Honors Numerical Analysis

MATH-UA 396

Yao Xiao

Spring 2023

Contents

1/23 Lecture

Instructor. Mike O'Neil, `oneil@cims.nyu.edu`.

Class time. Mondays & Wednesdays 2:00–3:15pm, 201 WWH.

Office hours. TBA, 1119 WWH.

Programming language. Julia, with `git` version control and Github Classroom.

Grading. Final 40%, Midterm 30%, Homework each 5%.

Course materials.

- Overton, *Numerical Computing with IEEE Floating Point Arithmetic*, SIAM, 2001.
- Driscoll and Braun, *Fundamentals of Numerical Computation*, SIAM, 2017.
- Trefethen and Bau, *Numerical Linear Algebra*, SIAM, 1997.
- Suli and Mayers, *An Introduction to Numerical Analysis*, Cambridge, 2003.

0 Introduction

- **Nonlinear Equations.** Nonlinear equations, nonlinear equations of several variables, and systems of nonlinear equations of several variables.
- **Linear Algebra.** Solving $A\vec{x} = \vec{b}$, eigenproblems $A\vec{v} = \lambda\vec{v}$, decompositions $A = QR = USV^\top$, and optimizations $y = \arg\min f(x)$.
- **Interpolation and Approximation.** Interpolants and approximation.
- **Numerical Integration.** Riemann integrals and quadratures.
- **Fast Algorithms.** Fast Fourier transform $O(n \log n)$ and Monte Carlos.

1/25 Lecture

1 IEEE Arithmetic

1.1 Computer Representation of Numbers

Fixed point and floating point representation. A fixed point representation specifies a fix number of bits for the integer and fraction part. For instance, with specified format "XXXX.XXXX", 7.5 becomes 0007.5000. A floating point representation, on the other hand, is represented $\pm M \times 2^E$, where $M \in [1, 2)$ is called the mantissa and E is called the exponent. For instance, $23 = (10111)_2 = (1.0111)_2 \times 2^4$, $0.125 = (0.001)_2 = (1.0)_2 \times 2^{-3}$, etc. A floating point number is one that can be stored *exactly* on a computer.

Single precision and double precision. A single precision number is 32 bits in total, which include 1 bit for the sign, 8 bits for the exponent, and 23 bits for the mantissa. For instance,

$$5.5 = 1.011 \times 2^2 = (\underbrace{0}_{\text{sign}} | \underbrace{E = 00000010}_{\text{exponent}} | \underbrace{M = 101100\cdots0}_{\text{mantissa, 23 bits}}). \tag{1}$$

On the other hand, a double precision number is 64 bits in total, which include 1 bit for the sign, 11 bits for the exponent, and 52 bits for the mantissa. Single precision and double precision are standardized by IEEE in the 1980s. In Julia, `x = 7.0` stores a `Float64` type variable, and `x = 7` stores a `Int64` type variable (an 8-byte integer).

To represent negative exponent E, the assumption is that E is offset by 1023, that is,

$$x = \pm M \times 2^{E-1023}. \tag{2}$$

In Julia, `bitstring(x)` returns the IEEE representation of `x`.

1.2 Rounding

Rounding. For instance, $(0.1)_{10} = (1.100\overline{1100})_2 \times 2^{-4}$, so the mantissa must be rounded and its approximation is stored as a floating point number. The dafault rounding strategy is rounding to the nearest. The important thing is the error made in rounding.

$$\text{Absolute rounding error} = |\text{round}(x) - x|, \quad \text{Relative rounding error} = \frac{|\text{round}(x) - x|}{|x|}. \tag{3}$$

IEEE standards. IEEE standard says that it must be the case that

$$\text{round}(a+b) = \text{round}(a) \underbrace{\bigoplus}_{\text{IEEE floating point addition}} \text{round}(b) = (a+b)(1+\delta), \tag{4}$$

where $|\delta| < \epsilon = \textsc{MachEp}$, where $\textsc{MachEp}$ denotes the machine precision. Machine precision is the distance between 1 and the next floating point number. In double precision, that is,

$$\epsilon = -(0|011\cdots1|00\cdots0) + (0|011\cdots1|00\cdots01) = 2^{-52}. \tag{5}$$

IEEE standard also says the following about relative accuracy of computations that

$$\frac{|\text{round}(a+b) - (a+b)|}{|a+b|} = |\delta| < \epsilon. \tag{6}$$

1/30 Lecture

2 Solving Nonlinear Equations

We can find solutions to linear equations such as $3x + 7 = 2$, but we cannot generally find solutions to nonlinear equations such as $\cos x + x^2 - 7 = 5$. This is a root finding problem. A *sufficient* condition for a solution $f(x) = 0$ (f is continuous) to exist is stated in the following theorem.

Theorem 2.1. If f is continuous on $[a, b]$, and if $f(a) \cdot f(b) \leq 0$, then there exists $x \in [a, b]$, such that $f(x) = 0$.

Proof. This is immediate from the Intermediate Value Theorem. □

2.1 Bisection

Idea. We will make use of Theorem 2.1. Split the interval in two, test the condition of a sign change and repeat. Let $a_0 = a$, $b_0 = b$, and from the original interval, let $[a_l, b_l]$ be the interval obtained after l splittings. Then,

$$b_l - a_l = \frac{b_0 - a_0}{2^l} = \frac{L}{2^l}. \tag{7}$$

Let $x_l = \frac{a_l + b_l}{2}$ be the approximation of the solution to $f(x) = 0$ in the lth step. The question is, when do we stop the splittings, and how many steps of this algorithm do we take? If we want $|x_l - x^*| < \epsilon$ where x^* is the true solution $f(x^*) = 0$, then we choose l such that

$$|x_l - x^*| \leq \frac{b_l - a_l}{2} = \frac{L}{2^{l+1}} < \epsilon \implies 2^{l+1} > \frac{L}{\epsilon} \implies l > 1 + \log_2\left(\frac{L}{\epsilon}\right). \tag{8}$$

Convergence behavior. If $e_l = |x_l - x^*|$ represents the error in the lth step, then $e_{l+1} = \frac{1}{2}e_l$, meaning that the absolute error goes down by a factor of 2 in each iteration. Therefore, we can see that the bisection method is not efficient enough.

2.2 Secant Method

Idea. Bisection only used the sign information. If we use the values, we can do better. The idea is to approximate by a line with an initial guess, find the root of the line, update the guess and repeat. The secant line is

$$s(x) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) + f(x_0), \tag{9}$$

which has the root

$$x = x_1 - f(x_1) \cdot \frac{x_1 - x_0}{f(x_1) - f(x_0)}. \tag{10}$$

We take this root as x_2 and repeat the process with x_0 substituted by x_1 and x_1 substituted by x_2. Let $f_k = f(x_k)$ for the sake of simplicity. The secant method generates a sequence of approximations to the root of f by

$$x_{k+1} = x_k - f_k \cdot \frac{x_k - x_{k-1}}{f_k - f_{k-1}}. \tag{11}$$

The convergence properties will be revisited later in this course.

2.3 Newton's Method

Idea. In addition to the values of the function, we can make use of both the value and the derivative of a function f at a single point to form an approximation line, find its root, and update and iterate just as in the secant method. The equation of the tangent line is

$$f(x) = f(x_0) + f'(x_0)(x - x_0), \tag{12}$$

which has the root

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}. \tag{13}$$

We take this root as x_1 and repeat the process with x_0 substituted by x_1. The Newton's method generates a sequence of approximations to the root of f by

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{14}$$

Note that (12) is the first-order Taylor approximation of f.

Convergence behavior. Let $e_k = |\xi - x_k|$ be the absolute error, where ξ is the true root of f. Newton's method converges quadratically, meaning that $e_{k+1} = Ae_k^2$ for some constant A. Note that the absolute error is very small, for instance, initially 10^{-1}. Then with only 5 iterations, the error will be near machine precision.

Theorem 2.2 (Theorem 1.8 from Suli and Mayers)**.** Suppose that f is twice continuously differentiable on the interval $I_\delta = [\xi - \delta, \xi + \delta]$, $\delta > 0$, with $f(\xi) = 0$ and $f'(\xi) \neq 0$. Also, assume $A > 0$ such that

$$\left| \frac{f''(x)}{f'(y)} \right| \leq A, \qquad \forall x, y \in I_\delta. \tag{15}$$

If the initial guess satisfies $|\xi - x_0| \leq h$, with $h \leq \min\left(\delta, \frac{1}{A}\right)$, then the sequence $\{x_k\}$ defined by Newton's method (14) converges quadratically to ξ.

2/1 Lecture

Proof. By Taylor's Theorem, we have that

$$f(\xi) = 0 = f(x_0) + f'(x_0)(\xi - x_0) + \frac{f''(\xi_0)}{2}(\xi - x_0)^2, \tag{16}$$

where $\xi_0 \in [\xi, x_0] \subseteq I_\delta$. By Newton's method, we have that

$$\xi - x_1 = \xi - x_0 + \frac{f(x_0)}{f'(x_0)} = -\frac{f''(\xi_0)}{2f'(x_0)}(\xi - x_0)^2. \tag{17}$$

Therefore, we can deduce that

$$|\xi - x_1| \leq \frac{1}{2}\left|\frac{f''(\xi_0)}{f'(x_0)}\right| |\xi - x_0|^2 \leq \frac{1}{2}A\,|\xi - x_0|^2 \leq \frac{1}{2}Ah \cdot \frac{1}{A} = \frac{h}{2}. \tag{18}$$

Repeating this step k times, we can thus show that

$$|\xi - x_k| \leq \frac{h}{2^k} \implies \lim_{k\to\infty} x_k = \xi, \tag{19}$$

proving convergence. Furthermore, since

$$|\xi - x_{k+1}| \leq \frac{1}{2}\left|\frac{f''(\xi_k)}{f'(x_k)}\right| |\xi - x_k|^2, \tag{20}$$

we can see that

$$\lim_{k\to\infty} \frac{|\xi - x_{k+1}|}{|\xi - x_k|^2} = \lim_{k\to\infty}\left|\frac{f''(\xi_k)}{f'(x_k)}\right| = \frac{1}{2}\left|\frac{f''(\xi)}{f'(\xi)}\right| = \texttt{const}. \tag{21}$$

This is exactly the *definition of quadratic convergence.* □

Failures of Newton's method.

- When Newton's method fails to converge (quadratically), it is usually because $f'(\xi) = 0$ at the root. In this case, (15) may not hold, *i.e.*, the quantity may not remain bounded.
- For some initial guesses, Newton's method may fail to converge at all.

Rates of Convergence

For the convergent sequence $e_k = |x_k - x^*|$, we can consider the following limit

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^\alpha} = \mu. \tag{22}$$

For instance, Bisection method has $\alpha = 1$ (*i.e.*, linear convergence), Secant method has approximately $\alpha = 1.62$, and Newton's method has $\alpha = 2$ (*i.e.*, quadratic convergence). For $\alpha = 1$, we further define the *order* of convergence as follows: For $\alpha = 1$ and $\mu < 1$, set $\rho = -\log_{10}\mu$ as the *asymptotic rate of convergence.* As an illustration, if $\mu = 1/10$, this means that

$$\frac{e_{k+1}}{e_k} \approx \frac{1}{10}, \tag{23}$$

i.e., the error goes down by 10% every iteration. That is to say that x_k gets one more correct digit every iteration, and we can see that

$$\rho = -\log_{10}\mu = -\log_{10}\frac{1}{10} = 1. \tag{24}$$

3 Solving Nonlinear Systems

A nonlinear system of equations in n variables $x_1, \cdots, x_n$ is given by

$$\begin{aligned} f_1(x_1, \cdots, x_n) &= 0, \\ &\vdots \\ f_n(x_1, \cdots, x_n) &= 0, \end{aligned} \tag{25}$$

which can be condense into the notation

$$\vec{f}(\vec{x}) = \vec{0}, \tag{26}$$

where

$$\vec{f} = \begin{pmatrix} f_1(\vec{x}) \\ \vdots \\ f_n(\vec{x}) \end{pmatrix}, \qquad \vec{x} = \begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix}. \tag{27}$$

Note that:

- Bisection does not, in general, extend to higher dimensions, since sign changes do not necessarily indicate roots.
- The idea behind Secant method and Newton's method directly extends to higher dimensions: Linearize (approximate by a linear function) and find the root of the approximant.

Vector and Matrix Norms

Definition 3.1. Suppose $\vec{u}, \vec{v}, \vec{x} \in \mathbb{C}^n$, then $\|\cdot\|$ is a *norm* on $\mathbb{C}^n$ if

(1) $\|\vec{x}\| \geq 0$, and $\|\vec{x}\| = 0$ if and only if $\vec{x} = \vec{0}$.

(2) $\|\alpha\vec{x}\| = |\alpha|\|\vec{x}\|$ for $\alpha \in \mathbb{C}$.

(3) $\|\vec{u} + \vec{v}\| \leq \|\vec{u}\| + \|\vec{v}\|$ (triangle inequality).

The most useful norm in Numerical Analysis is the l_2-norm, defined as $\|\vec{u}\|_2 = \sqrt{\sum_i |u_i|^2}$. Other norms commonly used include:

- l_∞-norm: $\|\vec{u}\|_\infty = \max_i |u_i|$.
- l_1-norm: $\|\vec{u}\|_1 = \sum_i |u_i|$.
- l_p-norm: $\|\vec{u}\|_p = (\sum_i |u_i|^p)^{1/p}$.

Definition 3.2. Suppose $A, B \in \mathbb{C}^{m\times n}$, then $\|\cdot\|$ is a *matrix norm* if

(1) $\|A\| \geq 0$, and $\|A\| = 0$ if and only if $A = 0$.

(2) $\|\alpha A\| = |\alpha|\|A\|$ for $\alpha \in \mathbb{C}$.

(3) $\|A + B\| \leq \|A\| + \|B\|$.

If $\|\cdot\|$ is any vector norm, then the *induced matrix norm* is defined as

$$\|A\| = \max_{\|\vec{u}\|=1} \|A\vec{u}\| = \max_{\vec{u}} \frac{\|A\vec{u}\|}{\|\vec{u}\|}. \tag{28}$$

2/6 Lecture

For instance, if $\|\vec{u}\| = \|\vec{u}\|_1 = \sum_{i=1}^n |u_i|$, then the induced matrix norm on $A \in \mathbb{C}^{m\times n}$ is

$$\|A\|_1 = \max_{j=1,\cdots,n} \sum_{i=1}^n |A_{ij}|. \tag{29}$$

To see this, note that

$$\|A\vec{u}\|_1 = \left\|\sum_{j=1}^{n} u_j \vec{A}_j\right\|_1 \leq \sum_{j=1}^{n} |u_j| \|\vec{A}_j\|_1 \leq \left(\max_j \|\vec{A}_j\|_1\right)\left(\sum_{j=1}^{n} |u_j|\right) = \left(\max_j \sum_{i=1}^{n} |A_{ij}|\right)\|\vec{u}\|_1. \tag{30}$$

Therefore, we have that

$$\frac{\|A\vec{u}\|_1}{\|\vec{u}\|_1} \leq \max_j \sum_{i=1}^{n} |A_{ij}|. \tag{31}$$

But taking $\vec{u}$ as a zero vector with only the jth entry as 1, we can see that $\|A\vec{u}\|_1 \geq \|\vec{A}_j\|_1$, so the equality must hold for some $\vec{u}$. Therefore, we can conclude that

$$\|A\|_1 = \max_{\vec{u}} \frac{\|A\vec{u}\|}{\|\vec{u}\|} = \max_{j=1,\cdots,n} \sum_{i=1}^{n} |A_{ij}|. \tag{32}$$

Theorem 3.3. Let $A \in \mathbb{R}^{m\times n}$ be a matrix. Then

$$\|A\|_2 = \sqrt{\max_j \lambda_j}, \tag{33}$$

where λ_j is the jth eigenvalue of $A^\top A$.

Proof. We have that

$$\|A\vec{u}\|^2 = (A\vec{u}, A\vec{u}) = (A\vec{u})^\top A\vec{u} = \vec{u}^\top A^\top A\vec{u}. \tag{34}$$

Note that $A^\top A$ is real and symmetric, it is diagonalizable, so we can write

$$\|A\vec{u}\|^2 = \vec{u}^\top P^\top DP\vec{u}, \tag{35}$$

where D is a diagonal matrix of eigenvalues and P is orthogonal (*i.e.*, $P^\top = P^{-1}$). Therefore, we can compute that

$$\max_{\|\vec{u}\|=1} \|A\vec{u}\|_2^2 = \max_{\|\vec{u}\|=1} \|\vec{u}^\top P^\top DP\vec{u}\|_2^2 = \max_{\|\vec{y}\|=1} \|\vec{y}^\top D\vec{y}\|_2^2 = \max_{\|\vec{y}\|=1} \sum_{i=1}^{n} y_i^2\lambda_i = \max_i \lambda_i, \tag{36}$$

where we can change $\vec{y} = P\vec{u}$ without changing the norm since P is diagonal, and the proof of the last equality is left as an *exercise*. □

Condition number

The *condition number* of the problem is the sensitivity of the "problem" at the solution. For instance, consider a function $y = f(x)$, the how sensitive is y to x?

- In an absolute sense, we can write that

$$|y - y'| = C(x)|x - x'|, \tag{37}$$

  where $C(x)$ is the absolute condition number at x. Note that $C(x) \approx |f'(x)|$ when x' is close to x.

- In a relative sense, we can write that

$$\frac{|y' - y|}{|y|} = K(x)\frac{|x' - x|}{|x|}, \tag{38}$$

  where $K(x)$ is the relative condition number at x. Note that $K(x) \approx \left|\frac{xf'(x)}{f(x)}\right|$ when x' is close to x.

Conditioning for matrix equations. Consider solving $A\vec{x} = \vec{b}$, where the input is $\vec{b}$ and the output is $\vec{x} = A^{-1}\vec{b}$. Let $\|\vec{b}' - \vec{b}\|$ be small, and let $\vec{x}' = A^{-1}\vec{b}'$, $\vec{x} = A^{-1}\vec{b}$. Then we have that

$$\|\vec{x}' - \vec{x}\| = \|A^{-1}\vec{b}' - A^{-1}\vec{b}\| \leq \|A^{-1}\| \cdot \|\vec{b}' - \vec{b}\|, \tag{39}$$

where $\|A^{-1}\|$ denotes the induced matrix norm, and in this case the absolute condition number of A at $\vec{b}$. However, recall that the absolute condition number tells us nothing about the number of correct digits in the answer, so we need the relative condition number, which can be computed as

$$\frac{\|\vec{x}' - \vec{x}\|}{\|\vec{x}\|} \leq \|A^{-1}\| \cdot \frac{\|\vec{b}' - \vec{b}\|}{\|\vec{x}\|} = \|A^{-1}\| \cdot \frac{\|\vec{b}' - \vec{b}\|}{\|\vec{b}\|} \cdot \frac{\|\vec{b}\|}{\|\vec{x}\|} = \|A^{-1}\| \cdot \frac{\|\vec{b}' - \vec{b}\|}{\|\vec{b}\|} \cdot \frac{\|A\vec{x}\|}{\|\vec{x}\|} \leq \|A^{-1}\|\|A\| \cdot \frac{\|\vec{b}' - \vec{b}\|}{\|\vec{b}\|}, \tag{40}$$

so that $\|A^{-1}\|\|A\|$ is the relative condition number of A at $\vec{b}$.

The Singular Value Decomposition

If $A \in \mathbb{R}^{m\times n}$ with $m \geq n$, then we can write

$$A = U\Sigma V^\top, \tag{41}$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are diagonal, and $\Sigma \in \mathbb{R}^{m\times n}$ is a rectangular diagonal matrix.

2/8 Lecture

Recall that the Taylor series of a scalar-valued function of several variables is

$$f = f(x_1, \cdots, x_n) = f(\vec{y}) + \sum_j \frac{\partial f}{\partial x_j}(x_j - y_j) + \textit{higher order terms.} \tag{42}$$

3.1 Multivariable Vector-Valued Taylor Series

Consider the multivariable vector-valued function

$$\vec{f}(\vec{x}) = \begin{pmatrix} f_1(x_1, \cdots, x_m) \\ \vdots \\ f_n(x_1, \cdots, x_m) \end{pmatrix}. \tag{43}$$

Place the expansions back into $\vec{f}$ and collect the terms, we have that

$$\begin{aligned} \vec{f}(\vec{x}) &= \begin{pmatrix} f_1(\vec{y}) + \sum_{k=1}^n \frac{\partial f_1(\vec{y})}{\partial x_k}(x_k - y_k) + \cdots \\ \vdots \\ f_n(\vec{y}) + \sum_{k=1}^n \frac{\partial f_n(\vec{y})}{\partial x_k}(x_k - y_k) + \cdots \end{pmatrix} = \vec{f}(\vec{y}) + \begin{pmatrix} \frac{\partial f_1(\vec{y})}{\partial x_1} & \cdots & \frac{\partial f_1(\vec{y})}{\partial x_m} \\ \vdots & & \vdots \\ \frac{\partial f_n(\vec{y})}{\partial x_1} & \cdots & \frac{\partial f_n(\vec{y})}{\partial x_m} \end{pmatrix} \begin{pmatrix} x_1 - y_1 \\ \vdots \\ x_m - y_m \end{pmatrix} \\ &= \vec{f}(\vec{y}) + J(\vec{y})(\vec{x} - \vec{y}) + (\vec{x} - \vec{y})^\top \cdot \frac{Q(\vec{y})}{2} \cdot (\vec{x} - \vec{y}) + \textit{higher order terms}, \end{aligned} \tag{44}$$

where J is called the *Jacobian matrix*, and Q is a *tensor* defined so that the ith entry of the second-order term $(\vec{x} - \vec{y})^\top \cdot \frac{Q(\vec{y})}{2} \cdot (\vec{x} - \vec{y})$ is given by

$$\frac{1}{2}\sum_{k,l} \frac{\partial^2 f_i(\vec{y})}{\partial x_k \partial x_l}(x_k - y_k)(x_l - y_l) = \frac{1}{2}(\vec{x} - \vec{y})^\top H_i(\vec{y})(\vec{x} - \vec{y}), \tag{45}$$

and H_i is the *Hessian* for f_i.

3.2 Multivariate Newton's Method

We want to solve a square system $\vec{f}(\vec{x}) = \vec{0}$ (*i.e.*, $m = n$). In general, determining whether there is one, none, or multiple solutions is exceedingly difficult. However, if a approximate $\vec{f}$ at $\vec{x}_0$ near the root $\vec{\xi}$, then we have

$$\vec{f}(\vec{x}) \approx \vec{f}(\vec{x}_0) + J(\vec{x}_0)(\vec{x} - \vec{x}_0), \tag{46}$$

also known as the truncated multivariable Taylor series. Evaluting at $\vec{xi}$, we have that

$$\vec{f}(\vec{x}_i) = \vec{0} \approx \vec{f}(\vec{x}_0) + J(\vec{x}_0)(\vec{\xi} - \vec{x}_0) \implies -\vec{f}(\vec{x}_0) \approx J(\vec{x}_0)(\vec{\xi} - \vec{x}_0) \implies \vec{\xi} \approx \vec{x}_0 - J^{-1}(\vec{x}_0)\vec{f}(\vec{x}_0). \tag{47}$$

Therefore, the Newton iteration is

$$\vec{x}_{k+1} = \vec{x}_k - J^{-1}(\vec{x}_k)\vec{f}(\vec{x}_k). \tag{48}$$

It can be shown that convergence is also quadratic, meaning that

$$\|\vec{x}_{k+1} - \vec{\xi}\|_2 \approx \|\vec{x}_k - \vec{\xi}\|_2^2. \tag{49}$$

Much like Newton's method, quadratic convergence may suffer if we have a singular Jacobian $J(\vec{x}_k)$. Another commonly seen form of the multivariate Newton's method is

$$J(\vec{x}_k)(\vec{x}_{k+1} - \vec{x}_k) = f(\vec{x}_k), \tag{50}$$

which can be done by solving $J(\vec{x}_k)\vec{s}_k = f(\vec{x}_k)$ and setting $\vec{x}_{k+1} = \vec{x}_k + \vec{s}_k$ in the iterative step. We will revise this when we talk about optimization.

2/13 Lecture

4 Optimization

4.1 Single Variable Optimization

Recall that root finding solves $f(x) = 0$. If we instead want to maximize or minimize f, we want to look for x such that $f'(x) = 0$. Therefore, we can apply our root finding methods to the equation $f'(x) = 0$ instead. Newton's method then becomes

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}. \tag{51}$$

However, note that for a general continuous function, $f'(x) = 0$ does not necessarily imply a global optimum. It can imply local optimums, or even non-optimal points (for instance, $x = 0$ for $f(x) = x^3$). One (common) regime where we can guarantee a minimum or maximum is when f is convex, defined as follows.

Definition 4.1. f is *convex* on $[a, b]$ if

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y), \tag{52}$$

for all $0 \leq \alpha \leq 1$ and $x, y \in [a, b]$.

Theorem 4.2. If f is convex on $[a, b]$, then any local minimum of f is also a global minimum of f.

Proof. We prove by contradiction. Let x^* be a local minimum of f, and $\tilde{x} \neq x^*$ be such that $f(\tilde{x}) < f(x^*)$. Then, since f is convex, we have that

$$f(\alpha x^* + (1 - \alpha)\tilde{x}) \leq \alpha f(x^*) + (1 - \alpha)f(\tilde{x}) < \alpha f(x^*) + (1 - \alpha)f(x^*) = f(x^*). \tag{53}$$

But since f is convex and x^* is a local minimum of f, there exists some small neighborhood $B_\epsilon(x^*)$ around x^*, such that $f(x) \geq f(x^*)$ for all $x \in B_\epsilon(x^*)$. Taking $\alpha = 1 - \frac{\epsilon}{2|x^* - \tilde{x}|}$ (ϵ can be arbitrarily small so we can guarantee that $0 \leq \alpha \leq 1$), we observe that

$$|\alpha x^* + (1 - \alpha)\tilde{x} - x^*| = (1 - \alpha)|x^* - \tilde{x}| = \frac{\epsilon}{2} < \epsilon, \tag{54}$$

so $\alpha x^* + (1 - \alpha)\tilde{x} \in B_\epsilon(x^*)$. But (53) then leads to a contradiction, so the proof is complete. □

Theorem 4.3. If f is *strictly convex*, then there is only one local minimum.

Proof. Strictly convexity requires the inequality in (52) to be strict. The proof is trivial. □

4.2 Multivariate Optimization

Multivariate Newton's method. Now consider that we want to find

$$\vec{x}^* = \arg\min_{\vec{x}\in\mathscr{X}} f(\vec{x}) = \arg\min_{\vec{x}\in\mathscr{X}} f(x_1, x_2, \cdots, x_n). \tag{55}$$

Assume that f is smooth and strictly convex, then we can (hopefully) solve this using Newton's method on the system $\vec{\nabla} f = \vec{0}$, which is equivalent to $\partial f/\partial x_i = 0$ for all $i = 1, \cdots, n$. Then Newton's method says that

$$\begin{aligned}\vec{x}_{k+1} &= \vec{x}_k - \begin{pmatrix} \frac{\partial}{\partial x_1}\left(\frac{\partial f(\vec{x}_k)}{\partial x_1}\right) & \cdots & \frac{\partial}{\partial x_n}\left(\frac{\partial f(\vec{x}_k)}{\partial x_1}\right) \\ \vdots & & \vdots \\ \frac{\partial}{\partial x_1}\left(\frac{\partial f(\vec{x}_k)}{\partial x_n}\right) & \cdots & \frac{\partial}{\partial x_n}\left(\frac{\partial f(\vec{x}_k)}{\partial x_n}\right) \end{pmatrix}^{-1} \vec{\nabla} f(\vec{x}_k) \\ &= \vec{x}_k - \begin{pmatrix} \frac{\partial^2 f(\vec{x}_k)}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f(\vec{x}_k)}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f(\vec{x}_k)}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(\vec{x}_k)}{\partial x_n \partial x_n} \end{pmatrix}^{-1} \vec{\nabla} f(\vec{x}_k) = \vec{x}_k - H^{-1}(\vec{x}_k)\vec{\nabla} f(\vec{x}_k),\end{aligned} \tag{56}$$

where H denotes the Hessian matrix. There is one computational challenge with this: evaluating and inverting the Hessian matrix are expensive, taking $O(n^2/2)$ and $O(n^3)$ time respectively. To avoid explicit construction of H, we consider *Quasi-Newton methods*, and recall that the secant method is the most basic Quasi-Newton method.

Broyden's Update. The goal is to form an approximation to H on each step of the multivariate Newton's method for optimization, and to be able to cheaply update this approximation. For instance, we want something like

$$\vec{x}_1 = \vec{x}_0 - H^{-1}(\vec{x}_0)\vec{\nabla} f(\vec{x}_0), \quad \vec{x}_2 = \vec{x}_1 - H^{-1}(\vec{x}_1)\vec{\nabla} f(\vec{x}_1), \quad H(\vec{x}_1) = H(\vec{x}_0) + \textit{something}. \tag{57}$$

Now consider the implied linear approximation from the multivariate Newton's method for optimization, that is,

$$\vec{\nabla} f(\vec{x}_{k+1}) \approx \vec{0} \approx \underbrace{\vec{\nabla} f(\vec{x}_k)}_{=:\vec{\nabla} f_k} + \underbrace{H(\vec{x}_k)}_{:=H_k}\underbrace{(\vec{x}_{k+1} - \vec{x}_k)}_{=:\vec{s}_k} \implies H_k \vec{s}_k \approx -\vec{\nabla} f_k. \tag{58}$$

To continue the iteration, we want to update the approximate Hessian matrix to H_{k+1}. Similar to the secant method, it should also be the case that

$$\vec{\nabla} f_{k+1} - \vec{\nabla} f_k \approx H_{k+1}(\vec{x}_{k+1} - \vec{x}_k) \implies H_{k+1}\vec{s}_k \approx \vec{\nabla} f_{k+1} - \vec{\nabla} f_k. \tag{59}$$

Of course this does not uniquely determine H_{k+1} given $\vec{s}_k$, $\vec{\nabla} f_{k+1}$, and $\vec{\nabla} f_k$ are known. We need an extra constraint that $H_{k+1} - H_k$ is of rank 1. This gives that

$$H_{k+1} = H_k + \frac{1}{\vec{s}_k^\top \vec{s}_k}\left(\vec{\nabla} f_{k+1} - \vec{\nabla} f_k - H_k \vec{s}_k\right)\vec{s}_k^\top. \tag{60}$$

Note that the form $\vec{u}\vec{v}^\top$ is the outer product $\vec{u} \otimes \vec{v}$, while the form $\vec{u}^\top \vec{v}$ is the inner product $\vec{u} \cdot \vec{v}$. Therefore, we propose the corresponding algorithm as follows. First, we initialize H_0 (finite differences, diagonal, etc.). Then, we set $\vec{x}_1 = \vec{x}_0 - H_0^{-1}\vec{\nabla} f(\vec{x}_0)$ and update to H_1 based on (60). Iterate this step until the terminating condition is met.

Remark 4.4. Since we are interested only in H^{-1}, there is actually a better way to update H_{k+1}^{-1} given H_k^{-1} without referring to H_{k+1} or H_k. Note that the updated Hessian matrix is in the form

$$H_{k+1} = H_k + \vec{u}\vec{v}^\top. \tag{61}$$

Therefore, we can use the *Sherman-Morrison-Woodbury formula*, which states that

$$(A + \vec{u}\vec{v}^\top)^{-1} = A^{-1} - \frac{A^{-1}\vec{u}\vec{v}^\top A^{-1}}{1 + \vec{v}^\top A^{-1}\vec{u}}. \tag{62}$$

2/15 Lecture

5 Numerical Linear Algebra

In infinite precision-exact arithmetic, solving linear systems comes down to a question of whether the system is consistent or not. On the computer, however, the situation is different due to the round-off error and finite precision calculations. We first make a few remarks on Julia. In Julia, the jth column of a matrix can be accessed by `A[:,j]`. A submatrix can be extracted as `A[i1:i2,j1:j2]`. To generate a matrix, one can do something like `A=rand(5,4)`. To compute the induced p-norm, one can use `opnorm(A,p)`. Note that `p=Inf` is also an option.

5.1 Gaussian Elimination

We all know how Gaussian elimination works. Consider a 2×2 matrix

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}. \tag{63}$$

Then $A\vec{x} = \vec{b}$ can be solved by eliminating a_{21}, such that

$$\left(\begin{array}{cc|c} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \end{array} \right) \sim \left(\begin{array}{cc|c} a_{11} & a_{12} & b_1 \\ 0 & a_{22} - \frac{a_{12}a_{21}}{a_{11}} & b_2 - \frac{a_{12}a_{21}}{a_{11}} \end{array} \right). \tag{64}$$

This implies that

$$x_2 = \frac{b_2 - \frac{a_{12}a_{21}}{a_{11}}}{a_{22} - \frac{a_{12}a_{21}}{a_{11}}}, \qquad x_1 = \frac{b_1 - a_{12}x_2}{a_{11}}. \tag{65}$$

This process is called the *backward substitution*, but it can easily fail. For instance, if one or some of the denominators evaluate to zero, then the process can no longer proceed. A successful algorithm would need to involve pivoting rows, and we will return to this later. One question to consider for this moment is how expensive is Gaussian elimination. To see this, we need to count the number of floating-point operations required to put A in echelon form. Consider the following algorithm:

1: **for** columns $j = 1, \cdots, n-1$ **do**
2: **for** rows $i = j+1, \cdots, n$ **do**
3: Compute `val` $= a_{ij}/a_{jj}$;
4: Compute $\text{row}_i - \texttt{val} \cdot \text{row}_j$;
5: Compute $b_i - \texttt{val} \cdot b_j$;
6: **end for**
7: **end for**

Within each inner loop, the first computation takes 1 operation, the second computation takes $2(n-j)$ computations, and the third operation takes 2 flops. Therefore, the total number of floating-point operations can be computed as

$$\sum_{j=1}^{n-1} \sum_{i=j+1}^{n} (2(n-j)+3) = \sum_{j=1}^{n-1} (n-j)(2(n-j)+3) = \Theta(n^3). \tag{66}$$

5.2 LU Factorization

Another way to think about Gaussian elimination is the LU factorization. Each row operation corresponds to a matrix multiplication by an *elementary* lower triangular matrix on the left. For instance, consider the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}. \tag{67}$$

We can perform some elementary row operations on A such that

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -7 & 0 & 1 \end{pmatrix}}_{L_1} \underbrace{\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}}_{A} = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -21 \end{pmatrix}, \tag{68}$$

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix}}_{L_2} \underbrace{\begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -21 \end{pmatrix}}_{L_1 A} = \underbrace{\begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & -9 \end{pmatrix}}_{U}. \tag{69}$$

Therefore, we can see that $L_2 L_1 A = U$, which implies $A = L_1^{-1} L_2^{-1} U$. Note that $L_1^{-1} L_2^{-1}$ is also lower triangular, then solving $A\vec{x} = \vec{b}$ is equivalent to solving $LU\vec{x} = \vec{b}$, where L is a lower triangular matrix and U is an upper triangular matrix. To do this, we only need to solve $L\vec{y} = \vec{b}$ by forward substitution, and then solve $U\vec{x} = \vec{y}$ using backward substitution. Each of these steps take $O(n^2)$ operations. However, the LU factorization step requires the same number of operations as the Gaussian elimination, which is $O(n^3)$, thus dominating the runtime.

Gaussian elimination and LU factorization failure. We have to notice that Gaussian elimination and LU factorization may fail if there is a zero or a small number in the pivot position. The solution is to interchange some rows, resulting in a factorization of the form $PA = LU$, where P is a permutation (row exchanging) matrix. For instance, let $\epsilon <$ MachEp, then consider

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}. \tag{70}$$

The exact solution to this linear system is

$$x_1 = 1 + \frac{\epsilon}{\epsilon - 1}, \qquad x_2 = 1 - \frac{\epsilon}{\epsilon + 1}, \tag{71}$$

and note that x_1 and x_2 are both approximately 1. Without pivoting, in floating point arithmetic, Gaussian elimination will give us

$$\left(\begin{array}{cc|c} \epsilon & 1 & 1 \\ 1 & 1 & 2 \end{array}\right) \sim \left(\begin{array}{cc|c} \epsilon & 1 & 1 \\ 0 & 1 - 1/\epsilon & 2 - 1/\epsilon \end{array}\right) \approx \left(\begin{array}{cc|c} \epsilon & 1 & 1 \\ 0 & -1/\epsilon & -1/\epsilon \end{array}\right), \tag{72}$$

which would further result in

$$x_2 = 1, \qquad x_1 = \frac{1 - 1}{\epsilon} = 0, \tag{73}$$

throwing the wrong answer.

2/22 Lecture

Before discussing pivoted LU factorization, there is one special case where it can be shown that pivoting is not necessary: $\vec{x}^\top A\vec{x} > 0$, *i.e.*, A is a *symmetric positive definite* matrix. Note that this fact is one of the earliest results in numerical analysis.

5.3 Cholesky Factorization

If A is a symmetric positive definite matrix, then we can write it as $A = U^\top U$, since then $A^\top = U^\top U$ as well. The algorithm is straightforward. Set

$$U = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix}, \tag{74}$$

where $u_{jj} \neq 1$ for all j (in fact, this condition is not necessary). Then, we have that

$$U^\top U = \begin{pmatrix} u_{11} & 0 & \cdots & 0 \\ u_{12} & u_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ u_{1n} & u_{2n} & \cdots & u_{nn} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix} = \begin{pmatrix} u_{11}^2 & u_{11}u_{12} & \cdots & u_{11}u_{1n} \\ u_{12}u_{11} & \sum_{i=1}^2 u_{i2}^2 & \cdots & \sum_{i=1}^2 u_{i2}u_{in} \\ \vdots & \vdots & \ddots & \vdots \\ u_{1n}u_{11} & \sum_{i=1}^2 u_{in}u_{i2} & \cdots & \sum_{i=1}^n u_{in}^2 \end{pmatrix}. \tag{75}$$

Therefore, if we set $A = U^\top U$, we can iteratively compute $u_{11} = \sqrt{a_{11}}$, $u_{12} = a_{12}/u_{11}$, and so on. The total cost would be $\Theta(n^3/3)$ for the Cholesky factorization algorithm.

5.4 Row Pivoting

When pivoting is necessary when doing an LU decomposition, we can effectively write

$$L_m P_{m-1} \cdots L_3 P_2 L_2 P_1 L_1 A = U, \tag{76}$$

where each L_j is a lower triangular matrix and each P_j is a permutation matrix. By the term "permutation matrix", we refer to some matrix P_0 that can for instance interchange two rows, such that

$$P_0 A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} - & \vec{a}_1^\top & - \\ - & \vec{a}_2^\top & - \\ - & \vec{a}_3^\top & - \\ - & \vec{a}_4^\top & - \end{pmatrix} = \begin{pmatrix} - & \vec{a}_1^\top & - \\ - & \vec{a}_2^\top & - \\ - & \vec{a}_4^\top & - \\ - & \vec{a}_3^\top & - \end{pmatrix}. \tag{77}$$

The question is, how to obtain an equivalent form of the factorization $PA = LU$, where P is the permutation matrix resulting from all of the row swaps in the factorization (possibly reordering all rows), L is a lower triangular matrix, and U is an upper triangular matrix. Moreover, note that since each P_j involves only a single row interchange (to be distinguished from P), we have that $P_j^{-1} = P_j = P_j^\top$. As an example, suppose that $L_2 P_1 L_1 A = U$. We then take $L_2 = L_2'$ and $L_1 = P_1 L_1' P_1$. This thus implies that

$$L_2' P_1 P_1 L_1' P_1 A = U \implies L_2' L_1' P_1 A = U \implies P_1 A = (L_2' L_1')^{-1} U. \tag{78}$$

It can be easily shown that $L_1' = P_1 L_1 P_1$ and $L_2' = L_2$ are lower triangular, and therefore in the general case, all permutation matrices can be "moved" to the right by the above argument (details omitted, but make heavy use of the fact that $P_j = P_j^{-1}$ as above).

Gaussian elimination and LU factorization failure (revisited). Recall a previous example, in which we set $\epsilon <$ MachEp and consider

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}. \tag{79}$$

The exact solution to this linear system is

$$x_1 = 1 + \frac{\epsilon}{\epsilon - 1}, \qquad x_2 = 1 - \frac{\epsilon}{\epsilon + 1}, \tag{80}$$

but Gaussian elimination without pivoting would result in

$$x_2 = 1, \qquad x_1 = \frac{1 - 1}{\epsilon} = 0. \tag{81}$$

Now with row pivoting, we swap the first and second rows, so that

$$\left(\begin{array}{cc|c} 1 & 1 & 2 \\ \epsilon & 1 & 1 \end{array}\right) \sim \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1 - \epsilon & 1 - 2\epsilon \end{array}\right) \approx \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1 & 1 \end{array}\right), \tag{82}$$

resulting in the correct answer

$$x_2 = 1, \qquad x_1 = \frac{2 - 1}{1} = 1. \tag{83}$$

5.5 Conditioning and Backward Stability

Recall from before that we computed the (relative) condition number of solving $A\vec{x} = \vec{b}$, which is

$$K(A) = \|A\|\|A^{-1}\|. \tag{84}$$

Theorem 5.1 (Perturbations of $A, \vec{x}$)**.** If $A(\vec{x} + \Delta\vec{x}) = \vec{b} + \Delta\vec{b}$, then

$$\frac{\|\Delta\vec{x}\|}{\|\vec{x}\|} \leq K(A)\frac{\|\Delta\vec{b}\|}{\|\vec{b}\|}. \tag{85}$$

If $(A + \Delta A)(\vec{x} + \Delta\vec{x}) = \vec{b}$, then

$$\frac{\|\Delta\vec{x}\|}{\|\vec{x}\|} \leq K(A)\frac{\|\Delta A\|}{\|A\|}, \qquad \text{as } \|\Delta A\| \to 0. \tag{86}$$

Note that for any induced matrix norm, we have that

$$1 = \|I\| = \|AA^{-1}\| \leq \|A\|\|A^{-1}\| = K(A), \tag{87}$$

so a condition number of 1 is the best we can hope for. In that case, the relative perturbation of the solution has the same size as that of the data. A condition number of size 10^t indicates that in floating point arithmetic, roughly t digits are lost (*i.e.*, become incorrect) in computing the solution $\vec{x}$. Moreover, if $K(A) > \textsc{MachEp}^{-1}$, then for computational purposes the matrix A is singular.

Residual and backward error. Suppose that $\tilde{x}$ is the computed solution to the exact linear system $A\vec{x} = \vec{b}$, then $\tilde{x} = \vec{x}$ is unknown since the exact solution $\vec{x}$ is unknown. Define the *residual* as

$$\vec{r} = \vec{b} - A\tilde{x}, \tag{88}$$

and note that $\tilde{x}$ solves the linear system $A\tilde{x} = \vec{b} - \vec{r}$, a perturbed linear system. The *backward error* is then $\|\vec{r}\|$, which the the perturbation from the original problem to the one that is solved exactly. However, note that small backward error does not necessarily imply small $\|\tilde{x} - \vec{x}\|$. To see this, let $\vec{h} = \tilde{x} - \vec{x}$ and thus $A\vec{h} = -\vec{r}$. By Theorem 5.1, we have that

$$\frac{\|\vec{x} - \tilde{x}\|}{\|\vec{x}\|} \leq K(A)\frac{\|r\|}{\|b\|}. \tag{89}$$

This says that the relative error can be much larger than the relative residual when the condition number $K(A)$ is large. In other words, when solving a linear system, all that can be expected is that the backward error be small, but we cannot guarantee a bound on the error.

5.6 QR Factorization

For the moment, let $A \in \mathbb{R}^{m\times n}$ with $m \geq n$ and $\text{rank}(A) = n$ (*i.e.*, n linearly independent columns). For a general $\vec{b} \in \mathbb{R}^m$, the system $A\vec{x} = \vec{b}$ has no solution if $m > n$. We can, however, ask for the least-squares solution

$$\vec{x}^* = \arg\min_{\vec{x}} \|A\vec{x} - \vec{b}\|_2. \tag{90}$$

In order to achieve this, we require that $\vec{x}^*$ satisfies the *normal equations*

$$A^\top(A\vec{x} - \vec{b}) = \vec{0} \implies A^\top A\vec{x} = A^\top\vec{b}. \tag{91}$$

To see this, let $\vec{y} \in \mathbb{R}^n$ be any vector, then

$$\begin{aligned}\|A(\vec{x} + \vec{y}) - \vec{b}\|_2^2 &= (A\vec{x} + A\vec{y} - \vec{b})^\top(A\vec{x} + A\vec{y} - \vec{b}) = (A\vec{x} - \vec{b})^\top(A\vec{x} - \vec{b}) + 2(A\vec{y})^\top(A\vec{x} - \vec{b}) + (A\vec{y})^\top(A\vec{y}) \\ &= \|A\vec{x} - \vec{b}\|_2^2 + 2y^\top A^\top(A\vec{x} - \vec{b}) + \|A\vec{y}\|_2^2 = \|A\vec{x} - \vec{b}\|_2^2 + \|A\vec{y}\|_2^2 \geq \|A\vec{x} - \vec{b}\|_2^2,\end{aligned} \tag{92}$$

which completes the proof. Furthermore, if we can write $A = QR$ with $Q \in \mathbb{R}^{m\times n}$ orthogonal (*i.e.*, $Q^\top Q = I$) and $R \in \mathbb{R}^{n\times n}$ upper triangular, the normal equations would become

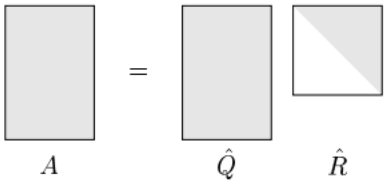
$$R^\top Q^\top QR\vec{x} = R^\top Q^\top \vec{b} \implies R^\top R\vec{x} = R^\top Q^\top \vec{b}. \tag{93}$$

Clearly $R^\top$ is non-singular, so this is equivalent to solving $R\vec{x} = Q^\top\vec{b}$. This is an $n \times n$ square triangular linear system that can be easily solved by backward substitution, and has a solution $\vec{x} = R^{-1}Q^\top\vec{b}$, which is the desired least-squares solution.

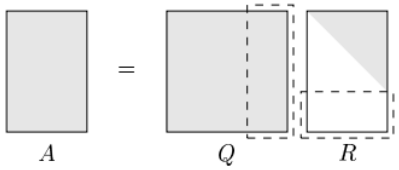
2/27 Lecture

There are in general two scenarios of QR factorization in the case $m \geq n$ and full rank n.

(I) Reduced QR factorization:

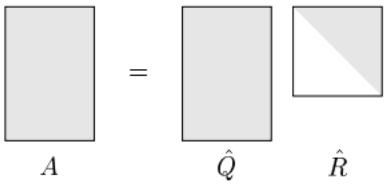


(II) Full QR factorization:



5.7 Gram-Schmidt (Triangular) Orthogonalization

In this section, consider the reduced QR factorization, *i.e.*, factorization of Type (I).



The simple (and naive) way to compute Q is via the *Gram-Schmidt process* which creates an orthonormal basis for the span of the columns of A, *i.e.*, $\vec{a}_1, \cdots, \vec{a}_n \in \mathbb{R}^m$. Write

$$A = \begin{pmatrix} | & | & & | \\ \vec{a}_1 & \vec{a}_2 & \cdots & \vec{a}_n \\ | & | & & | \end{pmatrix}, \qquad Q = \begin{pmatrix} | & | & & | \\ \vec{q}_1 & \vec{q}_2 & \cdots & \vec{q}_n \\ | & | & & | \end{pmatrix} \tag{94}$$

If $A = QR$, then since R is an upper triangular matrix, we have that

$$\begin{cases} \vec{a}_1 = & r_{11}\vec{q}_1, \\ \vec{a}_2 = & r_{12}\vec{q}_1 + r_{22}\vec{q}_2, \\ & \vdots \\ \vec{a}_n = & r_{1n}\vec{q}_1 + \cdots + r_{nn}\vec{q}_n. \end{cases} \tag{95}$$

The Gram-Schmidt process says: turn $\vec{a}_1, \cdots, \vec{a}_n$ into $\vec{q}_1, \cdots, \vec{q}_n$, so that Q can be a matrix with orthonormal columns (*i.e.*, an orthogonal matrix). The algorithm is then described as follows: on the jth step, set

$$\vec{v}_j = \vec{a}_j - (\vec{q}_1^{\top}\vec{a}_j)\vec{q}_1 - \cdots - (\vec{q}_{j-1}^{\top}\vec{a}_j)\vec{q}_{j-1}, \qquad \vec{q}_j = \frac{\vec{v}_j}{\|\vec{v}_j\|}. \tag{96}$$

Comparing with (95), this implies that

$$r_{ij} = \vec{q}_i^{\top}\vec{a}_j, \qquad |r_{jj}| = \left\| \vec{a}_j - \sum_{i=1}^{j-1} r_{ij}\vec{q}_i \right\|_2, \tag{97}$$

since we have that

$$r_{jj}\vec{q}_j = \vec{a}_j - r_{ij}\vec{q}_1 - \cdots - r_{j-1,j}\vec{q}_{j-1}. \tag{98}$$

The pseudocode of this classical Gram-Schmidt orthogonalization algorithm is shown as follows.

for $j = 1$ to n **do**
  $\vec{v}_j \leftarrow \vec{a}_j$;
  for $i = 1$ to $j - 1$ **do**
    $r_{ij} \leftarrow \vec{q}_i^{\top}\vec{a}_j$;
    $\vec{v}_j \leftarrow \vec{v}_j - r_{ij}\vec{q}_i$; `// round-off error can accumulate`
  end for
  $r_{jj} \leftarrow \|\vec{v}_j\|_2$;
  $\vec{q}_j \leftarrow \vec{v}_j / r_{jj}$;
end for

Unfortunately, this algorithm is numerically unstable. The updating step in the first equation of (96) will require an iteration over $i = 1, \cdots, j-1$, potentially causing the the round-off error to accumulate. An alternative way to think about the Gram-Schmidt orthogonalization algorithm is as follows. Define

$$Q_{j-1} = \begin{pmatrix} | & | & & | \\ \vec{q}_1 & \vec{q}_2 & \cdots & \vec{q}_{j-1} \\ | & | & & | \end{pmatrix}, \tag{99}$$

and let P_j be the projector onto the subspace orthogonal to the columns of Q_{j-1}, that is,

$$P_j = I - Q_{j-1}Q_{j-1}^{\top}. \tag{100}$$

The classical Gram-Schmidt orthogonalization algorithm can then be written as

$$\vec{q}_1 = \frac{\vec{a}_1}{\|\vec{a}_1\|} = \frac{P_1\vec{a}_1}{\|P_1\vec{a}_1\|}, \qquad \cdots, \qquad \vec{q}_j = \frac{P_j\vec{a}_j}{\|P_j\vec{a}_j\|}. \tag{101}$$

This modified Gram-Schmidt algorithm is stable, and implicitly writes the projection in a different form

$$P_j = I - Q_{j-1}Q_{j-1}^{\top} = \underbrace{(I - \vec{q}_{j-1}\vec{q}_{j-1}^{\top})}_{P_{\perp\vec{q}_{j-1}}}\underbrace{(I - \vec{q}_{j-2}\vec{q}_{j-2}^{\top})}_{P_{\perp\vec{q}_{j-2}}}\cdots\underbrace{(I - \vec{q}_1\vec{q}_1^{\top})}_{P_{\perp\vec{q}_1}}. \tag{102}$$

This is mathematically equivalent to (96), but numerically different. The pseudocode of this modified Gram-Schmidt orthogonalization algorithm is shown as follows.

for $i = 1$ to n **do**
  $\vec{v}_i \leftarrow \vec{a}_i$; // $\Theta(m)$
end for
for $i = 1$ to n **do**
  $r_{ii} \leftarrow \|\vec{v}_i\|_2$; `// m multiplications + m-1 additions + 1 square root`
  $\vec{q}_i \leftarrow \vec{v}_i / r_{ii}$; `// m divisions`
  for $j = i + 1$ to n **do**
    $r_{ij} \leftarrow \vec{q}_i^{\top}\vec{v}_j$; `// m multiplications + m-1 additions`
    $\vec{v}_j \leftarrow \vec{v}_j - r_{ij}\vec{q}_i$; `// m multiplications + m subtractions`

10: **end for**
11: **end for**

This modified Gram-Schmidt algorithm is the one used in practice, since it is less sensitive to the effects of rounding errors, and thus is a stable algorithm. The number of floating-point operations it need can be computed as

$$\sum_{i=1}^{n}\left(3m+\sum_{j=i+1}^{n}(4m-1)\right)\sim\sum_{i=1}^{n}\sum_{j=i+1}^{n}4m\sim\Theta(2mn^2). \tag{103}$$

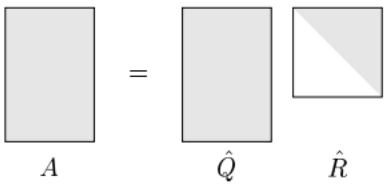
Therefore, this is a cubic algorithm (like Gaussian elimination, basically), and it is suited to reduced QR factorization, *i.e.*, factorization of Type (I).

5.8 Householder Reflection (Orthogonal) Triangularization

As we have seen previously, the Gram-Schmidt iteration applies a succession of elementary triangular matrices R_k on the right of A, so that the resulting matrix

$$A\underbrace{R_1R_2\cdots R_n}_{\hat{R}^{-1}}=\hat{Q} \tag{104}$$

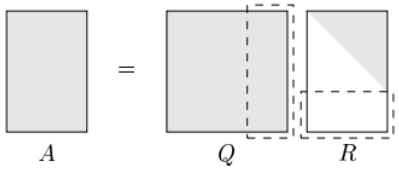
has orthonormal columns, *i.e.*, is an orthogonal matrix. The product $\hat{R}=R_n^{-1}\cdots R_2^{-1}R_1^{-1}$ is an upper triangular matrix, and thus $A=\hat{Q}\hat{R}$ is a reduced QR factorization of A, *i.e.*, a factorization of Type (I).



In contrast, the *Household method* that we are going to introduce applies a succession of elementary matrices Q_k on the left of A, so that the resulting matrix

$$\underbrace{Q_n\cdots Q_2Q_1}_{Q^\top}A=R \tag{105}$$

is an upper triangular matrix. The product $Q=Q_1^\top Q_2^\top\cdots Q_n^\top$ is orthogonal, and thus $A=QR$ would be a full QR factorization of A, *i.e.*, a factorization of Type (II).



As follows is an example of the Household method.

$$\begin{pmatrix}\times&\times&\times\\ \times&\times&\times\\ \times&\times&\times\\ \times&\times&\times\\ \times&\times&\times\end{pmatrix}\xrightarrow{Q_1}\begin{pmatrix}\star&\star&\star\\ 0&\star&\star\\ 0&\star&\star\\ 0&\star&\star\\ 0&\star&\star\end{pmatrix}\xrightarrow{Q_2}\begin{pmatrix}\times&\times&\times\\ &\star&\star\\ &0&\star\\ &0&\star\\ &0&\star\end{pmatrix}\xrightarrow{Q_3}\begin{pmatrix}\times&\times&\times\\ &\times&\times\\ &&\star\\ &&0\\ &&0\end{pmatrix}. \tag{106}$$

The question is, how can we construct orthogonal matrices Q_k, so as to introduce zeros as indicated above? The standard approach is as follows. Each Q_k is chosen to be a matrix of the form

$$Q_k=\begin{pmatrix}I_{k-1}&0\\ 0&F\end{pmatrix}, \tag{107}$$

where I_{k-1} is the $(k-1)\times(k-1)$ identity matrix and F is an $(m-k+1)\times(m-k+1)$ orthogonal matrix. Multiplication by F must introduce zeros in the kth column. The Householder algorithm chooses F to be a particular matrix called a *Household reflector*. Suppose that at the beginning of Step k, the entries $k, \cdots, m$ of the kth column are given by $\vec{x} \in \mathbb{R}^{m-k+1}$. To introduce the correct zeros into the kth column, the Household reflector F should effect the following map, such that

$$\vec{x} = \begin{pmatrix} \times \\ \times \\ \times \\ \vdots \\ \times \end{pmatrix} \quad \xrightarrow{F} \quad F\vec{x} = \begin{pmatrix} \|\vec{x}\| \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \|\vec{x}\|\vec{e}_1. \tag{108}$$

The idea is that, the reflector F should reflect the space $\mathbb{R}^{m-k+1}$ across the hyperplane H orthogonal to $\vec{v} = \|\vec{x}\|\vec{e}_1 - \vec{x}$, as is demonstrated in Figure 1.

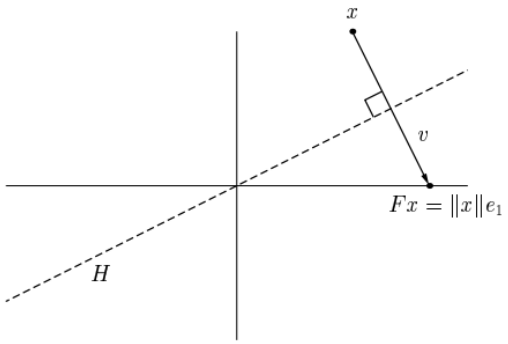


Figure 1: An illustration of a Household reflector.

Recall that for any $\vec{y} \in \mathbb{R}^p$, the *orthogonal projection* of $\vec{y}$ onto a hyperplane H that is orthogonal to some vector $\vec{v} \in \mathbb{R}^p$ can be written as

$$P\vec{y} = \left(I - \frac{\vec{v}\vec{v}^\top}{\vec{v}^\top\vec{v}}\right)\vec{y} = \vec{y} - \vec{v}\left(\frac{\vec{v}^\top\vec{y}}{\vec{v}^\top\vec{v}}\right). \tag{109}$$

Since a reflection would need to go twice the distance as a projection, the reflection would be written as

$$F\vec{y} = \vec{y} - 2\vec{v}\left(\frac{\vec{v}^\top\vec{y}}{\vec{v}^\top\vec{v}}\right) = \left(I - 2\frac{\vec{v}\vec{v}^\top}{\vec{v}^\top\vec{v}}\right)\vec{y}. \tag{110}$$

Hence, back to our case, the Householder reflector F and thus be selected as

$$F = I - 2\frac{\vec{v}\vec{v}^\top}{\vec{v}^\top\vec{v}}. \tag{111}$$

3/1 Lecture

Note that one could also reflect $\vec{x}$ to $-\|\vec{x}\|\vec{e}_1$ instead of reflecting to $\|\vec{x}\|\vec{e}_1$. However, we should always choose the one that moves $\vec{x}$ a longer distance for the reason of numerical stability. This can be simply done. For instance, if $x_1 \geq 0$ we can choose to reflect to $-\|\vec{x}\|\vec{e}_1$, and otherwise we can choose to reflect to $\|\vec{x}\|\vec{e}_1$. Now that we have

$$Q_n \cdots Q_2 Q_1 A = R, \tag{112}$$

then it suffices to construct Q by $Q = Q_1^\top Q_2^\top \cdots Q_n^\top = Q_1 Q_2 \cdots Q_n$ (recall that Q_k are orthogonal, and by construction symmetric). What would the algorithm be exactly, and what is the runtime?

5.9 The Singular Value Decomposition

Let $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ and $\text{rank}(A) = n$. Then A can be factorized as

$$A = USV^\top, \tag{113}$$

where $U \in \mathbb{R}^{m \times n}$ is orthogonal (*i.e.*, with orthonormal columns), $S \in \mathbb{R}^{n \times n}$ is diagonal with entries $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$, and $V^\top \in \mathbb{R}^{n \times n}$ is orthogonal as well. We can think of U and $V^\top$ as rotation and reflection, and S as positive scaling. Mathematically, the singular value decomposition can be computed by observing that $A^\top A$ is positive semi-definite and therefore can be written as

$$A^\top A = VS^2V^\top, \tag{114}$$

according to *eigen-decomposition*. Then, by setting $U = AVS^{-1}$, we will see that $US = AV$, and thus $A = USV^\top$. However, this is numerically more unstable than dealing with A directly since

$$K(A^\top A) = K^2(A). \tag{115}$$

Instead, observe (assuming $m = n$) that the matrix

$$H = \begin{pmatrix} 0 & A^\top \\ A & 0 \end{pmatrix} \in \mathbb{R}^{2m \times 2m} \tag{116}$$

is orthogonal, and that if $A = USV^\top$, we will be able to deduce that

$$H \begin{pmatrix} V & V \\ U & -U \end{pmatrix} = \begin{pmatrix} V & V \\ U & -U \end{pmatrix} \begin{pmatrix} S & 0 \\ 0 & -S \end{pmatrix}. \tag{117}$$

This does not involve matrix squaring, but deals with a $2m \times 2m$ matrix instead of a $m \times n$ one. We will introduce more on how to efficiently compute singular value decompositions later.

Pseudoinverse

Recall that for $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ and $\text{rank}(A) = n$, the solution to the least-squares problem must satisfy the normal equations, that is,

$$A^\top A\vec{x} = A^\top \vec{b} \implies \vec{x} = (A^\top A)^{-1}A^\top \vec{b}. \tag{118}$$

The matrix $(A^\top A)^{-1}A^\top$ is known as the *pseudoinverse* of A, which is commonly denoted by A^+. Substituting A with its singular value decomposition, we have that

$$A^+ = (A^\top A)^{-1}A^\top = (VS^2V^\top)^{-1}(USV^\top)^\top = VS^{-1}S^{-1}V^{-1}VSU^\top = VS^{-1}U^\top. \tag{119}$$

In the case that $m > n$, clearly A is not invertible, but with the pseudoinverse, we have that

$$A^+A = VS^{-1}U^\top USV^\top = VS^{-1}SV^\top = VV^\top = I. \tag{120}$$

Midterm Review

- **Bisection:** Linear convergence, absolute error goes down by half in each iteration.
- **Secant method:** Converges with $\alpha \approx 1.62$, using the update formula

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}. \tag{121}$$

- **Newton's method:** Quadratic convergence, using the update formula

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{122}$$

Note that Newton's method may fail because the first derivative evaluates to zero. It may also fail to converge for certain initial guesses. As for the multivariate version, the update step rule is

$$\vec{x}_{k+1} = \vec{x}_k - J^{-1}(\vec{x}_k)\vec{f}(\vec{x}_k), \tag{123}$$

where the (i,j)th entry of the Jacobian matrix is $J_{ij} = \partial f_i/\partial x_j$ (each row is an f_i, each column is an x_j).

- **Asymptotic rate of convergence:** Sometimes called *order*, defined as

$$\rho = -\log_{10}\mu. \tag{124}$$

Note that this is defined only for $\alpha = 1$, *i.e.*, linear convergence.

- **Induced matrix norms:** The induced l_1 norm of A satisfies

$$\|A\|_1 = \max_{j=1,\cdots,n} \sum_{i=1}^{n} |A_{ij}|. \tag{125}$$

The induced l_2 norm of A satisfies

$$\|A\|_2 = \sqrt{\max_{j=1,\cdots,n} \lambda_j}, \tag{126}$$

where λ_j are the eigenvalues of $A^\top A$. In other words, the induced l_2 norm of A is its largest singular value. Moreover, note that the induced l_2 norm of A^{-1} is the inverse of the smallest singular value of A.

- **Condition number:** The absolute condition number of A is $\|A^{-1}\|$, and relative $\|A^{-1}\| \cdot \|A\|$. Moreover, note that $K(A^\top A) = K^2(A)$.

- **Convexity:** f is convex on $[a,b]$ if for any $x, y \in [a,b]$ and $\alpha \in [0,1]$,

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y). \tag{127}$$

Moreover, f is convex if its Hessian matrix of second-order derivatives is *positive semidefinite*, *i.e.*, symmetric with all eigenvalues real and positive.

- **Optimization:** The update formula is

$$\vec{x}_{k+1} = \vec{x}_k - H^{-1}(\vec{x}_k)\vec{\nabla} f(\vec{x}_k). \tag{128}$$

However, explicit Hessian matrices are expensive to compute, so we use the *quasi-Newton* method, such that

$$H_{k+1} = H_k - \frac{H_k \vec{s}_k \vec{s}_k^\top H_k}{\vec{s}_k^\top H_k \vec{s}_k} + \frac{\vec{y}_k \vec{y}_k^\top}{\vec{y}_k^\top \vec{s}_k}, \qquad \vec{s}_k = \vec{x}_{k+1} - \vec{x}_k, \quad \vec{y}_k = \vec{\nabla} f(\vec{x}_{k+1}) - \vec{\nabla} f(\vec{x}_k). \tag{129}$$

- **Sherman-Morrison-Woodbury:** It states that

$$(A + \vec{u}\vec{v}^\top)^{-1} = A^{-1} - \frac{A^{-1}\vec{u}\vec{v}^\top A^{-1}}{1 + \vec{v}^\top A^{-1}\vec{u}}. \tag{130}$$

With this formula, the update of Hessian matrix inverse can be written as

$$H_{k+1}^{-1} = \left(I - \frac{\vec{s}_k \vec{y}_k^\top}{\vec{y}_k^\top \vec{s}_k}\right) H_k^{-1} \left(I - \frac{\vec{y}_k \vec{s}_k^\top}{\vec{y}_k^\top \vec{s}_k}\right) + \frac{\vec{s}_k \vec{s}_k^\top}{\vec{y}_k^\top \vec{s}_k}. \tag{131}$$

- **Real symmetric matrices:** Such matrices can be diagonalized as $A = P^\top D P$, where D is a diagonal matrix of eigenvalues and P is orthogonal, *i.e.*, with orthonormal columns so that $P^\top = P^{-1}$.

- **LU factorization:** $A = LU$, where L is lower triangular and U is upper triangular. This is done similar to Gaussian elimination, by multiplying elementary row operation matrices on the left of A.

- **Cholesky factorization:** If A is symmetric positive definite, then we can write $A = U^\top U$, where U is upper (lower) triangular.

- **Residual:** The residual $\vec{r} = \vec{b} - A\tilde{x}$, where $\tilde{x}$ is the computed solution of $A\vec{x} = \vec{b}$.

- **QR factorization:** $A \in \mathbb{R}^{m\times n}$ with $m \geq n$ and $\text{rank}(A) = n$. We want $\vec{x}$ that minimizes the l_2 norm of the residual, and this minimizer must satisfy the normal equations $A^\top A\vec{x} = \vec{A}^\top \vec{b}$. We can write $A = QR$, where Q is orthogonal (*i.e.*, $Q^\top Q = I$) and R is upper triangular.

- **Singular value decomposition:** $A = USV^\top$, where $U \in \mathbb{R}^{m\times n}$ is orthogonal, $S \in \mathbb{R}^{n\times n}$ is diagonal with entries as singular values, and $V \in \mathbb{R}^{n\times n}$ also orthogonal.

3/6 Lecture

6 Eigenvalue Problems

Recall that $\lambda, \vec{v}$ is an eigenvalue-eigenvector pair of a matrix A if $A\vec{v} = \lambda\vec{v}$. The direct computation would be via the characteristic equations, that is,

$$p(\lambda) = \det(A - \lambda I) = 0. \tag{132}$$

However, p would be a polynomial of degree n if $A \in \mathbb{R}^{n\times n}$. This is extremely expensive: forming p would cost $n!$ flops, and then a nonlinear root finding algorithm must be used to solve $p(\lambda) = 0$. Before introducing cheaper computation methods, we first see its application in solving systems of linear initial value problems.

6.1 Solving Systems of Linear Initial Value Problems

Consider the initial value problem $\vec{y}' = A\vec{y}$, where we have that

$$\vec{y}(t) = \begin{pmatrix} y_1(t) \\ \vdots \\ y_n(t) \end{pmatrix}, \qquad \vec{y}'(t) = \begin{pmatrix} y_1'(t) \\ \vdots \\ y_n'(t) \end{pmatrix} \tag{133}$$

We will ignore the given initial value $(t_0, \vec{y}(t_0))$ for now. One solution to solve this problem is to diagonalize A, such that $A = PDP^{-1}$ with D being the diagonal matrix of eigenvalues and each column of P being an eigenvector corresponding to the eigenvalue (with the same order). Then, we have that

$$\vec{y}' = PDP^{-1}\vec{y} \implies P^{-1}\vec{y}' = DP^{-1}\vec{y}. \tag{134}$$

By denoting $\vec{u} = P^{-1}\vec{y}$, we have that $\vec{u}' = P^{-1}\vec{y}'$ (since this is taking derivative with respect to t), and thus the above equation can be rewritten as

$$\vec{u}' = D\vec{u} \implies \begin{cases} u_1' = \lambda_1 u_1, \\ \vdots \\ u_n' = \lambda_n u_n. \end{cases} \tag{135}$$

This solves to

$$u_i = c_i e^{\lambda_i t}, \qquad u_i' = \lambda_i c_i e^{\lambda_i t}, \tag{136}$$

where c_i can be easily determined by the given initial values (which we will ignore here). Changing the variables back, we can obtain the solution by $\vec{y} = P\vec{u}$. Therefore, we can see that the problem of solving a system of linear initial value problems can be reduced to finding eigenvalues and eigenvectors of A.

6.2 The Gerschgorin Theorems

Theorem 6.1 (Gerschgorin)**.** Let $n \geq 2$ and $A \in \mathbb{C}^{n\times n}$, then all eigenvalues of the matrix A must lie in the region $D = \bigcup_{i=1}^{n} D_i$, where each D_i is the *Gerschogorin disc* defined as the closed circular region

$$D_i = \{z \in \mathbb{C};\ |z - a_{ii}| \leq R_i\}, \qquad R_i = \sum_{j\neq i} |a_{ij}|. \tag{137}$$

Proof. Suppose that $\lambda \in \mathbb{C}$ and $\vec{v} \in \mathbb{C}^n$ are an eigenvalue and the corresponding nonzero eigenvector of A, so that

$$(A\vec{v})_i = \sum_{j=1}^{n} a_{ij} v_j = \lambda v_i. \tag{138}$$

Suppose that v_k is the component of $\vec{v}$ that is largest (or one of the largest) in modulus. Then we have that

$$|\lambda - a_{kk}||v_k| = |\lambda v_k - a_{kk}v_k| = \left|\sum_{j=1}^{n} a_{kj}v_j - a_{kk}v_k\right| = \left|\sum_{j\neq k} a_{kj}v_j\right| \leq \sum_{j\neq k} |a_{kj}||v_j|. \tag{139}$$

Dividing both sides by $|v_k|$, we thus obtain that

$$|\lambda - a_{kk}| \leq \sum_{j\neq k} |a_{kj}|\frac{|v_j|}{|v_k|} \leq \sum_{j\neq k} |a_{kj}| = R_k. \tag{140}$$

Therefore, $\lambda \in D_k$ assuming v_k is the component of $\vec{v}$ that is largest (or one of the largest) in modulus. Hence, we can guarantee that $\lambda \in \bigcup_{i=1}^{n} D_i = D$, so the proof is complete. □

Theorem 6.2 (Gerschgorin's second theorem)**.** In the above scenario, if m disks are connected, then there are m eigenvalues in that connected region.

We will note prove this theorem now, but revisit it in detail when discussing Jacobi's method later.

6.3 The Power Method

We want to calculate the eigenvalue with the largest magnitude and its associated eigenvector (assuming that the matrix A is diagonalizable). We start with a random vector $\vec{w}$, which must be a linear combination of the eigenvectors of A, such that

$$\vec{w} = \sum_{j=1}^{n} c_j\vec{v}_j. \tag{141}$$

Applying A to $\vec{w}$, we will obtain that

$$A\vec{w} = A\left(\sum_{j=1}^{n} c_j\vec{v}_j\right) = \sum_{j=1}^{n} c_jA\vec{v}_j = \sum_{j=1}^{n} c_j\lambda_j\vec{v}_j. \tag{142}$$

Now, if we apply another A to this resulting vector, we can see that

$$A^2\vec{w} = A\left(\sum_{j=1}^{n} c_j\lambda_j\vec{v}_j\right) = \sum_{j=1}^{n} c_j\lambda_jA\vec{v}_j = \sum_{j=1}^{n} c_j\lambda_j^2\vec{v}_j \tag{143}$$

Therefore, for sufficiently large k, we will have that

$$A^k\vec{w} = \sum_{j=1}^{n} c_j\lambda_j^k\vec{v}_j, \tag{144}$$

which will be dominated by $c_1\lambda_1^k\vec{v}_1$, assuming that $|\lambda_1| > |\lambda_2| > |\lambda_3| > \cdots$ (*i.e.*, λ_1 is the largest eigenvalue of A in modulus). So eventually, if $\vec{y}^{(k)} = A^k\vec{w}$, then $\vec{y}^{(k+1)} = A^{k+1}\vec{w} = A\vec{y}^{(k)} \approx \lambda_1\vec{y}^{(k)}$. Now, normalizing iterants in every step, we will have that

$$\vec{w}_0 = \frac{\vec{w}}{\|\vec{w}\|}, \qquad \vec{w}_1 = \frac{A\vec{w}_0}{\|A\vec{w}_0\|}, \qquad \cdots, \qquad \vec{w}_k = \frac{A\vec{w}_{k-1}}{\|A\vec{w}_{k-1}\|}. \tag{145}$$

Under this normalization, the largest eigenvalue λ_1 is approximately equal to

$$\lambda_1 = \frac{w_{k,i}}{w_{(k-1),i}}, \tag{146}$$

where $w_{k,i}$ represents the ith component of $\vec{w}_k$. Furthermore, we can in fact obtain a better estimation of λ_1 by

$$\lambda_1 = \langle A\vec{w}_k, \vec{w}_k\rangle, \tag{147}$$

since $A\vec{w}_k \approx \lambda_1\vec{w}_k$, which means that $\vec{w}_k^{\top}A\vec{w}_k = \lambda_1\vec{w}_k^{\top}\vec{w}_k = \lambda_1$ given $\vec{w}_k$ is already normalized. Finally, the $\vec{w}_k$'s will approach $\vec{v}_1$ as $k \to \infty$.

Rate of convergence. If k is sufficiently large, then $A^k\vec{w} = c_1\lambda_1^k\vec{v}_1$ (assuming $c_1 > 0$). Then we can compute that

$$\vec{v}_1 \approx \frac{1}{c_1\lambda_1^k}A^k\vec{w} = \frac{1}{c_1\lambda_1^k}\sum_{j=1}^{n} c_j\lambda_j^k\vec{v}_j = \vec{v}_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^k\vec{v_2} + \frac{c_3}{c_1}\left(\frac{\lambda_3}{\lambda_1}\right)^k\vec{v_3} + \cdots + \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^k\vec{v_n}. \tag{148}$$

Since $|\lambda_j| < |\lambda_1|$ for $j \geq 2$, we know that $(\lambda_j/\lambda_1)^k \to 0$ as $k \to \infty$. Since the power method normalizes the iterants $\vec{w}_k$ in every step and we assume that the eigenvectors have unit norm, clearly $\vec{w}_k$ converges to $\vec{v}_1$ as $k \to \infty$. Moreover, we can see that

$$\|\vec{w}_k - \vec{v}_1\| \approx \left|\frac{c_2}{c_1}\right|\left|\frac{\lambda_2}{\lambda_1}\right|^k + \texttt{lower order terms} \sim O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right), \tag{149}$$

so that the rate of convergence of the power method depends on the gap in the eigenvalues, specifically the relative size of λ_2 to λ_1. In other words, if $|\lambda_1| \gg |\lambda_2|$, the power method converges fast but if $|\lambda_1| \approx |\lambda_2|$, the convergence would be very slow. We would need certain techniques to guarantee a sufficiently large gap between the eigenvalues.

The power method with shift. If a matrix A has eigenvalues $\lambda_1, \cdots, \lambda_n$, then the matrix $A - sI$ would have eigenvalues $\lambda_1 - s, \cdots, \lambda_n - s$. The reason is, if $\vec{v}$ is an eigenvector of A corresponding to the eigenvalue λ, then

$$(A - sI)\vec{v} = A\vec{v} - s\vec{v} = \lambda\vec{v} - s\vec{v} = (\lambda - s)\vec{v}. \tag{150}$$

Based on this observation, we can use the power method on $A - sI$ instead of A, so that we can choose s to increase the convergence rate. How to choose s? Under which conditions is the shifted power method useful? Power method with shift allows for computing the most negative or the most positive eigenvalue.

3/20 Lecture

The inverse power method with shift. How do we compute eigenvalues in the middle, *i.e.*, if $\lambda_1 > \lambda_2 > \cdots > \lambda_{n-1} > \lambda_n$, then the power method with shift can compute either λ_1 or λ_n. To compute $\lambda_2, \cdots, \lambda_{n-1}$, we would need a different algorithm. The idea is to apply the power method to find eigenvalues of $(A - sI)^{-1}$, which is called the *inverse power method with shift*. If A has some eigenvalue λ, then A^{-1} has the eigenvalue λ^{-1}. Furthermore, $(A - sI)^{-1}$ should have the eigenvalue $(\lambda - s)^{-1}$. If we choose s properly to make $(\lambda - s)^{-1}$ large, then the inverse power method with shift can converge very rapidly.

Choose s close to λ_l so that $(\lambda_l - s)^{-1}$ becomes very large in absolute value, while $(\lambda_j - s)^{-1}$ for $j \neq l$ remains bounded. This scheme is of course much more expensive since applying A^{-1} requires solving a linear system, which takes $\Theta(n^3)$ flops (versus $\Theta(n^2)$ for the power method). The algorithm can be written as follows.

1: set $\vec{w}_0$ to be random;
2: solve $(A - sI)\vec{y}_1 = \vec{w}_0$, which is equivalent to $\vec{y}_1 = (A - sI)\vec{w}_0$;
3: set $\vec{w}_1 = \vec{y}_1/\|\vec{y}_1\|$;
4: proceed as in the power method;

The hard part is knowing what to choose for s, which would require estimates for the eigenvalues. Moreover, both the power method with shift and the inverse power method with shift compute only one pair of eigenvalue and eigenvector at a time.

6.4 The Jacobi Method

Can we compute all the eigenvalues and their corresponding eigenvectors at the same time? Note that if A were diagonal, then we immediately know its eigenvalues, so the problem is how to make A diagonal. Recall the *similarity transform*, which states that if $B = M^{-1}AM$, then B is similar to A, *i.e.*, they have the same eigenvalues.

Proof. Look at the characteristic polynomial, such that

$$\begin{aligned}\rho_B(\lambda) &= \det(B - \lambda I) = \det(M^{-1}AM - \lambda I) = \det(M^{-1}AM - \lambda M^{-1}M)\\ &= \det(M^{-1}(A - \lambda I)M) = \det(M^{-1})\det(A - \lambda I)\det(M) = \rho_A(\lambda).\end{aligned} \tag{151}$$

Since eigenvalues are just the roots of the characteristic polynomials, the proof is complete. □

Note that if M were chosen to be the matrix of eigenvectors of A, then $A = MDM^{-1}$ with D being the diagonal matrix of eigenvalues. Then, $D = M^{-1}AM$ would be the diagonalization of A.

Example 6.3. Let A be a real symmetric 2×2 matrix, such that

$$A = \begin{pmatrix} a & b \\ b & d \end{pmatrix}. \tag{152}$$

Real symmetric matrices have real eigenvalues, and are in fact *orthogonally diagonalizable*, *i.e.*, can be diagonalized by orthogonal matrices. Note that all 2×2 orthogonal matrices can be parametrized as

$$V = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix}, \tag{153}$$

i.e., 2×2 rotation matrices. We thus want

$$V^\top AV = V^{-1}AV = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}. \tag{154}$$

By expanding the expression, we have that

$$\begin{aligned} &\begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} a & b \\ b & d \end{pmatrix} \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \\ \implies &\begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} a\cos\phi - b\sin\phi & a\sin\phi + b\cos\phi \\ b\cos\phi - d\sin\phi & b\sin\phi + d\cos\phi \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}. \end{aligned} \tag{155}$$

By comparing the diagonal entries, we have that

$$a\sin\phi\cos\phi + b\cos^2\phi - b\sin^2\phi - d\sin\phi\cos\phi = 0, \tag{156}$$

$$a\sin\phi\cos\phi - b\sin^2\phi + b\cos^2\phi - d\sin\phi\cos\phi = 0. \tag{157}$$

The two equations above are exactly the same. We rewrite any one of them to obtain that

$$(a-d)\sin\phi\cos\phi + b(\cos^2\phi - \sin^2\phi) = 0 \implies \frac{a-d}{2}\sin 2\phi + b\cos 2\phi = 0. \tag{158}$$

This implies that, when $d - a \neq 0$, ϕ is given by

$$\tan 2\phi = \frac{2b}{d-a} \implies \phi = \frac{1}{2}\arctan\left(\frac{2b}{d-a}\right). \tag{159}$$

If $d - a = 0$, then clearly $\cos 2\phi = 0$ (as long as A is not diagonal, *i.e.*, $b \neq 0$). This would give $\phi = \pi/4$. In C or Fortran we use `phi = atan(d-a,2b) / 2`. Up till now, we have determined ϕ, such that

$$V^\top AV = D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}. \tag{160}$$

The Jacobi method for an $n \times n$ matrix. Define

$$R^{pq}(\phi) = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \cos\phi & & \sin\phi & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \\ & & & -\sin\phi & & \cos\phi & \\ & & & & & & 1 \\ & & & & & & & 1 \\ & & & & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \leftarrow \texttt{row p} \\ \\ \\ \\ \leftarrow \texttt{row q} \\ \\ \\ \\ \end{matrix} \quad (161)$$

$$\begin{matrix} \uparrow \\ \texttt{col p} \end{matrix} \qquad \begin{matrix} \uparrow \\ \texttt{col q} \end{matrix}$$

$R^{pq}(\phi)$ can be used to set the (p,q)th and (q,p)th entries of a real symmetric matrix A to zero. Moreover, the transformation $R^{pq}(\phi)^\top A R^{pq}(\phi)$ leaves all rows and columns unchanged except for the pth row and column and the qth row and column of A. The algorithm is given as follows.

1: set $A^{(0)} = A$;
2: $k \leftarrow 0$;
3: **repeat**
4: find the (p,q)th element in $A^{(k)}$ with maximum absolute value with $p \neq q$;
5: compute $\phi_k = \arctan\left(2a_{pq}^{(k)}/(a_{qq}^{(k)} - a_{pp}^{(k)})\right)/2$;
6: set $A^{(k+1)} = R^{pq}(\phi_k)^\top A^{(k)} R^{pq}(\phi_k)$;
7: $k \leftarrow k+1$;
8: **until** all entries $|a_{ij}^{(k)}| < \epsilon$, $i \neq j$;

The reason this algorithm works is that

$$A^{(k)} \to \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{pmatrix} \qquad \text{as } k \to \infty. \quad (162)$$

To summarize, we apply a sequence of $R(\phi_k)$'s to A that zero out all off-diagonal elements. Furthermore, this sequence will converge to the matrix of eigenvectors.

Convergence of the Jacobi method. The question now is, do elements that are set to zero stay zero forever in the Jacobi method? The answer is no. The idea behind convergence here is, every Jacobi rotation (*i.e.*, applying $R^\top$ and R) moves the "mass" of the matrix from off-diagonal positions to diagonal positions. We first state a lemma.

Lemma 6.4. If R is an orthogonal transformation and $A^\top = A$, then the Frobenius norms $\|A\|_F = \|R^\top A R\|_F$.

Proof. Recall that the Frobenius norm is defined as

$$\|A\|_F = \left(\sum_{i,j} |a_{ij}|^2\right)^{1/2}. \quad (163)$$

Let $B = R^\top A R$, then A and B have the same eigenvalues, and $B^2 = (R^\top A R)(R^\top A R) = R^\top A^2 R$, implying that A^2 and B^2 also have the same eigenvalues. Therefore, $\mathrm{tr}(A^2) = \mathrm{tr}(B^2)$. However, note that

$$\|A\|_F^2 = \mathrm{tr}(A^\top A) = \mathrm{tr}(A^2) = \mathrm{tr}(B^2) = \|B\|_F^2, \quad (164)$$

so the proof is complete. The fact that $\|A\|_F^2 = \mathrm{tr}(A^\top A)$ is already prove in some previous Homework. □

3/22 Lecture

Now we split the Frobenius norm into a diagonal piece and an off-diagonal piece, such that

$$S(A) =: \|A\|_F^2 = \sum_{i,j} |a_{ij}|^2 = \sum_i |a_{ii}|^2 + \sum_{i \neq j} |a_{ij}|^2 =: D(A) + L(A). \tag{165}$$

Theorem 6.5. Let $A^{(k)}$ be the kth iterant in the Jacobi algorithm, then

$$\lim_{k\to\infty} L(A^{(k)}) = 0, \qquad \lim_{k\to\infty} D(A^{(k)}) = \operatorname{tr}(A^2). \tag{166}$$

Proof. Let a_{pq} be the off-diagonal element of A with the largest absolute value. Let $B = R^{pq}(\phi)^\top A R^{pq}(\phi)$, a single Jacobi rotation. Then we can compute that

$$\underbrace{\begin{pmatrix} b_{pp} & b_{pq} \\ b_{qp} & b_{qq} \end{pmatrix}}_{\tilde{B}} = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix}^\top \underbrace{\begin{pmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{pmatrix}}_{\tilde{A}} \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix}, \tag{167}$$

and do not forget that $b_{pq} = b_{qp} = 0$ by construction (*i.e.*, taking the suitable ϕ value). But from the lemma, we see that $\|B\|_F^2 = \|A\|_F^2$, so that $b_{pp}^2 + b_{qq}^2 = a_{pp}^2 + 2a_{pq}^2 + a_{qq}^2$. Now since $S(A) = S(B)$ as in the previous lemma, we have that $D(A) + L(A) = D(B) + L(B)$. Also note that the diagonal entries of B are the same as those of A, except the ones in rows p and q. Therefore, we can deduce that

$$D(B) - D(A) = (b_{pp}^2 + b_{qq}^2) - (a_{pp}^2 + a_{qq}^2) = 2a_{pq}^2 \implies D(B) = D(A) + 2a_{pq}^2. \tag{168}$$

Consequently, we also have that

$$L(B) = L(A) - 2a_{pq}^2. \tag{169}$$

Now since a_{pq} was the largest off-diagonal element of A, we have that

$$L(A) \leq n(n-1)a_{pq}^2 \implies a_{pq}^2 \geq \frac{L(A)}{n(n-1)}. \tag{170}$$

Therefore, we can see that

$$L(B) = L(A) - 2a_{pq}^2 \geq L(A) - \frac{2L(A)}{n(n-1)} = L(A)\left(1 - \frac{2}{n(n-1)}\right). \tag{171}$$

Now we relabel the matrices such that $A^{(0)} = A$ and $A^{(1)} = B$, then

$$L(A^{(1)}) \leq L(A^{(0)})\left(1 - \frac{2}{n(n-1)}\right), \qquad L(A^{(2)}) \leq L(A^{(0)})\left(1 - \frac{2}{n(n-1)}\right)^2, \qquad \text{etc.} \tag{172}$$

After k iterations, we will thus obtain that

$$L(A^{(k)}) \leq L(A^{(0)})\left(1 - \frac{2}{n(n-1)}\right)^k. \tag{173}$$

This implies that $L(A^{(k)}) \to 0$ as $k \to \infty$. Moreover, since $S(A^{(k)}) = \operatorname{tr}(A^2)$, we can easily see that $D(A^{(k)}) \to \operatorname{tr}(A^2)$ as $k \to \infty$. This completes the proof. □

Applying the Gerschgorin theorem. $A^{(k)}$ and A have the same eigenvalues, and $L(A^{(k)}) \to 0$ as $k \to \infty$. Therefore, the Gerschgorin disks of $A^{(k)}$ have radii going towards 0 as well. Therefore, the eigenvalues of $A^{(k)}$, and thus of A, are the limit of the diagonal of $A^{(k)}$.

Rate of convergence. We have shown that

$$L(A^{(k)}) \leq L(A^{(0)}) \left(1 - \frac{2}{n(n-1)}\right)^k. \tag{174}$$

If $n = 1000$, then $1 - \frac{2}{n(n-1)} \approx 0.99999799799\cdots$. In this case, even if $k = 10000$, $\left(1 - \frac{2}{n(n-1)}\right)^k$ is just approximately 0.98, far from 0. However, real-life convergence is often *much* faster than indicated in the proof. One final remark: the Jacobi method can be terminated when $L(A^{(k)}) < \epsilon$. Then we have that

$$A^{(k)} = \underbrace{R(\phi_k)^\top \cdots R(\phi_1)^\top}_{R^\top} A \underbrace{R(\phi_1) \cdots R(\phi_k)}_{R} \approx \texttt{diag}. \tag{175}$$

Then, $A = RA^{(k)}R^\top$. We say that R diagonalizes A. Here R is the matrix in which each column is the approximate of an eigenvector of A, and $A^{(k)}$ is approximately a diagonal matrix, whose diagonal entries are approximately the eigenvalues of A (in the same order as the eigenvectors).

6.5 The QR Method

For general matrices, the *QR method* can be used to find all eigenvalues. We will examine the algorithm first, and then analyze it. The QR algorithm is described as follows.

1: set $A^{(0)} = A$;
2: $k \leftarrow 1$;
3: **repeat**
4: factorize $A^{(k-1)} = Q^{(k)}R^{(k)}$;
5: compute $A^{(k)} = R^{(k)}Q^{(k)}$;
6: $k \leftarrow k + 1$;
7: **until** the stopping criterion is met;

Under certain assumptions, $A^{(k)}$ would converge to some upper triangular matrix. Furthermore, we have that

$$A^{(k)} = R^{(k)}Q^{(k)} = (Q^{(k)})^{-1}A^{(k-1)}Q^{(k)}, \tag{176}$$

so $A^{(k)}$ and $A^{(k-1)}$ have the same eigenvalues, implying that $A^{(k)}$ and A also have the same eigenvalues via iterations. Three things must be done in order for this to be a usable and accelerated algorithm.

(1) First, reduce A to tridiagonal (*i.e.*, the only nonzero entries must be on the main diagonal, the subdiagonal right below that, and the subdiagonal right above that). This can also be called *Hessenburg* (*i.e.*, both *upper Hessenburg* and *lower Hessenburg*). We do this by using the Householder reflections.

(2) Apply to the shifted matrices $A^{(k)} - \mu^{(k)}I$, with $\mu^{(k)}$ an estimate of some λ.

(3) Use a deflation, *i.e.*, decouple into smaller subproblems.

3/27 Lecture

6.6 Singular Value Decomposition (Revisited)

Assume that A is square, with $m = n$. This is no essential restriction, since we can bidiagonalize rectangular matrices via Householder reflections, and thus reduce to the case of square matrices. Consider the $2m \times 2m$ symmetric matrix

$$H = \begin{pmatrix} 0 & A^\top \\ A & 0 \end{pmatrix}. \tag{177}$$

Recall that the singular value decomposition can be written as $A = USV^\top$, where U and V are orthogonal and S is diagonal. This implies that $AV = US$, and that $U^\top A = SV^\top \implies A^\top U = VS$. Therefore, we can see that

$$\begin{pmatrix} 0 & A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} V & V \\ U & -U \end{pmatrix} = \begin{pmatrix} V & V \\ U & -U \end{pmatrix} \begin{pmatrix} S & 0 \\ 0 & -S \end{pmatrix}, \tag{178}$$

which amounts to an eigendecomposition of H. Thus, we can see that the singular values of A are the absolute values of the eigenvalues of H, and the singular vectors can be extracted from the eigenvectors of H. How?

7 Polynomial Interpolation

Since computers can only perform multiplications and additions, essentially the only type of functions that a computer can evaluate is the family of polynomials. In general, $n+1$ unique points in the xy-plane uniquely defines a polynomial of degree n, since it is equivalent to solving $n+1$ equations for $n+1$ unknowns (the coefficients $a_0, \cdots, a_n$). The point of polynomial interpolation is that, most functions (*e.g.*, triangular functions, solutions to differential equations) are not polynomials, and do not have closed form solutions. However, most of the times, they can be *locally* approximated by polynomials (just think of Taylor's series). With this in mind, given (x_j, y_j)'s, $j = 0, \cdots, n$ with $x_j \in [a, b]$, how do we compute the *interpolant*? Note that polynomial interpolation is at the core of Numerical Analysis.

7.1 Lagrange Interpolation

Given a nonnegative integer n, let P_n denote the family of all polynomials of degree at most n defined over $\mathbb{R}$. We restate the problem as follows. Suppose that x_i, $i = 0, \cdots, n$ are distinct real numbers and y_i, $i = 0, \cdots, n$ are real numbers. We wish to find $p_n \in P_n$, such that $p_n(x_i) = y_i$ for all $i = 0, \cdots, n$. To prove that this problem has a unique solution, we begin with a useful lemma.

Lemma 7.1. Suppose that $n \geq 1$, then there exists polynomials $L_k \in P_n$, $k = 0, \cdots, n$, such that

$$L_k(x_i) = \delta_{ik} = \begin{cases} 1, & \text{if } i = k, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall i, k = 0, \cdots, n. \tag{179}$$

Moreover,

$$p_n(x) = \sum_{k=0}^{n} L_k(x) y_k \tag{180}$$

satisfies the interpolation conditions. In other words, $p_n \in P_n$ and $p_n(x_i) = y_i$, $i = 0, \cdots, n$.

Proof. For each fixed $0 \leq k \leq n$, L_k is required to have n zeros, *i.e.*, all x_i's except x_k. Therefore, we can write it as

$$L_k(x) = C_k \prod_{i=0, i\neq k}^{n} (x - x_i), \tag{181}$$

where $C_k \in \mathbb{R}$ is a constant to be determined. Then by using $L_k(x_k) = 1$, we can deduce that

$$1 = C_k \prod_{i=0, i\neq k}^{n} (x_k - x_i) \implies C_k = \prod_{i=0, i\neq k}^{n} \frac{1}{x_k - x_i}. \tag{182}$$

Therefore, we can conclude that

$$L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}. \tag{183}$$

Since the function p_n is defined as a linear combination of polynomials of degree at most n, it must itself also be a polynomial of degree at most n, thus $p_n \in P_n$. Moreover,

$$p_n(x_i) = \sum_{k=0}^{n} L_k(x_i) y_k = \sum_{k=0}^{n} \delta_{ik} y_k = y_i, \qquad \forall i = 0, \cdots, n, \tag{184}$$

so the proof is complete. □

Theorem 7.2 (Lagrange's interpolation theorem)**.** Assume that $n \geq 0$. Let x_i, $i = 0, \cdots, n$ be distinct real numbers and y_i, $i = 0, \cdots, n$ be real. Then, there exists a unique polynomial $p_n \in P_n$, such that

$$p_n(x_i) = y_i, \qquad \forall i = 0, \cdots, n. \tag{185}$$

Proof. For $n = 0$ the proof is trivial. Therefore we assume that $n \geq 1$. The existence of the required polynomial follows directly from the previous lemma, so it suffices to prove the uniqueness. Assume for contradiction that there exists $q_n \in P_n$ different from p_n as specified in the previous lemma, yet $q_n(x_i) = y_i$ for all $i = 0, \cdots, n$. Then, clearly $p_n - q_n \in P_n$, and $p_n - q_n$ has $n+1$ distinct roots $x_0, \cdots, x_n$. However, a polynomial of degree at most n cannot have more than n roots, unless it is constant zero. Therefore, $p_n \equiv q_n$, leading to a contradiction. Hence, we have shown the uniqueness, and the proof is complete. □

3/29 Lecture

Theorem 7.3. Suppose that $n \geq 0$, and that f is a real-valued continuous function on the closed interval $[a, b]$. Moroever, assume that the $(n+1)$th order derivative of f exists and is also continuous on $[a, b]$, *i.e.*, f is $n+1$ times continuously differentiable. Then, given $x \in [a, b]$, there exists $\xi = \xi(x) \in (a, b)$, such that

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i). \tag{186}$$

Moreover, the error can be bounded such that

$$|f(x) - p_n(x)| \leq \frac{M_{n+1}}{(n+1)!} \prod_{i=0}^{n} |x - x_i|, \tag{187}$$

where $M_{n+1} = \max_{\xi \in [a,b]} |f^{(n+1)}(\xi)|$, the maximum value of the $(n+1)$th derivative of f in the given interval.

Proof. When $x = x_i$ for some i, $i = 0, \cdots, n$, both sides of (186) are zero, and thus the equality is trivially satisfied. Therefore, we assume that $x \in [a, b]$ and $x \neq x_i$, $i = 0, \cdots, n$. For such a value of x, let us consider the auxiliary function $t \mapsto \phi(t)$, defined on $[a, b]$ such that

$$\phi(t) = f(t) - p_n(t) - \frac{f(x) - p_n(x)}{\prod_{i=0}^{n}(x - x_i)} \prod_{i=0}^{n} (t - x_i). \tag{188}$$

Clearly, $\phi(x_i) = 0$, $i = 0, \cdots, n$ and $\phi(x) = 0$. Thus, ϕ vanishes at $n+2$ points which are all distinct in $[a, b]$. Therefore, $\phi'(t)$ would vanish at $n+1$ distinct points in (a, b), one between each pair of consecutive points at which ϕ vanishes[1]. Now we discuss the case $n = 0$ separately. If $n = 0$, there exists $\xi = \xi(x) \in (a, b)$ such that $\phi'(\xi) = 0$. Therefore, we have that

$$0 = \phi'(\xi) = f'(\xi) - p_0'(\xi) - \frac{f(x) - p_0(x)}{\prod_{i=0}^{0}(x - x_i)} = f'(\xi) - \frac{f(x) - p_0(x)}{\prod_{i=0}^{0}(x - x_i)}, \tag{189}$$

since $p_0(x) = f(x_0)$ which is a constant function, thus having zero derivative. Rewriting the equation above, we can obtain the desired result for the case $n = 0$ that

$$f(x) - p_0(x) = f'(\xi) \prod_{i=0}^{0} (x - x_i). \tag{190}$$

For the case $n \geq 1$, we iteratively apply Rolle's theorem to obtain the desired result, similar to above. Now note that f is $n+1$ times continuously differentiable, so $|f^{(n+1)}|$ is bounded on the closed interval $[a, b]$, being able to attain its maximum value in that interval. The bound then follows trivially, and the proof is complete. □

Convergence. An important theoretical question is whether or not a sequence (p_n) of interpolation polynomials for a continuous function f converges to f as $n \to \infty$. This question needs to be made more specific, as p_n depends on the distribution of the interpolation points x_j, $j = 0, \cdots, n$, not just on the value of n. Suppose, for instance, that we agree to choose equally spaced points, such that

$$x_j = a + \frac{j}{n}(b - a), \qquad j = 0, \cdots, n. \tag{191}$$

[1] This is a consequence of Rolle's theorem, which we will not explicitly introduce here.

The question of convergence then clearly depends on the behavior of M_{n+1} as n increases. In particular, if

$$\lim_{n\to\infty} \frac{M_{n+1}}{(n+1)!} \max_{x\in[a,b]} \prod_{i=0}^{n} |x - x_i| = 0, \tag{192}$$

then by the previous theorem, we can guarantee that

$$\lim_{n\to\infty} \max_{x\in[a,b]} |f(x) - p_n(x)| = 0. \tag{193}$$

However, the numerator could have increased faster than the denominator as $n \to \infty$, so the convergence is not guaranteed. For instance, consider Runge's function

$$f(x) = \frac{1}{1+x^2}. \tag{194}$$

This function has singularities (in fact, poles) at $x = \pm i$ in the complex plane. The radius of convergence is only 1. Even if it has the properties of infinite continuous differentiability (*i.e.*, smoothness), it can cause the Runge effect, *i.e.*, when n is large, it oscillates a lot. Moreover, the numerical stability of evaluating p_n in the basic Lagrange form $p_n(x) = \sum_{k=0}^{n} L_k(x) y_k$ can be unstable, *i.e.*, have large condition number. It is very sensitive to overflow/underflow, round-off error, etc.

Cost. The cost of evaluating p_n depends on the form it is written in. Suppose that we are evaluating p_n using the basic Lagrange form, the the number of flops can be counted as

$$p_n(x) = \underbrace{\sum_{k=0}^{n} y_k \underbrace{\underbrace{\prod_{i=0, i\neq k}^{n} \underbrace{\frac{x - x_i}{x_k - x_i}}_{3}}_{3n+(n-1)=4n-1}}_{4n}}_{(n+1)\cdot 4n + n = \Theta(n^2)}. \tag{195}$$

Therefore, we will need overall $\Theta(n^2)$ flops to evaluate p_n in the basic Lagrange form. However, we can set

$$b_0 = a_0 + b_1 x, \tag{196}$$

$$\vdots$$

$$b_{n-2} = a_{n-2} + b_{n-1} x, \tag{197}$$

$$b_{n-1} = a_{n-1} + a_n x. \tag{198}$$

In this way, we have that

$$\begin{aligned} p_n(x) = b_0 &= a_0 + b_1 x = a_0 + (a_1 + b_2 x)x = a_0 + a_1 x + b_2 x^2 = a_0 + a_1 x + (a_2 + b_3 x)x^2 \\ &= a_0 + a_1 x + a_2 x^2 + b_3 x^3 = \cdots = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n. \end{aligned} \tag{199}$$

Evaluating each b_i takes only 2 flops, so that the total number of flops would be $\Theta(2n)$. In this sense, we can see that the Lagrange form is extremely inefficient. Both given the inefficiency and instability of the Lagrange form, we seek for a better form of evaluating p_n.

7.2 Barycentric Forms of Interpolation

The numerical stability of evaluating an interpolating polynomial can be fixed by rearranging its terms, yet not changing what the actual interpolant is. As a motivation, we want to examine the Barycentric coordinates on a triangle. Suppose we have $\triangle ABC$ and a point P inside the triangle. The Barycentric coordinates of P are then given by $P = \alpha A + \beta B + \gamma C$ with $\alpha + \beta + \gamma = 1$, $\alpha, \beta, \gamma \geq 0$. For instance, the center of mass of the triangle is given

by $(\alpha, \beta, \gamma) = (1/3, 1/3, 1/3)$. Now, the idea is to replace A, B, and C with functions that sum to 1. We can rewrite the basic Lagrange form as

$$\begin{aligned} p_n(x) &= \sum_{k=0}^{n} y_k \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i} = \sum_{k=0}^{n} y_k \left(\prod_{j=0}^{n} (x - x_j) \right) \frac{1}{x - x_k} \prod_{i=0, i\neq k}^{n} \frac{1}{x_k - x_i} \\ &= \left(\prod_{j=0}^{n} (x - x_j) \right) \sum_{k=0}^{n} \frac{y_k}{x - x_k} \prod_{i=0, i\neq k}^{n} \frac{1}{x_k - x_i} =: \phi(x) \sum_{k=0}^{n} \frac{y_k}{x - x_k} w_k. \end{aligned} \tag{200}$$

This is called the modified Lagrange form, or the *first Barycentric formula*. We can even simplify this form by "dividing by 1". The polynomial interpolation of the constant function 1 can be simply written as

$$1 = \phi(x) \sum_{k=0}^{n} \frac{1}{x - x_k} w_k. \tag{201}$$

Thus, we can further simplify the modified Lagrange form as

$$p_n(x) = \frac{\phi(x) \sum_{k=0}^{n} \frac{y_k}{x - x_k} w_k}{\phi(x) \sum_{k=0}^{n} \frac{1}{x - x_k} w_k} = \frac{\sum_{k=0}^{n} \frac{w_k}{x - x_k} y_k}{\sum_{k=0}^{n} \frac{w_k}{x - x_k}}. \tag{202}$$

This is known as the *second Barycentric formula*. This form would be stable for any reasonable choices of x_j's, but we will not show it here. One should always use this form to do polynomial interpolation.

8 Function Approximation

Polynomial interpolation mainly has applications in function approximation, with respect to some norm. For functions, some example norms include

$$\|f\|_\infty = \max_{x \in [a,b]} |f(x)|, \qquad (l_\infty \text{ norm})$$

$$\|f\|_2 = \sqrt{\int_a^b |f(x)|^2 dx}, \qquad (l_2 \text{ norm})$$

$$\|f\|_1 = \int_a^b |f(x)| dx. \qquad (l_1 \text{ norm})$$

Norms of functions satisfy the same properties as those in the finite dimensional vector case, such that

- $\|f\| \geq 0$, and $\|f\| = 0$ if and only if $f \equiv 0$.
- $\|cf\| = |c| \, \|f\|$.
- (triangle inequality) $\|f + g\| \leq \|f\| + \|g\|$.

For instance, the l_2 norm of a function can be generalized by introducing a "weight" function $w > 0$, such that

$$\|f\|_{2,w} = \sqrt{\int_a^b |f(x)|^2 w(x) dx}. \tag{203}$$

Now as for function approximation, the problem is to find the polynomial p_n of degree at most n that best approximates a function f in some function norm, in a sense that

$$\min_{p_n \in P_n} \|p_n - f\|. \tag{204}$$

Note that one should never think of p_n here as an interpolation of f. From analysis, we know that continuous functions f on some finite interval can be approximated arbitrarily well by a polynomial of "some" degree, which is known as the *Weierstrass approximation theorem*, *i.e.*, for any $\epsilon > 0$, there exists a polynomial p, such that $\|f - p\| < \epsilon$. Unfortunately, this theorem is helpless for numerical approximation since it does not tell you how to find that p. In the numerical sense, restricting $p \in P_n$ is much more interesting and useful.

8.1 Chebyshev polynomials

Now we stick to the l_∞ function norm. To pose the problem, for $n \geq 1$, we want to find $p_n \in P_n$, such that

$$\|f - p_n\|_\infty = \min_{q \in P_n} \|f - q\|_\infty . \tag{205}$$

In general, however, one cannot write down the minmax polynomial, *i.e.*, the polynomial p_n such that

$$\|f - p_n\|_\infty = \min_{q \in P_n} \max_{x \in [a,b]} |f(x) - q(x)|. \tag{206}$$

However, we can explicitly write down the minmax polynomial approximation to the monomial $f(x) = x^{n+1}$ on $[0, 1]$.

Theorem 8.1. Let $n \geq 0$, then $\|p_n - f\|_\infty$, with $f(x) = x^{n+1}$, is minimized when

$$p_n(x) = x^{n+1} - \frac{1}{2^n} \cos((n+1) \arccos x). \tag{207}$$

Note that p_n above is indeed a polynomial of degree n. Moreover, $T_n(x) = \cos(n \arccos x)$ is known as the *Chebyshev polynomial* of degree n. These functions play a very important role in numerical analysis.

4/3 Lecture

Chebyshev polynomials satisfy the recursive relation

$$T_0(x) = 1, \qquad T_1(x) = x, \qquad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \tag{208}$$

This can be shown using trigonometric identities. Note that usually, we are only concerned with Chebyshev polynomials for $x \in [-1, 1]$. Trivially, the zeros of T_n can be computed as

$$\cos(n \arccos x) = 0 \implies n \arccos x = \frac{\pi}{2}(2m+1) \implies \arccos x = \frac{\pi}{2n}(2m+1) \implies x = \cos\left(\frac{(2m+1)\pi}{2n}\right), \tag{209}$$

for $m = 0, 1, \cdots, n-1$ since the roots repeat for $m \geq n$. The roots on $[-1, 1]$ can thus be ordered as

$$x_j = -\cos\left(\frac{(2j-1)\pi}{2n}\right), \qquad j = 1, \cdots, n. \tag{210}$$

Note that the angles are equispaced. Now with $f(x)$ and $p_n(x)$ specified as in Theorem 8.1, we have that

$$|f(x) - p_n(x)| = 2^{-n}|T_{n+1}(x)| \leq 2^{-n}, \qquad x \in [-1, 1]. \tag{211}$$

8.2 Approximation in the l_2-Norm

Definition 8.2. Let V be a linear space over the field of real numbers. A real-valued function $\langle \cdot, \cdot \rangle$ defined on $V \times V$ is called an *inner product* on V if it satisfies the following axioms.

- $\langle f + g, h \rangle = \langle f, h \rangle + \langle g, h \rangle$ for all $f, g, h \in V$.
- $\langle \lambda f, g \rangle = \lambda \langle f, g \rangle$ for all $\lambda \in \mathbb{R}$ and $f, g \in V$.
- $\langle f, g \rangle = \langle g, f \rangle$ for all $f, g \in V$.
- $\langle f, f \rangle \geq 0$ for all $f \in V$, and $\langle f, f \rangle = 0$ if and only if $f = 0$.

A linear space with an inner product is then called an *inner product space*.

Suppose that V is an inner product space, and $f, g \in V$, then we say that f is *orthogonal to* g if $\langle f, g\rangle = 0$. Moreover, we define the *induced* l_2*-norm* of f such that $\|f\| = \sqrt{\langle f, f\rangle}$. Note that the induced norm (of course) satisfies the triangle inequality, and furthermore it satisfies the Cauchy-Schwarz inequality, such that

$$|\langle f, g\rangle| \leq \|f\| \, \|g\|, \qquad \forall f, g \in V. \tag{212}$$

Now we denote by $L^2_w(a, b)$ the set of all real-valued functions f defined on (a, b) such that $w(x)|f(x)|^2$ is integrable on (a, b). The set $L^2_w(a, b)$ is equipped with the inner product and the induced l_2-norm. Note that when $w(x) \equiv 1$ on (a, b), we simply write $L^2(a, b)$. Now, the problem of best approximation in the l_2-norm can be formulated as follows. Given that $f \in L^2_w(a, b)$, find $p_n \in P_n$, such that

$$\|f - p_n\|_2 = \inf_{q \in P_n} \|f - q\|_2 . \tag{213}$$

As an example, let $n = 0$ and $f(x) = -2x^2$ on $[-1, 1]$, where $w(x) \equiv 1$. We seek to find $p_0(x) = c$ that minimizes $\|f - p_0\|_2$. Note that

$$\|f - p_0\|_2^2 = \int_{-1}^{1} (-2x^2 - c)^2 dx = 2c^2 + \frac{8c}{3} + \frac{8}{5}. \tag{214}$$

Taking its derivative with respect to c, we have that

$$\frac{d}{dc} \|f - p_0\|_2^2 = 4c + \frac{8}{3}. \tag{215}$$

We seek this derivative to be zero so that the objective would be minimized. Therefore, $p_0(x) = c = -2/3$. Note that the polynomial of best approximation from P_n to a function in the l_2-norm can be vastly different from the minimax approximation from P_n to the same function. Now to figure out a general approach, suppose that ϕ_j, $j = 0, \cdots, n$ for a basis of P_n, $n \geq 0$. We seek the polynomial of best approximation as the linear combination

$$p_n(x) = \gamma_0 \phi_0(x) + \cdots + \gamma_n \phi_n(x), \tag{216}$$

where $\gamma_0, \cdots, \gamma_n \in \mathbb{R}$ are to be determined. Setting $w(x) \equiv 1$ for now, we then have that

$$\begin{aligned}
\|f - p_n\|_2^2 &= \int_a^b \left(f(x) - \sum_{j=0}^{n} \gamma_j \phi_j(x) \right)^2 dx = \int_a^b f^2(x) dx - 2 \sum_{j=0}^{n} \gamma_j \int_a^b f(x) \phi_j(x) dx + \int_a^b \left(\sum_{j=0}^{n} \gamma_j \phi_j(x) \right)^2 dx \\
&= \int_a^b f^2(x) dx - 2 \sum_{j=0}^{n} \gamma_j \int_a^b f(x) \phi_j(x) dx + \sum_{j=0}^{n} \sum_{k=0}^{n} \gamma_j \gamma_k \int_a^b \phi_j(x) \phi_k(x) dx \\
&= \langle f, f\rangle - 2 \sum_{j=0}^{n} \gamma_j \langle f, \phi_j\rangle + \sum_{j=0}^{n} \sum_{k=0}^{n} \gamma_j \gamma_k \langle \phi_j, \phi_k\rangle .
\end{aligned} \tag{217}$$

In order to minimize this objective, we require each partial derivative with respect to γ_l to be 0. We have that

$$\frac{\partial}{\partial \gamma_l} \|f - p_n\|_2^2 = -2 \langle f, \phi_l\rangle + 2 \sum_{k=0}^{n} \gamma_k \langle \phi_l, \phi_k\rangle . \tag{218}$$

Setting it to zero for each $l = 0, \cdots, n$ gives that

$$\sum_{k=0}^{n} \gamma_k \langle \phi_l, \phi_k\rangle = \langle f, \phi_l\rangle , \tag{219}$$

and if we write this system of equations in the matrix form, we can see that

$$\begin{pmatrix} \langle \phi_0, \phi_0\rangle & \langle \phi_0, \phi_1\rangle & \cdots & \langle \phi_0, \phi_n\rangle \\ \langle \phi_1, \phi_0\rangle & \langle \phi_1, \phi_1\rangle & \cdots & \langle \phi_1, \phi_n\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \phi_n, \phi_0\rangle & \langle \phi_n, \phi_1\rangle & \cdots & \langle \phi_n, \phi_n\rangle \end{pmatrix} \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \vdots \\ \gamma_n \end{pmatrix} = \begin{pmatrix} \langle f, \phi_0\rangle \\ \langle f, \phi_1\rangle \\ \vdots \\ \langle f, \phi_n\rangle \end{pmatrix} . \tag{220}$$

Therefore, once this linear system is solved, the polynomial of best l_2-norm approximation to f would just be

$$p_n(x) = \gamma_0 \phi_0(x) + \cdots + \gamma_n \phi_n(x), \tag{221}$$

as we have designed. Furthermore, if $\phi_0, \cdots, \phi_n$ form an orthonormal basis, then $\langle \phi_l, \phi_k\rangle = \delta_{lk}$, and thus the left-hand side of the above system becomes $I\vec{\gamma} = \vec{\gamma}$, so taking $\gamma_l = \langle f, \phi_l\rangle$ for each l would suffice.

4/5 Lecture

8.3 Orthogonal Polynomials

From the analysis above, we can see that the approximation of f is equivalent to finding its orthogonal projection onto P_n under the inner product

$$\langle f, g\rangle = \int_a^b f(x)g(x)w(x)dx, \tag{222}$$

where $w(x)$ is the weight function and in the above scenario $w(x) \equiv 1$. Recall that we have defined what orthogonal polynomials are, that is, $\langle f, g\rangle = 0$. As an example of finding a system of orthogonal polynomials, we consider finding ϕ_0, ϕ_1, ϕ_2 on $[-1, 1]$ with weight function $w(x) \equiv 1$. First, set $\phi_0(x) = 1$, and let $\phi_1(x) = x + b$. By $\langle \phi_0, \phi_1\rangle = 0$, we can deduce that

$$\int_{-1}^1 1 \cdot (x + b)dx = 2b = 0 \implies b = 0. \tag{223}$$

Therefore, we set $\phi_1(x) = x$. Now let $\phi_2(x) = x^2 + bx + c$, then two conditions must be satisfied, such that

$$\int_{-1}^1 \phi_0(x)\phi_2(x)dx = \int_{-1}^1 (x^2 + bx + c)dx = \frac{2}{3} + 2c = 0 \implies c = -\frac{1}{3}, \tag{224}$$

$$\int_{-1}^2 \phi_1(x)\phi_2(x)dx = \int_{-1}^1 (x^3 + bx^2 + cx)dx = \frac{2}{3}b = 0 \implies b = 0. \tag{225}$$

Therefore, we set $\phi_2(x) = x^2 - 1/3$. Now by construction, 1, x, and $x^2 - 1/3$ are orthogonal on $[-1, 1]$. We could do the same steps for $\phi_3(x), \phi_4(x), \cdots$. The resulting polynomials are known as *Legendre polynomials*. They form an orthogonal basis for all $L^2(-1, 1)$ under the inner product

$$\langle f, g\rangle = \int_{-1}^1 f(x)g(x)dx. \tag{226}$$

Recall that we say $f \in L^2(-1, 1)$ if and only if $\int_{-1}^1 |f(x)|^2 dx < \infty$, *i.e.*, $\|f\|_2^2 < \infty$. Legendre polynomials can also be constructed in another way, *i.e.*, via the Gram-Schmidt process. We start with $\phi_0(x) = 1$ and $\phi_1(x) = x$, which are automatically orthogonal. Then we set $m_2(x) = x^2$, which is linearly independent from ϕ_0 and ϕ_1. Now by Gram-Schmidt process, we take

$$\phi_2(x) = m_2(x) - \frac{\langle m_2, \phi_0\rangle}{\langle \phi_0, \phi_0\rangle}\phi_0(x) - \frac{\langle m_2, \phi_1\rangle}{\langle \phi_1, \phi_1\rangle}\phi_1(x) = x^2 - \frac{1}{3}, \tag{227}$$

where the result is exactly the same as before. So in general, we compute

$$\phi_n(x) = m_n(x) - \sum_{l=0}^{n-1} \frac{\langle m_n, \phi_l\rangle}{\langle \phi_l, \phi_l\rangle}\phi_l(x) = x^n - \sum_{l=0}^{n-1} \frac{\phi_l(x)}{\|\phi_l\|_2^2} \int_{-1}^1 x^n \phi_l(x)dx. \tag{228}$$

These polynomials can be scaled to any interval, and moreover, we can easily add weight function (as long as ϕ_0 and ϕ_1 are chosen properly). Now, we examine the Chebyshev polynomials. We know that for $m \neq n$,

$$\int_0^\pi \cos(mt)\cos(nt)dt = 0, \tag{229}$$

and by substituting $t = a\cos x$, we have that $dt = dx/\sqrt{1 - x^2}$, which means

$$\int_{-1}^1 \cos(m \arccos x)\cos(n \arccos x)\frac{dx}{\sqrt{1 - x^2}} = 0 \implies \int_{-1}^1 T_m(x)T_n(x)\frac{1}{\sqrt{1 - x^2}}dx = 0, \qquad m \neq n. \tag{230}$$

Therefore, we can conclude that the Chebyshev polynomials $T_0, T_1, T_2, \cdots$ are orthogonal on $[-1, 1]$ with respect to the weight function $w(x) = 1/\sqrt{1 - x^2}$.

Theorem 8.3. If $f \in L^2_w(a,b)$, there is a unique polynomial p_n of degree n, such that

$$\|f - p_n\|_{2,w} = \min_{q \in P_n} \|f - q\|_{2,w}\,, \tag{231}$$

where we recall that

$$\|f\|^2_{2,w} = \int_a^b |f(x)|^2 w(x) dx. \tag{232}$$

Proof. The Gram-Schmidt process solve directly for the coefficients of the associated orthogonal polynomial expansion to compute the approximation. Therefore, existence and uniqueness both follows from the Gram-Schmidt process. □

There are some famous sets of orthogonal polynomials. For instance, Legendre polynomials on $(-1,1)$ with $w(x) \equiv 1$, Chebyshev polynomials on $(-1,1)$ with $w(x) = 1/\sqrt{1-x^2}$, Laguerre polynomials on $(0,\infty)$ with $w(x) = e^{-x}$, and Hermite polynomials on $(-\infty,\infty)$ with $w(x) = e^{-x^2}$. Now, let us take an explicit example to elaborate on how to compute best l_2-norm polynomial approximation. Consider $f(x) = \sin x$ on $(0,\pi)$ with weight function $w(x) \equiv 1$. Recall that the first three Legendre polynomials on $(-1,1)$ are

$$\phi_0(x) = 1, \qquad \phi_1(x) = x, \qquad \phi_2(x) = x^2 - \frac{1}{3}. \tag{233}$$

To define these functions on $(0,\pi)$, we substitute $t = \frac{x+1}{2}\pi$, which means $x = 2t/\pi - 1$. The shifted orthogonal polynomials would then be

$$\tilde{\phi}_0(t) = 1, \qquad \tilde{\phi}_1(t) = \frac{2}{\pi}\left(t - \frac{\pi}{2}\right), \qquad \tilde{\phi}_2(t) = \frac{4}{\pi^2}\left(t^2 - \pi t + \frac{\pi^2}{6}\right). \tag{234}$$

The best approximation is given by the projection of f onto the linear space spanned by $\tilde{\phi}_0, \tilde{\phi}_1, \tilde{\phi}_2$, such that

$$p_2(t) = \frac{\langle f, \tilde{\phi}_0\rangle}{\langle \tilde{\phi}_0, \tilde{\phi}_0\rangle}\tilde{\phi}_0(t) + \frac{\langle f, \tilde{\phi}_1\rangle}{\langle \tilde{\phi}_1, \tilde{\phi}_1\rangle}\tilde{\phi}_1(t) + \frac{\langle f, \tilde{\phi}_2\rangle}{\langle \tilde{\phi}_2, \tilde{\phi}_2\rangle}\tilde{\phi}_2(t). \tag{235}$$

9 Numerical Integration

Almost no integral can be computed analytically, they must be evaluated numerically. For instance, we have the Gaussian *error function*

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2} dt. \tag{236}$$

Moreover, numerical integration has wide applications in ODEs for initial value problems, for instance, $y'(t) = f(t)$, $y(0) = y_0$. The analytical solution would be

$$y(t) = y_0 + \int_0^t f(\tau) d\tau, \tag{237}$$

but the integral may need to be evaluated numerically. In fact, all numerical methods for solving such initial value problems are based on numerically approximating the integral.

4/10 Lecture

9.1 Newton-Cotes Formulae

Let f be a real-valued function, defined and continuous on the closed real interval $[a,b]$, and suppose we have to evaluate the integral

$$\int_a^b f(x) dx. \tag{238}$$

Since polynomials are easy to integrate, the idea, roughly speaking, is to approximate the function f by its Lagrange interpolation polynomial p_n of degree n, and integrate p_n instead. Therefore, we have that

$$\int_a^b f(x)dx \approx \int_a^b p_n(x)dx. \tag{239}$$

This can be done with arbitrarily high degree interpolation, but remember that the interpolating polynomial may suffer from *Runge's phenomenon*. Now to be more specific, let x_i, $i = 0, \cdots, n$ denote the interpolation points. For the sake of simplicity, we shall assume that these are equispaced, *i.e.*, $x_i = a + ih$ where $h = (b-a)/n$. The Lagrange interpolation polynomial of degree n for the function f, with these interpolation points, is of the form

$$p_n(x) = \sum_{k=0}^{n} L_k(x) f(x_k), \tag{240}$$

where

$$L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}. \tag{241}$$

Inserting this expression of p_n into our approximate integration, we have that

$$\int_a^b f(x)dx \approx \int_a^b \sum_{k=0}^{n} L_k(x) f(x_k) dx = \sum_{k=0}^{n} \underbrace{\left(\int_a^b L_k(x)dx \right)}_{=:w_k} f(x_k) = \sum_{k=0}^{n} w_k f(x_k). \tag{242}$$

The values w_k, $k = 0, \cdots, n$ are referred to as the *quadrature weights*, while the interpolation points x_k, $k = 0, \cdots, n$ are called the *quadrature nodes*. The numerical quadrature rule as stated above, with equispaced quadrature points, is called the *Newton-Cotes formula* of order n. Here are a few remarks. Firstly, the quadrature weights w_k depends only on the locations of the quadrature nodes, but not on the function itself. Secondly, using large n may result in inaccurate approximation by the interpolating polynomial due to the Runge phenomenon. The remedy would be to use smaller n (*i.e.*, lower order interpolations), but repeat multiple times and glue the results together.

Now, based on the Newton-Cotes formula, we introduce the following simple rule called the *Trapezium rule*. Take $n = 1$, so that $x_0 = a$ and $x_1 = b$. Then we can compute that

$$p_1(x) = L_0(x)f(a) + L_1(x)f(b) = \frac{x-b}{a-b} f(a) + \frac{x-a}{b-a} f(b) = \frac{(b-x)f(a) + (x-a)f(b)}{b-a}. \tag{243}$$

Integrating $p_1(x)$ from a to b then yields

$$\int_a^b f(x)dx \approx \int_a^b p_1(x)dx = \frac{(b-a)(f(a)+f(b))}{2}. \tag{244}$$

The reason why this is called the Trapezium rule is that, the expression is the area of the trapezium (tī xíng) with vertices $(a, 0), (b, 0), (a, f(a)), (b, f(b))$.

9.2 Error Estimates

Our next task is to estimate the size of error in the numerical integration formula, Newton-Cotes formula. The error is defined by

$$E_n(f) = \int_a^b f(x)dx - \sum_{k=0}^{n} w_k f(x_k). \tag{245}$$

Theorem 9.1. Let $n \geq 1$. Suppose that f is a real-valued function, defined and $n+1$ times continuously differentiable on the interval $[a, b]$, *i.e.*, $f^{(n+1)}$ exists and is continuous on $[a, b]$. Then,

$$|E_n(f)| \leq \frac{M_{n+1}}{(n+1)!} \int_a^b |\pi_{n+1}(x)|dx, \tag{246}$$

where

$$M_{n+1} = \max_{\xi \in [a,b]} |f^{(n+1)}(\xi)|, \qquad \pi_{n+1}(x) = \prod_{i=0}^{n} (x - x_i). \tag{247}$$

Proof. Recalling the the definition of the quadrature weights w_k, we can write the error as

$$E_n(f) = \int_a^b f(x)dx - \int_a^b \left(\sum_{k=0}^{n} L_k(x) f(x_k) \right) dx = \int_a^b (f(x) - p_n(x))dx. \tag{248}$$

Therefore, by the bound on the interpolation error (187), we can obtain that

$$|E_n(f)| \leq \int_a^b |f(x) - p_n(x)|dx \leq \frac{M_{n+1}}{(n+1)!} \int_a^b |\pi_{n+1}(x)|dx, \tag{249}$$

as desired, so the proof is complete. □

Applying this theorem to the Trapezium rule, *i.e.*, when $n = 1$, we can see that

$$|E_1(f)| \leq \frac{M_2}{2} \int_a^b |(x-a)(x-b)|dx = \frac{M_2}{2} \int_a^b (x-a)(b-x)dx = \frac{(b-a)^3}{12} M_2, \tag{250}$$

where $M_2 = \max_{\xi \in [a,b]} |f''(\xi)|$.

9.3 Composite Formulae

We shall consider only some very simple quadrature rules, for instance the composite Trapezium rule. Higher orders include the composite Simpsons rule, which can be derived from Simpsons rule by taking $n = 2$ in the Newton-Cotes formula, but we will not discuss here. Now, suppose that f is a function, defined and continuous on $[a, b]$. In order to construct an approximation to its integral over $[a, b]$, we divide the interval into m equal subintervals, each of width $h = (b - a)/m$, so that

$$\int_a^b f(x)dx = \sum_{i=1}^{m} \int_{x_{i-1}}^{x_i} f(x)dx, \tag{251}$$

where $x_i = a + ih = a + i(b - a)/m$, $i = 0, \cdots, m$. Each of these integrals is then evaluated by the Trapezium rule, such that

$$\int_a^b f(x)dx = \sum_{i=1}^{m} \int_{x_{i-1}}^{x_i} f(x)dx \approx \sum_{i=1}^{m} \frac{h(f(x_{i-1}) + f(x_i))}{2} = h \left(\frac{f(x_0)}{2} + \sum_{i=1}^{m-1} f(x_i) + \frac{f(x_m)}{2} \right). \tag{252}$$

This is called the *composite Trapezium rule*. As for the error, we can estimate it using the error bound for the normal Trapezium rule on each subinterval. To this end, we define

$$\mathscr{E}_1(f) = \sum_{i=1}^{m} \left(\int_{x_{i-1}}^{x_i} f(x)dx - \frac{h(f(x_{i-1}) + f(x_i))}{2} \right). \tag{253}$$

Then, we can apply (250) to obtain that

$$|\mathscr{E}_1(f)| \leq \sum_{i=1}^{m} \frac{(x_i - x_{i-1})^3}{12} \left(\max_{\xi \in [x_{i-1}, x_i]} |f''(\xi)| \right) \leq \frac{((b-a)/m)^3}{12} \sum_{i=1}^{m} \max_{\xi \in [a,b]} |f''(\xi)| = \frac{(b-a)^2}{12m^2} M_2, \tag{254}$$

where $M_2 = \max_{\xi \in [a,b]} |f''(\xi)|$. This error is $1/m^2$ of the error bound of the normal Trapezium rule. In other words, it is an $O(h)$ error bound where h is the subinterval width.

9.4 The Euler-Maclaurin Expansion

We have seen that the error in the composite Trapezium rule is bounded by a term involving $1/m^2$, where m is the number of subdivisions of the interval $[a, b]$. The *Euler-Maclaurin expansion* that we are going to introduce expresses this error as a series in powers of $1/m^2$, making it possible to improve accuracy via extrapolation methods. We need the following theorem as a preparation.

Theorem 9.2. Suppose that the function g is defined and continuous on the interval $[-1, 1]$ and has a continuous derivative of order $2k$ over this interval. Then,

$$\int_{-1}^{1} g(t)dt - (g(-1) + g(1)) = \int_{-1}^{1} -tg'(t)dt = \sum_{r=1}^{k} q_{2r}(1)\left(g^{(2r-1)}(1) - g^{(2r-1)}(-1)\right) - \int_{-1}^{1} q_{2k}(t)g^{(2k)}(t)dt, \quad (255)$$

where q_r, $r = 1, 2, \cdots$ is a sequence of polynomials, such that (1) q_r is of degree r, (2) $q'_{r+1} = q_r$, (3) q_r is odd if r is odd and even if r is even, (4) $q_r(1) = q_r(-1) = 0$ for $r \geq 3$ odd, and (5) $q_1(t) = -t$.

Proof. The first equality follows from integration by parts. Then, note that $q_1(t) = -t$ and by repeatedly applying integration by parts in the other direction, we can compute that

$$\begin{aligned}
\int_{-1}^{1} -tg'(t)dt &= \int_{-1}^{1} q_1(t)g'(t)dt = g'(t)q_2(t)\Big|_{t=-1}^{1} - \int_{-1}^{1} q_2(t)g''(t)dt \\
&= \left(g'(t)q_2(t) - g''(t)q_3(t)\right)\Big|_{t=-1}^{1} + \int_{-1}^{1} q_3(t)g'''(t)dt \\
&= \cdots \\
&= \left(g'(t)q_2(t) - g''(t)q_3(t) + \cdots + g^{(2k-1)}(t)q_{2k}(t)\right)\Big|_{t=-1}^{1} - \int_{-1}^{1} q_{2k}(t)g^{(2k)}(t)dt. \quad (256)
\end{aligned}$$

Since that $q_r(1) = q_r(-1) = 0$ for $r \geq 3$ odd, we can simplify the above expression as

$$\begin{aligned}
\int_{-1}^{1} -tg'(t)dt &= \left(g'(t)q_2(t) + g'''(t)q_4(t) + \cdots + g^{(2k-1)}(t)q_{2k}(t)\right)\Big|_{t=-1}^{1} - \int_{-1}^{1} q_{2k}(t)g^{(2k)}(t)dt \\
&= \left(\sum_{r=1}^{k} g^{(2r-1)}(t)q_{2r}(t)\right)\Big|_{t=-1}^{1} - \int_{-1}^{1} q_{2k}(t)g^{(2k)}(t)dt \\
&= \sum_{r=1}^{k}\left(g^{(2r-1)}(1)q_{2r}(1) - g^{(2r-1)}(-1)q_{2r}(-1)\right) - \int_{-1}^{1} q_{2k}(t)g^{(2k)}(t)dt. \quad (257)
\end{aligned}$$

Since q_{2r} is even (because $2r$ is even), we have that $q_{2r}(-1) = q_{2r}(1)$, and thus we can obtain that

$$\int_{-1}^{1} -tg'(t)dt = \sum_{r=1}^{k} q_{2r}(1)\left(g^{(2r-1)}(1) - g^{(2r-1)}(-1)\right) - \int_{-1}^{1} q_{2k}(t)g^{(2k)}(t)dt, \quad (258)$$

concluding the second equality, so the proof is complete. □

Theorem 9.3 (Euler-Maclaurin expansion)**.** Suppose that the real-valued function f is defined and continuous on the interval $[a, b]$ and has a continuous derivative of order $2k$ on this interval. Consider the subdivision of $[a, b]$ into $m \geq 1$ closed intervals $[x_{i-1}, x_i]$, $i = 1, \cdots, m$, where $x_i = a + ih$, $i = 0, \cdots, m$, and $h = (b-a)/m$. Writing $T(m)$ for the result of approximating the integral $I = \int_a^b f(x)dx$ by the composite Trapezium rule with these aforementioned m subintervals, we have that

$$I - T(m) = \sum_{r=1}^{k} c_r h^{2r}\left(f^{(2r-1)}(b) - f^{(2r-1)}(a)\right) - \left(\frac{h}{2}\right)^{2k} \sum_{i=1}^{m} \int_{x_{i-1}}^{x_i} q_{2k}(t)f^{(2k)}(x)dx, \quad (259)$$

where $t = t(x) = -1 + 2(x - x_{i-1})/h$ for $x \in [x_{i-1}, x_i]$, $i = 1, \cdots, m$, and $c_r = 2^{-2r}q_{2r}(1)$ for $r = 1, \cdots, k$.

Proof. We express the integral as a sum over the m subintervals $[x_{i-1}, x_i]$, $i = 1, \cdots, m$. In each interval $[x_{i-1}, x_i]$, we make change of variable such that $x = x_{i-1} + h(t+1)/2$, so that

$$\int_{x_{i-1}}^{x_i} f(x)dx = \int_{-1}^{1} f\left(x_{i-1} + \frac{h(t+1)}{2}\right)\frac{hdt}{2} = \frac{h}{2}\int_{-1}^{1} g(t)dt, \tag{260}$$

where $g(t) = f(x) = f(x_{i-1} + h(t+1)/2)$. Now using the previous theorem, we can write that

$$\int_{-1}^{1} g(t)dt = \sum_{r=1}^{k} q_{2r}(1)\left(g^{(2r-1)}(1) - g^{(2r-1)}(-1)\right) - \int_{-1}^{1} q_{2k}(t)g^{(2k)}(t)dt + g(-1) + g(1). \tag{261}$$

Note that for each $l = 1, 2, \cdots, 2k$, we have that

$$g^{(l)}(t) = \left(\frac{h}{2}\right)^{l} f^{(l)}(x), \qquad \frac{dx}{dt} = \frac{h}{2} \implies dt = \frac{2dx}{h}. \tag{262}$$

Therefore, we can rewrite the above expression as

$$\begin{aligned}\int_{-1}^{1} g(t)dt &= \sum_{r=1}^{k} q_{2r}(1)\left(\frac{h}{2}\right)^{2r-1}\left(f^{(2r-1)}(x_i) - f^{(2r-1)}(x_{i-1})\right) - \int_{x_{i-1}}^{x_i} q_{2k}(t)\left(\frac{h}{2}\right)^{2k} f^{(2k)}(x)\frac{2dx}{h} + f(x_{i-1}) + f(x_i) \\ &= \sum_{r=1}^{k} 2c_r h^{2r-1}\left(f^{(2r-1)}(x_i) - f^{(2r-1)}(x_{i-1})\right) - \left(\frac{h}{2}\right)^{2k-1}\int_{x_{i-1}}^{x_i} q_{2k}(t)f^{(2k)}(x)dx + f(x_{i-1}) + f(x_i).\end{aligned} \tag{263}$$

Therefore, we have that

$$\begin{aligned}\int_{x_{i-1}}^{x_i} f(x)dx &= \frac{h}{2}\int_{-1}^{1} g(t)dt \\ &= \sum_{r=1}^{k} c_r h^{2r}\left(f^{(2r-1)}(x_i) - f^{(2r-1)}(x_{i-1})\right) - \left(\frac{h}{2}\right)^{2k}\int_{x_{i-1}}^{x_i} q_{2k}(t)f^{(2k)}(x)dx + \frac{h(f(x_{i-1}) + f(x_i))}{2}.\end{aligned} \tag{264}$$

Summing up over all the subintervals, the first summand in the above expression sums up to

$$\begin{aligned}&\sum_{i=1}^{m}\left(\sum_{r=1}^{k} c_r h^{2r}\left(f^{(2r-1)}(x_i) - f^{(2r-1)}(x_{i-1})\right)\right) \\ &\qquad = \sum_{r=1}^{k} c_r h^{2r}\sum_{i=1}^{m}\left(f^{(2r-1)}(x_i) - f^{(2r-1)}(x_{i-1})\right) = \sum_{r=1}^{k} c_r h^{2r}\left(f^{(2r-1)}(b) - f^{(2r-1)}(a)\right).\end{aligned} \tag{265}$$

Therefore, we have that

$$\begin{aligned}I = \int_a^b f(x)dx &= \sum_{i=1}^{m}\int_{x_{i-1}}^{x_i} f(x)dx \\ &= \sum_{r=1}^{k} c_r h^{2r}\left(f^{(2r-1)}(b) - f^{(2r-1)}(a)\right) + \left(\frac{h}{2}\right)^{2k}\sum_{i=1}^{m}\int_{x_{i-1}}^{x_i} q_{2k}(t)f^{(2k)}(x)dx + \underbrace{\sum_{i=1}^{m}\frac{h(f(x_{i-1}) + f(x_i))}{2}}_{\text{composite Trapezium } T(m)}.\end{aligned} \tag{266}$$

Consequently, we can conclude that

$$I - T(m) = \sum_{r=1}^{k} c_r h^{2r}\left(f^{(2r-1)}(b) - f^{(2r-1)}(a)\right) - \left(\frac{h}{2}\right)^{2k}\sum_{i=1}^{m}\int_{x_{i-1}}^{x_i} q_{2k}(t)f^{(2k)}(x)dx, \tag{267}$$

where for each subinterval $[x_{i-1}, x_i]$, we have defined $x = x_{i-1} + h(t+1)/2 \implies t = t(x) = -1 + 2(x - x_{i-1})/h$, thus the proof is complete. □

We note that the coefficients c_r are actually given by

$$c_r = \frac{q_{2r}(1)}{2^{2r}} = -\frac{B_{2r}}{(2r)!}, \tag{268}$$

where B_{2r} are the Bernoulli numbers with even index, determined from the Taylor expansion such that

$$\frac{x}{2}\coth\left(\frac{x}{2}\right) = \sum_{r=0}^{\infty} \frac{B_{2r}x^{2r}}{(2r)!}. \tag{269}$$

The calculation of B_{2r} was the output of arguably the first "computer program" written by Ada Lovelace and Charles Babbage. We provide the first few values of c_r here as

$$c_1 = -\frac{1}{12}, \qquad c_2 = \frac{1}{720}, \qquad c_3 = -\frac{1}{30240}, \qquad c_4 = \frac{1}{1209600}, \qquad \cdots \tag{270}$$

Implications of Euler-Maclaurin. If $f \in C^\infty[a,b]$ and periodic with $f^{(j)}(a) = f^{(j)}(b)$ (think of Fourier series, $\cos mx$, $\sin mx$, etc.), then this error $|I - T(m)|$ decays *superalgebraically* as $n \to \infty$. We say that $\epsilon_n \to 0$ superalgebraically as $n \to \infty$ if

$$\lim_{n\to\infty} \frac{\epsilon_n}{h^{pn}} = 0, \qquad \forall p > 0, \tag{271}$$

which means that $\epsilon_n \to 0$ faster than any power of h. For this reason, the composite Trapezium rule is very important in various numerical fields.

9.5 Clenshaw-Curtis Quadrature

This is a special case of Newton-Cotes formula, where we interpolate f at Chebyshev nodes and integrate each Chebyshev polynomial. More specifically, we approximate

$$\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} p_n(x)dx, \tag{272}$$

where p_n is the Lagrange interpolant expressed in Chebyshev polynomials

$$p_n(x) = \sum_{j=0}^{n} a_j T_j(x), \tag{273}$$

where a_j are coefficients. Then, we have that

$$\begin{aligned}\int_{-1}^{1} f(x)dx &\approx \sum_{j=0}^{n} a_j \int_{-1}^{1} T_j(x)dx = \sum_{j=0}^{n} a_j \int_{-1}^{1} \cos(j\arccos x)dx = \sum_{j=0}^{n} a_j \int_0^{\pi} \cos(j\theta)\sin\theta d\theta \\ &= \sum_{j=0}^{n} a_j \frac{1+\cos(j\pi)}{1-j^2} = \sum_{i=0}^{n/2} a_{2i}\frac{1+\cos(2i\pi)}{1-(2i)^2} = \sum_{i=0}^{n/2}\frac{2a_{2i}}{1-4i^2}.\end{aligned} \tag{274}$$

We will return to a "fast" method for computing a_i's when we talk about the *discrete Fourier transform*, but for now, you can imagine computing them with the composite Trapezium rule since $f(\cos\theta)\sin\theta$ is a periodic function.

4/12 Lecture

9.6 Extrapolation Methods

In general the calculation of the higher order derivatives involved in the Euler-Maclaurin expansion is not possible. However, the existence of the expansion allows us to eliminate successive terms by repeated calculation of the Trapezium rule approximation. For instance, the case $k = 2$ in the Euler-Maclaurin expansion can be written as

$$\int_a^b f(x)dx - T(m) = c_1 h^2 \left(f'(b) - f'(a)\right) + O(h^4). \tag{275}$$

This would also mean that

$$\int_a^b f(x)dx - T(2m) = c_1 \frac{h^2}{4} \left(f'(b) - f'(a)\right) + O(h^4). \tag{276}$$

Thus, we can eliminate the h^2 term via subtracting four times the second equation from the first one, such that

$$-3\int_a^b f(x)dx - T(m) + 4T(2m) = O(h^4) \implies \int_a^b f(x)dx = \frac{4T(2m) - T(m)}{3} + O(h^4). \tag{277}$$

The same elimination process could be used for any two values of m, but the advantage of using m and $2m$ is that in the computation of $T(2m)$, half of the required values of $f(x_i)$ are already known from $T(m)$, so we do not have to compute them again. This process of eliminating the term in h^2 from the expression of the error is known as *Richardson extrapolation* or h^2 *extrapolation*. It is easy to extend the process to higher-order terms. For instance,

$$\int_a^b f(x)dx - T(m) = c_1 h^2 \left(f'(b) - f'(a)\right) + c_2 h^4 \left(f'''(b) - f'''(a)\right) + c_3 h^6 \left(f^{(5)}(b) - f^{(5)}(a)\right) + O(h^8), \tag{278}$$

$$\int_a^b f(x)dx - T(2m) = c_1 \frac{h^2}{4} \left(f'(b) - f'(a)\right) + c_2 \frac{h^4}{16} \left(f'''(b) - f'''(a)\right) + c_3 \frac{h^6}{64} \left(f^{(5)}(b) - f^{(5)}(a)\right) + O(h^8). \tag{279}$$

Eliminating the h^2 term, we will obtain that

$$\begin{aligned} -3\int_a^b f(x)dx - T(m) + 4T(2m) &= c_2 \frac{3h^4}{4} \left(f'''(b) - f'''(a)\right) + c_3 \frac{15h^6}{16} \left(f^{(5)}(b) - f^{(5)}(a)\right) + O(h^8) \\ \implies \int_a^b f(x)dx - \underbrace{\frac{4T(2m) - T(m)}{3}}_{=:T_1(m)} &= -c_2 \frac{h^4}{4} \left(f'''(b) - f'''(a)\right) - c_3 \frac{5h^6}{16} \left(f^{(5)}(b) - f^{(5)}(a)\right) + O(h^8). \end{aligned} \tag{280}$$

Computing also with $2m$, we can obtain a second equation such that

$$\int_a^b f(x)dx - T_1(2m) = -c_2 \frac{h^4}{64} \left(f'''(b) - f'''(a)\right) - c_3 \frac{5h^6}{1024} \left(f^{(5)}(b) - f^{(5)}(a)\right) + O(h^8). \tag{281}$$

Eliminating the h^4 term, we will obtain that

$$\begin{aligned} -15\int_a^b f(x)dx - T_1(m) + 16T_1(2m) &= -c_3 \frac{15h^6}{64} \left(f^{(5)}(b) - f^{(5)}(a)\right) + O(h^8) \\ \implies \int_a^b f(x)dx - \underbrace{\frac{16T_1(2m) - T_1(m)}{15}}_{=:T_2(m)} &= c_3 \frac{h^6}{64} \left(f^{(5)}(b) - f^{(5)}(a)\right) + O(h^8). \end{aligned} \tag{282}$$

We can again repeat this process, *i.e.*, compute with $2m$ and eliminate the h^6 term, but we will not explicit do it here. We observe that

$$T_1(m) = \frac{4T(2m) - T(m)}{3}, \qquad T_2(m) = \frac{16T_1(2m) - T_1(m)}{15}, \qquad \cdots \tag{283}$$

By adopting the notational convention $T_0(m) = T(m)$, we can thus recursively obtain that

$$T_k(m) = \frac{4^k T_{k-1}(2m) - T_{k-1}(m)}{4^k - 1}, \tag{284}$$

which will approximate $\int_a^b f(x)dx$ to accuracy $O(h^{2k+2})$, as long as $f^{(2k+2)}$ exists and is continuous on the interval $[a, b]$. This extrapolation process is also known as the *Romberg integration method*.

9.7 Construction of Gauss Quadrature Rules

Before we were all talking about the Newton-Cotes family of formulae for numerical integration. But now, we are going to introduce another family of numerical integration rules, namely *Gauss quadrature formulae*.

Suppose that the function f is defined on the closed interval $[a, b]$ and that it is continuous and differentiable on this interval. Also suppose that w is a weight function, defined, positive, continuous, and integrable on (a, b). We wish to construct quadrature formulae for the approximate evaluation of the integral

$$\int_a^b w(x)f(x)dx. \tag{285}$$

For a nonnegative integer n, let x_i, $i = 0, \cdots, n$ be $n + 1$ points in the interval $[a, b]$, where we will determine the precise location of these points later. The *Hermite interpolation polynomial* of degree $2n + 1$ for the function f is given by the expression

$$p_{2n+1}(x) = \sum_{k=0}^{n} H_k(x)f(x_k) + \sum_{k=0}^{n} K_k(x)f'(x_k), \tag{286}$$

where

$$H_k(x) = L_k^2(x)(1 - 2L_k'(x_k)(x - x_k)), \tag{287}$$
$$K_k(x) = L_k^2(x)(x - x_k), \tag{288}$$
$$L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i} \in P_n. \tag{289}$$

If $n = 0$, we let $L_0(x) = 1$ and thus $H_0(x) = 1$ and $K_0(x) = x - x_0$. Now, we can deduce that

$$\int_a^b w(x)f(x)dx \approx \int_a^b w(x)p_{2n+1}(x)dx = \sum_{k=0}^{n} W_k f(x_k) + \sum_{k=0}^{n} V_k f'(x_k), \tag{290}$$

where

$$W_k = \int_a^b w(x)H_k(x)dx, \qquad V_k = \int_a^b w(x)K_k(x)dx. \tag{291}$$

There is an obvious advantage in choosing the points x_k in such a way that all the coefficients V_k are zero, so that the derivative values $f'(x_k)$ need not be computed. Recalling the form of the polynomial K_k and inserting it into the definition for V_k, we have that

$$V_k = \int_a^b w(x)L_k^2(x)(x - x_k)dx = C_n \int_a^b w(x)\pi_{n+1}(x)L_k(x)dx, \tag{292}$$

where

$$\pi_{n+1}(x) = \prod_{i=0}^{n}(x - x_i), \tag{293}$$
$$C_n = \begin{cases} \prod_{i=0, i\neq k}^{n}(x_k - x_i)^{-1}, & \text{if } n \geq 1, \\ 1, & \text{if } n = 0. \end{cases} \tag{294}$$

Since π_{n+1} is a polynomial of degree $n + 1$ while L_k is of degree n for each $0 \leq k \leq n$, then each V_k would be zero if the polynomial π_{n+1} is orthogonal to every polynomial of lower degree with respect to the weight function w. Therefore, we can construct the desired quadrature formula with $V_k = 0$, $k = 0, \cdots, n$ by choosing the points x_k, $k = 0, \cdots, n$ to be the zeros of the polynomial of degree $n + 1$ in a system of orthogonal polynomials over the interval (a, b) with respect to w. In this case,

$$\begin{aligned} W_k &= \int_a^b w(x)H_k(x)dx = \int_a^b w(x)L_k^2(x)(1 - 2L_k'(x_k)(x - x_k)) \\ &= \int_a^b w(x)L_k^2(x)dx - 2L_k'(x_k)V_k = \int_a^b w(x)L_k^2(x)dx. \end{aligned} \tag{295}$$

Therefore, the desired quadrature formula would become

$$\int_a^b f(x)dx \approx \sum_{k=0}^{n} W_k f(x_k), \tag{296}$$

where the *quadrature weights* are

$$W_k = \int_a^b w(x) L_k^2(x) dx, \tag{297}$$

and the *quadrature points* x_k, $k = 0, \cdots, n$ are chosen as the zeros of the polynomial of degree $n+1$ from a system of orthogonal polynomial over the interval (a, b) with respect to the weight function w. The quadrature formula for numerical integration as above is known as the *Gauss quadrature rule*. Note that when f is a polynomial of degree $2n+1$ or less, the Gauss quadrature rule gives the exact result. Also recall that the system of orthogonal polynomials can be constructed via Gram-Schmidt orthogonalization, as in Section 8.3.

4/17 Lecture

10 The Discrete Fourier Transform

We must first agree on what discrete Fourier transform means. The simple response is to give a formula, such as

$$F_k = \frac{1}{N} \sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f_n \exp\left(-\frac{i2\pi nk}{N}\right), \tag{298}$$

and state that this holds for k equal to any N consecutive integers. This is indeed the definition that we shall use, but we are also interested in its relation with the Fourier transform. In the course of this section, we will arrive at the DFT along the following paths: (1) approximation to the Fourier transform of a function; (2) approximation to the Fourier coefficients of a function; (3) trigonometric approximation; and (4) the Fourier transform of a spike train.

10.1 DFT Approximation to the Fourier Transform

For the moment, we assume that f is defined on $(-\infty, \infty)$ and has some known properties, one of which is that f is absolutely integrable on the real line, such that

$$\int_{-\infty}^{\infty} |f(x)| dx < \infty. \tag{299}$$

Then we may define a function $\hat{f}(\omega)$ by

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(x) \exp\left(-i2\pi\omega x\right) dx, \tag{300}$$

where $-\infty < \omega < \infty$ and $i = \sqrt{-1}$. The function $\hat{f}$ is called the *Fourier transform* of f and is uniquely determined by the equation above. The transform $\hat{f}$ is said to be defined in the *frequency domain (transform domain)*, while the input function f is said to be defined in the *spatial domain* (spacial coordinate) or in the *time domain* (time-dependent). Of tremendous importance is the fact that there is an inverse relation between f and $\hat{f}$ given by

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\omega) \exp\left(i2\pi\omega x\right) d\omega, \tag{301}$$

also known as the *inverse Fourier transform* of $\hat{f}(\omega)$.

We begin by looking at the *kernel* of the Fourier transform, *i.e.*, $\exp\left(-i2\pi\omega x\right) = \cos(2\pi\omega x) - i\sin(2\pi\omega x)$. For a fixed value of ω, the kernel consists of waves with period (wavelength) of $1/\omega$, measured in the units of x (either

length or time). These waves are called *modes*. On the other hand, the inverse Fourier transform can be regarded as a recipe for assembling the function f as a combination of modes of all frequencies $-\infty < \omega < \infty$. The mode associated with a particular frequency ω has a certain weight in this combination, and the weight is given by $\hat{f}(\omega)$. This process of assembling a function from all of the various modes is often called *synthesis*, and the complete set of values of $\hat{f}$ is often called the *spectrum* of f since it gives the entire "frequency content" of the function or signal f.

With this capsule account of the Fourier transform, next we see how the DFT emerges as a natural approximation. First a practical observation is needed: when a function is given, either it already has well-defined boundaries, or it is assumed zero outside some finite interval for the sake of computation. For now we assume that $f(x) = 0$ for $|x| > A/2$, then the Fourier transform with limited extent is given by

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(x) \exp\left(-i2\pi\omega x\right) dx = \int_{-A/2}^{A/2} f(x) \exp\left(-i2\pi\omega x\right) dx. \tag{302}$$

We wish to approximate this integral numerically, so we divide the interval of integration $[-A/2, A/2]$ into N subintervals of length $\Delta x = A/N$. Assume that N is even, then let the grid points be denoted as

$$x_{-N/2} = -\frac{A}{2}, \qquad x_{-N/2+1} = -\frac{A}{2} + \Delta x, \qquad \cdots, \qquad x_0 = 0, \qquad \cdots, \qquad x_{N/2} = \frac{A}{2}. \tag{303}$$

Now assume that the function f is known at the grid points. Letting the integrand be $g(x) = f(x) \exp\left(-i2\pi\omega x\right)$, we may apply the Trapezium rule to this integral, which leads to the approximation

$$\int_{-A/2}^{A/2} g(x)dx \approx \frac{\Delta x}{2}\left(g\left(-\frac{A}{2}\right) + 2\sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}-1} g(x_n) + g\left(\frac{A}{2}\right)\right). \tag{304}$$

For now we will add the requirement that $g(-A/2) = g(A/2)$, an assumption that will be the subject of great scrutiny later. With this assumption, the Trapezium rule may be rewritten as

$$\hat{f}(\omega) = \int_{-A/2}^{A/2} g(x)dx \approx \Delta x \sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} g(x_n) = \frac{A}{N} \sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f(x_n) \exp\left(-i2\pi\omega x_n\right). \tag{305}$$

Now, since N values of f are used in the Trapezium rule approximation, we also want to choose N values for ω at which to approximate $\hat{f}$. Closely related to the spatial or temporal domain $[-A/2, A/2]$ for x is a frequency domain which we will denote by $[-\Omega/2, \Omega/2]$, also equipped with N equally spaced grid points. Imagine all modes (sines and consines) that have an integer number of periods on $[-A/2, A/2]$ and fit exactly on the interval. Of these waves, we consider the one with the largest possible period. This is often called the *one-mode* or the *fundamental mode*, having period A and thus frequency $1/A$. This will be used as the fundamental unit of frequency $\Delta\omega = 1/A$, thus $\Omega = N/A$.

Reciprocity relations. From the observations above, we can conclude that

$$A\Omega = N, \qquad \Delta x \Delta\omega = \frac{A}{N} \cdot \frac{1}{A} = \frac{1}{N}. \tag{306}$$

These are two reciprocity relations, not independent but both are useful. With these reciprocity relations established, we can now return to the Trapezium rule approximation and extract the DFT in short order. For simplicity, denote $f_n = f(x_n)$, for $n = -N/2+1, -N/2+2, \cdots, N/2-1, N/2$. Then, agreeing to approximate $\hat{f}$ at the frequency grid points $\omega_k = k\Delta\omega = k/A$, we note that

$$x_n\omega_k = (n\Delta x)(k\Delta\omega) = nk \cdot \Delta x \Delta\omega = \frac{nk}{N}. \tag{307}$$

The sum in the Trapezium rule then becomes

$$\begin{aligned} \hat{f}\left(\frac{k}{A}\right) = \hat{f}(\omega_k) &\approx \frac{A}{N} \sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f(x_n) \exp\left(-i2\pi\omega_k x_n\right) = \frac{A}{N} \sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f_n \exp\left(-\frac{i2\pi nk}{N}\right) \\ &= A \cdot \underbrace{\frac{1}{N} \sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f_n \exp\left(-\frac{i2\pi nk}{N}\right)}_{=:F_k}, \qquad k \in \left[-\frac{N}{2}+1, \frac{N}{2}\right] \cap \mathbb{Z}. \end{aligned} \tag{308}$$

Given the set of N sample values f_n, the DFT consists of N coefficients F_k. In addition to identifying the DFT, we can conclude that the approximations to the Fourier transform $\hat{f}(\omega_k)$ are given by $\hat{f}(\omega_k) = AF_k$. This approximation and the errors it entails will be investigated later. Next we will officially introduce the DFT.

10.2 The DFT-IDFT Pair

For convenience, we will adopt the notation

$$\omega_N = \exp\left(\frac{i2\pi}{N}\right) = \cos\left(\frac{2\pi}{N}\right) + i\sin\left(\frac{2\pi}{N}\right), \tag{309}$$

so that $\omega_N^{-nk} = \exp(-i2\pi nk/N)$, and $\omega_N^{nk} = \exp(i2\pi nk/N)$.

Definition 10.1 (Discrete Fourier transform)**.** Let N be an even positive integer and let f_n be a sequence of N complex numbers where $n = -N/2+1, \cdots, N/2$. Then its *discrete Fourier transform* is another sequence of N complex numbers given by

$$F_k = \frac{1}{N}\sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f_n \omega_N^{-nk}, \qquad k = -\frac{N}{2}+1, \cdots, \frac{N}{2}. \tag{310}$$

Analogously if N is odd, the range of n will be $-(N-1)/2, \cdots, (N-1)/2$, and the sum will be in this range as well.

An alternate form of the DFT can be as follows. Let N be a positive integer and let f_n be a sequence of N complex numbers where $n = 0, \cdots, N-1$. Then its discrete Fourier transform is another sequence of N complex numbers given by

$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n \omega_N^{-nk}, \qquad k = 0, \cdots, N-1. \tag{311}$$

This is equivalent to the original form, and is independent of the parity (even or odd) of N. Moreover, note that the DFT can be split into the real and imaginary parts. We can compute that

$$\begin{aligned} F_k &= \frac{1}{N}\sum_{-\frac{N}{2}+1}^{\frac{N}{2}} f_n \omega_N^{-nk} = \frac{1}{N}\sum_{-\frac{N}{2}+1}^{\frac{N}{2}} (\mathrm{Re}(f_n) + i\mathrm{Im}(f_n))\left(\cos\left(\frac{2\pi nk}{N}\right) - i\sin\left(\frac{2\pi nk}{N}\right)\right) \\ &= \frac{1}{N}\sum_{-\frac{N}{2}+1}^{\frac{N}{2}} \left(\mathrm{Re}(f_n)\cos\left(\frac{2\pi nk}{N}\right) + \mathrm{Im}(f_n)\sin\left(\frac{2\pi nk}{N}\right)\right) \\ &\quad + \frac{i}{N}\sum_{-\frac{N}{2}+1}^{\frac{N}{2}} \left(\mathrm{Im}(f_n)\cos\left(\frac{2\pi nk}{N}\right) - \mathrm{Re}(f_n)\sin\left(\frac{2\pi nk}{N}\right)\right). \end{aligned} \tag{312}$$

Inverse discrete Fourier transform (IDFT). Let N be an even positive integer and let F_k be a sequence of N complex numbers, where $k = -N/2+1, \cdots, N/2$. Then its *inverse discrete Fourier transform* is another sequence of N complex numbers, given by

$$f_n = \sum_{k=-N/2+1}^{N/2} F_k \omega_N^{nk}, \qquad n = -\frac{N}{2}+1, \cdots, \frac{N}{2}. \tag{313}$$

Note that this definition confers periodicity on the sequence f_n, *i.e.*, $f_n = f_{n+N}$. We denote $\mathscr{D}(f_n)$ to mean the DFT of the sequence f_n and $\mathscr{D}(f_n)_k$ to mean the kth element of the DFT. We also denote $\mathscr{D}^{-1}(F_k)$ to mean the IDFT of the sequence F_k and $\mathscr{D}^{-1}(F_k)_n$ to mean the nth element of the IDFT. Therefore, we have that

$$\mathscr{D}(f_n)_k = F_k, \qquad \mathscr{D}^{-1}(F_k)_n = f_n, \qquad \mathscr{D}\left(\mathscr{D}^{-1}(F_k)\right)_k = F_k, \qquad \mathscr{D}^{-1}\left(\mathscr{D}(f_n)\right)_n = f_n. \tag{314}$$

To verify the last two equations, we must first develop the orthogonality property of the complex exponential. We introduce the notion of *modular Kronecker delta*, such that

$$\hat{\delta}_N(k) = \begin{cases} 1, & \text{if } k = 0 \text{ or an integer multiple of } N, \\ 0, & \text{otherwise.} \end{cases} \tag{315}$$

Theorem 10.2. Let j and k be integers and let N be a positive integer. Then

$$\sum_{n=0}^{N-1} \exp\left(\frac{i2\pi nj}{N}\right) \exp\left(-\frac{i2\pi nk}{N}\right) = N\hat{\delta}_N(j-k) = \begin{cases} N, & \text{if } j = k \text{ or } j-k \text{ is an integer multiple of } N, \\ 0, & \text{otherwise.} \end{cases} \tag{316}$$

Given the theorem (of orthogonality) above, we can see that

$$\begin{aligned} \mathscr{D}^{-1}\left(\mathscr{D}(f_n)_k\right)_n &= \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} F_k \omega_N^{nk} = \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} \left(\frac{1}{N} \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} f_j \omega_N^{-jk} \right) \omega_N^{nk} = \frac{1}{N} \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} f_j \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} \omega_N^{k(n-j)} \\ &= \frac{1}{N} \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} f_j \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} \omega_N^{k(n-j)} = \frac{1}{N} \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} f_j \underbrace{\sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} \exp\left(\frac{i2\pi k(n-j)}{N}\right)}_{=N\hat{\delta}_N(n-j)} \\ &= \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} f_j \hat{\delta}_N(n-j) = f_n, \qquad \forall k \implies \mathscr{D}^{-1}\left(\mathscr{D}(f_n)\right)_n = f_n. \end{aligned} \tag{317}$$

It is equally easy to show the other side round, *i.e.*, $\mathscr{D}\left(\mathscr{D}^{-1}(F_k)\right)_k = F_k$.

10.3 DFT Approximations to Fourier Series Coefficients

As closely as the DFT is related to the Fourier transform, it may be argued that it holds even more kinship to the coefficients of the Fourier series.

Definition 10.3 (Fourier series)**.** Let f be a function that is A-periodic, then the *Fourier series* associated with f is the trigonometric series

$$f(x) \sim \sum_{k=-\infty}^{\infty} c_k \exp\left(\frac{i2\pi kx}{A}\right), \tag{318}$$

where the coefficients c_k are given by

$$c_k = \frac{1}{A} \int_{-A/2}^{A/2} f(x) \exp\left(-\frac{i2\pi kx}{A}\right) dx. \tag{319}$$

We have a similar orthogonality relation

$$\int_{-A/2}^{A/2} \exp\left(\frac{i2\pi jx}{A}\right) \exp\left(-\frac{i2\pi kx}{A}\right) dx = A\delta(j-k) = \begin{cases} A, & \text{if } j = k, \\ 0, & \text{otherwise,} \end{cases} \tag{320}$$

which follows from direct integration. To find the coefficients c_k, we assume that the A-periodic function f is the sum of its Fourier series, so that

$$f(x) = \sum_{j=-\infty}^{\infty} c_j \exp\left(\frac{i2\pi jx}{A}\right), \tag{321}$$

then we can find that

$$\begin{aligned}\frac{1}{A}\int_{-A/2}^{A/2} f(x)\exp\left(-\frac{i2\pi kx}{A}\right)dx &= \frac{1}{A}\int_{-A/2}^{A/2}\sum_{j=-\infty}^{\infty} c_j \exp\left(\frac{i2\pi(j-k)x}{A}\right)dx \\ &= \frac{1}{A}\sum_{j=-\infty}^{\infty} c_j \cdot \underbrace{\int_{-A/2}^{A/2}\exp\left(\frac{i2\pi(j-k)x}{A}\right)dx}_{=A\delta(j-k)} \\ &= \frac{1}{A}\sum_{j=-\infty}^{\infty} c_j A\delta(j-k) = c_k. \end{aligned} \tag{322}$$

Fourier Series for real-valued f. Let f be a real-valued function that is A-periodic. Then the Fourier series associated with f is the trigonometric series

$$f(x) \sim \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi kx}{A}\right) + \sum_{k=1}^{\infty} b_k \sin\left(\frac{2\pi kx}{A}\right), \tag{323}$$

where the coefficients are given by

$$a_k = \frac{2}{A}\int_{-A/2}^{A/2} f(x)\cos\left(\frac{2\pi kx}{A}\right)dx, \qquad k = 0, 1, 2, \cdots, \tag{324}$$

$$b_k = \frac{2}{A}\int_{-\frac{A}{2}}^{\frac{A}{2}} f(x)\sin\left(\frac{2\pi kx}{A}\right)dx, \qquad k = 1, 2, \cdots. \tag{325}$$

Approximating Fourier coefficients. Let $g(x) = f(x)\exp(-i2\pi kx/A)$ be the integrand and apply the Trapezium rule. We will have the approximation that

$$c_k = \frac{1}{A}\int_{-A/2}^{A/2} g(x)dx \approx \frac{1}{A}\cdot\frac{\Delta x}{2}\left(g\left(-\frac{A}{2}\right) + 2\sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}-1} g(x_n) + g\left(\frac{N}{2}\right)\right). \tag{326}$$

Note that for this choice of g, dictated by the convergence properties of the Fourier series, we can guarantee that $g(-A/2) = g(A/2)$ without making this unnecessary assumption. Therefore, the above approximation can be rewritten as

$$c_k \approx \frac{\Delta x}{A}\sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} g(x_n) = \frac{1}{N}\sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f(x_n)\exp\left(-\frac{i2\pi kx_n}{A}\right) = \frac{1}{N}\sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f(x_n)\exp\left(-\frac{i2\pi nk}{N}\right). \tag{327}$$

Denoting $f_n = f(x_n)$, this simplifies to

$$c_k \approx \frac{1}{N}\sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f_n\omega_N^{-nk} = \mathscr{D}(f_n)_k, \qquad k = -\frac{N}{2}+1, \cdots, \frac{N}{2}. \tag{328}$$

10.4 DFT from Trigonometric Approximation

The derivations shown so far have evolved from the problem of approximating either the Fourier series coefficients or the Fourier transform of a particular function. Another way to uncover the DFT follows by considering the problem of approximating (or fitting) a set of data with a function known as a trigonometric polynomial. The goal is to find a linear combination of sines and cosines that best approximates a given dataset.

Suppose that we are given N data pairs that we will denote (x_n, f_n), where $n = -(N-1)/2, \cdots, (N-1)/2$. Note that this derivation is one instance in which it is more convenient to work with N odd. The x_n's are real and are

assumed to be equally spaced in the interval $[-A/2, A/2]$. That is, $x_n = n\Delta x$ with $\Delta x = A/N$. The f_n's may be complex-valued. We seek the best possible approximation of the data using the N-term trigonometric polynomial ψ_N, given by

$$\psi_N(x) = \sum_{k=-\frac{N-1}{2}}^{\frac{N-1}{2}} \alpha_k \exp\left(\frac{i2\pi kx}{A}\right). \tag{329}$$

We will use the least squares criterion here, that is, we seek to choose the coefficients α_k to minimize the *discrete least squares error*

$$E = \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} |f_n - \psi_N(x_n)|^2 = (f_n - \psi_N(x_n))\,(f_n - \psi_N(x_n))^*\,. \tag{330}$$

Therefore, we desire that

$$\frac{\partial E}{\partial \alpha_k} = \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} \left(\exp\left(-\frac{i2\pi nk}{N}\right)\left(f_n - \sum_{p=-\frac{N-1}{2}}^{\frac{N-1}{2}} \alpha_p \exp\left(\frac{i2\pi np}{N}\right)\right)\right) = 0. \tag{331}$$

Proper rearrangement of terms gives that

$$\sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} f_n \exp\left(-\frac{i2\pi nk}{N}\right) = \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} \sum_{p=-\frac{N-1}{2}}^{\frac{N-1}{2}} \alpha_p \exp\left(\frac{i2\pi n(p-k)}{N}\right). \tag{332}$$

Use previous results, we can obtain that

$$\sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} f_n \omega_N^{-nk} = \sum_{p=-\frac{N-1}{2}}^{\frac{N-1}{2}} \alpha_p \underbrace{\sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} \omega_N^{(p-k)n}}_{=N\hat{\delta}_N(p-k)} = \sum_{p=-\frac{N-1}{2}}^{\frac{N-1}{2}} \alpha_p N\hat{\delta}_N(p-k) = N\alpha_k. \tag{333}$$

Bringing this choice of α_k into the expression of the error, we thus have that

$$\begin{aligned}
E &= \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} |f_n|^2 - \underbrace{\sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} f_n \psi_N^*(x_n)}_{N\sum_n |\alpha_n|^2} - \underbrace{\sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} f_n^* \psi_N(x_n)}_{N\sum_n |\alpha_n|^2} + \underbrace{\sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} |\psi_N(x_n)|^2}_{N\sum_n |\alpha_n|^2} = \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} |f_n|^2 - N \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} |\alpha_n|^2 \\
&= \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} |f_n|^2 - \frac{N}{N^2} \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} \sum_{p=-\frac{N-1}{2}}^{\frac{N-1}{2}} \sum_{m=-\frac{N-1}{2}}^{\frac{N-1}{2}} \left(f_p f_m^* \exp\left(\frac{i2\pi n(m-p)}{N}\right)\right) \\
&= \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} |f_n|^2 - \frac{1}{N} \sum_{p=-\frac{N-1}{2}}^{\frac{N-1}{2}} \sum_{m=-\frac{N-1}{2}}^{\frac{N-1}{2}} f_p f_m^* \underbrace{\sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} \exp\left(\frac{i2\pi n(m-p)}{N}\right)}_{=N\hat{\delta}_N(m-p)} \\
&= \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} |f_n|^2 - \sum_{p=-\frac{N-1}{2}}^{\frac{N-1}{2}} \sum_{m=-\frac{N-1}{2}}^{\frac{N-1}{2}} f_p f_m^* \hat{\delta}_N(m-p) = \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} |f_n|^2 - \sum_{m=-\frac{N-1}{2}}^{\frac{N-1}{2}} |f_m|^2 = 0.
\end{aligned} \tag{334}$$

Since the sum of squares of the individual errors at the grid points is zero, it necessarily follows that the individual errors at each grid point must be zero. In other words, ψ_N is not only the (discrete) least squares approximation of the data, but also an interpolating function. Moreover, with $E = 0$ being known now, we have the intermidiate result that

$$\sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} |f_n|^2 = N \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} |\alpha_n|^2, \tag{335}$$

which is a fundamental property of the DFT known as *Parseval's relation*.

4/19 Lecture

No class.

4/24 Lecture

10.5 Fast Fourier Transform

Recall that the goal is to compute the DFT of a (possibly complex) sequence x_n which has length N and is assumed to have period N. As we have noted many times, there are several commonly used forms of the DFT. The FFT can be developed for any of these forms, but a specific choice must be made for the sake of exposition. For this part, we will use the definition in which both indices n and k belong to the set $\{0, \cdots, N-1\}$, and

$$X_k = \sum_{n=0}^{N-1} x_n \omega_N^{-nk}, \qquad \omega_N = \exp\left(\frac{i2\pi}{N}\right). \tag{336}$$

Note that we are dispending the $1/N$ scaling factor, which can always be included at the end of the calculation if needed. We begin with the cases where $N = 2^M$ for some natural number M, and split x_n into its even and odd sequences $y_n = x_{2n}$ and $z_n = x_{2n+1}$. Then

$$X_k = \sum_{n=0}^{\frac{N}{2}-1} \left(y_n \omega_N^{-2nk} + z_n \omega_N^{-(2n+1)k}\right). \tag{337}$$

We note that

$$\omega_N^{-2nk} = \exp\left(-\frac{i4\pi nk}{N}\right) = \exp\left(-\frac{i2\pi nk}{N/2}\right) = \omega_{N/2}^{-nk}, \tag{338}$$

or more generally, $\omega_N^{-pq} = \omega_{N/q}^{-p}$ for $p, q \in \mathbb{R}$. Therefore, the expression of X_k can be rewritten as

$$X_k = \underbrace{\sum_{n=0}^{\frac{N}{2}-1} y_n \omega_{N/2}^{-nk}}_{\text{DFT of length } N/2} + \omega_N^{-k} \underbrace{\sum_{n=0}^{\frac{N}{2}-1} z_n \omega_{N/2}^{-nk}}_{\text{DFT of length } N/2}. \tag{339}$$

And now, if we stand back, we see that the original DFT has been expressed as a simple combination of the DFT of the sequence y_n and the DFT of the sequence z_n, both of length $N/2$, denoted by Y_k and Z_k, respectively. Then

$$X_k = Y_k + \omega_N^{-k} Z_k, \qquad X_{k+\frac{N}{2}} = Y_{k+\frac{N}{2}} + \omega_N^{-\left(k+\frac{N}{2}\right)} X_{k+\frac{N}{2}}. \tag{340}$$

Note that

$$\omega_N^{-\left(k+\frac{N}{2}\right)} = \exp\left(-\frac{i2\pi k + i\pi N}{N}\right) = \exp\left(-\frac{i2\pi k}{N}\right)\exp\left(-i\pi\right) = -\exp\left(-\frac{i2\pi k}{N}\right) = -\omega_N^k, \tag{341}$$

so that

$$X_{k+\frac{N}{2}} = Y_{k+\frac{N}{2}} - \omega_N^{-k} X_{k+\frac{N}{2}} = Y_k - \omega_N^{-k} Z_k. \tag{342}$$

These is often called the *combined formula* or *butterfly relations*. They give a recipe for combining two DFTs of length $N/2$ (corresponding to the even and odd subsequences of the original sequence) to form the DFT of the original sequence. It is worthwhile to note the savings that have already been achieved. Computing the sequence X_k explicitly from its definition costs approximately N^2 complex multiplications and N^2 complex additions. However with the splitting method, the DFT requires only $2(N/2)^2 = N^2/2$ multiplications and $2(N/2)^2 = N^2/2$ additions, which is a factor-of-two savings.

However, we are not finished yet. We just assumed that the DFT sequences Y_k and Z_k would be computed as matrix-vector products. However, a full FFT algorithm results when the splitting idea is applied to the computation of Y_k and Z_k. In a divide-and-conquer spirit, it is then repeated on and on for $M = \log_2 N$ steps, and eventually the original problem of computing a DFT of length N will be replaced by computing N DFTs of length 1 (this actually has nothing to be done; the DFT of a sequence of length 1 is just itself).

On the first level, we combine two problems of size $N/2$ into one problem of size N. This will take N butterflies. On the second level, we combine two problems of size $N/4$ into one problem of size $N/2$, and this happens twice. They will take $2 \cdot N/2 = N$ butterflies. We iterate until the last level, where we combine two problems of size 1 into one problem of size 2, and this happens $N/2$ times since there are N problems of size 1 on the last level. They will take $N/2 \cdot 2 = N$ butterflies. Overall, each level takes N butterflies, and there are $\log_2 N$ levels in total, so the whole process takes $N \log_2 N$ butterflies.

Now note that each two butterflies take one complex multiplication and two complex additions. This is because

$$X_k, X_{k+\frac{N}{2}} = Y_k \pm \omega_N^{-k} Z_k, \tag{343}$$

so one complex multiplication can serve two butterflies. Therefore, the total operation count of an N-point complex DFT would involve $N \log_2 N$ complex additions and $N \log_2 N/2$ complex multiplications. Therefore, FFT achieves $O(N \log N)$ computational complexity, achieving an $O(N/\log N)$ computational savings over the evaluation of DFT by definition.

4/26 Lecture

10.6 Spectral Differentiation

A common numerical technique is to differentiate some sampled function $y(x)$ via FFTs. Equivalently, one differentiates an approximate Fourier series (or a trigonometric interpolation). These are also known as the *spectral differentiation methods*. Consider one dimension here for simplicity, one has a periodic function $y(x)$ with period L that one conceptually expands as a Fourier series

$$y(x) = \sum_{k=-\infty}^{\infty} Y_k \exp\left(\frac{i2\pi kx}{L}\right), \tag{344}$$

for the Fourier coefficients

$$Y_k = \frac{1}{L}\int_0^L \exp\left(-\frac{i2\pi kx}{L}\right) y(x)dx. \tag{345}$$

Differentiation is performed term-by-term in the Fourier domain, and multiplication by some function $c(x)$ is done by transforming back to space domain. For instance,

$$\frac{d}{dx}y(x) = \sum_{k=-\infty}^{\infty} \left(\frac{i2\pi k}{L} \cdot Y_k\right) \exp\left(\frac{i2\pi kx}{L}\right). \tag{346}$$

To implement this on a computer, one consider the DFT and IDFT

$$Y_k = \frac{1}{N}\sum_{n=0}^{N-1} y_n \exp\left(-\frac{i2\pi nk}{N}\right), \qquad y_n = \sum_{k=0}^{N-1} Y_k \exp\left(\frac{i2\pi nk}{N}\right). \tag{347}$$

Since we have FFT that costs $\Theta(N \log N)$ operations, we can quickly transform back and forth between the spacial domain (where multiplications are easy) and the Fourier domain (where derivatives are easy). Now in order to compute derivatives like $y'(x)$, we need to do more than expressing y_n. We need to use the IDFT expression to define a continuous interpolation between the samples y_n and then differentiate this interpolation. This is often called a *trigonometric interpolation*. However, it is not unique. We note that

$$\exp\left(\frac{i2\pi nk}{N}\right) = \exp\left(\frac{i2\pi nk}{N}\right) \underbrace{\exp\left(i2\pi nm\right)}_{=1, \text{ for } m,n\in\mathbb{Z}} = \exp\left(\frac{i2\pi(k+mN)n}{N}\right), \tag{348}$$

so that

$$\sum_{k=0}^{N-1} Y_k \exp\left(\frac{i2\pi(k+mN)n}{N}\right) \tag{349}$$

for any integer m still gives the same samples y_n, essentially just oscillating m extra times in between the sample points. This has no effect on y_n but has a huge effect on the derivatives.

We define a more arbitrary interpolated function $y(x)$ by substituting $n = Nx/L$ into the IDFT and allowing any aliasing integer m_k for each Y_k, such that

$$y(x) = \sum_{k=0}^{N-1} Y_k \exp\left(\frac{i2\pi(k+m_kN)x}{L}\right), \tag{350}$$

but still limit to N frequency components. Regardless of m_k, this gives the same sample points $y_n = y(nL/N)$, but changing m_k modifies $y(x)$ between the samples. In order to uniquely determine m_k, we wish to oscillate as little as possible between the sample points y_n. One way to express this idea is to assume that $y(x)$ is *bandlimited* to frequencies $|k + m_kN| \le N/2$. If $0 \le k < N/2$, then $|k + m_kN|$ is minimized when $m_k = 0$. If $N/2 < k < N$, then $|k + m_kN|$ is minimized when $m_k = -1$. If $k = N/2$, we split between $m_k = 0$ and $m_k = -1$. This results in the *unique minimal-oscillation trigonometric interpolation* of order N, such that

$$y(x) = Y_0 + \sum_{0<k<N/2} \left(Y_k \exp\left(\frac{i2\pi kx}{L}\right) + Y_{N-k} \exp\left(-\frac{i2\pi kx}{L}\right)\right) + Y_{N/2} \cos\left(\frac{\pi Nx}{L}\right), \tag{351}$$

where the $N/2$ (called the Nyquist) term is absent for odd N.

First derivative. We consider how to compute the derivatives $y'_n = y'(nL/N)$ and $y''_n = y''(nL/N)$ at the sample points, using FFTs to compute the trigonometric interpolation coefficients. The first derivative of $y(x)$ is

$$y'(x) = \sum_{0<k<N/2} \frac{i2\pi k}{L} \left(Y_k \exp\left(\frac{i2\pi kx}{L}\right) - Y_{N-k} \exp\left(-\frac{i2\pi kx}{L}\right)\right) - \frac{\pi N}{L} Y_{N/2} \sin\left(\frac{\pi Nx}{L}\right). \tag{352}$$

When we evaluate this at the sample points $x = nL/N$, we obtain

$$y'_n = y\left(\frac{nL}{N}\right) = \sum_{0<k<N/2} \frac{i2\pi k}{L} \left(Y_k \exp\left(\frac{i2\pi nk}{N}\right) - Y_{N-k} \exp\left(-\frac{i2\pi nk}{N}\right)\right) =: \sum_{k=0}^{N-1} Y'_k \exp\left(i\frac{2\pi nk}{N}\right), \tag{353}$$

where the $Y_{N/2}$ term has vanished since $\sin(n\pi) = 0$. The resulting procedure is as follows.

1: Given y_n for $0 \le n < N$, use an FFT to compute Y_k for $0 \le k < N$;
2: Multiply Y_k by $i2\pi k/L$ for $k < N/2$, by $i2\pi(k-N)/L$ for $k > N/2$, and by *zero* for $k = N/2$ to obtain Y'_k;
3: Compute y'_n from Y'_k via an inverse FFT;

Second derivative. On the other hand, the second derivativeof $y(x)$ is

$$y''(x) = -\sum_{0<k<N/2} \left(\frac{2\pi k}{L}\right)^2 \left(Y_k \exp\left(\frac{i2\pi kx}{L}\right) + Y_{N-k} \exp\left(-\frac{i2\pi kx}{L}\right)\right) - \left(\frac{\pi N}{L}\right)^2 Y_{N/2} \cos\left(\frac{\pi Nx}{L}\right), \tag{354}$$

which at the sample points gives that

$$\begin{aligned} y''_n = y''\left(\frac{nL}{N}\right) &= -\sum_{0<k<N/2} \left(\frac{2\pi k}{L}\right)^2 \left(Y_k \exp\left(\frac{i2\pi nk}{N}\right) + Y_{N-k} \exp\left(-\frac{i2\pi nk}{N}\right)\right) - \left(\frac{\pi N}{L}\right)^2 Y_{N/2}(-1)^n \\ &=: \sum_{k=0}^{n-1} Y''_k \exp\left(\frac{i2\pi nk}{N}\right), \end{aligned} \tag{355}$$

where the $Y_{N/2}$ term has not vanished. The resulting procedure is the same as that for computing the first derivative, except that the step to obtain Y''_k needs to be slightly modified according to the expression above. In fact, all other derivatives can be obtained in a similar way.

10.7 Spectral Integration

To be done...

5/1 Lecture

10.8 Convolution

The *discrete convolution theorem* is among the most important properties of the DFT. It underlies essentially all of the Fourier transform-based signal processing done today, and by itself accounts for much of the utility of the DFT. We begin by defining the notion of discrete convolution. Given two N-periodic sequences f_n and g_n defined for the indices $n = -N/2+1, \cdots, N/2$, their *discrete (cyclic) convolution*, denoted $f_n * g_n$, is another sequence h_n given by

$$h_n = f_n * g_n = \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} f_j g_{n-j}, \qquad n = -\frac{N}{2}+1, \cdots, \frac{N}{2}. \tag{356}$$

Notice that h_n is also an N-periodic sequence. Now it is easy to see that the convolution is commutative, *i.e.*, $f_n * g_n = g_n * f_n$. Moreover, a scalar multiple can be passed through, such that $(\alpha f_n) * g_n = \alpha(f_n * g_n) = f_n * (\alpha g_n)$. Further insight into discrete convolution might be offered by considering a shifting property of the (Kronecker) δ sequence. For a fixed integer n_0, we can easily see that

$$f_n * \delta(n - n_0) = \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} f_j \delta(n - n_0 - j) = f_{n-n_0}. \tag{357}$$

Note that we may view the convolution of two sequences as a filtering operation in which one of the sequences is the input data and the other is a filter. We now arrive at one of the most important of the DFT properties.

Theorem 10.4 (Discrete convolution theorem)**.** Let f_n and g_n be periodic sequences of length N whose DFTs are F_k and G_k. Then the DFT of the convolution $h_n = f_n * g_n$ is

$$H_k = \mathscr{D}(f_n * g_n)_k = NF_kG_k. \tag{358}$$

That is, the DFT of the convolution is the pointwise product of the DFTs. This is often expressed by saying that convolution in the spatial (or time) domain corresponds to multiplication in the frequency domain.

Proof. We begin by writing f_n and g_{n-j} as the IDFTs of their DFTs. By the *modulation property*, we know that $g_{n-j} = \mathscr{D}^{-1}(G_k \omega_N^{-jk})$. Hence, we can write that

$$\begin{aligned} f_n * g_n &= \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} f_j g_{n-j} = \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} \left(\sum_{p=-\frac{N}{2}+1}^{\frac{N}{2}} F_p \omega_N^{jp} \right) \left(\sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} G_k \omega_N^{(n-j)k} \right) \\ &= \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} G_k \omega_N^{nk} \sum_{p=-\frac{N}{2}+1}^{\frac{N}{2}} F_p \underbrace{\sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} \omega_N^{jp-jk}}_{=N\hat{\delta}_N(p-k)} = \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} NF_kG_k\omega_N^{nk}. \end{aligned} \tag{359}$$

This shows that $f_n * g_n$ is the IDFT of NF_kG_k. Stated differently, we conclude that

$$f_n * g_n = \mathscr{D}^{-1}(NF_kG_k)_n, \qquad \mathscr{D}(f_n * g_n)_k = NF_kG_k, \tag{360}$$

as desired, so the proof is complete. □

Frequency convolution. Just as the DFT of a convolution of two sequences is the product of their DFTs, it can be shown that the DFT of the product of two sequences is the convolution of their DFTs, *i.e.*, $\mathscr{D}(f_n g_n)_k = F_k * G_k$. To see this, we consider the inverse transform of the convolution and write

$$\begin{aligned}\mathscr{D}^{-1}(F_k * G_k)_n &= \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} \left(\sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} F_j G_{k-j} \right) \omega_N^{nk} = \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} F_j \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} G_{k-j} \omega_N^{nk} \\ &= \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} F_j \omega_N^{nj} \underbrace{\sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} G_{k-j} \omega_N^{n(k-j)}}_{=g_n} = g_n \underbrace{\sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} F_j \omega_N^{nj}}_{=f_n} = f_n g_n. \end{aligned} \tag{361}$$

Two important properties. We will introduce two important properties of DFT, the modulation property and the shift property. The *shift property* can be expressed as

$$\mathscr{D}(f_{n-j})_k = \omega_N^{-jk} F_k, \qquad \mathscr{D}^{-1}(\omega_N^{-jk} F_k)_n = f_{n-j}. \tag{362}$$

The *modulation property* can be expressed as

$$\mathscr{D}^{-1}(F_{k-j})_n = \omega_N^{nj} f_n, \qquad \mathscr{D}(\omega_N^{nj} f_n)_k = F_{k-j}. \tag{363}$$

10.9 Digital Signal Processing

For our purposes, a *digital signal* is a sequence of numbers occurring at regular intervals, often obtained by recording some fluctuating voltage or current in an electronic device that measures some physical quantity. As a motivating example, consider a signal consisting of $N = 24$ samples such that

$$f_n = \cos(2\pi n \Delta x) + \frac{1}{2}\cos(10\pi n \Delta x) + \frac{1}{3}\cos(12\pi n \Delta x), \qquad n = -11, -10, \cdots, 12, \qquad \Delta x = \frac{1}{24}. \tag{364}$$

Suppose that a new sequence g_n is formed by computing a five-point weighted running average of f_n. Assuming f_n is 24-periodic, we let

$$g_n = \frac{1}{8} f_{n-2} + \frac{1}{4} f_{n-1} + \frac{1}{4} f_n + \frac{1}{4} f_{n+1} + \frac{1}{8} f_{n+2}, \qquad n = -11, -10, \cdots, 12. \tag{365}$$

Here, g_n will be much smoother than f_n. Note that the process of forming the five-point weighted running average may be expressed as a cyclic convolution

$$g_n = f_n * h_n = \sum_{j=-11}^{12} f_j h_{n-j}, \tag{366}$$

where h_n is the sequence

$$\underbrace{0, 0, 0, 0, 0, 0, 0, 0, 0}_{9 \text{ zeros}}, \frac{1}{8}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \underbrace{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}_{10 \text{ zeros}}. \tag{367}$$

Recall that $\mathscr{D}(f_n * h_n)_k = N F_k H_k$. Here, h_n is called the *filter*. Note that computing the filtered output by convolution method is an $O(N^2)$ operation, since we need to pointwise multiply to sequence of length N. Using FFTs, however, we need two DFTs and one IDFT, all on some sequence of length N, thus taking $O(N \log N)$ time. The pointwise product only takes N operations, dominated by $O(N \log N)$. Thus, the use of FFT greatly improves the efficiency of *digital filtering of signals*.

5/3 Lecture

11 Iterative Methods

11.1 Krylov Subspace

Let A be an $n \times n$ invertible matrix and suppose we want to solve $A\vec{x} = \vec{b}$, but we only knew A via matrix-vector products, *i.e.*, a mapping $\vec{v} \to A\vec{v}$. Let us consider the sequence of vectors $\vec{b}, A\vec{b}, \cdots, A^{n-1}\vec{b}, A^n\vec{b}, \cdots$. If we consider the first $n+1$ of them, then we are guaranteed for these vectors to be linearly dependent since our space is only n-dimensional. Therefore, there exist coefficients $\alpha_0, \cdots, \alpha_n$ (not all zero), such that

$$\alpha_0\vec{b} + \alpha_1 A\vec{b} + \cdots + \alpha_{n-1}A^{n-1}\vec{b} + \alpha_n A^n\vec{b} = \vec{0}. \tag{368}$$

Let k be the smallest integer such that $\alpha_k \neq 0$, then since A^{-1} exists, we can write that

$$A^{-1}\vec{b} = -\frac{1}{\alpha_k}\left(\alpha_{k+1}\vec{b} + \cdots + \alpha_n A^{n-k-1}\vec{b}\right). \tag{369}$$

This is also known as the *weak Cayley-Hamilton theorem*. This shows that $\vec{x} = A^{-1}\vec{b}$ can be computed by only matrix-vector products, giving the idea to search for fast solutions from *Krylov subspaces*.

Definition 11.1. Let A be a matrix and $\vec{c}$ a vector. The *rth Krylov subspace*, denoted by $\mathscr{K}_r(A, \vec{c})$, is the vector space spanned by

$$\vec{c}, A\vec{c}, \cdots, A^{r-1}\vec{c}. \tag{370}$$

Usually, though not necessarily, we set $\vec{c}$ to be the right-hand side in the linear system $A\vec{x} = \vec{b}$.

11.2 Generalized Minimum Residual Method (GMRES)

The GMRES is a Krylov subspace method that computes the rth step the best least squares solution $\vec{x}_r$ from the Krylov subspace $\mathscr{K}_r(A, \vec{b})$. That is, the GMRES method successively solves the following least squares problems.

$$\text{Step 1} \qquad \min_{\vec{x}_1 \in \mathscr{K}_1(A,\vec{b})} \|\vec{b} - A\vec{x}_1\|_2, \tag{371}$$

$$\text{Step 2} \qquad \min_{\vec{x}_2 \in \mathscr{K}_2(A,\vec{b})} \|\vec{b} - A\vec{x}_2\|_2, \tag{372}$$

$$\vdots \qquad\qquad \vdots$$

$$\text{Step } r \qquad \min_{\vec{x}_r \in \mathscr{K}_r(A,\vec{b})} \|\vec{b} - A\vec{x}_r\|_2. \tag{373}$$

These problems may seem to be awkward to solve at first because we do not know how to find the least squares solution over Krylov subspaces. However, if the columns of Q_r form an orthonormal basis for $\mathscr{K}_r(A, \vec{b})$, then we can equivalently write the GMRES method as

$$\text{Step 1} \qquad \min_{\vec{c} \in \mathbb{R}} \|\vec{b} - AQ_1\vec{c}\|_2, \qquad \vec{x}_1 = Q_1\vec{c}, \tag{374}$$

$$\text{Step 2} \qquad \min_{\vec{c} \in \mathbb{R}^2} \|\vec{b} - AQ_2\vec{c}\|_2, \qquad \vec{x}_2 = Q_2\vec{c}, \tag{375}$$

$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$

$$\text{Step } r \qquad \min_{\vec{c} \in \mathbb{R}^r} \|\vec{b} - AQ_r\vec{c}\|_2, \qquad \vec{x}_r = Q_r\vec{c}. \tag{376}$$

Therefore, the rth GMRES step should find the least squares solution to $AQ_r\vec{c} = \vec{b}$. We do this by using the QR factorization of AQ_r, *i.e.*, $AQ_r = \tilde{Q}_r\tilde{R}_r$. Thus, we can rewrite GMRES again as the following steps.

$$\begin{array}{llll} \text{Step 1} & \text{Solve } \tilde{R}_1\vec{c} = \tilde{Q}_1^\top\vec{b}, & \tilde{Q}_1\tilde{R}_1 = AQ_1, & \vec{x}_1 = Q_1\vec{c}, \end{array} \tag{377}$$

$$\begin{array}{llll} \text{Step 2} & \text{Solve } \tilde{R}_2\vec{c} = \tilde{Q}_2^\top\vec{b}, & \tilde{Q}_2\tilde{R}_2 = AQ_2, & \vec{x}_2 = Q_2\vec{c}, \end{array} \tag{378}$$

$$\begin{array}{llll} \vdots & \vdots & \vdots & \end{array}$$

$$\begin{array}{llll} \text{Step } r & \text{Solve } \tilde{R}_r\vec{c} = \tilde{Q}_r^\top\vec{b}, & \tilde{Q}_r\tilde{R}_r = AQ_r, & \vec{x}_r = Q_r\vec{c}. \end{array} \tag{379}$$

Convergence of GMRES. We begin with two observations. Let $\vec{r}_n$ denote the residual, then GMRES converges monotonically, *i.e.*, $\|\vec{r}_{n+1}\| \leq \|\vec{r}_n\|$. The reason is that $\|\vec{r}_n\|$ is as small as possible for the subspace $\mathscr{K}_n$, so by enlarging $\mathscr{K}_n$ to $\mathscr{K}_{n+1}$, we can only decrease the residual norm or at worst leave it unchanged. Suppose that A is an $m \times m$ matrix, then the second observation is that after m steps, the GMRES must converge, at least in the absence of rounding errors, $\|\vec{r}_m\| = 0$. This is because $\mathscr{K}_m = \mathbb{R}^m$, and in special cases the convergence would arrive even earlier.

Now we would like to know how many iterations of GMRES do we require to achieve a particular tolerance. A useful exercise is to translate the GMRES minimization problem to an external problem in polynomial approximation. By doing this, we are able to understand the convergence better from intuition on polynomials. A simple derivation shows that

$$\min_{\vec{x_r} \in \mathscr{K}_r(A,\vec{b})} \|\vec{b} - A\vec{x}_r\|_2 = \min_{\vec{c} \in \mathbb{R}^r} \|\vec{b} - (c_1 A\vec{b} + c_2 A^2\vec{b} + \cdots + c_r A^r\vec{b})\|_2 = \min_{p \in \mathscr{P}_r, p(0)=1} \|p(A)\vec{b}\|_2. \tag{380}$$

Theorem 11.2. Let A be a diagonalizable matrix such that $A = V^{-1}\Lambda V$. Then

$$\min_{\vec{x}_r \in \mathscr{K}_r(A,\vec{b})} \|\vec{b} - A\vec{x}_r\|_2 \leq \underbrace{\|V\|_2\|V^{-1}\|_2}_{=K(V)} \|\vec{b}\|_2 \min_{p \in \mathscr{P}_r, p(0)=1} \sup_{z \in \mathrm{diag}(\Lambda)} |p(z)|. \tag{381}$$

Proof. The proof is simple by observing that

$$\|p(A)\|_2 \leq \|V\|_2\|p(\Lambda)\|_2\|V^{-1}\|_2 = K(V) \sup_{z \in \mathrm{diag}(\Lambda)} |p(z)|, \tag{382}$$

and details will be ignored here. □

Here we can expect rapid convergence of GMRES if the eigenvector matrix of A is well conditioned (so that $K(V)$ is small), or if the eigenvalues of A are clustered and far from the origin (so that properly normalized degree n polynomials can be found whose size on the spectrum of A, *i.e.*, $\mathrm{diag}(\Lambda)$ decreases quickly with n).

Topics Recap

- Floating point arithmetic
- Solving nonlinear equations (bisection, secant, and Newton; their convergence rates)
- Solving systems of nonlinear equations (multivariate Newton)
- Numerical linear algebra (vector and matrix norms; condition number and SVD; optimization methods, convexity, and quasi-Newton methods)
- Solving $A\vec{x} = \vec{b}$ (Gaussian elimination, LU factorization, and Cholesky decomposition; their flop counts $O(n^3)$; pivoting in the cases where LU factorization may fail)
- QR decomposition (least-squares problems; Gram-Schmidt process; Householder reflections)
- Eigenvalues and eigenvectors (inverse power method with shift; Jacobi's method *symmetric*; *QR algorithm will not appear in final*)

- Krylov methods (Krylov subspace; GMRES *just a least-squares problem*)
- Function approximation (interpolation in Lagrange form and the Runge phenomenon; minimax approximation; Chebyshev interpolation, Chebyshev polynomials and computing their roots; 2-norm approximation; orthogonal polynomials)
- Numerical integration (Trapezoidal rule and its order of approximation; Gaussian quadrature; Clenshaw-Curtis quadrature and its concept; Richardson extrapolation)
- Fourier analysis (DFT and IDFT; FFT, its splitting, and its complexity; spectral differentiation and integration; convolutions using the FFT)

Last Modified: May 10, 2023.