University of
Cambridge

Mathematics Tripos

Part IB

Numerical Analysis

Lent, 2018

Lectures by
H. Fawzi

Notes by
Qiangru Kuang

Contents

0 Introduction

Numerical analysis is the study of *algorithms* for continuous mathematics. Examples of problems in continuous mathematics are:

- solve $f(x) = 0$ where $f : \mathbb{R}^n \to \mathbb{R}$,
- solve $\frac{dx}{dt} = f(x)$ where $f : \mathbb{R}^n \to \mathbb{R}^n$
- optimisation: find $\min f(x)$ where $x \in \mathbb{R}^n, f : \mathbb{R}^n \to \mathbb{R}$.

A note on complexity: we measure the complexity of an algorithm by the number of *elementary operations* $(+, \times, -, /)$ it needs.

Big O notation: for example $O(n), O(n^2)$, where n is input size. We also have complexity $O(f(n))$ if the number of operations is at most $cf(n)$ where $c > 0$.

1 Polynomial Interpolations

Denote a degree n polynomial by

$$p(x) = p_0 + p_1 x + \cdots + p_n x^n$$

and let $P_n[x]$ be the vector space of polynomials of degree at most n. The interpolation problem is, given $x_0, x_1, \ldots, x_n \in \mathbb{R}$ and $f_0, f_1, \ldots, f_n \in \mathbb{R}$, find $p \in P_n[x]$ such that $p(x_i) = f_i$ for $i = 0, \ldots, n$.

1.1 Lagrange formula

Claim that

$$p(x) = \sum_{k=0}^{n} f_k \underbrace{\prod_{\ell \neq k} \frac{x - x_\ell}{x_k - x_\ell}}_{L_k(x)}$$

solves the problem.

Note that $L_k(x_j) = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases}$ and the result easily follows.

Now we prove the uniqueness of the solution. Assume $q \in P_n[x]$ is another polynomial that interpolates the data. Then $p - q$ has $n+1$ zeros. But a non-zero polynomial in $P_n[x]$ has at most n zeros. Thus $p - q$ must be zero.

This is an easy solution but what is its complexity? For each k, the complexity of evaluating $L_k(x)$ is $O(n)$ so the total complexity of evaluating $p(x)$ is $O(n^2)$.

1.1.1 Error of polynomial interpolation

Let $C^s[a, b]$ be the space of functions $[a, b] \to \mathbb{R}$ that are s times continuously differentiable.

Theorem 1.1. *Let $f \in C^{n+1}[a, b]$ and let $p \in P_n[x]$ interpolate f at distinct $x_0, \ldots x_n$, i.e. $p(x_i) = f(x_i)$ for $i = 0, \ldots, n$. Then for all $x \in [a, b]$, there exists $\xi \in [a, b]$ such that*

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

The last term $\prod_{i=0}^{n}(x - x_i)$ is called the *nodal polynomial*.

Proof. If $x = x_i$ for some i then the result is trivially true. Assume x is distinct from x_i's for all i. Define

$$\varphi(t) = f(t) - \left(p(t) + (f(x) - p(x)) \prod_{i=0}^{n} \frac{t - x_i}{x - x_i} \right).$$

Note that the second term is by construction the interpolating polynomial of f at $x_0, \ldots, x_n$ and x. Thus

$$\varphi(x_0) = \varphi(x_1) = \cdots = \varphi(x_n) = \varphi(x) = 0$$

so φ has $n+2$ zeros. Recall from IA Analysis I Rolle's Theorem: if $g \in C^1[a,b]$ such that $g(a) = g(b)$ then there exists $\alpha \in (a,b)$ such that $g'(\alpha) = 0$. Apply it to φ, we deduce that φ' has $n+1$ zeros. Inductively we find that $\varphi^{(n+1)}$ has 1 zero, i.e. there exists $\xi \in [a,b]$ such that $\varphi^{(n+1)}(\xi) = 0$. Thus

$$0 = \varphi^{(n+1)}(\xi) = f^{(n+1)}(\xi) - \left(\underbrace{p^{(n+1)}(\xi)}_{=0} + (f(x) - p(x)) \frac{(n+1)!}{\prod_{i=0}^n (x - x_i)} \right).$$

Rearrange,

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

□

Example. $[a,b] = [-5,5]$, $x_j = -5 + 10\frac{j}{n}$ for $j = 0, \dots, n$. Plot $\prod_{i=0}^n (x - x_i)$, we note that it vanishes at x_i's but blows up near the endpoints.

This is called *Runge's phenomenon*. If one attempts to interpolate $f(x) = \frac{1}{1+x^2}$ using equispaced points on $[-5,5]$ and plots the error $f(x) - p(x)|$.

Thus equispaced points may not be the most suitable to minimise the error. The remedy is to look for points $x_0, \dots, x_n$ such that $|\prod_{i=0}^n (x - x_i)|$ is small. This leads us to *Chebyshev points*. For example in the previous case we should choose

$$x_j = 5 \cos \frac{(n-j)\pi}{n}.$$

This choice of points is the one that minimises

$$\max_{x \in [a,b]} \left| \prod_{i=0}^n (x - x_i) \right|.$$

1.2 Divided difference and Newton's interpolation formula

Definition (Divided difference)**.** Given pairwise distinct points $x_0, \dots, x_n$, let $p \in P_n[x]$ interpolate $f \in C[a,b]$ at these points. The coefficient of x^n in p is called the *divided difference* of f at $(x_0, \dots, x_n)$ and is denoted $f[x_0, \dots, x_n]$.

We know from the last lecture that the interpolent is

$$p(x) = \sum_{k=0}^n f(x_k) \prod_{\ell \neq k} \frac{x - x_\ell}{x_k - x_\ell}.$$

Thus we get

$$f[x_0, \dots, x_n] = \sum_{k=0}^n f(x_k) \prod_{\ell \neq k} \frac{1}{x_k - x_\ell}.$$

The next example illustrates the reason behind the name "divided difference".

Example.

- $f[x_0] = f(x_0)$.
- $f[x_0, x_1] = f(x_0)\frac{1}{x_0 - x_1} + f(x_1)\frac{1}{x_1 - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$.

This observation is generalised by

Theorem 1.2.

$$f[x_0, x_1, \dots, x_{n+1}] = \frac{f[x_1, \dots, x_{n+1}] - f[x_0, \dots, x_n]}{x_{n+1} - x_0}.$$

Proof. Let $p \in P_n[x]$ interpolate f at $x_0, \dots, x_n$ and $q \in P_n[x]$ interpolate f at $x_1, \dots, x_{n+1}$ and let

$$r(x) = \frac{x - x_{n+1}}{x_0 - x_{n+1}} p(x) + \frac{x - x_0}{x_{n+1} - x_0} q(x) \in P_{n+1}[x].$$

Observe that r interpolates f at $x_0, \dots, x_{n+1}$. So the divided difference, i.e. coefficient of x^{n+1} in $r(x)$ is

$$\frac{1}{x_0 - x_{n+1}} f[x_0, \dots, x_n] + \frac{1}{x_{n+1} - x_0} f[x_1, \dots, x_{n+1}].$$

□

Theorem 1.3. *Assume $x_0, \dots, x_n$ are pairwise distinct in $[a, b]$ and $f \in C^n[a, b]$, then there exists $\xi \in [a, b]$ such that*

$$f[x_0, \dots, x_n] = \frac{1}{n!} f^{(n)}(\xi).$$

Proof. Let $p \in P_n[x]$ interpolate f at $x_0, \dots, x_n$. Then function $f - p$ has $n + 1$ zeros in $[a, b]$. Applying Rolle's Theorem n times, we get that $(f - p)^{(n)}$ has at least one zero in $[a, b]$. Let $\xi \in [a, b]$ be such that $f^{(n)}(\xi) - p^{(n)}(\xi) = 0$. As p has degree n, $p^{(n)}(x) = n! \cdot$ leading coefficient. The result thus follows. □

Theorem 1.4 (Newton's interpolation formula)**.** *Let $x_0, \dots, x_n$ be pairwise distinct and let*

$$p(x) = f[x_0] + f[x_0, x_1](x - x_0) + \cdots + f[x_0, \dots, x_n] \prod_{i=0}^{n-1} (x - x_i).$$

Then $p(x_i) = f(x_i)$ for $i = 0, \dots, n$.

Proof. Induction on n. If $n = 0$ then $p(x) = f[x_0]$ which trivially satisfies $p(x_0) = f[x_0] = f(x_0)$.

Suppose it holds for $n-1$. Let $p_n \in P_{n-1}[x]$ interpolate f at $x_0, \dots, x_{n-1}$ and $p \in P_n[x]$ interpolate f at $x_0, \dots, x_{n-1}, x_n$. $p - p_n$ is a polynomial of degree n that vanishes at $x_0, \dots, x_{n-1}$, implying that

$$p - p_n = \alpha \prod_{i=0}^{n-1} (x - x_i).$$

But α is the coefficient of x^n in p and so we must have $\alpha = f[x_0, \dots, x_n]$. Thus we deduce that

$$p = p_n + f[x_0, \dots, x_n] \prod_{i=0}^{n-1} (x - x_i).$$

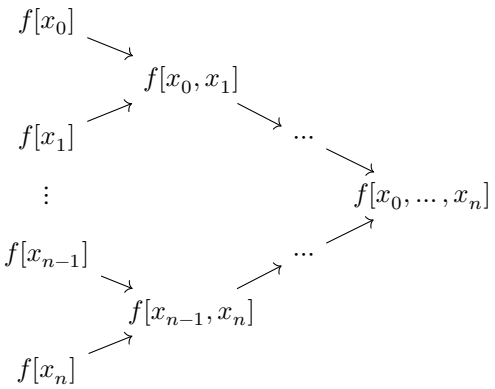
By using induction hypothesis, the result follows. □

1.2.1 Evaluating Newton's interpolating formula

Recall that the divided difference is defined recursively by

$$f[x_0, x_1, \dots, x_{n+1}] = \frac{f[x_1, \dots, x_{n+1}] - f[x_0, \dots, x_n]}{x_{n+1} - x_0}.$$

We can draw a table to evaluate them:



The output of this procedure is $f[x_j, x_{j+1}, \dots, x_\ell]$ for all $0 \le j \le \ell \le n$. To evaluate a new divided difference requires 3 operations (2 substractions and 1 division) so the complexity is

$$3(n-1) + 3(n-2) + \cdots + 3 \approx 3 \cdot \frac{n^2}{2} \sim O(n^2).$$

Having obtained the divided differences, we can use Horner's scheme to evaluate Newton's formula in $O(n)$. Note that the term $(x - x_0)$, for example, appears in every term except the first one in Newton's interpolating formula so we may group them together

$$\begin{aligned} p(x) = f[x_0] + (x - x_0)(f[x_0, x_1] + (x - x_1)(f[x_0, x_1, x_2] + \cdots \\ + (x - x_{n-1}) f[x_0, x_1, \dots, x_n])). \end{aligned}$$

This way the evaluation requires $O(n)$ operations.

2 Orthogonal Polynomials

2.1 Definitions

Definition (Inner product)**.** An *inner product* is a function $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$ where V is a real vector space which is

1. symmetric: for all $x, y \in V$, $\langle x, y \rangle = \langle y, x \rangle$,
2. positive-definite: for all $x \in V$, $\langle x, x \rangle \geq 0$ with equality if and only if $x = 0$,
3. linear: for all $x, y, z \in V$, $a, b \in \mathbb{R}$, $\langle ax + by, z \rangle = a\langle x, z \rangle + b\langle y, z \rangle$.

Definition (Orthogonality)**.** Given an inner product, $x, y \in V$ are *orthogonal* if $\langle x, y \rangle = 0$.

In this section, let $V = C[a, b]$ and let $w \in V$ be a positive function and define an inner product

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

It is easy to verify the axioms of inner product.

Definition (Orthogonal polynomial)**.** Given an inner product in $V = P[x]$, the *orthogonal polynomials* are a sequence of polynomials $p_0, p_1, ...$ such that

1. $\deg p_n = n$ for all $n \geq 0$ and
2. $\langle p_n, p_m \rangle = 0$ for all $n \neq m$.

Remark. For any $n \geq 0$, $\{p_0, \dots, p_n\}$ is an orthogonal basis of $P_n[x]$.

The next theorem shows that this object with the desired property does exist:

Theorem 2.1. *For every $n \geq 0$ there exists a unique monic orthogonal polynomial p_n of degree n.*

Proof. Let $p_0(x) = 1$ and proceed by induction on n. Suppose $p_0, \dots, p_n$ satisfy the induction hypothesis. To define p_{n+1} let $q(x) = x^{n+1} \in P_{n+1}[x]$ and conduct Gram-Schmidt on $q(x)$:

$$p_{n+1}(x) = q(x) - \sum_{k=0}^{n} \frac{\langle q, p_k \rangle}{\langle p_k, p_k \rangle} p_k(x).$$

Clearly $p_{n+1} \in P_{n+1}[x]$ and is monic. The orthogonality is a consequence of Gram-Schmidt.

To prove uniqueness, suppose there exists $\tilde{p}_{n+1} \in P_{n+1}[x]$ also monic orthogonal. Then $p = p_{n+1} - \tilde{p}_{n+1} \in P_n[x]$ and thus we have

$$0 = \langle p_{n+1}, p \rangle - \langle \tilde{p}_{n+1}, p \rangle = \langle p, p \rangle$$

so $p = 0$. □

Example (Legendre polynomials)**.** Define an inner product

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

for $f, g \in P[x]$. The orthogonal polynomials arising from the scalar product is called *Legendre polynomials*. The first few terms in the sequence are

$$\begin{aligned} p_0(x) &= 1 \\ p_1(x) &= x \\ p_2(x) &= x^2 - \frac{1}{3} \\ p_3(x) &= x^3 - \frac{3}{5}x \\ f_4(x) &= x^4 - \frac{30}{35}x^2 + \frac{3}{35} \end{aligned}$$

Some well-known examples of orthogonal polynomials:

Name	Notation	$[a, b]$	$w(x)$
Legendre	P_n	$[-1, 1]$	1
Chebyshev	T_n	$[-1, 1]$	$(1 - x^2)^{-1/2}$
Laguerre	L_n	$[0, \infty)$	e^{-x}
Hermite	H_n	$(-\infty, \infty)$	e^{-x^2}

Table 1: Common orthogonal polynomials

2.2 Three-term recurrence relation

Gram-Schmidt gives us a way to construct orthogonal polynomials, but it suffers from loss of accuracy due to imprecisions in the calculation of scalar products. A considerably better procedure follows from our next theorem, which relies on the mild assumption that the inner product satisfies

$$\langle xp, q\rangle = \langle p, xq\rangle$$

for all $p, q \in P[x]$.

Theorem 2.2. *Assume the scalar product on $P[x]$ satisfies $\langle xp, q\rangle = \langle p, xq\rangle$ for all $p, q \in P[x]$. The orthogonal polynomials are given by*

$$\begin{aligned} p_{-1}(x) &= 0 \\ p_0(x) &= 1 \\ p_{n+1}(x) &= (x - \alpha_n)p_n(x) - \beta_n p_{n-1}(x),\ n \geq 0 \end{aligned}$$

where

$$\alpha_n = \frac{\langle p_n, xp_n\rangle}{\langle p_n, p_n\rangle}$$
$$\beta_n = \frac{\langle p_n, p_n\rangle}{\langle p_{n-1}, p_{n-1}\rangle}$$

Proof. Let $n \geq 0$ and $\psi(x) = p_{n+1}(x) - (x-\alpha_n)p_n(x) + \beta_n p_{n-1}(x)$. Since p_n and p_{n+1} are monic, it follows that $\psi \in P_n[x]$. Moreover, because of orthogonality of p_{n-1}, p_n, p_{n+1},

$$\langle \psi, p_\ell\rangle = 0$$

for $\ell = 0, \ldots, n-2$ by linearity. Because of monicity, $q = xp_{n-1} - p_n \in P_{n-1}[x]$. Thus from the definition of α_n and β_n,

$$\langle \psi, p_{n-1}\rangle = -\langle p_n, xp_{n-1}\rangle + \beta_n\langle p_{n-1}, p_{n-1}\rangle = -\langle p_n, p_n\rangle + \beta_n\langle p_{n-1}, p_{n-1}\rangle = 0$$
$$\langle \psi, p_n\rangle = -\langle xp_n, p_n\rangle + \alpha_n\langle p_n, p_n\rangle = 0$$

Every $p \in P_n[x]$ that obeys $\langle p, p_\ell\rangle = 0$ for all $0 \leq \ell \leq n$ must necessarily be the zero polynomials. Thus $\psi = 0$ and the result follows. □

Example (Chebyshev polynomials)**.** Define an inner product on $C[-1, 1]$

$$\langle f, g\rangle = \int_{-1}^{1} \frac{f(x)g(x)}{\sqrt{1-x^2}}dx$$

and define

$$T_n \in P_n[x]$$
$$T_n(\cos\theta) = \cos(n\theta)$$

The first three terms are thus

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_2(x) = 2x^2 - 1$$

The inner product can be easily found by a change of vairable:

$$\begin{aligned}\langle T_n, T_m\rangle &= \int_{-1}^{1} \frac{T_n(x)T_m(x)}{\sqrt{1-x^2}}dx \\ &= \int_0^\pi \cos n\theta \cos m\theta d\theta \\ &= \frac{1}{2}\int_0^\pi (\cos(n+m)\theta + \cos(n-m)\theta)d\theta \\ &= 0\end{aligned}$$

whenever $n \neq m$.

The recurrence relation for Chebyshev polynomials is particularly simple:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

which can be verified at once from the identity

$$\cos(n+1)\theta + \cos(n-1)\theta = 2\cos\theta\cos(n\theta).$$

Note that T_n's are not monic. To obtain monic polynomials take $T_n/2^{n-1}$ for $n \geq 1$.

2.3 Least-squares polynomial fitting

Question. Fix a scalar product on $C[a,b]$ of the form

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

where $w : [a,b] \to \mathbb{R}$ is positive. Given $f \in C[a,b]$, find $p \in P_n[x]$ such that

$$\|f-p\|^2 = \langle f-p, f-p\rangle$$

is minimised.

Let $\widehat{p}_n$ be the polynomial of degree n minimising $\|f-p\|^2$.

Theorem 2.3. *Let p_0, p_1, p_2 be orthogonal polynomials with respect to $\langle\cdot,\cdot\rangle$. Then*

$$\widehat{p}_n = \sum_{k=0}^{n} \frac{\langle f, p_k\rangle}{\langle p_k, p_k\rangle} p_k.$$

Proof. We know that any $p \in P_n[x]$ can be written as

$$p = \sum_{i=0}^{n} c_k p_k$$

where $c_0, \ldots, c_n \in \mathbb{R}$. Then

$$\begin{aligned}
\langle f-p, f-p\rangle &= \left\langle f - \sum_{k=0}^{n} c_k p_k, f - \sum_{k=0}^{n} c_k p_k \right\rangle \\
&= \langle f, f\rangle - 2\left\langle f, \sum_{k=0}^{n} c_k p_k \right\rangle + \left\langle \sum_{k=0}^{n} c_k p_k, \sum_{k=0}^{n} c_k p_k \right\rangle \\
&= \langle f, f\rangle - 2\left\langle f, \sum_{k=0}^{n} c_k p_k \right\rangle + \sum_{k=0}^{n}\sum_{k'=0}^{n} c_k c_{k'} \underbrace{\langle p_k, p_k\rangle}_{=0 \text{ if } k\neq k'} \\
&= \langle f, f\rangle - 2\left\langle f, \sum_{k=0}^{n} c_k p_k \right\rangle + \sum_{k=0}^{n} c_k^2 \langle p_k, p_k\rangle
\end{aligned}$$

As usual, to minimise the quantity we take derivative with respect to c_i's:

$$\frac{\partial}{\partial c_k}\langle f-p, f-p\rangle = -2\langle f, p_k\rangle + 2c_k\langle p_k, p_k\rangle.$$

Set the derivative to 0, we get

$$c_k = \frac{\langle f, p_k\rangle}{\langle p_k, p_k\rangle}.$$

□

In this case, the minimal error is

$$\|f-\hat{p}_n\|^2 = \langle f-\hat{p}_n, f-\hat{p}_n\rangle = \langle f,f\rangle - \sum_{k=0}^{n}\frac{\langle f,p_k\rangle^2}{\langle p_k,p_k\rangle} = \langle f,f\rangle - \langle \hat{p}_n,\hat{p}_n\rangle.$$

The can be interpreted as the generalised Pythagoras Theorem for inner product spaces.

It is obvious that $\|f-\hat{p}_n\|^2$ is a non-increasing function of n, but

Question. Does $\|f-\hat{p}_n\|^2 \to 0$ as $n\to\infty$?

The answer is yes and can be proved using Weierstrass theorem, which we state without giving a proof:

Theorem 2.4 (Weierstrass)**.** *Let $f\in C[a,b]$ where $[a,b]$ is bounded. For all $\varepsilon>0$, there exists a polynomial p of high enough degree such that*

$$|f(x)-p(x)|<\varepsilon$$

for all $x\in[a,b]$.

Using Weierstrass theorem, we can prove the claim above. For any p we have

$$\|f-p\|^2 = \int_a^b w(x)(f(x)-p(x))^2dx \le \left(\max_{x\in[a,b]}|f(x)-p(x)|\right)^2\int_a^b w(x)d(x).$$

Give any $\delta>0$, by Weierstrass theorem applied with $\varphi=\sqrt{\delta/\int_a^b w(x)dx}$, there is a polynomial p of degree N such that

$$|f(x)-p(x)|<\varepsilon$$

for all $x\in[a,b]$. Then for any $N\ge n$,

$$\|f-\hat{p}_N\|^2 \le \|f-p\|^2 \le \varepsilon^2\int_a^b w(x)dx = \delta$$

as required.

Theorem 2.5 (Parseval identity)**.**

$$\sum_{k=0}^{\infty}\frac{\langle f,p_k\rangle^2}{\langle p_k,p_k\rangle} = \langle f,f\rangle.$$

Proof. Reformulation of the above claim. □

2.4 Least-squares fitting to discrete data

Given the value of a function at pairwise distinct points $x_1,\dots,x_m$, the goal of this section is to find $p\in P_n[x]$ that minimises

$$\sum_{k=1}^{m}(f(x_k)-p(x_k))^2.$$

This can be thought as a generalisation of the interpolation problem, where $m = n + 1$ and the minimised error is always 0. In the generalised setting, $n \leq m - 1$ (and usually much smaller).

This question can be reformulated as in terms of orthogonal polynomial, where the inner product is defined to be

$$\langle g, h \rangle = \sum_{k=1}^{m} g(x_k)h(x_k)$$

since then

$$\langle f - p, f - p \rangle = \sum_{k=0}^{n} (f(x_k) - p(x_k))^2.$$

We can proceed as follow, given $n \leq m - 1$:

1. use three-term recurrence for the inner product to compute orthogonal polynomials $p_0, \dots, p_n$.
2. Form
$$\hat{p}_n = \sum_{k=0}^{n} \frac{\langle f, p_k \rangle}{\langle p_k, p_k \rangle} p_k.$$

The reason we require $n \leq m - 1$ is for positive definiteness.

2.5 Gaussian quadrature

Question. Let $w : [a, b] \to \mathbb{R}$ be positive. We want to approximate the integral $\int_a^b w(x)f(x)dx$ where $f \in C[a, b]$.

A *quadrature formula* is an approximation of the above expression

$$\int_a^b w(x)f(x)dx \approx \sum_{k=1}^{\nu} b_k f(c_k) \qquad (*)$$

where b_k are *weights* and c_k are *nodes*.

We require the quadrature formula to be exact for $f \in P_n[x]$.

Claim that if the quadrature formula (*) is exact for any $f \in P_n[x]$ then necessarily $n \leq 2\nu - 1$.

Proof. We will construct a polynomial of degree 2ν such that LHS and RHS are different. Let

$$f(x) = \prod_{k=1}^{\nu} (x - c_k)^2$$

which has degree 2ν. RHS is equal to 0 while LHS is $\int_a^b w(x)f(x)dx > 0$. □

Can we find a quadrature formula that is exact for all polynomials of degree $\leq 2\nu - 1$? Yes, and this is *Gaussian quadrature*.

Let

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

and let p_0, p_1, p_2 be orthogonal polynomials associated to this inner product.

Theorem 2.6. *For any $n \geq 0$, p_n has n real distinct roots in $[a, b]$.*

Proof. Let $\xi_1, \dots, \xi_m \in (a, b)$ be the points where p_n changes sign and let

$$q(x) = \prod_{i=1}^{m} (x - \xi_i).$$

Note that $p_n(x)q(x)$ has constant sign in $[a, b]$, i.e. in this step we make every root have even multiplicity. Since $p_n q$ has constant sign on $[a, b]$,

$$|\langle p_n, q \rangle| = \left| \int_a^b w(x) p_n(x) q(x) dx \right| = \int_a^b w(x) |p_n(x) q(x)| dx > 0$$

This implies that $\deg q \geq n$ so $m \geq n$. But $m \leq n$ because p_n has degree n. Thus p_n has n distinct real roots in (a, b). □

Given nodes $c_1, \dots, c_\nu$, define the *interpolatory weights*

$$b_k = \int_a^b w(x) \prod_{j=1, j \neq k}^{\nu} \frac{x - c_j}{c_k - c_j} dx \qquad (**)$$

for $k = 1, \dots, \nu$.

Theorem 2.7.

1. *The quadrature formula with weights given by (**) is exact for polynomials of degree up to $\nu - 1$.*
2. *If furthermore the c_k's are the roots of p_ν, then the quadrature formula is exact for polynomials up to degree $2\nu - 1$.*

Proof.

1. Let $f \in P_{\nu-1}[x]$. Write f in terms of its interpolating formula at the c_k's,

$$f(x) = \sum_{k=1}^{\nu} f(c_k) \prod_{j \neq k} \frac{x - c_j}{c_k - c_j}.$$

Then

$$\begin{aligned} \int_a^b w(x) f(x) dx &= \int_a^b w(x) \sum_{k=1}^{\nu} f(c_k) \prod_{j \neq k} \frac{x - c_j}{c_k - c_j} dx \\ &= \sum_{k=1}^{\nu} f(c_k) \int_a^b w(x) \prod_{j \neq k} \frac{x - c_j}{c_k - c_j} dx \end{aligned}$$

2. Assume now that $c_1, \dots, c_\nu$ are the roots of p_ν. Let $f \in P_{2\nu-1}[x]$. We can write

$$f = p_\nu q + r$$

with $\deg r \leq \nu - 1, \deg q \leq \nu - 1$. Then

$$\begin{aligned}\int_a^b w(x)f(x)dx &= \int_a^b w(x)p_\nu(x)q(x)dx + \int_a^b w(x)r(x)dx \\ &= \langle p_\nu, q\rangle + \sum_{k=1}^{\nu} b_k r(c_k) \\ &= 0 + \sum_{k=1}^{\nu} b_k f(c_k)\end{aligned}$$

□

2.6 Peano Kernel Theorem

How does the error behave when we use the quadrature formula to approximate a function that is not a polynomial, or a polynomial of higher degree?

The error from a quadrature formula is

$$L(f) = \int_a^b w(x)f(x)dx - \sum_{k=1}^{\nu} b_k f(c_k)$$

Assume $L(f) = 0$ for all $f \in P_n[x]$. The goal is to bound error $|L(f)|$ for $f \in C^{n+1}[a, b]$. Recall from IA Analysis I the Taylor expansion formula with integral remainder

$$f(x) = \sum_{i=0}^{n} f^{(i)}(a)\frac{(x-a)^i}{i!} + \frac{1}{n!}\int_a^x (x-\theta)^n f^{(n+1)}(\theta)d\theta.$$

Call the first part $g(x)$. Since $g \in P_n[x]$, $L(g) = 0$. Thus by linearity of L we get

$$L(f) = L\left(x \mapsto \frac{1}{n!}\int_a^x (x-\theta)^n f^{(n+1)}(\theta)d\theta\right).$$

Let

$$(x-\theta)_+^n = \begin{cases}(x-\theta)^n & \text{if } \theta \leq x \\ 0 & \text{otherwise}\end{cases}$$

Then

$$\begin{aligned}\int_a^x (x-\theta)^n f^{(n+1)}(\theta)d\theta &= \int_a^b \mathbb{1}_{\theta\leq x}(x-\theta)^n f^{(n+1)}(\theta)d\theta \\ &= \int_a^b (x-\theta)_+^n f^{(n+1)}(\theta)d\theta\end{aligned}$$

Assuming we can exchange L and the integral, we get

$$L(f) = \frac{1}{n!}\int_a^b K(\theta)f^{(n+1)}(\theta)dx$$

where

$$K(\theta) = L(x \mapsto (x-\theta)_+^n)$$

is the *Peano kernel*.

Example (Simpson's rule)**.**

$$\int_{-1}^{1} f(x)dx \approx \frac{1}{3}[f(-1) + 4f(0) + f(1)].$$

Then

$$L(f) = \int_{-1}^{1} f(x)dx - \frac{1}{3}[f(-1) + 4f(0) + f(1)].$$

We can check that $L(f) = 0$ for all $f \in P_2[x]$. The Peano kernel for L is

$$K(\theta) = L(x \mapsto (x-\theta)_+^2) = \begin{cases} -\frac{1}{3}\theta(1+\theta)^2 & \text{if } -1 \le \theta \le 0 \\ -\frac{1}{3}\theta(1-\theta)^2 & \text{if } 0 \le \theta \le 0 \\ 0 & \text{otherwise} \end{cases}$$

Assume $0 \le \theta \le 1$ for example,

$$\begin{aligned} K(\theta) &= L(x \mapsto (x-\theta)_+^2) \\ &= \int_{-1}^{1} (x-\theta)_+^2 dx - \frac{1}{3}\left[(-1-\theta)_+^2 + 4(0-\theta)_+^2 + (1-\theta)_+^2\right] \\ &= \int_{\theta}^{1} (x-\theta)^2 dx - \frac{1}{3}[0 + 0 + (1-\theta)^2] \\ &= \frac{(x-\theta)^3}{3}\Big|_0^1 - \frac{1}{3}(1-\theta)^2 \\ &= -\frac{1}{3}\theta(1-\theta)^2 \end{aligned}$$

For any $f \in C^3[-1,1]$,

$$\begin{aligned} |L(f)| &= \frac{1}{2}\left|\int_{-1}^{1} K(\theta) f^{(3)}(\theta) d\theta\right| \\ &\le \frac{1}{2} \max_{\theta \in [-1,1]} |f^{(3)}(\theta)| \int_{-1}^{1} |K(\theta)| d\theta \\ &\le \frac{1}{36} \|f^{(3)}(\theta)\|_\infty \end{aligned}$$

In particular, applying the result to $P_2[x]$ tells us that the quadrature is exact for polynomials of degree 2 or smaller.

In fact Peano Kernel Theorem applies to other kind of numerical algorithms as well.

Example (Numerical differentiation)**.**

$$f'(0) \approx -\frac{3}{2}f(0) + 2f(1) - \frac{1}{2}f(2).$$

We can check that the formula is exact for $f(x) = 1, x, x^2$ so by linearity this extends to all elements of $P_2[x]$. The error is

$$L(f) = f'(0) - [-\frac{3}{2}f(0) + 2f(1) - \frac{1}{2}f(2)]$$

so

$$K(\theta) = L(x \mapsto (x-\theta)_+^2) = \begin{cases} 2\theta - \frac{3}{2}\theta^2 & \text{if } 0 \le \theta \le 1 \\ \frac{1}{2}(2-\theta)^2 & \text{if } 1 \le \theta \le 2 \\ 0 & \text{otherwise} \end{cases}$$

For any $f \in C^3[0,2]$,

$$\begin{aligned} |L(f)| &= \frac{1}{2}\left|\int_0^2 K(\theta) f^{(3)}(0) d\theta d\theta\right| \\ &\le \frac{1}{2}\int_0^2 |K(\theta)||f^{(3)}(\theta)| f d\theta \\ &\le \frac{1}{2}\|f^{(3)}\|_\infty \int_0^2 |K(\theta)| d\theta \\ &\le \frac{1}{3}\|f^{(3)}\|_\infty \end{aligned}$$

3 Ordinary Differential Equations

Given an ordinary differential equation of the form

$$\frac{dy}{dt} = y' = f(t, y),\ y(0) = y_0,\ t \geq 0$$

where $f : \mathbb{R} \times \mathbb{R}^N \to \mathbb{R}^N, y \in \mathbb{R}^N$, we want to solve it by compute $y(t_n)$ where $t_n = nh$. h can be thought as the time step.

3.1 One-step methods

A *one-step method* is defined as a map

$$y_{n+1} = \varphi_h(t_n, y_n)$$

where y_n is our approximation for $y(nh)$. The map only depends on one previous value, ergo the name.

The *Euler method* is a one-step method defined by

$$y_{n+1} = y_n + hf(t_n, y_n), n \geq 0.$$

The "deviation" is

$$\begin{aligned} y((n+1)h) &= y(nh + h) \\ &\approx y(nh) + hy'(nh) \\ &= y_n + hf(t_n, y_n) \end{aligned}$$

Definition (Convergence of algorithm)**.** We say that a method *converges* if given $t^* > 0$,

$$\lim_{h \to 0} \max_{0 \leq n \leq \lfloor \frac{t^*}{h} \rfloor} \|y_n(h) - y(nh)\| = 0.$$

Theorem 3.1. *Assume f is Lipschitz in the second argument. Then Euler's method is convergent.*

Proof. Let $e_n(h) = y_n(h) - y(nh)$. Then

$$\begin{aligned} e_{n+1}(h) &= y_{n+1}(h) - y((n+1)h) \\ &= [y_n(h) + hf(nh, y_n(h))] - [y(nh) + hy'(nh) + \eta(h)] \end{aligned}$$

where we Taylor expand the second term and $\eta(h) \sim O(h^2)$

$$\begin{aligned} &= y_n(h) - y(nh) + h[f(nh, y_n(h)) - y'(nh)] - \eta(h) \\ &= e_n(h) + h[f(nh, y_n(h)) - f(nh, y(nh))] - \eta(h) \end{aligned}$$

Assume the Lipschitz constant is λ, we thus get

$$\begin{aligned} \|e_{n+1}(h)\| &\leq \|e_n(h)\| + h\lambda\|y_n(h) - y(nh)\| + \|\eta(h)\| \\ &= \|e_n(h)\|(1 + h\lambda) + Ch^2 \end{aligned}$$

Continue recursively,

$$\begin{aligned}
\|e_n(h)\| &\leq \|e_{n-1}(h)\|(1+h\lambda) + Ch^2 \\
&\leq \|e_{n-2}(h)\|(1+h\lambda)^2 + Ch^2(1+h\lambda) + Ch^2 \\
&\leq \cdots \\
&\leq \underbrace{\|e_0(h)\|}_{0}(1+h\lambda)^n + Ch^2 \sum_{j=0}^{n-1}(1+h\lambda)^j \\
&= Ch^2 \frac{(1+h\lambda)^n - 1}{h\lambda} \\
&= \frac{Ch}{\lambda}[(1+h\lambda)^n - 1]
\end{aligned}$$

As $e^x \geq 1 + x$,

$$\leq \frac{Ch}{\lambda}(e^{h\lambda n} - 1)$$

Since $n \leq \frac{t^*}{h}$,

$$\begin{aligned}
&\leq \frac{Ch}{\lambda}(e^{\lambda t^*} - 1) \\
&\to 0
\end{aligned}$$

as $h \to 0$. □

3.2 Multistep methods

An *s-step method* is defined by a recursion rule

$$\sum_{\ell=0}^{s} \rho_\ell y_{n+\ell} = h \sum_{\ell=0}^{s} \sigma_\ell f(t_{n+\ell}, y_{n+\ell}) \tag{1}$$

where $\rho_s = 1$.

The method is called *explicit* if $\sigma_s = 0$. Otherwise it is called *implicit*.

For example for Euler's method,

$$y_{n+1} - y_n = hf(t_n, y_n)$$

$s = 1, \rho_1 = 1, \rho_0 = -1, \sigma_1 = 0, \sigma_s = 1$. This is an explicit method.

By constrast, the implicit Euler method is

$$y_{n+1} - y_n = hf(t_{n+1}, y_{n+1}).$$

An example of a 2-step method is 2-*step Adams-Bashforth* defined by

$$y_{n+2} - y_{n+1} = h\left(\frac{3}{2}f(t_{n+1}, y_{n+1}) - \frac{1}{2}f(t_n, y_n)\right)$$

Definition (Order)**.** The *order* of a multistep method is the biggest integer $p \geq 0$ such that

$$\sum_{\ell=0}^{s} \rho_\ell y(t_{n+\ell}) - h \sum_{\ell=0}^{s} \sigma_\ell y'(t_{n+\ell}) = O(h^{p+1})$$

for all sufficiently smooth functions y.

If we think $y'(t_{n+\ell}) = f(t_{n+\ell}, y(t_{n+\ell}))$, this is the error of starting at the previous iteration, i.e. the "local error".

Now we calculate the order of Euler method.

$$\begin{aligned} y(t_{n+1}) - y(t_n) - hy'(t_n) &= y(t_n + h) - y(t_n) - hy'(t_n) \\ &= y(t_n) + hy'(t_n) + O(h^2) - y(t_n) - hy'(t_n) \\ &= O(h^2) \end{aligned}$$

so it has order 1.

Another example: the *theta-method* is given by

$$y_{n+1} = y_n + h[\theta f(t_n, y_n) + (1-\theta) f(t_{n+1}, y_{n+1})]$$

which can thought as a parameterised Euler method, with the parameter θ controlling the "explicitness". For $\theta = \frac{1}{2}$, it is given a special name: trapezoidal rule.

The order of the theta-method is

$$\begin{aligned} & y(t_{n+1}) - y(t_n) - h[\theta y'(t_n) + (1-\theta) y'(t_{n+1})] \\ &= y(t_n) + hy'(t_n) + \frac{h^2}{2} y''(t_n) + \frac{h^3}{3!} y'''(t_n) + O(h^4) \\ &\quad - y(t_n) - h[\theta y'(t_n) + (1-\theta)(y'(t_n) + hy''(t_n) + \frac{h^2}{2} y'''(t_n) + O(h^3))] \\ &= h^2 \left(\frac{y''(t_n)}{2} - (1-\theta) y''(t_n) \right) + O(h^3) \\ &= y''(t_n)(\theta - \frac{1}{2}) h^2 + O(h^3) \end{aligned}$$

Thus if $\theta = \frac{1}{2}$ then it has order 2, otherwise order 1[1].

Theorem 3.2. *Let $\rho(w) = \sum_{\ell=0}^{s} \rho_\ell w^\ell$ and $\sigma(w) = \sum_{\ell=0}^{s} \sigma_\ell w^\ell$. Then the order of the multistep method is the largest integer $p \geq 0$ such that*

$$\rho(e^z) - z\sigma(e^z) = O(z^{p+1})$$

as $z \to 0$.

Proof. This is just a tedious exercise using Taylor expansion. For analytic y, we

[1] Strictly speaking we conclude that the order when $\theta = \frac{1}{2}$ is at least 2. To show the order is 2 we need to in addition show that the coefficient of h^3 does not vanish.

have

$$\begin{aligned}
&\sum_{\ell=0}^{s} \rho_\ell y(t_{n+\ell}) - h \sum_{\ell=0}^{s} \sigma_\ell y'(t_{n+\ell}) \\
&= \sum_{\ell=0}^{s} \rho_\ell \sum_{k=0}^{\infty} \frac{(\ell h)^k}{k!} y^{(k)}(t_n) - h \sum_{\ell=0}^{s} \sigma_\ell \sum_{k=0}^{\infty} \frac{(\ell h)^k}{k!} y^{(k+1)}(t_n) \\
&= \sum_{\ell=0}^{s} \rho_\ell y(t_n) + \sum_{k=1}^{\infty} \frac{h^k}{k!} \left[\sum_{\ell=0}^{s} \rho_\ell \ell^k y^{(k)}(t_n) - \sum_{\ell=0}^{s} \sigma_\ell \ell^{k-1} k y^{(k)}(t_n) \right] \\
&= \sum_{\ell=0}^{s} \rho_\ell y(t_n) + \sum_{k=1}^{\infty} \frac{h^k}{k!} \left[\sum_{\ell=0}^{s} \rho_\ell \ell^k - \sum_{\ell=0}^{s} \sigma_\ell \ell^{k-1} k \right] y^{(k)}(t_n)
\end{aligned}$$

This shows that the method has order p if and only if

$$\begin{aligned}
\sum_{\ell=0}^{s} \rho_\ell &= 0 \\
\sum_{\ell=0}^{s} \rho^\ell \ell^k - \sum_{\ell=0}^{s} \sigma_\ell \ell^{k-1} k &= 0
\end{aligned}$$

for all $1 \le k \le p$.

Now repeat the same for the exponentials,

$$\begin{aligned}
&\rho(e^z) - z\sigma(e^z) \\
&= \sum_{\ell=0}^{s} \rho_\ell e^{\ell z} - z \sum_{\ell=0}^{s} \sigma_\ell e^{\ell z} \\
&= \sum_{\ell=0}^{s} \rho_\ell \sum_{k=0}^{\infty} \frac{(\ell z)^k}{k!} - z \sum_{\ell=0}^{s} \sigma_\ell \sum_{k=0}^{\infty} \frac{(\ell z)^k}{k!} \\
&= \sum_{\ell=0}^{s} \rho_\ell + \sum_{k=1}^{\infty} \frac{z^k}{k!} \left[\sum_{\ell=0}^{s} \rho_\ell \ell^k - \sum_{\ell=0}^{s} \sigma_\ell \ell^{k-1} k \right]
\end{aligned}$$

which is of $O(z^{p+1})$ if and only if

$$\begin{aligned}
\sum_{\ell=0}^{s} \rho_\ell &= 0 \\
\sum_{\ell=0}^{s} \rho_\ell \ell^k - \sum_{\ell=0}^{s} \sigma_\ell \ell^{k-1} k &= 0
\end{aligned}$$

for all $1 \le k \le p$, which is exactly the same condition as before. The result thus follows. □

Example (Adams-Bashforth)**.** Recall

$$y_{n+2} - y_{n+1} = h \left(\frac{3}{2} f(t_{n+1}, y_{n+1}) - \frac{1}{2} f(t_n, y_n) \right).$$

We have

$$\begin{aligned}
\rho(w) &= w^2 - w \\
\sigma(w) &= \frac{3}{2} w - \frac{1}{2}
\end{aligned}$$

Thus the exponential formula says

$$\begin{aligned}\rho(e^z) - z\sigma(e^z) &= e^{2z} - e^z - z(\frac{3}{2}e^z - \frac{1}{2}) \\ &= 1 + 2z + \frac{(2z)^2}{2} + \frac{(2z)^3}{3!} \\ &\quad - (1 + z + \frac{z^2}{2} + \frac{z^3}{3!}) \\ &\quad - z(\frac{3}{2} + \frac{3z}{2} + \frac{3z^2}{4} - \frac{1}{2}) + O(z^4) \\ &= z^3(\frac{8}{6} - \frac{1}{6} - \frac{3}{4}) + O(z^4)\end{aligned}$$

so it has order 2.

3.2.1 Convergence of multistep methods

Definition (Root condition)**.** We say that a polynomial $\rho(w) = \sum_{\ell=0}^{s} \rho_\ell w^\ell$ satisfies the *root condition* if all the roots of ρ lie inside the unit disc $\{z \in \mathbb{C} : |z| \leq 1\}$ and any root with modulus 1 is simple.

Theorem 3.3 (Dahlquist equivalence theorem)**.** *The general s-step method given by (1) is convergent if and only if it has order $p \geq 1$ and $\rho(w) = \sum_{\ell=0}^{s} \rho_\ell w^\ell$ satisfies the root condition.*

Example (Adams-Bashforth)**.**

$$y_{n+2} - y_{n+1} = h\left(\frac{3}{2}f(t_{n+1}, t_{n+2}) - \frac{1}{2}f(t_n, y_n)\right).$$

We have checked last time that the order is 2. $\rho(w) = w^2 - w$ with roots 0 and 1 so the method satisfies the root condition and converges.

Example. Consider

$$y_{n+2} - 2y_{n+1} + y_n = 0.$$

To find the order,

$$\begin{aligned}y(t_{n+2}) - 2y(t_{n+1}) + y(t_n) &= y(t_n + 2h) - 2y(t_n + h) + y(t_n) \\ &= ... \\ &= O(h^2)\end{aligned}$$

and further computation shows $p = 1$. $\rho(w) = w^2 - 2w + 1$ so 1 is a root of multiplicity 2 so root condition is not satisfied.

To construct a convergent s-step method, follow the two steps:

1. Choose a polynomial ρ of degree s that satisfies the root condition and $\rho(1) = 0$.

2. To get an implicit method, take $\sigma(w)$ to be the s-degree truncation of Taylor expansion of $\frac{\rho(w)}{\log w}$ around $w = 1$ (which exists since $\rho(1) = 0$). For an explicit method, take σ to be the $(s-1)$-degree Taylor truncation of $\frac{\rho(w)}{\log w}$.

We verify that the order $\geq s$: for an implicit method, by construction

$$\sigma(w) = \frac{\rho(w)}{\log w} + O(|w-1|^{s+1})$$

as $w \to 1$. Substitute $w = e^z$ and since $e^z - 1 = z + O(z^2)$

$$\sigma(e^z) = \frac{\rho(e^z)}{z} + O(z^{s+1})$$

as $z \to 0$. So the order $\geq s+1$. In the case of an explicit method order $\geq s$.

Example. $s = 2, \rho(w) = w^2 - w$ satisfies the root equation.

$$\begin{aligned}\frac{\rho(w)}{\log w} &= \frac{w(w-1)}{\log w} \\ &= \frac{(v+1)v}{\log(1+v)} \\ &= \frac{(v+1)v}{v - \frac{v^2}{2} + \frac{v^3}{3} + O(v^4)} \\ &= \dots \\ &= 1 + \frac{3}{2}v + \frac{5}{12}v^2 + O(v^3)\end{aligned}$$

Thus

$$\sigma(w) = 1 + \frac{3}{2}(w-1) + \frac{5}{12}(w-1)^2$$

for an implicit method (which is called Adams-Moulton method) and

$$\sigma(w) = 1 + \frac{3}{2}(w-1) = -\frac{1}{2} + \frac{3}{2}w$$

which gives Adams-Bashforth method.

3.3 Backward differentiation formula

Definition (Backward differentiation formula)**.** A *backward differentiation formula* (BDF) is an s-step methods with

$$\sigma(w) = \sigma_s w^s. \tag{2}$$

Theorem 3.4. *For (2) to have order s, we must have*

$$\rho(w) = \sigma_s \sum_{\ell=1}^{s} \frac{1}{\ell} w^{s-\ell}(w-1)^\ell$$

with $\sigma_s = \left(\sum_{\ell=1}^{s} \frac{1}{\ell}\right)^{-1}$.

Proof. We want ρ to satisfy

$$\rho(w) - \sigma_s w^s \log w = O(|w-1|^{s+1})$$

as $w \to 1$. We have

$$\log w = -\log \frac{1}{w} = -\log\left(1 - \frac{w-1}{w}\right) = \sum_{\ell=1}^{\infty} \frac{1}{\ell}\left(\frac{w-1}{w}\right)^{\ell}.$$

But our choice of ρ is precisely the first s-terms in this expansion. The value of σ_s is chosen such that $\rho_s = 1$. □

Example. Let $s = 2, \sigma_s = (1 + \frac{1}{2})^{-1} = \frac{2}{3}$, then

$$\sigma(w) = \frac{2}{3}(w(w-1) + \frac{1}{2}(w-1)^2) = w^2 - \frac{4}{3}w + \frac{1}{3}$$

which gives 2-step BDF

$$y_{n+2} - \frac{4}{3}y_{n+1} + \frac{1}{3}y_n = \frac{2}{3}hf(t_{n+2}, y_{n+2}).$$

3.4 Runge-Kutta methods

Suppose we want to solve the ODE

$$y' = f(t, y),\ y(0) = y_0.$$

The solution of this ODE satisfies

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + \int_{t_n}^{t_{n+1}} y'(\tau)d\tau \\ &= y(t_n) + \int_{t_n}^{t_{n+1}} f(\tau, y(\tau))d\tau \\ &= y(t_n) + h\int_0^1 f(t_n + \alpha h, y(t_n + \alpha h))d\alpha \end{aligned}$$

at this point we can apply quadrature formulæ

$$\approx y(t_n) + h\sum_{j=1}^{\nu} b_j \underbrace{f(t_n + c_j h, y(t_n + c_j h))}_{\approx k_j}$$

and Runge-Kutta will give estimates to the k_j's.

The *explicit Runge-Kutta method* is

$$y_{n+1} = y_n + h\sum_{j=1}^{\nu} b_j k_j$$

with

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + c_2 h, y_n + c_2 h k_1) \\ &\vdots \\ k_\nu &= f(t_n + c_\nu h, y_n + h \sum_{j=1}^{\nu-1} a_{j,\nu} k_j) \end{aligned}$$

where $\sum_{j=1}^{\nu-1} a_{j,\nu} = c_\nu$.

Example. $\nu = 2$. We have

$$y_{n+1} = y_n + h(b_1 k_1 + b_2 k_2)$$

where

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + c_2 h, y_n + c_2 h k_1) \end{aligned}$$

The order of the Runge-Kutta method is defined to be the largest integer $p \geq 0$ such that

$$y(t_{n+1}) - y(t_n) - h(b_1 k_1 + b_2 k_2) = O(h^{p+1})$$

where y is the solution of the ODE. For the above case we have

$$\begin{aligned} k_1 &= f(t_n, y(t_n)) = y'(t_n) \\ k_2 &= f(t_n + c_2 h, y(t_n + c_2 h)) \\ &= f(t_n + c_2 h, y(t_n) + c_2 h y'(t_n) + O(h^2)) \\ &= f(t_n, y(t_n)) + \frac{\partial f}{\partial t} c_2 h + \frac{\partial f}{\partial y}(c_2 h y'(t_n) + O(h^2)) + O(h^2) \\ &= f(t_n, y(t_n)) + h c_2 \underbrace{\left(\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} y'(t_n) \right)}_{y''(t_n)} + O(h^2) \end{aligned}$$

Thus after some calculation we find that

$$\begin{aligned} & y(t_{n+1}) - y(t_n) - h(b_1 k_1 + b_2 k_2) \\ &= h y'(t_n)(1 - b_1 - b_2) + h^2 y''(t_n)(\frac{1}{2} - b_2 c_2) + O(h^3) \end{aligned}$$

Thus if

$$\begin{aligned} 1 - b_1 - b_2 &= 0 \\ \frac{1}{2} - b_2 c_2 &= 0 \end{aligned}$$

the method has order ≥ 2.

Now let's prove that the order cannot be ≥ 3, by exhibiting a ODE whose analytic solution is known. Consider

$$\begin{aligned} y' &= y \\ y(0) &= 1 \end{aligned}$$

The solution of this ODE is $y(t) = e^t$. Thus

$$k_1 = f(t_n, y(t_n)) = y(t_n) = e^{t_n}$$
$$k_2 = f(t_n + c_2 h, y(t_n) + c_2 h k_1) = y(t_n) + c_2 h k_1 = e^{t_n}(1 + c_2 h)$$

so

$$\begin{aligned} & y(t_{n+1}) - y(t_n) - h(b_1 k_1 + b_2 k_2) \\ &= e^{t_n + h} - e^{t_n} - h e^{t_n}(b_1 + b_2(1 + c_2 h)) \\ &= e^{t_n}(e^h - 1 - h(b_1 + b_2(1 + c_2 h))) \\ &= e^{t_n}(h + \frac{h^2}{2} + \frac{h^3}{6} + O(h^3) - h(b_1 + b_2) - h^2 b_2 c_2) \\ &= e^{t_n}(h(1 - b_1 - b_2) + h^2(\frac{1}{2} - b_2 c_2) + \frac{h^3}{6} + O(h^3)) \end{aligned}$$

which has a non-vanishing h^3 term.

In general, a Runge-Kutta method has the form

$$y_{n+1} = y_n + h \sum_{\ell=1}^{\nu} b_\ell k_\ell$$

where

$$k_\ell = f(t_n + c_\ell h, y_n + h \sum_{j=1}^{\nu} a_{\ell,j} k_j)$$

and $\sum_{j=1}^{\nu} a_{\ell,j} = c_\ell$.

For explicit Runge-Kutta method, $a_{\ell,j} = 0$ for $j \geq \ell$, i.e. the matrix $a_{\ell,j}$ is lower triangular.

3.5 Stiffness

Consider the ODE

$$y' = \lambda y,\ y(0) = 1.$$

The solution is $y(t) = e^{\lambda t}$ and if $\lambda < 0$, $\lim_{t\to\infty} y(t) = 0$. If we solve our ODE using a numerical method, we want $\lim_{n\to\infty} y_n = 0$.

For example, explicit Euler gives

$$y_n = (1 + h\lambda)^n$$

which goes to zero as $n \to \infty$ if and only if $|1 + h\lambda| < 1$, i.e. $h < \frac{2}{|\lambda|}$.

On the other hand, implicit Euler gives

$$y_{n+1} = (1 - h\lambda)^{-n}$$

which goes to 0 as $n \to \infty$ for any $h > 0$.

Definition (Linear stability domain)**.** Consider a numerical method for the ODE

$$y' = \lambda y,\ y(0) = 1$$

that produces sequence $(y_n)_{n\geq 0}$. The *linear stability domain* of the method

is

$$\mathcal{D} = \{h\lambda \in \mathbb{C} : \lim_{n\to\infty} y_n = 0\}.$$

Definition (A-stable)**.** We say the method is *A-stable* if $\mathbb{C}^- \subseteq \mathcal{D}$ where $\mathbb{C}^- = \{z \in \mathbb{C} : \operatorname{Re} z < 0\}$.

Example. For explicit Euler method,

$$\mathcal{D} = \{h\lambda \in \mathbb{C} : |1 + h\lambda| < 1\} = \{z \in \mathbb{C} : |1 + z| < 1\}.$$

which is not A-stable.

For implicit Euler method,

$$\mathcal{D} = \{z \in \mathbb{C} : |(1 - z)^{-1}| < 1\} = \{z \in \mathbb{C} : |1 - z| > 1\}$$

which is A-stable.

Example (Trapezoidal rule)**.**

$$y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n), f(t_{n+1}, y_{n+1}))$$

Assume $f(t, y) = \lambda y$ so

$$y_{n+1}(1 - \frac{1}{2}h\lambda) = (1 + \frac{1}{2}h\lambda)y_n$$
$$y_n = \left(\frac{1 + \frac{1}{2}h\lambda}{1 - \frac{1}{2}h\lambda}\right)^n y_0$$

so the linear stability domain is

$$\mathcal{D} = \{z \in \mathbb{C} : \operatorname{Re} z < 0\}.$$

Thus trapezoidal rule is A-stable.

Theorem 3.5 (Second Dahlquist barrier)**.** *Any A-stable multistep method has order ≤ 2.*

This is a bad news for multistep methods. One way to get around this is to relax A-stability condition. Recall that the backward differentiation formula (BDF) is an s-step method with $\sigma(w) = \sigma_s w^s$. It is almost A-stable (see pictures in lecture).

3.5.1 Stiffness and Runge-Kutta

Consider the 2-stage implicit Runge-Kutta method

$$y_{n+1} = y_n + \frac{1}{4}h(k_1 + 3k_2)$$

where

$$k_1 = f(t_n, y_n + \frac{1}{4}h(k_1 - k_2))$$
$$k_2 = f(t_n, +\frac{2}{3}h, y_n + \frac{1}{12}h(3k_1 + 5k_2))$$

Let's compute its linear stability domain. Assume $f(t, y) = \lambda y$ so

$$k_1 = \lambda y_n + \frac{1}{12}\lambda h(k_1 - K_2)$$
$$k_2 = \lambda y_n + \frac{1}{12}\lambda h(3k_1 + 5k_2)$$

Solve to get

$$\begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \frac{\lambda y_n}{1 - \frac{2}{3}h\lambda + \frac{1}{6}(h\lambda)^2} \begin{pmatrix} 1 - \frac{2}{3}h\lambda \\ 1 \end{pmatrix}$$

The solution to the original problem is

$$y_{n+1} = \frac{1 + \frac{1}{3}h\lambda}{1 - \frac{2}{3}h\lambda + \frac{1}{6}(h\lambda)^2} y_n$$

so the linear stability domain is the region where the modulus of the factor is small than 1, i.e.

$$\mathcal{D} = \{z \in \mathbb{C} : |r(z)| < 1\}$$

where $r(z) = \frac{1+\frac{1}{3}z}{1-\frac{2}{3}z+\frac{1}{6}z^2}$. To show it is A-stable, we use maximum modulus principle from IB Complex Analysis. First notice that $r(z)$ is a rational function with poles $2 \pm \sqrt{2}i$. As $r(z)$ is analytic in the negative real half plane, $\max |r(z)|$ is attained at the boundary of the domain, i.e. $i\mathbb{R}$. For all $t \in \mathbb{R}$,

$$|r(it)| = \frac{|1 + \frac{1}{3}it|}{|1 - \frac{2}{3}it - \frac{1}{6}t^2|}$$

so $|r(it)|^2 \leq 1$ if and only if

$$1 + \frac{1}{9}t^2 \leq (1 - \frac{1}{6}t^2)^2 + (\frac{2}{3}t)^2,$$

if and only if

$$\frac{1}{36}t^4 \geq 0.$$

Thus the method is A-stable.

Let's show that our implicit Runge-Kutta method has order ≥ 3. We will restrict our attention to only scalar autonomous equations $y' = f(t)$. Let y be the solution to our ODE. Write y for y_n. Then

$$\begin{aligned} k_1 &= f(y + \frac{1}{4}h(k_1 - k_2)) \\ &= f(y) + \frac{1}{4}h(k_1 - k_2)f' + \frac{1}{32}h^2(k_1 - k_2)^2 f'' + O(h^3) \\ k_2 &= f(y + \frac{1}{12}h(3k_1 + 5k_2) \\ &= f(y) + \frac{1}{12}h(3k_1 + 5k_2)f' + \frac{1}{288}h^2(3k_1 + 5k_2)^2 f'' + O(h^3) \end{aligned}$$

It is difficult and tedious to solve this directly, but we could use a trick to extract the coefficients iteratively. To first order

$$k_1 = f(y) + O(h)$$
$$k_2 = f(y) + O(h)$$

Plug in to get

$$k_1 = f(y) + O(h^2)$$
$$k_2 = f(y) + \frac{8}{12}hf'f(y) + O(h^2)$$

Plug in again,

$$k_1 = f(y) - \frac{1}{6}h^2(f')^2 f(y) + O(h^3)$$
$$k_2 = f(y) + \frac{2}{3}hf'f(y) + h^2(\frac{5}{18}(f')^2 f(y) + \frac{2}{9}f''f(y)^2) + O(h^3)$$

Use these values to compute local error,

...

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

3.6 Implementation of ODE methods

3.6.1 Error control

Milne device Given the trapezoidal rule

$$y_{n+1} = y_n + \frac{1}{2}h(f(t_n, y_n) + f(t_{n+1}, y_{n+1}),$$

we want to get an estimate of $y_{n+1} - y(t_{n+1})$,

$$y(t_{n+1}) - y_{n+1} = -\frac{1}{12}h^3 y'''(t_n) + O(h^4)$$

where $-\frac{1}{12}$ is called the *error constant* of the trapezoidal rule.

We secretly run another method in the background: Adams-Bashforth

$$y_{n+1} = y_n + ...$$

has order 2. By expansion,

$$y(t_{n+1}) - y_{n+1} = \frac{5}{12}h^3 y'''(t_n) + O(h^4)$$

so the error constant is $\frac{5}{12}$.

Then

$$y_{n+1}^{\mathrm{AB}} - y_{n+1}^{\mathrm{TR}} = (-\frac{1}{12} - \frac{5}{12})h^3 y'''(t_n) + O(h^4)$$

so

$$h^3 y'''(t_n) \approx -2(t_{n+1}^{\mathrm{AB}} - y_{n+1}^{\mathrm{TR}}.$$

Plugging into the eqution for trapezoidal rule,

$$y_{n+1}^{\mathrm{TR}} - y(t_{n+1}) \approx \frac{1}{6}(y_{n+1}^{\mathrm{TR}} - y_{n+1}^{\mathrm{AB}}.$$

More generally, we work with a pair of multistep methods of the same order

predictor	explicit
corrector	implicity

There are two goals for having a predictor:

1. get initial guess when solving algebraic equation for the (implicit) corrector method.
2. error control: provide an estimate of the error incurred by the corrector.

Let $tol > 0$ be the error tolerance probided by user. Let $\|\mathrm{error}\|$ be the estimate of the local error as computed in the previous.

1. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

3.6.2 Embedded Runge-Kutta

The two-step Runge-Kutta method is

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1) \\ y_{n+1}^1 &= y_n + hk_2 \end{aligned}$$

which has order 2. We can further define

$$k_3 = f(t_n + h, y_n - h + 2hk_1)$$
$$y^2_{n+1} = y_n + \frac{1}{6}h(k_1 + hk_2 + k_3)$$

which has order 3. Then we can use the second method as the "true" value of the function to estimate the error the first method, i.e.

$$y^1_{n+1} - y(t_{n+1}) \approx y^1_{n+1} - y^2_{n+2}.$$

Zadunaisky device: the idea is to define another ODE

$$z' = g(t, z) \tag{$*$}$$

"close" to our ODE

$$y' = f(t, y) \tag{$**$}$$

and such that we know the exact solution to ($*$). Our estimate of the error will be

$$y_{n+1} - y(t_{n+1}) \approx z_{n+1} - z(t_{n+1})$$

where the first term on RHS is what we get by running the numerical method for ($*$) and the second term is its analytic solution, which is known.

Let $y_n, y_{n-1}, \dots, y_{n-p}$ be past solution values define by running the numerical method for ($**$). Let d be a degree p polynomial that interpolates these the values $d(t_{n-k}) = y_{n-k}$ for $0 \le k \le p$. Our approximate ODE is

$$z' = f(t, z) + (d' - f(t, d))$$

where the second term on RHS is small since $d \approx y$ and $y' = f(t, y)$. The solution of this ODE is just $z = d$.

That's all we are going to talk about error control.

3.7 Solving nonlinear algebraic equations using Newton-Raphson

There is one problem we still haven't discussed: solving nonlinear algebraic equations.

An implicit s-step method has the form

$$y_{n+s} = h\sigma_s f(t_{n+s}, y_{n+s}) + v$$

where v only depends on previous values $y_n, \dots, y_{n+s-1}$. One way is to use *functional iteration* or *fixed point iteration*:

1. given initial guess y^0_{n+s} for y_{n+s} (for example from predictor method)
2. iteration
$$y^{j+1}_{n+s} = h\sigma_s f(t_{n+s}, y^j_{n+s}) + v.$$
3. (Hope it converges to the actual value.)

4 Numerical Linear Algebra

4.1 LU factorisation

Definition (LU factorisation)**.** Let A be a real $n \times n$ matrix. An *LU factorisation* of A is a factorisation $A = LU$ where L is unit lower triangluar, i.e. $L_{ij} = 0$ for $j > i$, $L_{ii} = 1$ for $1 \leq i \leq n$ and U is upper triangular, i.e. $U_{ij} = 0$ if $j < i$.

Application.

1. Determinant: given $A = LU$, the determinant can be computed by

$$\det A = (\det L)(\det U) = \prod_{i=1}^{n} L_{ii} \prod_{i=1}^{n} U_{ii} = \prod_{i=1}^{n} U_{ii}$$

   which is much faster than using starting ab initio from the definition

$$\det A = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}$$

   which has $n!$ terms in the sum and can be prohibitively expensive computationally.

2. Testing non-singularity: A is singular if and only if $\det A = 0$, if and only if there exists $i \in \{1, \dots, n\}$ such that $U_{ii} = 0$.

3. Solving linear equations: suppose $Ax = b$ with x unknown. Then it suffices to solve

$$Ux = y$$
$$Ly = b$$

   which are triangular systems and can be solved using (forward/backward) substitution (the cost of which is $O(n^2)$).

Let $\ell_1, \dots, \ell_n$ be columns of L and $u_1^T, \dots, u_n^T$ be rows of U so

$$A = LU = \begin{pmatrix} \ell_1 & \cdots & \ell_n \end{pmatrix} \begin{pmatrix} u_1^T \\ \vdots \\ u_n^T \end{pmatrix} = \sum_{i=1}^{n} \ell_i u_i^T$$

Note that ℓ_k has 1 at kth component and 0 at ith component for $i < k$. Similarly u_k has 0 at ith component for $i < k$. Thus the first $k-1$ rows and columns of $\ell_k u_k^T$ are 0. Thus the only contribution to the first row and column of A comes from $\ell_1 u_1^T$, implying that

$$u_1^T = \text{1st row of } A$$
$$\ell_1 = \frac{1}{A_{11}} \cdot \text{1st column of } A$$

Subsequently, let $A_1 = A - \ell_1 u_1^T$. The only contribution to the second row and column of A_1 comes from $\ell_2 u_2^T$ so

$$u_2^T = \text{2nd row of } A_1$$
$$\ell_2 = \frac{1}{(A_1)_{22}} \cdot \text{2nd column of } A_2$$

The general algorithm is

$A_0 = A$;
for $k = 1$ **to** n **do**
  $u_k^T = k$th row of A_{k-1};
  $\ell_k = \frac{1}{(A_{k-1})_{kk}} \cdot$ kth column of A_{k-1};
  $A_k = A_{k-1} - \ell_k u_k^T$;
end

Algorithm 1: LU factorisation

Complexity analysis The dominant cost is the last line in the for loop, where the number of multiplications needed to form $\ell_k u_k^T$ is $(n-k+1)^2$ so the total cost is

$$\sum_{k=1}^{n} (n-k+1)^2 = \sum_{j=1}^{n} j^2 = O(n^3).$$

A quick remark for implementation: we don't have to store A_k's so it is possible to have even smaller spatial complexity.

The algorithm can be seen as a variant of Guassian elimination: the matrix A_k constructed in the algorithm is of the form

$$\begin{pmatrix} 0 & 0 \\ 0 & * \end{pmatrix}$$

and the lower right block is exactly the same as the one obtained after k steps of Gaussian elimination.

4.2 Pivoting

In the factorisation algorithm above, what happens if $(A_{k-1})_{kk} = 0$? Indeed, not all matrices are LU factorisable. We have to *pivot*.

If we unfold the algorithm, we get

$$\begin{aligned} A_1 &= P_1 A - \ell_1 u_1^T \\ A_2 &= P_2 A_1 - \ell_2 u_2^T = P_2 P_1 A - P_2 \ell_1 u_1^T - \ell_2 u_2^T \\ &\vdots \\ A_n &= P_{n-1} \cdots P_1 A - P_{n-1} \cdots P_2 \ell_1 u_1^T - \cdots - \ell_n u_n^T \end{aligned}$$

by noting that $P_n = I$. Since $A_n = 0$, we get

$$PA = \tilde{L}U$$

$A_0 = A$;
for $k = 1$ **to** n **do**
  let $p \geq k$ be such that $(A_{k-1})_{pk} \neq 0$;
  let P_k be the permutation matrix that swaps positions k and p;
  $l_k = \frac{1}{(P_k A_{k-1})_{kk}} \cdot$ kth column of $P_k A_{k-1}$;
  $u_k^T = k$th row of $P_k A_{k-1}$;
  $A_k = P_k A_{k-1} - l_k u_k^T$;
end

Algorithm 2: LU factorisation with pivoting

where

$$P = P_{n-1} \cdots P_1$$
$$\tilde{L} = (\tilde{\ell}_1 \ \cdots \ \tilde{\ell}_n)$$

and

$$\tilde{\ell}_k = P_{n-1} \cdots P_{k+1} \ell_k.$$

Note that $\tilde{\ell}_k$ has 1 at kth component and 0 for $1, \ldots, k-1$ since the permutations only act on the components $k+1, \ldots, n$. Thus L is a unit lower triangular matrix.

Note. The algorithm can still fail if $(A_{k-1})_{pk} = 0$ for all $p \geq k$. But we can make a choice so that ℓ_k have 1 in kth component and 0 elsewhere, and u_k^T to be kth row of A_{k-1} so $A_{k-1} - \ell_k u_k^T$ has its kth row and column 0 (the factorisation in this case is not unique and we made a choice in ℓ_k).

Pivoting is not only needed to prevent the algorithm from failing but also to prevent round-off errors from accummulating. Thus it is always used for numerical stability of the algorithm. A common choice of pivot is to take p such that $|(A_{k-1})_{pk}|$ is maximum for all $p \geq k$. With such a choice of pivot, all entries of L have abolute value ≤ 1.

4.3 Symmetric matrices

A symmetric matrix A is such that $A_{ij} = A_{ji}$ for all i, j. We aim to factorise it in a way to exhibit this symmetry.

Definition (LDL$^{\mathrm{T}}$ factorisation)**.** An LDL^T *factorisation* of a symmetric matrix A is

$$A = LDL^T$$

where L is unit lower triangular and D is diagonal.

Note that this is a special case of LU factorisation where $U = DL^T$.

We can construct an LDL$^{\mathrm{T}}$ factorisation using an algorithm very similar to the LU factorisation (without pivoting), with the final result being

$$A = D_{11} \ell_1 \ell_1^T + \cdots + D_{nn} \ell_n \ell_n^T = LDL^T$$

where

$$L = \ell_1 \ell_1^T + \cdots + \ell_n \ell_n^T.$$

$A_0 = A$;
for $k = 1$ **to** n **do**
  $\ell_k = \frac{1}{(A_{k-1})_{kk}} \cdot k$th column of A_{k-1};
  $D_{kk} = (A_{k-1})_{kk}$;
  $A_k = A_{k-1} - D_{kk}\ell_k\ell_k^T$;
end

Algorithm 3: $\mathrm{LDL}^{\mathrm{T}}$ factoriation

We will run into similar problems if all elements in a row below certain element is zero. We can still use pivoting, but note that to preserve the symmetry we must act it on both the rows and columns. Instead, in this section we will discuss cases where our naïve algorithm always works.

Recall that a symmetric positive definite matrix A is such that

$$x^T A x > 0$$

for all $x \in \mathbb{R}^n \setminus \{0\}$.

Theorem 4.1. *A symmetric matrix A is positive definite if and only if it has an LDL^T factorisation where $D_{kk} > 0$ for all $1 \le k \le n$.*

Proof. This is exactly the same as the proof in IB Linear Algebra, but numerical analysis flavoured.

- $\Longleftarrow$: Assume $A = LDL^T$ of the form given. For any $x \in \mathbb{R}^n \setminus \{0\}$,

$$x^T A x = x^T L D L^T x = y^T D y = \sum_{k=1}^{n} D_{kk} u_k^2 > 0$$

  where $y = L^T x \neq 0$ since $x \neq 0$ and L is non-singular.

- $\Longrightarrow$: We will show that at each step of the algorithm, $(A_{k-1})_{kk} > 0$. Proceed by induction. If $k = 1$,

$$(A_0)_{11} = A_{11} = e_1^T A e_1 > 0.$$

  By definition

$$A_{k-1} = A - D_{11}\ell_1\ell_1^T - \cdots - D_{k-1}\ell_{k-1}\ell_{k-1}^T.$$

  Define $x \in \mathbb{R}^n$ as the solution of the linear system

$$\begin{pmatrix} & & \ell_1^T & & & & \\ & & \cdots & & & & \\ & & \ell_{k-1}^T & & & & \\ 0 & \cdots & 0 & 1 & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} x = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

which has a unique solution as the matrix is non-singular. Then by construction

$$\begin{aligned}(A_{k-1})_{kk} &= x^T A_{k-1} x \\ &= x^T (A - \sum_{j=1}^{k-1} D_{jj} \ell_j \ell_j^T) x \\ &= x^t A x - \underbrace{\sum_{j=1}^{k-1} D_{jj} (\ell_j^T x)^2}_{=0} \\ &= x^T A x \\ &> 0\end{aligned}$$

□

For a positive definite matrix $A = LDL^T$ with $D_{kk} > 0$, let

$$D^{1/2} = \text{diag}(D_{11}^{1/2}, \dots, D_{nn}^{1/2}).$$

Then

$$A = LD^{1/2} D^{1/2} L^T = \tilde{L}\tilde{L}^T$$

where $\tilde{L} = LD^{1/2}$ is lower triangular. This the *Cholesky factorisation.*

4.4 Sparse matrices

Informally, a *sparse matrix* is one with "many" zeros.

Example. A matrix A is *banded* if $A_{ij} = 0$ whenever $|i - j| > r$ for some fixed r. For examlple, if $r = 1$ then A is diagonal. If $r = 2$ then A is tridiagonal.

We want the LU factorisation of a sparse matrix to inherit this property.

Theorem 4.2. *Let $A = LU$ be an LU factorisation of A (without pivoting). Then*

1. *all leading zeros in the rows of A to the left of the diagonal are inherited by L,*
2. *all leading zeros in the columns of A above the diagonal are inherited by U.*

Proof. We assume that the LU factorisation algorithm terminates without failing, which implies that $(A_{k-1})_{kk} = U_{kk} \neq 0$.

Assume $A_{i,1} = A_{i,2} = \cdots = A_{i,j} = 0$ where $j < i$. We want to show

$$L_{i,1} = L_{i,2} = \cdots = L_{i,j} = 0.$$

But

$$0 = A_{i,1} = L_{i,1} \underbrace{U_{11}}_{\neq 0} + \underbrace{L_{1,2} U_{2,1} + \dots}_{=0}$$

so $L_{i,1} = 0$. Similarly

$$0 = A_{i,2} = \underbrace{L_{i,1}}_{=0} U_{1,2} + L_{1,2} \underbrace{U_{2,2}}_{\neq 0} + \underbrace{L_{1,3}U_{3,2} + ...}_{=0}$$

so $L_{i,2} = 0$ and so on. The statement about columns are exactly the same. □

Application. If A is banded with bandwidth r, then an LU factorisation of A inherits this banded structure.

For banded matrices with bandwidth r, an LU facorisation can be computed in $O(r^2 n)$ time, and a linear system can be solved (given an LU factorisation) in $O(rn)$ time. To see this,

$A_0 = A$;
for $k = 1$ **to** n **do**
  $\ell_k = \frac{1}{(A_{k-1})_{kk}} \cdot$ kth column of A_{k-1};
  $U_k^T = k$th row of A_{k-1};
  $A_k = A_{k-1} - \ell_k u_k^T = A_{k-1} - \sum_{i=k}^{k+r-1} \ell_{ik} u_{ki}$;
end

Algorithm 4: LU factorisation for r-banded matrix

so we only need r^2 multiplications to form $\ell_k u_k^T$.

Recall that to solve a linear system $Ax = b$ given LU factorisation $A = LU$, we instead solve

$$Ux = y$$
$$Ly = b$$

using substitutions, which in general takes $O(n^2)$ opeartions. If A is r-banded, in each step at most r opeartions is needed so the complexity is $O(rn)$.

For a general sparse matrice A, however, an LU factorisation can have significantly more nonzero elements than A. Ideally we want an LU factorisation that minimises *fill-ins*, which is a zero entry of A that is nonzero in L or U. One way to minimise fill-ins is via pivoting.

4.5 QR factorisation

Recall from IB Linear Algebra: we endow $\mathbb{R}^n$ with the Euclidean inner product

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i = x^T y.$$

Two vectors are called *orthogonal* if $x^T y = 0$. An *orthonormal* system $(q_1, ..., q_m)$ in $\mathbb{R}^n$ is a system such that

$$q_i^T q_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

A matrix $Q \in \mathcal{M}_{n,n}(\mathbb{R})$ is *orthogonal* if its columns form an orthonormal basis of $\mathbb{R}^n$. Equivalently, a matrix Q is orthogonal if any one of the following holds:

- $Q^TQ = I$,
- $Q^{-1} = Q^T$,
- $QQ^T = I$.

Remark. If $Q \in \mathcal{M}_{n,n}(\mathbb{R})$ is orthogonal then $\det Q = \pm 1$.

Definition (QR factorisation)**.** Let $A \in \mathcal{M}_{m,n}(\mathbb{R})$. A QR *factorisation* of A is a factorisation $A = QR$ where

- $Q \in \mathcal{M}_{m,m}(\mathbb{R})$ is orthogonal,
- $R \in \mathcal{M}_{m,n}(\mathbb{R})$ is upper triangular.

In the case where $m \geq n$, a *reduced QR factorisation* is $A = QR$ where

- $Q \in \mathcal{M}_{m,n}(\mathbb{R})$ has orthogonal columns,
- $R \in \mathcal{M}_{n,n}(\mathbb{R})$ is upper triangular.

Application fo QR factorisation to linear system solving Suppose A is $n \times n$. $Ax = b$ is equivalent to

$$\begin{aligned} Rx &= y \\ Qy &= b \end{aligned}$$

but Q is orthogonal so $Q^{-1} = Q^T$ so

$$\begin{aligned} Rx &= y \\ y &= Q^Tb \end{aligned}$$

which takes $O(n^2)$ operations.

Gram-Schmidt algorithm Assume $m \geq n$, and we want to construct a reduced QR factorisation. Let $a_1, \dots, a_n \in \mathbb{R}^n$ be the columns of A. We want to find $q_1, \dots, q_n \in \mathbb{R}^m$ such that

1. q_k is a linear combination of $a_1, \dots, a_k$,
2. $(q_1, \dots, q_n)$ is an orthonormal system.

1. $q_1 = \frac{a_1}{\|a_1\|}, R_{11} = \|a_1\|$.
2. $d_2 = a_2 - (a_2^Tq_1)q_1$, $q_2 = \frac{d_2}{\|d_2\|}$. $R_{12} = a_2^Tq_1, R_{22} = \|d_2\|$.

Cost of algorithm Assume the columns of A to be linearly independent, at each iteration j we have to compute $\sim j$ inner products between vectors in $\mathbb{R}^m$, so $O(mj)$ operations at each iteration. Thus the total cost is $O(m(1 + \dots + n)) = O(mn^2)$.

However, note that this version of Gram-Schmidt is not numerically stable.

Gram-Schmidt is "triangular orthogonalisation" process, i.e. use triangular operations to put a matrix into orthogonal form. The next two algorithms are instances of "orthogonal triangularisation", i.e. use orthognal transformations to turn a matrix into a triangular one.

$q_1 = \frac{a_1}{\|a_1\|}$;
$R_{11} = \|a_1\|$;
for $j = 2$ **to** n **do**
  $d_j = a_j - \sum_{i=1}^{j-1}(a_j^T q_i)q_i$;
  $q_j = \frac{d_j}{\|d_j\|}$;
  for $i \leq j-1$ **do**
    $R_{ij} = a_j^T q_i$;
  end
  $R_{jj} = \|d_j\|$;
end

Algorithm 5: Gram-Schmidt

4.5.1 Givens algorithm

We seek orthogonal transformations $\Omega_1, \dots, \Omega_k$ such that $R = \Omega_k \cdots \Omega_1 A$ is upper triangular. Then $A = QR$ where $Q = (\Omega_k \cdots \Omega_1)^T$ is orthogonal.

Givens rotations Recall that clockwise rotation through θ in $\mathbb{R}^2$ is given by

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

Definition (Givens rotation matrix)**.** A *Givens rotation martrix* $\Omega \in \mathbb{R}^{m\times m}$ is specified by $1 \leq p < q \leq m$ and $\theta \in [-\pi, \pi)$ and defined by

$$\begin{aligned} \Omega_{p,p} &= \cos\theta \\ \Omega_{p,q} &= \sin\theta \\ \Omega_{q,p} &= -\sin\theta \\ \Omega_{q,q} &= \cos\theta \\ \Omega_{i,j} &= \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Denote it as $\Omega^{[p,q]}$.

Example. Let $m = 4$.

$$\Omega^{[1,2]} = \begin{pmatrix} \cos\theta & \sin\theta & & \\ -\sin\theta & \cos\theta & & \\ & & 1 & \\ & & & 1 \end{pmatrix}$$

$$\Omega^{[2,4]} = \begin{pmatrix} 1 & & & \\ & \cos\theta & & \sin\theta \\ & & 1 & \\ & -\sin\theta & & \sin\theta \end{pmatrix}$$

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Theorem 4.3. *Let $A \in \mathbb{R}^{m\times n}$. Let $1 \le p < q \le m$ and $j \in \{1, \dots, n\}$. There is a Givens rotation matrix $\Omega^{[p,q]}$ such that*

$$(\Omega^{[p,q]}A)_{q,j} = 0.$$

Furthermore, the rows of $\Omega^{[p,q]}A$, except for the pth and qth row, are identical to those of A. The pth and qth rows of $\Omega^{[p,q]}A$ are linear combinations of the pth and qth row of A.

Proof. $(A_{p,j}, A_{q,j}) \in \mathbb{R}^2$ is a vector and we can find a clockwise rotation through θ sending to the direction along postiive x-axis. θ satisfies

$$\cos\theta = \frac{A_{p,j}}{\sqrt{A_{p,j}^2 + A_{q,j}^2}}$$
$$\sin\theta = \frac{A_{q,j}}{\sqrt{A_{p,j}^2 + A_{q,j}^2}}$$

Then

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} A_{p,j} \\ A_{q,j} \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

□

The *Givens algorithm* is given by

```
for j = 1 to n do
    for i = j + 1 to m do
        Find a Givens rotation such that (Ω^[j,i] A)_{i,j} = 0;
        A := Ω^[j,i] A;
    end
end
```

Algorithm 6: Givens algorithm

At the end of the algorithm, A is upper triangular. Note that this does not explicitly compute Q. To do so we

But in practice this is rarely necessary. For example in solving linear equations, it suffices to act on the RHS by the Ω's, which can be done so by adding a few lines to the original code.

```
Q := I;
for j = 1 to n do
    for i = j + 1 to m do
        Find a Givens rotation such that (Ω^[j,i] A)_{i,j} = 0;
        A := Ω^[j,i] A;
        Q := Ω^[j,i] Q;
    end
end
Q := Q^T;
```

Algorithm 7: Givens algorithm, with Q stored

4.5.2 Householder algorithm

The reflection in $\mathbb{R}^n$ sending a vector u to $-u$ is given by

$$H = I - 2\frac{uu^T}{\|u\|^2}.$$

This is called the *Householder reflection matrix*. We can check that $H^T H = I$.

Given a matrix A which we which to use triangularise using relfection matrix, suppose v_1 is the first column, then we want Hv_1 to be a multiple of e_1. In general we are free to choose the sign. If we want it to be in $+e_1$ direction, choose $u_1 = a_1 - \|a_1\|e_1$. Otherwise, choose $u_1 = a_1 - (-\|a_1\|e_1)$.

Use notation

$$A_{k:m,j} = \begin{pmatrix} A_{k,j} \\ \vdots \\ A_{m,j} \end{pmatrix} \in \mathbb{R}^{m-k+1}$$

Suppose $A \in \mathbb{R}^{m\times n}, m \geq n$,

```
for k = 1 to n do
    ã_k = A_{k:m,k} = (A_{k,k}, ..., A_{m,k})^T;
    ũ_k = ã_k + sgn(A_{k,k}) ‖ã_k‖ ẽ_1;
    for j = k to n do
        A_{k:m,j} = A_{k:m,j} − 2 (ũ^T A_{k:m,j} / ‖ũ‖^2) ũ;
    end
end
```

Algorithm 8: Householder algorithm

This algorithm transforms A into upper triangular form by a series of orthogonal transformations. Like for the Givens algorithm, the matrix Q is not explicitly. If necessary, we can initialise an identity matrix, left multiply by the matrix operation corresponding to the operation in each inner loop, and finally take the transpose to get Q.

Cost of Householder algorithm For each $k = 1$ to n, and j from $k + 1$ to n, we have to compute the inner product $\tilde{u}^T A_{k:m,j}$, which has $m - k + 1 = O(n)$ multiplications. Thus the total cose is $O(mn^2)$, same as Givens.

4.6 Least-square problem

Suppose $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ where $m \geq n$. We want to find x such that $Ax \approx b$. A popular choice is the *least-squares* approximation, aiming to minimise $\|Ax - b\|^2$.

Theorem 4.4. *x is a solution to the least-squares problem if and only if*

$$A^T(Ax - b) = 0.$$

The equation is called *normal equation*.

Proof. Let

$$f(x) = \|Ax - b\|^2 = (Ax - b)^T(Ax - b) = x^T A^T Ax - 2b^T Ax + b^T b.$$

Then

$$\nabla f(x) = 2A^T Ax - 2A^T b.$$

Thus at optimality we have $\nabla f(x) = 0$, so $A^T Ax = A^T b$.

Since f is (by assumption) convex, it is also a sufficient condition for optimality. □

Note that $A^T(Ax - b) = 0$ if and only if $Ax - b$ is orthogonal to all the columns of A. Suppose

$$A = (a_1 \mid \cdots \mid a_n),$$

then this is equivalent to $a_i^T(Ax - b) = 0$ for all i. Geometrically, Ax is the projection of b onto the subspace spanned by $a_1, \ldots, a_n$.

Assume $A = QR$ is a reduce QR factorisation. By definition Q has orthonormal columns and these columns span the same subspace as the columns of A. Thus

$$\begin{aligned} & A^T(Ax - b) = 0 \\ \iff & Q^T(Ax - b) = 0 \\ \iff & Q^T Ax = Q^T b \\ \iff & Rx = Q^T b \end{aligned}$$

Check these conditions carefully using column space. The end result is a triangular system that can be solved using back substitution in $O(n^2)$ operations.

Index