:end,:]
        Q[:,j:end] = Q[:,j:end]*Q_j
    end
    Q,R
end
```

Computational cost: $O(m^2n^2)$. Higher computation cost but method 2 is more accurate than method 1.

Application: Least Square Problem We try to find the $\boldsymbol{x}$ that minimises $\|A\boldsymbol{x}-\boldsymbol{b}\|^2$.
QR-approach

$$\min_{x\in\mathbb{R}^n}\|A\boldsymbol{x}-\boldsymbol{b}\| = \min_{x\in\mathbb{R}^n}\|\hat{R}\boldsymbol{x}-\boldsymbol{c}\| + \|(Q^T\boldsymbol{b})_{n+1:m}\| \quad \text{where } c := (Q^T\boldsymbol{b})_{1:n} = \hat{Q}^T\boldsymbol{b}$$

so equivalent to finding $\boldsymbol{x}$ that minimises $\|\hat{R}\boldsymbol{x}-\boldsymbol{c}\|$.
Code:

```
    Q, R_hat = qr(A)    # Warning: all-0 rows are not stored for R
    Q_hat = Q[:, 1:n]  # takes first n columns
    x = R_hat \ Q_hat'b
```

6.2 LU, PLU decomposition

For square A, LU decomposition is $A = LU$, PLU is $A = P^TLU$ where P is a permutation matrix, L is lower triangular and U is upper triangular.

One-column lower triangular matrix

$$\mathcal{L}_j = \left\{ I + \begin{pmatrix}\boldsymbol{0}_j\\ \boldsymbol{l}\end{pmatrix}\boldsymbol{e}_j^T \;:\; \boldsymbol{l}\in\mathbb{R}^{n-j}\right\}$$

Inverse of $L_j \in \mathcal{L}_j$:

$$L_j^{-1} = I - \begin{pmatrix}\boldsymbol{0}_j\\ \boldsymbol{l}\end{pmatrix}\boldsymbol{e}_j^T$$

Multiplication of $L_j = I + \begin{pmatrix}\boldsymbol{0}_j\\ \boldsymbol{l}_1\end{pmatrix}\boldsymbol{e}_j^T \in \mathcal{L}_j, L_k = I + \begin{pmatrix}\boldsymbol{0}_k\\ \boldsymbol{l}_2\end{pmatrix}\boldsymbol{e}_k^T \in \mathcal{L}_k$:

$$L_jL_k = I + \begin{pmatrix}\boldsymbol{0}_j\\ \boldsymbol{l}_1\end{pmatrix}\boldsymbol{e}_j^T + \begin{pmatrix}\boldsymbol{0}_k\\ \boldsymbol{l}_2\end{pmatrix}\boldsymbol{e}_k^T$$

Pivoting property: If P_σ(a permutation) fixes the first j entries. i.e. $P_\sigma = \begin{pmatrix}I_j & 0\\ 0 & \tilde{P}\end{pmatrix}$, then

$$P_\sigma L_j = \widetilde{L_j}P_\sigma$$

where $\widetilde{L_j} \in \mathcal{L}_j$ is the same as L_j, but replacing $\boldsymbol{l}$ by $\tilde{P}\boldsymbol{l}$.

LU decomposition
If

$$L_1 = \begin{pmatrix} 1 & & & \\ -\frac{a_{21}}{a_{11}} & 1 & & \\ \vdots & & \ddots & \\ -\frac{a_{n1}}{a_{11}} & & & 1 \end{pmatrix}$$

then $L_1 A$ has 0 entries on first column except $1, 1$ entry. Repeat this process on $L_1 A[2 : end, 2 : end]$, and then repeat until we get an upper triangular matrix. i.e.

$$L_{n-1} ... L_2 L_1 A = U$$

then $A = LU$ where $L := L_1^{-1} L_2^{-1} ... L_{n-1}^{-1}$ must be lower triangular by rules of one-column lower triangular matrix mentioned above.

PLU decomposition
Sometimes we need to swaps rows to avoid problem of dividing very small numbers. Say P_1 swaps row 1 and row k where k maximises $|a_{k1}|$. Then we pick L_1 as in LU decomposition for $P_1 A$. Repeat this process, we can get

$$L_{n-1} P_{n-1} ... L_2 P_2 L_1 P_1 A = U$$

by pivoting property, we can swap P_2 and L_1, so $L_{n-1} P_{n-1} ... L_3 P_3 L_2 \widetilde{L_1} P_2 P_1 A = U$. Then we can swap L_3 with $L_2 \widetilde{L_1}$, getting $L_{n-1} P_{n-1} ... L_3 \widetilde{L_2} \widetilde{\widetilde{L_1}} P_3 P_2 P_1 A = U$. Repeat until getting $L_{n-1} L'_{n-2} ... L'_1 P_{n-1} ... P_1 A = U$. So $A = P^T LU$ where $P = P_1 ... P_{n-1}$, $L = (L'_1)^{-1} ... L_{n-1}^{-1}$.
code:

```
L,U,sigma = lu(A) # sigma is a vector encoding the permutation P
```

6.3 Cholesky decomposition

For square, symmetric and positive definite A, $A = LL^T$ for lower triangular L.
Positive Definite matrix $\forall x \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, $\boldsymbol{x}^T A \boldsymbol{x} > 0$.
Positive definite matrices have diagonal elements $a_{kk} > 0$.
If V is non-singular, A is positive definite, then $V^T AV$ is positive definite. In particular, if P_σ is a permutation, then $P_\sigma^T A P_\sigma$ is positive definite. If v is the vector representing P_σ, then in Julia $P_\sigma^T A P_\sigma = A[v, v]$

Theorem. A is symmetric positive definite $\Leftrightarrow$ it has Cholesky decomposition $A = LL^T$ where diagonal elements of L are all positive.
So if for symmetric matrix A, Cholesky decomposition fails (at some stage, first entry is negative), A cannot be positive definite.

Algorithm: Define

$$A_1 := A,\ \boldsymbol{v}_k := A_k[2 : n-k+1, 1],\ \alpha_k := A_k[1,1]$$

$$A_{k+1} := A_k[2 : n-k+1, 2 : n-k+1] - \frac{\boldsymbol{v}_k \boldsymbol{v}_k^T}{\alpha_k}$$

Then

$$L = \begin{pmatrix} \sqrt{a_1} & & & \\ \frac{v_1[1]}{\sqrt{a_1}} & \sqrt{a_2} & & \\ \vdots & \ddots & \ddots & \\ \frac{v_1[n-1]}{\sqrt{a_1}} & \cdots & \frac{v_{n-1}[1]}{\sqrt{a_{n-1}}} & \sqrt{a_n} \end{pmatrix}$$

The components of above decompositions are very easy to invert.
If a matrix A does not have Cholesky decomposition i.e. matrix is not positive definite,

```
cholesky(A)  # throws error
```

Positive symmetric matrix can also be decomposed into UU^T where U is upper triangular matrix.

6.4 Vector and matrix norm

p-norm Given $\boldsymbol{x} \in \mathbb{R}^n$, p-norm is defined as

$$\|\boldsymbol{x}\|_p := \left(\sum_{k=1}^{n} |x_k|^p \right)^{1/p}$$

Recall $\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2$ and $\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty$ **Fröbenius norm** For a matrix $A_{m\times n} = (a_{k,j})$, Fröbenius norm is basically treating matrix as a vector

$$\|A\|_F := \sqrt{\left(\sum_{k,j} |a_{k,j}|^2 \right)}$$

norm(A) in Julia means 2 norm.
Note $\|A\|_F = \sqrt{tr(A^T A)}$
And if Q is orthogonal, $\|QA\|_F = \|A\|_F$
Induced Matrix Norm Given $A_{m\times n}$ and $\|\cdot\|_X$ on $\mathbb{R}^m$, $\|\cdot\|_Y$ on $\mathbb{R}^n$.

$$\|A\|_{X\to Y} := \sup_{\boldsymbol{v}:\|v\|_X=1} \|Av\|_Y$$

If the matrix is square, define $\|A\|_X := \|A\|_{X\to X}$.
Properties of Induced Matrix Norm (abbreviated as $\|\cdot\|$ below)

- Triangular inequality: $\|A+B\| \leq \|A\| + \|B\|$
- Homogenecity: $\|cA\| = |c|\|A\|$
- Positive-definite: $\|A\| = 0 \Rightarrow A = 0$
- $\|Ax\|_Y \leq \|A\|_{X\to Y}\|x\|_X$
- Multiplicative inequality $\|AB\|_{X\to Z} \leq \|A\|_{Y\to Z}\|B\|_{X\to Y}$.

Note for induced p-norm

$$\left\|\begin{pmatrix} A \\ B \end{pmatrix}\right\|_p^p = \|A\|_p^p + \|B\|_p^p$$

Special cases:
$1-$norm: $\|A\|_1 = \max_j \|\boldsymbol{c}_j\|_1$ (max 1-norm of columns)
$\|A\|_\infty$ equals max 1-norm of rows.
$\|A\|_{1\to\infty}$ equals $\max_{k,j}|a_{k,j}|$
$\|A\|_2 =$ largest singular value. If A is diagonal, $\|A\|_2 = \max_k |d_k|$ (max diagonal entry). If Q is orthogonal, $\|Q\|_2 = 1$.
And $\|A\|_2 \leq \|A\|_F \leq \sqrt{r}\|A\|_2$ where $r = rank(A)$. So if A has rank 1, then 2-norm coincides with Fröbenius norm.
Denote $|A|$ as the matrix of absolute values of entries of A. $\||A|\|_1 = \|A\|_1, \||A|\|_\infty = \|A\|_\infty, \||A|\|_F = \|A\|_F$, but $\||A|\|_2 \neq \|A\|_2$ in general. But according to the above inequality,

$$\|A\|_2 \leq \|A\|_F = \||A|\|_F \leq \sqrt{s}\||A|\|_2 \leq \sqrt{\min(m,n)}\||A|\|_2$$

where $s = \text{rank}\,|A|$.

```
# 1 operator norm
opnorm(A, 1)
# 2 operator norm
opnorm(A)
# infinity operator norm
opnorm(A, Inf)
```

6.5 SVD decomposition

Reduced version
$A_{m\times n}$ with $rank(A) = r$ can be decomposed into $U\Sigma V^T$. $U_{m\times r}$, $V_{n\times r}$ have orthonormal columns. Σ is diagonal matrix with positive diagonal entries that are decreasing i.e. $d_1 \geq d_2 ... \geq d_r > 0$.
Full version
$A = U\Sigma V^T$. $U_{m\times m}, V_{n\times n}$ are orthogonal and Σ has only diagonal entries $d_{k,k}$.

Gram matrix Given matrix A, the Gram matrix is $A^T A$.

$Ker(A^TA) = Ker(A)$ and $A^TA = QDQ^T$ for some orthogonal Q and diagonal D with non-negative diagonal entries. Note singular values of A are exactly square root of Eigenvalues of A^TA.

Singular values and matrix norms

- $\|A\|_F^2 = \sigma_1^2 + ... + \sigma_m^2$, sum of squares of singular values
- $\|A\|_2$ is the largest singular value i.e. the first diagonal entry of Σ
- $\|A^{-1}\|_2$ is reciprocal of the smallest non-zero singular value

$rank(A)$ = number of non-zero singular values = number of non-zero Eigenvalues of A^TA

Application: low-rank approximation
If $k < r = rank(A)$, and A has SVD $A = U\Sigma V^T$. Let U_k, Σ_k, V_k be first k columns of U, Σ, V respectively. Then $A_k := U_k\Sigma_kV_k^T$ is the best 2-norm approximation to matrix A:

$$\forall B \text{ with rank } k, \|A - A_k\|_2 \leq \|A - B\|_2$$

this is a strong compression technique.
SVD in Julia

```
U, D, V = svd(A) # where D is the diagonal matrix with singular values.
sigma = svdvals(A)  # get singular values in descending order.
# Compute low-rank approximation
D_k = Diagonal(D[1:k])   # Diagonal matrices are treated as vectors in Julia
U_k = U[:, 1:k]
V_k = V[:, 1:k]
A_k = U_k * D_k * V_k'
```

7 Conditional Numbers

Theorem. *If $\widetilde{\boldsymbol{x}}$ is approximation to $A\boldsymbol{x} = \boldsymbol{b}$, where A is non-singular. For any natural norm (any p-norm and p-norm induced matrix norm), as long as they are consistent throughout the following inequalities,*

$$\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\| \leq \|\boldsymbol{r}\|\|A^{-1}\|$$

where $\boldsymbol{r} = \boldsymbol{b} - A\widetilde{\boldsymbol{x}}$ is called the residual vector. If given $\boldsymbol{x}, \boldsymbol{b} \neq 0$, we have

$$\frac{\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|} \leq \|A\|\|A^{-1}\|\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}$$

Proof.

$$\boldsymbol{r} = \boldsymbol{b} - A\widetilde{\boldsymbol{x}} = A\boldsymbol{x} - A\widetilde{\boldsymbol{x}} = A(\boldsymbol{x} - \widetilde{\boldsymbol{x}})$$

so $A^{-1}\boldsymbol{r} = \boldsymbol{x} - \widetilde{\boldsymbol{x}}$, by property of norm,

$$\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\| = \|A^{-1}\boldsymbol{r}\| \leq \|A^{-1}\|\|r\|$$

If given $\|x\|, \|b\| \neq 0$, as $\boldsymbol{b} = A\boldsymbol{x}$, $\|\boldsymbol{b}\| \leq \|A\|\|\boldsymbol{x}\|$. So $1/\|\boldsymbol{x}\| \leq \|A\|/\|\boldsymbol{b}\|$, and

$$\frac{\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|} \leq \|A\|\|A^{-1}\|\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}$$

□

Note the term $\frac{\|\boldsymbol{x}-\widetilde{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|}$ is called relative error. And $\|A\|\|A^{-1}\|$ provides a relation between residual vector and relative residual(i.e. $\|\boldsymbol{r}\|/\|\boldsymbol{b}\|$).

Therefore, we define $K(A) := \|A\|\|A^{-1}\|$. So the inequality becomes

$$\frac{\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|} \leq K(A)\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}$$

A is **well-conditioned** if $K(A)$ is close to 1, and **ill-conditioned** if it deviates far from 1. So for well-conditioned matrix, small residual vector implies accurate approximation.
$K_p(A)$ denotes $\|A\|_p\|A^{-1}\|_p$.

Proposition. *If $|\epsilon_i| \leq \epsilon$ and $n\epsilon < 1$, then*

$$\prod_{i=1}^{n}(1 + \epsilon_i) = 1 + \theta_n$$

where $|\theta_n| \leq \frac{n\epsilon}{1-n\epsilon}$

Proposition (dot product backward error)**.** *The floating point dot product satisfies*

$$dot(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x} + \delta\boldsymbol{x})^T\boldsymbol{y}$$

where $|\delta\boldsymbol{x}| \leq \frac{n\epsilon_m}{2-n\epsilon_m}|\boldsymbol{x}|$

Proposition (matrix multiplication backward error)**.** *Consider floating point matrix multiplication on $A \in \mathbb{R}^{m\times n}$*

$$mul(A, \boldsymbol{x}) = \begin{pmatrix} dot(A[1,:], \boldsymbol{x}) \\ dot(A[2,:], \boldsymbol{x}) \\ \vdots \\ dot(A[m,:], \boldsymbol{x}) \end{pmatrix}$$

then $mul(A, \boldsymbol{x}) = (A + \delta A)\boldsymbol{x}$ where entries of δA: δa satisfies $|\delta a| \leq \frac{n\epsilon_m}{2-n\epsilon_m}|a|$. So

$$\|\delta A\|_1 \leq \frac{n\epsilon_m}{2 - n\epsilon_m}\|A\|_1$$

$$\|\delta A\|_2 \leq \sqrt{\min\{m,n\}}\frac{n\epsilon_m}{2-n\epsilon_m}\|A\|_2$$
$$\|\delta A\|_\infty \leq \frac{n\epsilon_m}{2-n\epsilon_m}\|A\|_\infty$$

Theorem (Relative error)**.** *If $\|\delta A\| \leq \|A\|\epsilon$, then*

$$\frac{\|\delta A\boldsymbol{x}\|}{\|A\boldsymbol{x}\|} \leq K(A)\epsilon$$

the general conclusion is if $p = 1, \infty$.

$$\frac{\|mul(A,\boldsymbol{x}) - A\boldsymbol{x}\|_p}{\|A\boldsymbol{x}\|_p} \leq K_p(A)\frac{n\epsilon_m}{2-n\epsilon_m}$$

for $p = 2$, multiply $\sqrt{\min\{m,n\}}$ in front.

8 Differential Equations

8.1 Integration

Consider the differential equation

$$u(a) = c,\ u'(x) = f(x) \quad \forall x \in [a,b]$$

break interval $[a,b]$ into $n-1$ intervals using cutting points $a =: x_1, x_2, ..., x_n := b$ so each interval $[x_k, x_{k+1}]$ has length $(b-a)/(n-1)$.

using forward difference one can find a linear system

$$D_h := \frac{1}{h}\begin{pmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix}\boldsymbol{u} = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_{n-1}) \end{pmatrix} =: \boldsymbol{f}$$

such that $\boldsymbol{u} := (u_1\ \dots\ u_n)^T$ approximates $(u(x_1)\ \dots\ u(x_n))^T$. Note the matrix $D_h \in \mathbb{R}^{(n-1)\times n}$. Using inital condition:

$$\begin{pmatrix} e_1^T \\ D_h \end{pmatrix}\boldsymbol{u} = \begin{pmatrix} c \\ \boldsymbol{f} \end{pmatrix}$$

where now the matrix $\begin{pmatrix} e_1^T \\ D_h \end{pmatrix}$ is lower bi-diagonal and the system can be solved by forward substitution in $O(n)$. i.e.

$$u_1 = c,\ u_{k+1} = u_k + hf(x_k)$$

If using central difference instead, $\boldsymbol{f}$ becomes $(f(m_1) \ \dots \ f(m_{n-1}))^T$ where $m_k := (x_{k+1} + x_k)/2$ are midpoints. Other values are unchanged. As

$$u'(m_k) \approx \frac{u_{k+1} - u_k}{h} = f(m_k)$$

It turns out that error of using forward difference decreases in $O(\frac{1}{n})$ but error of central difference decreases in $O(\frac{1}{n^2})$. Errors of both methods stabilises in the end.

```
# Implementation
# build grid using x
x = range(0, 1; length=n)
h = step(x)
# build the lower bi-diagonal matrix mentioned above
L = Bidiagonal([1; fill(1/h, n-1)], fill(-1/h, n-1), :L)


m = (x[1:end-1] + x[2:end])/2 # midpoints

# evaluation of f
vec_f = f.(x[1:end-1])
vec_f_m = f.(m)   # evaluation at mid-points instead
u = L \ [c; vec_f]   # forward difference implementation
u_m = L \ [c; vec_f_m]    # central difference implementation
```

8.2 Time-evolution and Euler method

Consider

$$u(0) = c, \ u'(t) - a(t)u(t) = f(t) \quad \forall t \in [0, T]$$

again we cut $[0, T]$ into n-point grids with $h := T/(n-1), t_k := (k-1)h$.
Vector version:

$$\boldsymbol{u}(0) = \boldsymbol{c}, \ \boldsymbol{u}'(t) - A(t)\boldsymbol{u}(t) = \boldsymbol{f}(t) \quad \forall t \in [0, T]$$

Forward Euler using forward difference on linear version gives

$$\begin{pmatrix} 1 & & & \\ -a(t_1) - 1/h & 1/h & & \\ & \ddots & \ddots & \\ & & -a(t_{n-1}) - 1/h & 1/h \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} c \\ f(t_1) \\ \vdots \\ f(t_{n-1}) \end{pmatrix}$$

this gives iteration process

$$u_1 = c, \ u_{k+1} = (1 + ha(t_k))u_k + hf(t_k)$$

this generalise to vector differential equation:

$$\boldsymbol{u}_1 = \boldsymbol{c},\ \boldsymbol{u}_{k+1} = (I + hA(t_k))\boldsymbol{u}_k + h\boldsymbol{f}(t_k)$$

Backward Euler using backward difference on linear version gives

$$\begin{pmatrix} 1 & & & \\ -1/h & 1/h - a(t_2) & & \\ & \ddots & \ddots & \\ & & -1/h & 1/h - a(t_n) \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} c \\ f(t_2) \\ \vdots \\ f(t_n) \end{pmatrix}$$

this gives iteration process

$$u_1 = c,\ u_{k+1} = (1 - ha(t_{k+1}))^{-1}\left(u_k + hf(t_{k+1})\right)$$

generalise to vector differential equation:

$$\boldsymbol{u}_1 = \boldsymbol{c},\ \boldsymbol{u}_{k+1} = (I - hA(t_{k+1}))^{-1}\left(\boldsymbol{u}_k + h\boldsymbol{f}(t_{k+1})\right)$$

8.3 Non-linear simple case

Consider

$$\boldsymbol{u}(0) = \boldsymbol{c},\ \boldsymbol{u}'(t) = f(t, \boldsymbol{u}(t)) \quad \forall\, t \in [0, T]$$

forward Euler method gives iteration process:

$$\boldsymbol{u}_1 = \boldsymbol{c},\ \boldsymbol{u}_{k+1} = \boldsymbol{u}_k + h\boldsymbol{f}(t_k, \boldsymbol{u}_k)$$

backward is too complicated, omitted.

8.4 Laplace and Poisson equation

The Laplacian $\nabla^2 u$ of a scalar-valued function $u(x, y, z)$ can be understood as a measure of concavity of u. $\nabla^2 u < 0$ means values of u around this point are on average smaller than this point. ∇^2 is very similar to $\frac{d^2}{dx^2}$.

Poisson equation specifies the concavity of a curve,

$$\nabla^2 u = f(\boldsymbol{x})$$

with Dirichlet boundary condition

$$u(\boldsymbol{x}) = g(\boldsymbol{x}) \text{ for } \boldsymbol{x} \in D$$

where D is the boundary of region.
Laplace equation is a special case of Poisson equation where $f = 0$.

consider simple case $f : [a, b] \to \mathbb{R}$.

$$u(a) = c_0,\ u''(x) = f(x),\ u(b) = c_1$$

Cut $[a,b]$ into $n-1$ intervals at $a =: x_1, ..., x_n := b$.
Using finite difference of second derivative, we have

$$\begin{pmatrix} 1 & 0 & & & \\ \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} \\ & & & 0 & 1 \end{pmatrix} \boldsymbol{u} = \begin{pmatrix} c_0 \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ c_1 \end{pmatrix}$$

The above linear system is equivalent to

$$T\boldsymbol{u} := \begin{pmatrix} 1 & & \\ & \frac{1}{h^2}\Delta & \\ & & 1 \end{pmatrix} \boldsymbol{u} = \begin{pmatrix} c_0 \\ f(x_2) - \frac{c_0}{h^2} \\ f(x_3) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) - \frac{c_1}{h^2} \\ c_1 \end{pmatrix}$$

where the Laplacian of this graph

$$\Delta := \begin{pmatrix} -1 & 1 & & & \\ 1 & -2 & \ddots & & \\ & 1 & \ddots & 1 & \\ & & \ddots & -2 & 1 \\ & & & 1 & -1 \end{pmatrix}$$

this matrix is actually negative definite, i.e. $-\Delta = LL^T$ for some lower diagonal L.

```
# Julia implementation
x = range(0, 1; length = n)
h = step(x)
# build the matrix
T = Tridiagonal([fill(1/h^2, n-2); 0], [1; fill(-2/h^2, n-2); 1],
[0; fill(1/h^2, n-2)])
u = T \ [c_0; exp.(x[2:end-1]); c_1]
# plot the solution
scatter(x, u)
```

8.5 Convergence

Toeplitz matrix. matrices that are constant on all diagonals, i.e.

$$A = \begin{pmatrix} a_0 & a_{-1} & a_{-2} & \dots & \dots & a_{-(n-1)} \\ a_1 & a_0 & a_{-1} & \ddots & & \vdots \\ a_2 & a_1 & \ddots & \dots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \dots & \dots & a_2 & a_1 & a_0 \end{pmatrix}$$

One special case where we can find inverse

$$\begin{pmatrix} 1 & & & & \\ -l & 1 & & & \\ & -l & 1 & & \\ & & \ddots & \ddots & \\ & & & -l & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & & & & \\ l & 1 & & & \\ l^2 & l & 1 & & \\ \vdots & \ddots & \ddots & \ddots & \\ l^{n-1} & \dots & l^2 & l & 1 \end{pmatrix}$$

Convergence of

$$u(0) = c,\ u'(t) = au(t) + f(t) \text{ for } t \in [0, 1]$$

where a is a constant using forward/backward finite difference. If u is twice differentiable and u'' is uniformly bounded, then

$$\|\boldsymbol{u} - \boldsymbol{u}_{\text{ex}}\|_\infty = O(1/n)$$

where $\boldsymbol{u}$ is the approximation and $\boldsymbol{u}_{\text{ex}}$ is exact solution.

As for the Poisson equation, if $u^(4)$ exists and $u^(4)$ is uniformly bounded, then

$$\|\boldsymbol{u} - \boldsymbol{u}_{\text{ex}}\|_\infty = O(1/n^2)$$

where $\boldsymbol{u}$ is the approximation and $\boldsymbol{u}_{\text{ex}}$ is exact solution.

9 Fourier series

Recall that we can approximate periodic functions by finite sum, assume f is 2π-periodic,

$$f(\theta) \approx \sum_{k=-m}^{m} \hat{f}_k e^{ik\theta} =: f_m(\theta)$$

where $\hat{f}_k := \frac{1}{2\pi} \int_0^{2\pi} f(\theta) e^{-ik\theta}\, d\theta$.

Theorem. *If coefficients converge absolutely, i.e.* $\|\hat{f}\|_1 := \sum_{k=-\infty}^{\infty} |\hat{f}_k| < \infty$*, then* $f_m(\theta)$*, the finite sum, converges to* $f(\theta)$*.*

For coefficients to converge absolutely, it is enough to have f is periodic(so f' will be periodic) and f'' exist and is uniformly bounded. Note f, f', f'' are all 2π-periodic. The more derivatives exist, the faster f_m converges to f.

Note if $0 = \hat{f}_{-1} = \hat{f}_{-2} = \ldots$, and $\hat{f}_k$ converges absolutely, then let $z := e^{i\theta}$,

$$f(z) = \sum_{k=0}^{\infty} \hat{f}_k e^{ik\theta} = \sum_{k=0}^{\infty} \hat{f}_k z^k$$

i.e. a Taylor series.

In practice we might have to approximate the coefficients by cutting the interval $[0, 2\pi]$ into n pieces. And if $n = 2m+1$(i.e. n is odd), then we approximate coefficients $\hat{f}_{-m}, ..., \hat{f}_m$ using trapezium rule. For even $n = 2m$, we approximate coefficients $\hat{f}_{-m}, ..., \hat{f}_{m-1}$ instead.

Trapezium rule and average of function For 2π-periodic function f,

$$\int_0^{2\pi} f(\theta)\, d\theta \approx \frac{2\pi}{n} \sum_{j=1}^{n-1} f(\theta_j) =: 2\pi\Sigma_n[f(\theta)]$$

where $n = 2m+1$ and $\theta_j := \frac{2\pi j}{n}$ partitions $[0, 2\pi]$.

So in the case of Fourier coefficients.

$$\hat{f}_k \approx \frac{1}{n} \sum_{j=0}^{n-1} f(\theta_j) e^{-ik\theta j} = \Sigma_n[f(\theta)e^{-ik\theta}]$$

we use this as approximation of $\hat{f}_k$ and call it $\hat{f}_k^n$.

Proposition.

$$\Sigma_n[e^{ik\theta}] = \begin{cases} 1, & \text{if } k = ..., -n, 0, n, 2n, ... \\ 0, & \text{otherwise} \end{cases}$$

So

$$\Sigma_n[e^{ik\theta}e^{-ij\theta}] = \begin{cases} 1, & \text{if } j = k + pn \text{ for some } p \in \mathbb{Z} \\ 0, & \text{otherwise} \end{cases}$$

therefore,

$$\hat{f}_k^n = \sum_{p=-\infty}^{\infty} \hat{f}_{k+pn} \qquad (1)$$

Using this equation (1), we can prove that if $0 = \hat{f}_{-1} = \hat{f}_{-2} = \dots$ i.e. Taylor series case,

$$f_n(\theta) := \sum_{k=0}^{n-1} \hat{f}_k^n e^{ik\theta}$$

converges uniformly to f if given that $\sum_{k=0}^{\infty} |\hat{f}_k| < \infty$. This means our approximation using trapezium rule converges to the true function.
Another consequence of equation (1) is $\hat{f}_{k+pn}^n = \hat{f}_k^n$ for any $p \in \mathbb{Z}$. So Taylor series coefficients can be related to Fourier coefficients by permutation. Given Taylor coefficients $(\hat{f}_k^n)_{k=0,\dots,n-1}$, $n = 2m+1$, to build Fourier coefficients $(\hat{f}_k^n)_{k=-m,\dots,0,1,\dots,m}$, use permutation

$$\begin{pmatrix} 1 & \cdots & m & | & m+1 & \cdots & n \\ m+2 & \cdots & n & | & 1 & \cdots & m+1 \end{pmatrix}$$

9.1 Tricks for finding Fourier coefficients

Find expression for $\hat{f}_k$ first, then consider $0 \le k \le n-1$ (or use $-n \le k \le -1$, choose the convenient range) $\forall m \in \mathbb{Z}$,

$$\hat{f}_{k+mn}^n = \sum_{p=-\infty}^{\infty} \hat{f}_{k+pn}$$

By a famous integral in complex analysis, if $k \in \mathbb{Z}, C = \{e^{i\theta} : \theta \in [0, 2\pi]\}$

$$\oint_C \frac{1}{z^k}\, dz = \begin{cases} 2\pi i & \text{if } z = -1 \\ 0 & \text{otherwise} \end{cases}$$

substitution $z = e^{-i\theta}$ or $z = e^{i\theta}$ helps you with lots of integrals. e.g.

$$\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

use $z = e^{i\theta}$ for $\frac{b}{b - ce^{i\theta}}$ where $b > c$, and $z = e^{-i\theta}$ for $b < c$. Taylor series can be helpful here.

9.2 Discrete Fourier Transform (DFT)

Dividing $[0, 2\pi]$ into n pieces $0 =: \theta_0, \theta_1, \dots, \theta_n =: 2\pi$, one can recast discrete Fourier coefficients using matrix:

$$\begin{pmatrix} \hat{f}_0^n \\ \vdots \\ \hat{f}_{n-1}^n \end{pmatrix} = \frac{1}{\sqrt{n}} Q_n \begin{pmatrix} f(\theta_0) \\ \vdots \\ f(\theta_{n-1}) \end{pmatrix}$$

where matrix Q_n is

$$\frac{1}{\sqrt{n}}\begin{pmatrix}1 & 1 & 1 & \cdots & 1\\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)}\\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{(n-1)^2}\end{pmatrix} \quad \text{where } \omega = e^{i\frac{2\pi}{n}}$$

this matrix is unitary. i.e. its inverse is just Q_n^*. And using this linear system one can deduce $f_n(\theta_j) := \sum_{k=0}^{n-1} \hat{f}_k^n e^{ik\theta_j} = f(\theta_j)$, same applies to Fourier series (a sum from $-m$ to m instead). So Taylor/Fourier series correctly interpolates f at discretised points θ_j. (Warning: this does not mean $\theta \mapsto \sum_{k=0}^{n-1} \hat{f}_k^n e^{ik\theta}$ is a good estimate for f at all points in $[0, 2\pi]$. But this is true for Fourier series given that second derivative of f exists and is uniformly bounded.)

Julia implementation:

```
# take theta_0, ..., theta_{n-1}, dropping theta_n == 2*pi
theta = range(0,2*pi; length=n+1)[1:end-1]

# build the DFT matrix
Q = 1/sqrt(n) * [exp(-im*(k-1)*theta[j]) for k = 1:n, j=1:n]
hat_f = 1/sqrt(n) * Q * f.(theta)

function finitefourier(hat_f, theta)
    # taking digitisation hat_f and return an estimate of f at theta
    m = n ÷ 2    # use coefficients between -m:m
    ret = 0.0 + 0.0im   # coefficients are complex so we need complex arithmetic

    # we need to permute hat_f as it is taken as coefficients of Taylor series
    for k = 0:m
        ret += hat_f[k+1] * exp(im*k*theta)
    end
    for k = -m:-1
        ret += hat_f[end+k+1] * exp(im*k*theta)
    end
    ret
end

function finitetaylor(hat_f, theta)
    ret = 0.0 + 0.0im # coefficients are complex so we need complex arithmetic
    for k = 0:n-1
        ret += hat_f[k+1] * exp(im*k*theta)
    end
    ret
end
```

```
# build functions according to Taylor and Fourier series
f_n = theta -> finitefourier(hat_f, theta)
t_n = theta -> finitetaylor(hat_f, theta)
```

Time complexity of this algorithm $O(n^2)$. There is a faster way.

9.3 Fast Fourier Transform FFT

Complexity can be reduced to $O(n \log n)$ in this way.

Denote $\omega_n = e^{i\frac{2\pi}{n}}$ and $\omega_{2n} = e^{i\frac{\pi}{n}}$ so $\omega_{2n}^2 = \omega_n$. Using this we can express

$$\begin{pmatrix} 1 \\ \omega_{2n} \\ \vdots \\ \omega_{2n}^{2n-1} \end{pmatrix} = P_\sigma^T \begin{pmatrix} 1 \\ \omega_n \\ \omega_n^2 \\ \vdots \\ \omega_n^{n-1} \\ \omega_{2n} \\ \omega_{2n}\omega_n^2 \\ \vdots \\ \omega_{2n}\omega_n^{n-1} \end{pmatrix}$$

where permutation σ is

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & \cdots & n & n+1 & \cdots & 2n \\ 1 & 3 & 5 & \cdots & 2n-1 & 2 & \cdots & 2n \end{pmatrix}$$

so

$$Q_{2n}^* = \frac{1}{\sqrt{2n}} \left(P_\sigma^T \begin{pmatrix} \mathbf{1}_n \\ \mathbf{1}_n \end{pmatrix} \quad P_\sigma^T \begin{pmatrix} \boldsymbol{\omega}_n \\ \omega_{2n}\boldsymbol{\omega}_n \end{pmatrix} \quad P_\sigma^T \begin{pmatrix} \boldsymbol{\omega}_n^2 \\ \omega_{2n}^2\boldsymbol{\omega}_n^2 \end{pmatrix} \quad \cdots \quad P_\sigma^T \begin{pmatrix} \boldsymbol{\omega}_n^{2n-1} \\ \omega_{2n}^{2n-1}\boldsymbol{\omega}_n^{2n-1} \end{pmatrix} \right)$$

$$= \frac{1}{\sqrt{2}} P_\sigma^T \begin{pmatrix} Q_n^* & Q_n^* \\ Q_n^* D_n & -Q_n^* D_n \end{pmatrix} \qquad \text{where } D_n := diag\{1, \omega_{2n}, \cdots, \omega_{2n}^{n-1}\}$$

$$= \frac{1}{\sqrt{2}} P_\sigma^T \begin{pmatrix} Q_n^* & 0 \\ 0 & Q_n^* \end{pmatrix} \begin{pmatrix} I_n & I_n \\ D_n & -D_n \end{pmatrix}$$

So as a result, if $n = 2^q$, multiplication by Q_n^* can be done iteratively using at most $3nq$ additions and multiplications.

```
# Julia Implementation
# package FFTW deals with FFT

# evenly spaced points from 0:2pi, dropping last node
theta = range(0, 2*pi; length=n+1)[1:end-1]
```

```
# fft returns discrete Fourier coefficients n*[hat_f_0, ..., hat_f_(n-1)]
# This is a Taylor series
fc = fft(f.(theta))/n

# rearrange fc to Fourier series hat_f
hat_f = [fc[m+2:end]; fc[1:m+1]]

# build the function f_n estimating f
f_n = theta -> transpose([exp(im*k*theta) for k=-m:m]) * hat_f
```

10 Orthogonal polynomials

Fourier series approximation requires the function to be periodic, what about general smooth function (not periodic ones)? We use orthogonal polynomials. i.e. we seek sequence of polynomials p_k with desired property and try to express $f(x) = \sum c_k(x)p_k(x)$ where c_k are coefficients.

Definition 10.1 (Graded polynomials)**.** A sequence of polynomials $(p_n)_{n\geq 0}$ is called graded if each p_n has degree n. (Hidden in this assumption: $p_n \neq 0$ for all n) Usually we denote $p_n := c_n x^n + O(x^{n-1})$

Given integrable weight function $w(x)$ s.t. $w(x) > 0$ when $x \in (a, b)$, inner product can be defined for continuous functions:

$$\langle f, g\rangle := \int_a^b f(x)g(x)w(x)\,dx$$

Definition 10.2 (Orthogonal Polynomials)**.** Graded polynomials $(p_n)_{n\geq 0}$ is called orthogonal polynomials(OPs) if $\langle p_n, p_m\rangle = 0$ for all $m \neq n$.

One way to obtain OPs is performing Gram-Schmidt process on $\{1, x, ..., x^n\}$ to obtain a unique sequence of monic orthogonal polynomials $(p_k)_{0\leq k\leq n}$.

Note $\|p_n\|^2 := \langle p_n, p_n\rangle = \int_a^b p_n^2(x)w(x)\,dx > 0$. And $(q_n)_{n\geq 0}$ is called orthonormal if it is orthogonal and $\|q_n\|^2 = 1$. And we call $(p_n)_{n\geq 0}$ monic if p_n are all monic.

Proposition (Base expansion)**.** *If $r(x)$ is polynomial of degree n and $(p_n)_{n\geq 0}$ is orthogonal, then*

$$r(x) = \sum_{k=0}^{n} \frac{\langle p_k, r\rangle}{\|p_k\|^2} p_k(x)$$

basically, $\{p_0, ..., p_n\}$ is an orthogonal basis of set of all polynomials with degree $\leq n$. Expansion is easier if we have orthonormal polynomials $(q_n)_{n\geq 0}$

$$r(x) = \sum_{k=0}^{n} \langle q_k, r\rangle q_k(x)$$

Proposition (Orthogonal to lower degree)**.** *Given weight function $w(x)$, define associated inner product $\langle \cdot\ ,\ \cdot \rangle$ on the space of continuous functions. Fix $n \in \mathbb{N}$. Given OPs $(p_k)_{0 \le k \le n}$. For any polynomial p,*

$$\begin{cases} \deg p = n \\ \langle p, r \rangle = 0 \ \textit{whenever} \ \deg r < n \end{cases} \iff p(x) = cp_n(x)$$

Proof. ($\Rightarrow$) Say $p(x) = cx^n + O(x^{n-1})$, then $p - cp_n$ has degree $\le n-1$ as p_n is monic. So for all $k \le n-1$, by linearity of inner product,

$$\langle p_k, p - cp_n \rangle = \langle p_k, p \rangle - c\langle p_k, p_n \rangle$$

RHS is 0 as $deg(p_k) \le n-1$, by our assumption, $\langle p_k, p \rangle = 0$. And $\langle p_k, p_n \rangle = 0$ as they belong to an orthogonal family. Also note that $\{p_0, ..., p_{n-1}\}$ is a basis of polynomials of degree $\le n-1$, $\langle p_k, p - cp_n \rangle = 0$ for all $k \le n-1$ means $p - cp_n = 0$, i.e. $p = cp_n$.

($\Leftarrow$) Assume $p = cp_n$, $\{p_0, ..., p_{n-1}\}$ is a basis so for any $r(x)$ with degree $\le n-1$, we can write

$$r(x) = \sum_{k=0}^{n-1} r_k p_k(x)$$

for some constants r_k. Then by linearity of inner product,

$$\langle p, r \rangle = \langle cp_n, \sum_{k=0}^{n-1} r_k p_k \rangle = \sum_k cr_k \langle p_n, p_k \rangle = 0$$

□

Remark. This theorem is important, as given a constant k_n, we can define corresponding OP $p_n(x)$ to be the unique polynomial s.t. $\langle p_n, r \rangle = 0$ whenever $\deg r < n$.

Proposition (Three-term recurrence)**.** *Using the OPs $(p_k)_{0 \le k \le n}$ and inner product used in the last proposition, there exists constants a_n, b_n, c_{n-1} where last two are non-zero s.t.*

$$xp_0(x) = a_0 p_0(x) + b_0 p_1(x) \qquad \textit{(1)}$$

$$xp_n(x) = c_{n-1} p_{n-1}(x) + a_n p_n(x) + b_n p_{n+1}(x) \qquad \textit{(2)}$$

Proof. Again remember that $(p_k)_{0 \le k \le n}$ makes a basis. So (1) is true as $\deg(xp_0(x)) = 1$. And since $p_0 \ne 0$, $b_0 \ne 0$. For the second equation, notice $\deg(xp_n(x)) = n+1$ but

$$\langle xp_n, p_k \rangle = \langle xp_k, p_n \rangle$$

by definition of inner product. And if $\deg(xp_k(x)) < n$, i.e. $\deg(p_k) = k < n-1$, then we have RHS of equation being 0 by last proposition. So only the first three term are left when expanding xp_n with basis $(p_k)_{0\leq k\leq n+1}$. So (2) follows.

$$b_n = \frac{\langle xp_n, p_{n+1}\rangle}{\|p_{n+1}\|^2} \neq 0$$

as $\deg(xp_n(x)) = n+1$.

$$c_{n-1} = \frac{\langle xp_n, p_{n-1}\rangle}{\|p_{n-1}\|^2} = \frac{\langle xp_{n-1}, p_n\rangle}{\|p_{n-1}\|^2} = b_{n-1}\frac{\|p_n\|^2}{\|p_{n-1}\|^2} \neq 0$$

□

Remark. If p_k are monic, $b_n = 1$ for all n.
If $(p_k)_{1\leq k\leq n}$ is orthonormal, then we have $c_n = b_n$.

Using this property, one can quickly generate the OPs $(p_k)_{0\leq k\leq n}$. First, we need to know $p_0(x)$ (usually taken to be 1) and weight function, then

$$a_n = \frac{\langle xp_n, p_n\rangle}{\|p_n\|^2}, \quad b_n = \frac{\langle xp_n, p_{n+1}\rangle}{\|p_{n+1}\|^2}, \quad c_{n-1} = \frac{\|p_n\|^2}{\|p_{n-1}\|^2}b_{n-1}$$

(set $b_n = 1$ to get monic OPs) iteration process is given below:

$$b_0p_1(x) = xp_0(x) - a_0p_0(x), \quad b_np_{n+1}(x) = xp_n(x) - c_{n-1}p_{n-1}(x) - a_np_n(x)$$

We can recast these relationships into a matrix system, so if we write $P(x) := (p_0(x) \mid p_1(x) \mid p_2(x) \mid p_3(x)\cdots)$,

$$xP(x) = P(x)X$$

where matrix X is

$$\begin{pmatrix} a_0 & c_0 & & \\ b_0 & a_1 & c_1 & \\ & b_1 & a_2 & \ddots \\ & & \ddots & \ddots \end{pmatrix}$$

Note if given OPs are orthonormal, X will be symmetric. We call X^T the Jacobi matrix. If we are dealing with general degree n polynomial $f(x) = \sum_{k=0}^n d_kp_k(x)$, then $f(x) = P(x)\boldsymbol{d}$ where $\boldsymbol{d} = \begin{pmatrix} d_0 & d_1 & \cdots & d_n\end{pmatrix}^T$, and

$$xf(x) = P(x)X\boldsymbol{d}$$

and given polynomial $a(x)$, we have $a(x)f(x) = P(x)a(X)\boldsymbol{d}$ where $a(X)$ is the polynomial $a(x)$ applied to matrix X.

Using the above method, with a basic low degree polynomial e.g. $f(x) = x^2$, one can first expand f into $\sum d_k p_k(x)$ by $d_k = \langle f, p_k\rangle / \|p_k\|^2$. Then after multiplying f by another polynomial $a(x)$, we do not have to calculate each d_k by the same method again, but use $a(x)f(x) = P(x)a(X)\boldsymbol{d}$.

Orthonormal case If given OPs (p_k) and its corresponding Jacobi matrix X, and we build orthonormal (q_k) from (p_k) by defining $q_k := \frac{p_k}{\|p_k\|}$, then we know $Q(x) := (q_0(x) \mid q_1(x) \mid q_2(x) \mid q_3(x) \cdots) = P(x)D$ for some diagonal matrix D. Then $xQ(x) = xP(x)D = P(x)XD = Q(x)D^{-1}XD$. So Jacobi matrix for (q_k) is $D^{-1}XD$.

10.1 Classical polynomials

These are the polynomials studied in the history that has special meanings/properties.

Chebyshev Polynomial(first kind) Let T_n be OPs w.r.t. $w(x) = \frac{1}{\sqrt{1-x^2}}$ on $[-1,1]$, s.t. $T_0(x) = 1, T_n(x) = 2^{n-1}x^n + O(x^{n-1})$. ($2^{n-1}$ are the normalising constants, uniquely determine T_n due to orthogonal to lower degree proposition).

$T_n(\cos(\theta)) = \cos n\theta$, and so $T_n(x) = \cos(n \arccos x)$. (This trick of changing variable $x = \cos\theta$ is very useful, one can use it to calculate $\|T_n\|_2$) The three-term recurrence for T_n is $xT_0(x) = T_1(x), \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$.

Note if $f(x) = \sum_{k=0}^{\infty} a_k T_k(x)$, then

$$f(\cos\theta) = \sum_{k=0}^{\infty} a_k \cos k\theta = \sum_{k=0}^{\infty} a_k \left(\frac{e^{ik\theta} + e^{-ik\theta}}{2}\right)$$

$$= \sum_{k=-\infty}^{-1} \frac{a_k}{2} e^{ik\theta} + a_0 + \sum_{k=1}^{\infty} \frac{a_k}{2} e^{ik\theta}$$

so $a_0 = \hat{f}_0, a_k = 2\hat{f}_k$. where $\hat{f}_k$ are the Fourier coefficients.

Chebyshev Polynomial(second kind) $w(x) = \sqrt{1-x^2}$ on $[-1,1]$, $U_0(x) = 1$, $U_n(x) = 2^n x^n + O(x^{n-1})$.
It can be shown in similar manner that:

$$U_n(x) = \frac{\sin(n+1)\theta}{\sin\theta}$$

and three term recurrence formulae is

$$xU_0(x) = \frac{1}{2}U_1(x), \quad xU_n(x) = \frac{1}{2}U_{n-1}(x) + \frac{1}{2}U_{n+1}(x)$$

Legendre polynomials

Legendre polynomials P_n are polynomials w.r.t $w(x) = 1$ on $[-1, 1]$ s.t. $P_n(x) = \frac{1}{2^n}\binom{2n}{n} x^n + O(x^{n-1})$.
There is no clean closed form for Legendre polynomials, but we do have Rodriguez formulae:

$$P_n(x) = \frac{1}{(-2)^n n!}\left(\frac{d}{dx}\right)^n (1-x^2)^n$$

the proof of Rodriguez formulae is divided into the following steps:

- First try to find a three-term recurrence formulae. (Sometimes you can include derivatives of p_n, p_{n-1})
- Verify such P_n are graded polynomials.
- orthogonal to all lower degrees on $[-1, 1]$. (Using integration by part and the fact that for any polynomial p, first $n-1$ derivatives of p^n has factor p.)
- have correct normalising constants $k_n = \frac{1}{2^n}\binom{2n}{n}$. (Differentiate, but only taking care of highest power of x)

One can derive the first few terms by differentiating n times, taking care of the highest powers of x. e.g. First two terms are:

$$P_0(x) = 1, P_1(x) = x, \quad P_n(x) = \frac{(2n)!}{2^n(n!)^2}x^n - \frac{(2n-2)!}{2^n(n-2)!(n-1)!}x^{n-2} + O(x^{n-4})$$

Three-term recurrence formulae:

$$xP_0(x) = P_1(x), \quad (2n+1)xP_n(x) = nP_{n-1}(x) + (n+1)P_{n+1}(x)$$

this can be proved by matching terms: since $r(x) := (2n+1)xP_n(x) - nP_{n-1}(x) - (n+1)P_{n+1}(x)$ must be orthogonal to all polynomials $O(x^{n-2})$, we just have to prove coefficients for x^{n+1}, x^n, x^{n-1} of r are 0.

Name	$w(x)$	Interval	Leading coefficient	Three-term Recurrence	Exact Express/Rodriguez
Chebyshev(first) T_n	$\frac{1}{\sqrt{1-x^2}}$	$[-1,1]$	2^{n-1}	$b_0=1, a_0=0$ for $n \geq 1$, $b_n = c_{n-1} = 1/2$, $a_n = 0$	$\cos(n \arccos(x))$
Chebyshev(second) U_n	$\sqrt{1-x^2}$	$[-1,1]$	2^n	$b_0=1/2, a_0=0$ for $n \geq 1$, $b_n = c_{n-1} = 1/2$, $a_n = 0$	$\frac{\sin(n+1)\theta}{\sin\theta}$
Legendre P_n	1	$[-1,1]$	$\frac{(2n)!}{2^n(n!)^2}$	$b_0=1, a_0=0$ for $n \geq 1$, $b_n = \frac{n+1}{2n+1}, c_{n-1} = \frac{n}{2n+1}, a_n = 0$	$\frac{1}{(-2)^n n!}\left(\frac{d}{dx}\right)^n (1-x^2)^n$
Hermite H_n	e^{-x^2}	$\mathbb{R}$	2^n	$b_n = 1/2, c_{n-1} = n, a_n = 0$	$(-1)^n \exp(x^2) \frac{d^n}{dx^n} \exp(-x^2)$
Laguerre $L_n^{(\alpha)}$ $\alpha > -1$	$e^{-x}x^{\alpha}$	$(0,\infty)$	$\frac{(-1)^n}{n!}$	$b_n = -(n+1), c_{n-1} = -(n+\alpha)$ $a_n = 2n+\alpha+1$	$\frac{1}{n! e^{-x}x^{\alpha}} \frac{d^n}{dx^n}\left(e^{-x}x^{n+\alpha}\right)$
Jacobi $P_n^{(a,b)}$ $a,b > -1$	$(1-x)^a(1+x)^b$	$[-1,1]$	$\frac{(n+a+b+1)_n}{2^n n!}$	very complex	$\frac{\frac{1}{(-2)^n n! w(x)} \times \cdots}{\frac{d^n}{dx^n}\left(w(x)(1-x^2)^n\right)}$
Ultraspherical $C_n^{(\lambda)}$ $\lambda \neq 0, \lambda > -1/2$	$(1-x^2)^{\lambda-1/2}$	$[-1,1]$	$\frac{2^n(\lambda)_n}{n!}$	$b_n = \frac{n+1}{2(n+\lambda)}, c_{n-1} = \frac{n+2\lambda-1}{2(n+\lambda)}$ $a_n = 0$	$\frac{\frac{(2\lambda)_n}{(-2)^n(\lambda+1/2)_n n! w(x)} \times \cdots}{\frac{d^n}{dx^n}\left(w(x)(1-x^2)^n\right)}$

11 Interpolation polynomials

Given distinct points $x_1, ..., x_n \in \mathbb{R}$ and samples $f_1, ..., f_n \in \mathbb{R}$, degree $n-1$ interpolatory polynomial $p(x)$ is a polynomial s.t. $p(x_i) = f_i$.
Assume

$$p(x) = \sum_{k=0}^{n-1} c_k x^k = \begin{pmatrix} 1 & x & \cdots & x^{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}$$

then we know

$$\begin{pmatrix} p(x_1) \\ \vdots \\ p(x_n) \end{pmatrix} = \begin{pmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_{n-1} \end{pmatrix}$$

where the matrix is called Vandermonde matrix V. In fact, V is invertible, so we know interpolatory polynomial exists. And the polynomial is unique. But inverting matrix takes $O(n^3)$ complexity so we need a better algorithm:

Theorem (Lagrange's interpolation)**.** *Let*

$$l_k(x) := \prod_{j=1 \ j\neq k}^{n} \frac{x - x_j}{x_k - x_j}$$

then

$$l_k(x_j) = \begin{cases} 1 & j = k \\ 0 & \textit{otherwise} \end{cases}$$

by uniqueness of interpolatory polynomial, $p(x) = f_1 l_1(x) + \cdots + f_n l_n(x)$*.*

```
# Julia implementation of brute-force algorithm
using Plots, LinearAlgebra
f = x -> cos(10x)
n = 5

x = range(0, 1; length=n)  # evenly spaced points (BAD for interpolation)
V = x .^ (0:n-1)'    # Vandermonde matrix
c = V \ f.(x)   # coefficients of interpolatory polynomial
p = x -> dot(c, x .^ (0:n-1))

g = range(0,1; length=1000) # plotting grid
plot(g, f.(g); label="function")
plot!(g, p.(g); label="interpolation")
scatter!(x, f.(x); label="samples")
```

For some functions, doing polynomial interpolation at evenly spaced point may not give convergence. It turns out that for $n-1$ interpolatory polynomial, choosing $x_1, \cdots x_n$ to be the roots of $q_n(x)$ (where q_n are orthonormal polynomials) is efficient. One can show that there are exactly n distinct roots for p_n, so this choice for interpolation is valid.

Definition 11.1 (Truncated Jacobi Matrix)**.** Taking only the first n rows and columns, we define

$$X_n := \begin{pmatrix} a_0 & b_0 & & & \\ b_0 & \ddots & \ddots & & \\ & \ddots & a_{n-2} & b_{n-2} \\ & & b_{n-2} & a_{n-1} \end{pmatrix}$$

It turns out that zeros of q_n are the Eigenvalues of X_n i.e. $Q_n^T X_n Q_n = diag(x_1, \cdots, x_n)$, where

$$Q_n = \begin{pmatrix} q_0(x_1) & \cdots & q_0(x_n) \\ \vdots & \ddots & \vdots \\ q_{n-1}(x_1) & \cdots & q_{n-1}(x_n) \end{pmatrix} \begin{pmatrix} \alpha_1^{-1} & & \\ & \ddots & \\ & & \alpha_n^{-1} \end{pmatrix}$$

$$\alpha_j := \sqrt{q_0(x_j)^2 + \cdots + q_{n-1}(x_j)^2}$$

α_j are to make sure Q_n is orthogonal, i.e. $Q_n^T Q_n = I$.

You have to normalise OPs before use, e.g. Chebyshev's normalising constants are:

$$q_0(x) = \frac{1}{\sqrt{\pi}}, \quad q_k(x) = T_k(x)\frac{\sqrt{2}}{\sqrt{\pi}}$$

11.1 Interpolation to Quadrature

The target is to approximate $\int_a^b f(x)w(x)\,dx$ by $\sum_{j=1}^n w_j f(x_j)$ where w_j are called quadrature weights and x_j are called quadrature points. The trapezium rule can also be used to estimate this integral, where we fix points x_j lying with equal distance between a, b and choose w_j in require way. But quadrature method allows us to choose x_j freely.

Recall that we have Lagrange interpolation $\sum_{j=1}^n f(x_j) l_j(x)$ for $f(x)$, integrating this yields

$$\sum_{j=1}^{n} \underbrace{\int_a^b w(x) l_j(x)\,dx}_{=:w_j} f(x_j) =: \sum_{n}^{w,\boldsymbol{x}} [f]$$

and if we know $\int_a^b x^k w(x)\,dx$ for k up to $n-1$, we can calculate w_j explicitly(as $l_j(x)$ is a polynomial of degree $n-1$). This interpolation is exact for polynomials of degree less than n. i.e. if p is a polynomial with $\deg p < n$,

$$\sum_n^{w,\boldsymbol{x}}[p] = \int_a^b p(x)w(x)\ dx$$

(To prove it, recall for polynomial, $p(x) = \sum_j p(x_j)l_j(x)$) The quadrature is called (interpolatory) Gaussian quadrature and it can be introduced in another way: given weight function and orthonormal family of polynomials $(q_n)_{n\geq 0}$, define

$$\sum_n^{w}[f] := \sum_{j=1}^{n} w_j f(x_j)$$

where $w_j := \frac{1}{\alpha_j^2} = \frac{1}{q_0(x_j)^2+\cdots+q_{n-1}(x_j)^2}$ and we pick x_j to be roots of $q_n(x)$. Recall the truncated Jacobi matrix X_n and orthogonal matrix Q_n used to diagonalise X_n. So $X_nQ_n = Q_n\begin{pmatrix} x_1 & & \\ & \ddots & \\ & & x_n\end{pmatrix}$. This important relation yields another expression of w_j:

$$w_j = \int_a^b w(x)\ dx Q_n[1,j]^2$$

Proof. Recall definition of Q, $Q_n[1,j]^2 = \frac{q_0(x_j)^2}{\alpha_j^2}$, so we have

$$\int_a^b w(x)\ dx Q_n[1,j]^2 = \frac{1}{\alpha_j^2}\int_a^b w(x)q_0(x_j)^2\ dx$$

note degree of q_0 is 0, so q_0 is constant function i.e. $q_0(x_j) = q_0(x)$ for all x. So RHS of above equation equals

$$\frac{1}{\alpha_j^2}\int_a^b w(x)q_0(x)^2\ dx = \frac{1}{\alpha_j^2}\langle q_0, q_0\rangle = \frac{1}{\alpha_j^2} = w_j$$

□

One important property of Gaussian quadrature is discrete orthogonality

Proposition (Discrete Orthogonality). *For $0 \leq l, m \leq n-1$, we have*

$$\sum_n^{w}[q_lq_m] = \delta_{lm} = \langle q_l, q_m\rangle$$

Proof. Second equality is due to orthogonality of (q_n) family. For first one,

$$\sum_n^{w}[q_lq_m] = \sum_j^n \frac{q_l(x_j)q_m(x_j)}{\alpha_j^2} = \begin{pmatrix}\frac{q_l(x_1)}{\alpha_1} & \cdots & \frac{q_l(x_n)}{\alpha_n}\end{pmatrix}\begin{pmatrix}\frac{q_m(x_1)}{\alpha_1} \\ \cdots \\ \frac{q_m(x_n)}{\alpha_n}\end{pmatrix}$$

the vectors are exactly two rows of Q_n,

$$= e_l^T Q_n (e_m^T Q_n)^T = e_l^T Q_n Q_n^T e_m = (\text{as } Q_n \text{ is orthogonal}) e_l^T e_m = \delta_{lm}$$

$\square$

Before proving two versions of Gaussian quadrature are the same, we have to introduce interpolation function(we will show it interpolates f later) $f_n(x) := \sum_{k=0}^{n-1} c_k^n q_k(x)$ where $c_k^n := \sum_n^w [f q_k]$. This approximation is kind of like finite Fourier series, c_k^n approximates $\langle f, q_k \rangle$ for $k = 0, 1, ..., n-1$. It is worth noting that f_n depends on f, as c_k^n depends on f.

If $f = q_m$, by discrete orthogonality, $f_n = f$. And since $q_0, ..., q_{n-1}$ forms a basis for polynomials with degree $\leq n-1$, if f is any polynomial with $\deg f \leq n-1$, we have $f_n = f$. What about the relationship between f_n, f for general f? The following theorem justifies that f_n interpolates f at chosen nodes $x_1, ..., x_n$:

Theorem. *$f_n(x_j) = f(x_j)$ for $j = 1, ..., n$*

Proof. From the definition of $f_n(x)$, we can build a matrix that sends coefficients c_k^n to values $f_n(x_j)$. Define

$$\tilde{V} := \begin{pmatrix} q_0(x_1) & \cdots & q_{n-1}(x_1) \\ \vdots & & \vdots \\ q_0(x_n) & \cdots & q_{n-1}(x_n) \end{pmatrix}$$

then

$$\tilde{V} \begin{pmatrix} c_0^n \\ \vdots \\ c_{n-1}^n \end{pmatrix} = \begin{pmatrix} f_n(x_1) \\ \vdots \\ f_n(x_n) \end{pmatrix}$$

Now we need to relate the coefficients to $f(x_j)$.

$$\begin{pmatrix} c_0^n \\ \vdots \\ c_{n-1}^n \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^n w_j f(x_j) q_0(x_j) \\ \vdots \\ \sum_{j=1}^n w_j f(x_j) q_{n-1}(x_j) \end{pmatrix}$$

$$= \underbrace{\begin{pmatrix} w_1 q_0(x_1) & \cdots & w_n q_0(x_n) \\ \vdots & & \vdots \\ w_1 q_{n-1}(x_1) & \cdots & w_n q_{n-1}(x_n) \end{pmatrix}}_{=: Q_n^w} \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}$$

Collecting all the information we have,

$$\begin{pmatrix} f_n(x_1) \\ \vdots \\ f_n(x_n) \end{pmatrix} = \tilde{V} \begin{pmatrix} c_0^n \\ \vdots \\ c_{n-1}^n \end{pmatrix} = \tilde{V} Q_n^w \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}$$

left to show $\tilde{V}Q_n^w = I$, this can be proved by writing the matrix $\tilde{V}Q_n^w$ explicitly and use discrete orthogonality. $\square$

Corollary. $\sum_n^w[f] = \sum_n^{w,\boldsymbol{x}}[f]$. *This means two definitions of Gaussian quadrature coincide.*

Proof. We defined RHS as

$$\int_a^b p(x)w(x)\,dx$$

where $p(x) = \sum_{j=1}^n f(x_j)l_j(x)$ is Lagrange interpolation and it interpolates $f(x)$ at $x_1, ..., x_n$. And in the theorem above, we just proved that f_n also interpolates f at these points. So by uniqueness of interpolatory polynomial, we have $p(x) = f_n(x)$. (While reading the following lines, please remember q_0 has degree 0 so it is a constant.)

$$\int_a^b f_n(x)w(x)\,dx = \frac{1}{q_0(x)}\sum_{k=0}^{n-1} c_k^n \int_a^b q_k(x)q_0(x)w(x)\,dx = \frac{1}{q_0}\sum_{k=0}^{n-1} c_k^n \langle q_k, q_0\rangle = \frac{1}{q_0}\sum_{k=0}^{n-1} c_k^n \delta_{k0}$$

$$= \frac{c_0^n}{q_0} = \frac{\sum_n^w[fq_0]}{q_0} = \frac{\sum_{j=1}^n f(x_j)q_0 w_j}{q_0} = \sum_{j=1}^n f(x_j)w_j = \sum_n^w[f]$$

$\square$

Finally we prove the surprising result that exactness of Gaussian quadrature for polynomials extends up to degree $2n-1$. (Quick explanation: The weights w_j and nodes x_j are all free, so we get $2n$ degree of freedom, which corresponds to polynomials of degree $2n-1$)

Proposition. *Given polynomial $p(x)$ with $\deg p = 2n-1$. Then* $\sum_n^w[p] = \int_a^b p(x)w(x)\,dx$

Proof. We use the polynomial division, there exists two polynomials s, r where $\deg r < n$ s.t.

$$p(x) = q_n(x)s(x) + r(x)$$

we know $p(x)$ has degree $2n-1$ and q_n has degree n, so $\deg s = n-1$.

$$\sum_n^w[p] = \sum_n^w[q_n s] + \sum_n^w[r]$$

as $\sum_n^w[\cdot]$ is a summation(linear operator). First term is 0 as $\sum_n^w[q_n s] = \sum w_j q_n(x_j)s(x_j)$ but x_j are roots of q_n by definition. And second term $\sum_n^w[r] = \sum_n^{w,\boldsymbol{x}}[r] = \int_a^b r(x)w(x)\,dx$, since $\deg r < n$(interpolatory quadrature is exact for degree less than n). Now in order to recover $p(x)$ inside the integral, note $\langle q_n, s\rangle = \int_a^b q_n(x)s(x)w(x)\,dx = 0$ as $\deg s < n = \deg q_n$. So

$$\sum_n^w[p] = \int_a^b r(x)w(x)\,dx + 0 = \int_a^b r(x)w(x)\,dx + \int_a^b q_n(x)s(x)w(x)\,dx$$

$$= \int_a^b (q_n(x)s(x) + r(x))w(x)\, dx = \int_a^b p(x)w(x)\, dx$$

□

Remark. You may find the same technique can be used to prove first version of Gaussian quadrature is exact for $n \le \deg p < 2n$. And for $\deg p < n$, simply use that result of interpolatory quadrature and the fact that $\sum_n^w[p] = \sum_n^{w,\boldsymbol{x}}[p]$.

In practice, if you would like to find Gaussian Quadrature interpolation $f_n(x)$ for some function f, use the matrix

$$Q_n^w := \begin{pmatrix} w_1 q_0(x_1) & \cdots & w_n q_0(x_n) \\ \vdots & & \vdots \\ w_1 q_{n-1}(x_1) & \cdots & w_n q_{n-1}(x_n) \end{pmatrix}$$

defined in the proof of theorem above. As we showed that

$$\begin{pmatrix} w_1 q_0(x_1) & \cdots & w_n q_0(x_n) \\ \vdots & & \vdots \\ w_1 q_{n-1}(x_1) & \cdots & w_n q_{n-1}(x_n) \end{pmatrix} \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} = \begin{pmatrix} c_0^n \\ \vdots \\ c_{n-1}^n \end{pmatrix}$$

where c_k^n are the coefficients required to construct interpolation $f_n(x)$.