Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 1[1]

What is numerical analysis?

Numerical analysis is the study of algorithms for problems in continuous mathematics.[2] The key word here is *algorithms*. We are looking for algorithms that *run fast* and that are *stable* against various sources of errors and "noise". Some examples of problems from continuous mathematics include:

- Algebraic equations: Solve $f(x) = 0$ where $f : \mathbb{R}^n \to \mathbb{R}$ is a given function.
- Differential equations: Solve $\frac{dx}{dt} = f(x)$ where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a given function.
- Optimization: Find $\min\{f(x) : x \in \mathbb{R}^n\}$ where $f : \mathbb{R}^n \to \mathbb{R}$ is a given function.

Needless to say that such problems arise in many different application areas!

A note about computational complexity We measure the complexity of an algorithm by the number of *elementary operations* $(+,-,\times,\div)$ it needs. We use the Big-Oh notation, e.g., $\mathcal{O}(n)$ or $\mathcal{O}(n^2)$ where n is the size of the input. More precisely an algorithm has complexity $\mathcal{O}(f(n))$ if the number of operations it needs is at most $C \cdot f(n)$ where $C > 0$ is a constant.

1 Polynomial interpolation

We denote by $\mathbb{P}_n[x]$ the *linear space* of all real polynomials of degree at most n.

1.1 The interpolation problem

Given $n+1$ distinct real points $x_0, x_1, \ldots, x_n$ and real numbers $f_0, f_1, \ldots, f_n$, we seek a polynomial $p \in \mathbb{P}_n[x]$ such that $p(x_i) = f_i$, $i = 0, 1, \ldots, n$. Such a polynomial is called an *interpolant*. Note that a polynomial $p \in \mathbb{P}_n[x]$ has $n+1$ degrees of freedom, while interpolation at $x_0, x_1, \ldots, x_n$ constitutes $n+1$ conditions. This, intuitively, justifies seeking an interpolant from $\mathbb{P}_n[x]$.

1.2 The Lagrange formula

Although, in principle, we may solve a linear problem with $n+1$ unknowns to determine a polynomial interpolant, this can be accomplished more easily by using the explicit *Lagrange formula*. We claim that

$$p(x) = \sum_{k=0}^{n} f_k \prod_{\substack{\ell=0 \\ \ell \neq k}}^{n} \frac{x - x_\ell}{x_k - x_\ell}, \qquad x \in \mathbb{R}.$$

Note that $p \in \mathbb{P}_n[x]$, as required. We wish to show that it interpolates the data. Define

$$L_k(x) := \prod_{\substack{\ell=0 \\ \ell \neq k}}^{n} \frac{x - x_\ell}{x_k - x_\ell}, \qquad k = 0, 1, \ldots, n$$

(*Lagrange cardinal polynomials*). It is trivial to verify that $L_j(x_j) = 1$ and $L_j(x_k) = 0$ for $k \neq j$, hence

$$p(x_j) = \sum_{k=0}^{n} f_k L_k(x_j) = f_j, \qquad j = 0, 1, \ldots, n,$$

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

[2] This definition is taken from the essay *The definition of numerical analysis* by L. N. Trefethen.

and p is an interpolant,

Uniqueness Suppose that both $p \in \mathbb{P}_n[x]$ and $q \in \mathbb{P}_n[x]$ interpolate to the same $n+1$ data. Then the nth degree polynomial $p - q$ vanishes at $n+1$ distinct points. But the only nth-degree polynomial with $\geq n+1$ zeros is the zero polynomial. Therefore $p - q \equiv 0$ and the interpolating polynomial is unique.

Complexity For each $k = 0, \ldots, n$, evaluating $L_k(x)$ takes $\mathcal{O}(n)$ operations, and thus the total complexity of evaluating $p(x)$ using the Lagrange formula is $\mathcal{O}(n^2)$.

1.3 The error of polynomial interpolation

Let $[a, b]$ be a closed interval of $\mathbb{R}$. We denote by $C[a, b]$ the space of all continuous functions from $[a, b]$ to $\mathbb{R}$ and let $C^s[a, b]$, where s is a positive integer, stand for the linear space of all functions in $C[a, b]$ that possess s continuous derivatives.

Theorem Given $f \in C^{n+1}[a, b]$, let $p \in \mathbb{P}_n[x]$ interpolate the values $f(x_i)$, $i = 0, 1, \ldots, n$, where $x_0, \ldots, x_n \in [a, b]$ are pairwise distinct. Then for every $x \in [a, b]$ there exists $\xi \in [a, b]$ such that

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i). \tag{1.1}$$

Proof. The formula (1.1) is true when $x = x_j$ for $j \in \{0, 1, \ldots, n\}$, since both sides of the equation vanish. Let $x \in [a, b]$ be any other point and define

$$\phi(t) := f(t) - \left(p(t) + (f(x) - p(x)) \frac{\prod_{i=0}^{n} (t - x_i)}{\prod_{i=0}^{n} (x - x_i)} \right), \qquad t \in [a, b].$$

[Note: The variable in ϕ is t, whereas x is a fixed parameter.] Note that $\phi(x_j) = 0$, $j = 0, 1, \ldots, n$, and $\phi(x) = 0$. Hence, ϕ has at least $n+2$ distinct zeros in $[a, b]$. Moreover, $\phi \in C^{n+1}[a, b]$.
We now apply the *Rolle theorem:* if the function $g \in C^1[a, b]$ vanishes at two distinct points in $[a, b]$ then its derivative vanishes at an intermediate point. We deduce that ϕ' vanishes at (at least) $n+1$ distinct points in $[a, b]$. Next, applying Rolle to ϕ', we conclude that ϕ'' vanishes at n points in $[a, b]$. In general, we prove by induction that $\phi^{(s)}$ vanishes at $n + 2 - s$ distinct points of $[a, b]$ for $s = 0, 1, \ldots, n+1$. Letting $s = n+1$, we have $\phi^{(n+1)}(\xi) = 0$ for some $\xi \in [a, b]$. Hence

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

where we used the fact that $\frac{\mathrm{d}^{n+1}}{\mathrm{d}t^{n+1}} \prod_{i=0}^{n} (t - x_i) = (n+1)!$. Using the fact that $p^{(n+1)} \equiv 0$ (since p is a polynomial of degree n) we finally get (1.1). □

Runge's example We interpolate $f(x) = 1/(1 + x^2)$, $x \in [-5, 5]$, at the equally-spaced points $x_j = -5 + 10\frac{j}{n}$, $j = 0, 1, \ldots, n$. Some of the errors are displayed below

x	$f(x) - p(x)$	$\prod_{i=0}^{n} (x - x_i)$
0.75	3.2×10^{-3}	-2.5×10^{6}
1.75	7.7×10^{-3}	-6.6×10^{6}
2.75	3.6×10^{-2}	-4.1×10^{7}
3.75	5.1×10^{-1}	-7.6×10^{8}
4.75	$4.0 \times 10^{+2}$	-7.3×10^{10}

Table: Errors for $n = 20$

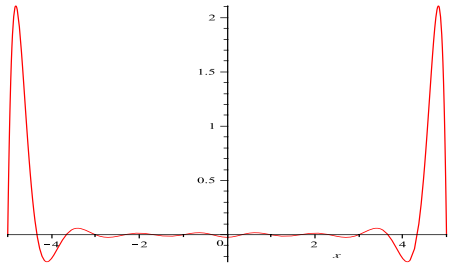


Figure: Errors for $n = 15$

The growth in the error is explained by the product term in (1.1) (the rightmost column of the table). Adding more interpolation points makes the largest error even worse. A remedy to this state of affairs is to cluster points toward the end of the range. A considerably smaller error is attained for $x_j = 5 \cos \frac{(n-j)\pi}{n}$, $j = 0, 1, \ldots, n$ (so-called *Chebyshev points*). It is possible to prove that this choice of points minimizes the magnitude of $\max_{x \in [-5,5]} |\prod_{i=0}^{n} (x - x_i)|$.

Mathematical Tripos Part IB: Lent 2019
Numerical Analysis – Lecture 2[1]

In this lecture, we will see another way to compute the interpolant polynomial, known as *Newton's interpolation formula*. Before presenting this formula we first need the definition of *divided differences*.

1.2 Divided differences: a definition

Given pairwise-distinct points $x_0, x_1, \ldots, x_n \in [a,b]$, we let $p \in \mathbb{P}_n[x]$ interpolate $f \in C[a,b]$ there. The coefficient of x^n in p is called the *divided difference* and denoted by $f[x_0, x_1, \ldots, x_n]$. We say that this divided difference is of *degree* n.
We can derive $f[x_0, \ldots, x_n]$ from the Lagrange formula,

$$f[x_0, x_1, \ldots, x_n] = \sum_{k=0}^{n} f(x_k) \prod_{\substack{\ell=0 \\ \ell \neq k}}^{n} \frac{1}{x_k - x_\ell}. \tag{1.2}$$

It is easy to verify that $f[x_0] = f(x_0)$ and $f[x_0, x_1] = \frac{f(x_1)-f(x_0)}{x_1-x_0}$. We can already see why it is called *divided differences*. In fact one has the following general fact:

Theorem Suppose that $x_0, x_1, \ldots, x_{n+1}$ are pairwise distinct, where $n \geq 0$. Then

$$f[x_0, x_1, \ldots, x_{n+1}] = \frac{f[x_1, x_2, \ldots, x_{n+1}] - f[x_0, x_1, \ldots, x_n]}{x_{n+1} - x_0}. \tag{1.3}$$

Proof. Let $p, q \in \mathbb{P}_n[x]$ be the polynomials that interpolate f at

$$\{x_0, x_1, \ldots, x_n\} \qquad \text{and} \qquad \{x_1, x_2, \ldots, x_{n+1}\}$$

respectively and define

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

We readily verify that $r(x_i) = f(x_i)$, $i = 0, 1, \ldots, n+1$. Hence r is the $(n+1)$-degree interpolating polynomial and $f[x_0, \ldots, x_{n+1}]$ is the coefficient of x^{n+1} therein. The recurrence (1.3) follows from the definition of divided differences. □

From this recursive definition it is natural to think that divided differences can approximate the derivatives of f. This is made precise in the following theorem:

Theorem Let $[a,b]$ be the shortest interval that contains $x_0, x_1, \ldots, x_n$ and let $f \in C^n[a,b]$. Then there exists $\xi \in [a,b]$ such that

$$f[x_0, x_1, \ldots, x_n] = \tfrac{1}{n!} f^{(n)}(\xi). \tag{1.4}$$

Proof. Let p be the interpolating polynomial. The error function $f - p$ has at least $n+1$ zeros in $[a,b]$ and, applying Rolle's theorem n times, it follows that $f^{(n)} - p^{(n)}$ vanishes at some $\xi \in [a,b]$. But $p(x) = \frac{1}{n!} p^{(n)}(\zeta) x^n +$ lower order terms (for any $\zeta \in \mathbb{R}$), therefore, letting $\zeta = \xi$,

$$f[x_0, x_1, \ldots, x_n] = \tfrac{1}{n!} p^{(n)}(\xi) = \tfrac{1}{n!} f^{(n)}(\xi)$$

and we deduce (1.4). □

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

1.3 The Newton interpolation formula

We now present the Newton interpolation formula, which is an alternative to the Lagrange formula we saw in Lecture 1. Again, $f(x_i)$, $i = 0, 1, \ldots, n$, are given and we seek $p \in \mathbb{P}_n[x]$ such that $p(x_i) = f(x_i)$, $i = 0, \ldots, n$.

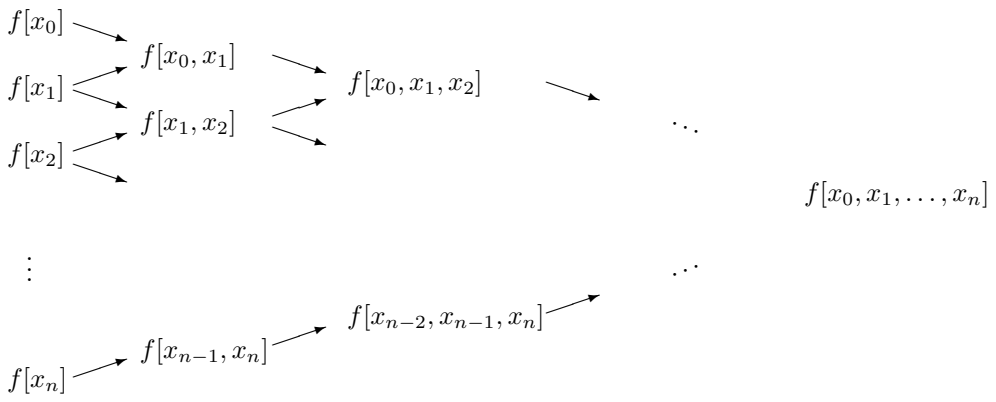
Theorem Suppose that $x_0, x_1, \ldots, x_n$ are pairwise distinct. The polynomial

$$p_n(x) := f[x_0] + f[x_0, x_1](x - x_0) + \cdots + f[x_0, x_1, \ldots, x_n] \prod_{i=0}^{n-1} (x - x_i) \in \mathbb{P}_n[x] \tag{1.5}$$

obeys $p_n(x_i) = f(x_i)$, $i = 0, 1, \ldots, n$.

Proof. By induction on n. The statement is obvious for $n = 0$ and we suppose that it is true for n. We now prove that $p_{n+1}(x) - p_n(x) = f[x_0, x_1, \ldots, x_{n+1}] \prod_{i=0}^{n}(x - x_i)$. Clearly, $p_{n+1} - p_n \in \mathbb{P}_{n+1}[x]$ and the coefficient of x^{n+1} therein is, by definition, $f[x_0, \ldots, x_{n+1}]$. Moreover, $p_{n+1}(x_i) - p_n(x_i) = 0$, $i = 0, 1, \ldots, n$, hence it is a multiple of $\prod_{i=0}^{n}(x - x_i)$, and this proves the asserted form of $p_{n+1} - p_n$. The explicit form of p_{n+1} follows by adding $p_{n+1} - p_n$ to p_n. □

Evaluating the interpolant using Newton's formula We saw last lecture that evaluating the interpolant using Lagrange's formula requires $\mathcal{O}(n^2)$ operations, where n is the number of interpolation points. What about the Newton formula? To evaluate the Newton formula we first need to compute the divided differences. These can be computed using the recursive formula via the *divided difference table,* in the following manner:



The recursive formula allows us to compute all the quantities in this table (i.e., all the $\{f[x_j, x_1, \ldots, x_l]\}_{0 \le j \le l \le n}$), column by column, in $\mathcal{O}(n^2)$ operations.

Having computed the divided differences, the evaluation of the Newton formula (1.5) can be done in just $\mathcal{O}(n)$ time using the Horner scheme, i.e., by writing the Newton formula in the following equivalent way

$$p_n(x) = f[x_0] + (x - x_0)\Big(f[x_0, x_1] + (x - x_1)\Big(f[x_0, x_1, x_2] + \ldots\Big)\Big).$$

On the other hand, the Lagrange formula is often better when we wish to manipulate the interpolation polynomial as part of a larger mathematical expression. We'll see an example in the section on *Gaussian quadrature.*

Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 3[1]

2 Orthogonal polynomials

2.1 Orthogonality in general linear spaces

We have already seen the scalar product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^n x_i y_i$, acting on $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. Likewise, given arbitrary *weights* $w_1, w_2, \ldots, w_n > 0$, we may define $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^n w_i x_i y_i$. In general, a *scalar* (or *inner*) *product* is any function $\mathbb{V} \times \mathbb{V} \to \mathbb{R}$, where $\mathbb{V}$ is a vector space over the reals, subject to the following three axioms:
Symmetry: $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{y}, \boldsymbol{x} \rangle \ \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{V}$;
Nonnegativity: $\langle \boldsymbol{x}, \boldsymbol{x} \rangle \geq 0 \ \forall \boldsymbol{x} \in \mathbb{V}$ and $\langle \boldsymbol{x}, \boldsymbol{x} \rangle = 0$ iff $\boldsymbol{x} = \boldsymbol{0}$; and
Linearity: $\langle a\boldsymbol{x} + b\boldsymbol{y}, \boldsymbol{z} \rangle = a\langle \boldsymbol{x}, \boldsymbol{z} \rangle + b\langle \boldsymbol{y}, \boldsymbol{z} \rangle \ \forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{V}$, $a, b \in \mathbb{R}$.
Given a scalar product, we may define *orthogonality:* $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{V}$ are orthogonal if $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0$.
Let $\mathbb{V} = C[a, b]$, $w \in \mathbb{V}$ be a fixed *positive* function and define

$$\langle f, g \rangle := \int_a^b w(x) f(x) g(x) \, \mathrm{d}x \tag{2.1}$$

for all $f, g \in \mathbb{V}$. It is easy to verify all three axioms of the scalar product.

2.2 Orthogonal polynomials – definition, existence, uniqueness

Given a scalar product in $\mathbb{V} = \mathbb{P}[x]$ (the vector space of polynomials in x, with no bound on the degree), we seek to define a sequence of polynomials $p_0, p_1, p_2, \ldots$ such that:

- $\deg(p_n) = n$ for all $n \geq 0$; and
- $\langle p_n, p_m \rangle = 0$ for all $n \neq m$.

This sequence will be called the *orthogonal polynomials*, and p_n will be called the *n'th orthogonal polynomial*. Observe that for such sequence, $(p_0, \ldots, p_n)$ is an orthogonal basis of $\mathbb{P}_n[x]$ for any $n \geq 0$. **Note**: Different scalar products in general lead to different orthogonal polynomials.

The existence of orthogonal polynomials is the object of the next theorem. A polynomial in $\mathbb{P}_n[x]$ is *monic* if the coefficient of x^n therein equals one.

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

Proof. We let $p_0(x) \equiv 1$ and prove the theorem by induction on n. Thus, suppose that $p_0, p_1, \ldots, p_n$ satisfy the induction hypothesis. To define p_{n+1} let $q(x) := x^{n+1} \in \mathbb{P}_{n+1}[x]$ and, motivated by the *Gram–Schmidt algorithm,* choose

$$p_{n+1}(x) = q(x) - \sum_{k=0}^{n} \frac{\langle q, p_k \rangle}{\langle p_k, p_k \rangle} p_k(x). \tag{2.2}$$

Clearly, $p_{n+1} \in \mathbb{P}_{n+1}[x]$ and it is monic (since all the terms in the sum are of degree $\leq n$).
Let $m \in \{0, 1, \ldots, n\}$. It follows from (2.2) and the induction hypothesis that

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

Hence, p_{n+1} is orthogonal to $p_0, \ldots, p_n$. To prove uniqueness, we suppose the existence of two monic orthogonal polynomials $p_{n+1}, \tilde{p}_{n+1} \in \mathbb{P}_{n+1}[x]$. Let $p := p_{n+1} - \tilde{p}_{n+1} \in \mathbb{P}_n[x]$, hence $\langle p_{n+1}, p \rangle = \langle \tilde{p}_{n+1}, p \rangle = 0$, and this implies

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

and we deduce $p \equiv 0$. □

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

Example *Legendre polynomials* Define the scalar product $\langle f, g\rangle := \int_{-1}^{1} f(x)g(x)\,\mathrm{d}x$ for $f, g \in \mathbb{P}[x]$. The orthogonal polynomials arising from this scalar product is called *Legendre polynomials*. The first polynomials of this sequence are:

$$p_0(x) = 1,\ \ p_1(x) = x,\ \ p_2(x) = x^2 - (1/3),\ \ p_3(x) = x^3 - (3/5)x,\ \ p_4(x) = x^4 - (30/35)x^2 + (3/35).$$

Well-known examples of orthogonal polynomials include:

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

The weight function refers to the function w in the scalar product definition of Equation (2.1).

2.3 The three-term recurrence relation

How to construct orthogonal polynomials? (2.2) might help, 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. For the next theorem we assume the scalar product satisfies $\langle xp, q\rangle = \langle p, xq\rangle$ for any $p, q \in \mathbb{P}[x]$.

Theorem Assuming the scalar product on $\mathbb{P}[x]$ satisfies $\langle xp, q\rangle = \langle p, xq\rangle$ for all $p, q \in \mathbb{P}[x]$, monic orthogonal polynomials are given by the formula

$$\begin{aligned} p_{-1}(x) &\equiv 0, \qquad\qquad p_0(x) \equiv 1, \\ p_{n+1}(x) &= (x - \alpha_n)p_n(x) - \beta_n p_{n-1}(x), \qquad n = 0, 1, \ldots, \end{aligned} \tag{2.3}$$

where

$$\alpha_n := \frac{\langle p_n, xp_n\rangle}{\langle p_n, p_n\rangle}, \qquad \beta_n = \frac{\langle p_n, p_n\rangle}{\langle p_{n-1}, p_{n-1}\rangle} > 0.$$

Remark: The assumption $\langle xp, q\rangle = \langle p, xq\rangle$ on the scalar product is satisfied by most common examples of scalar products. It is satisfied for example by (2.1).

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

$$\langle \psi, p_\ell\rangle = \langle p_{n+1}, p_\ell\rangle - \langle p_n, (x - \alpha_n)p_\ell\rangle + \beta_n\langle p_{n-1}, p_\ell\rangle = 0, \qquad \ell = 0, 1, \ldots, n-2.$$

Because of monicity, $xp_{n-1} = p_n + q$, where $q \in \mathbb{P}_{n-1}[x]$. Thus, from the definition of α_n, β_n,

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

Every $p \in \mathbb{P}_n[x]$ that obeys $\langle p, p_\ell\rangle = 0$, $\ell = 0, 1, \ldots, n$, must necessarily be the zero polynomial. For suppose that it is not so and let x^s be the highest power of x in p. Then $\langle p, p_s\rangle \neq 0$, which is impossible. We deduce that $\psi \equiv 0$, hence (2.3) is true. □

Example *Chebyshev polynomials* We choose the scalar product

$$\langle f, g\rangle := \int_{-1}^{1} f(x)g(x)\frac{\mathrm{d}x}{\sqrt{1 - x^2}}, \qquad f, g \in C[-1, 1]$$

and define $T_n \in \mathbb{P}_n[x]$ by the relation $T_n(\cos\theta) = \cos(n\theta)$. Hence $T_0(x) \equiv 1$, $T_1(x) = x$, $T_2(x) = 2x^2 - 1$ etc. Changing the integration variable,

$$\langle T_n, T_m\rangle = \int_{-1}^{1} T_n(x)T_m(x)\frac{\mathrm{d}x}{\sqrt{1 - x^2}} = \int_0^\pi \cos n\theta \cos m\theta\,\mathrm{d}\theta = \tfrac{1}{2}\int_0^\pi [\cos(n+m)\theta + \cos(n-m)\theta]\,\mathrm{d}\theta = 0$$

whenever $n \neq m$. The recurrence relation for Chebyshev polynomials is particularly simple, $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$, as can be verified at once from the identity $\cos[(n+1)\theta] + \cos[(n-1)\theta] = 2\cos(\theta)\cos(n\theta)$. Note that the T_ns aren't monic, hence the inconsistency with (2.3). To obtain monic polynomials take $T_n(x)/2^{n-1}$, $n \geq 1$.

Mathematical Tripos Part IB: Lent 2019
Numerical Analysis – Lecture 4[1]

2.4 Least-squares polynomial fitting

Consider a scalar product

$$\langle g, h\rangle = \int_a^b w(x)g(x)h(x)\,\mathrm{d}x \tag{2.3}$$

on $C[a,b]$ where $w(x) > 0$ for $x \in (a,b)$. Given $f \in C[a,b]$ we want to find $p \in \mathbb{P}_n[x]$ so as to minimise $\langle f-p, f-p\rangle = \|f-p\|^2$. Call the optimal polynomial $\hat{p}_n$. The following theorem shows that $\hat{p}_n$ can be easily expressed in terms of the orthogonal polynomials associated to (2.3).

Theorem Let $p_0, p_1, p_2, \ldots$ be orthogonal polynomials associated to the inner product (2.3). Let $f \in C[a,b]$. Then the polynomial $\hat{p}_n \in \mathbb{P}_n[x]$ that minimises $\|f-p\|^2 = \langle f-p, f-p\rangle$ is given by

$$\hat{p}_n(x) = \sum_{k=0}^{n} \frac{\langle f, p_k\rangle}{\langle p_k, p_k\rangle} p_k(x). \tag{2.4}$$

Proof. Because $p_0, \ldots, p_n$ form a basis of $\mathbb{P}_n[x]$, any $p \in \mathbb{P}_n$ can be written as $p = \sum_{k=0}^n c_k p_k$ for some coefficients c_k. Thus we have, using orthogonality of the p_ks

$$\langle f-p, f-p\rangle = \left\langle f - \sum_{k=0}^n c_k p_k, f - \sum_{k=0}^n c_k p_k \right\rangle = \langle f, f\rangle - 2\sum_{k=0}^n c_k \langle p_k, f\rangle + \sum_{k=0}^n c_k^2 \langle p_k, p_k\rangle.$$

To minimise this expression we find values of the c_ks that make the gradient equal to 0. We have:

$$\frac{\partial}{\partial c_k}\langle f-p, f-p\rangle = -2\langle p_k, f\rangle + 2c_k\langle p_k, p_k\rangle, \qquad k = 0, 1, \ldots, n,$$

hence setting these to zero we get $c_k = \frac{\langle f, p_k\rangle}{\langle p_k, p_k\rangle}$ which gives the expression (2.4). □

The approximation error we get with $\hat{p}_n$ is

$$\langle f-\hat{p}_n, f-\hat{p}_n\rangle = \langle f, f\rangle - \sum_{k=0}^n \{2c_k\langle p_k, f\rangle - c_k^2\langle p_k, p_k\rangle\} = \langle f, f\rangle - \sum_{k=0}^n \frac{\langle p_k, f\rangle^2}{\langle p_k, p_k\rangle}. \tag{2.5}$$

This identity can be rewritten as $\langle f-\hat{p}_n, f-\hat{p}_n\rangle + \langle \hat{p}_n, \hat{p}_n\rangle = \langle f, f\rangle$, reminiscent of the Pythagoras theorem. It is clear that increasing n brings the approximation error down. A natural question is: if we keep increasing n does the approximation error eventually reach 0? The answer is yes and this is a consequence of Weierstrass's theorem (which we are going to admit without proof):

Theorem *(Weierstrass theorem)* Let $[a,b]$ be finite and let $f \in C[a,b]$. For any $\epsilon > 0$ there is a polynomial p of high enough degree such that $|f(x) - p(x)| \le \epsilon$ for all $x \in [a,b]$.

To prove that $\|f-\hat{p}_n\|^2 \to 0$ as $n \to \infty$, note that by our definition of inner product (2.3) we have, for any p

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

For any $\delta > 0$ we know by Weierstrass theorem that there is a polynomial p of degree n (with n large enough) such that $\max_{x\in[a,b]} |f(x)-p(x)| \le \sqrt{\delta / \int_a^b w(x)dx}$. So for any $N \ge n$ we have

$$\|f-\hat{p}_N\|^2 \le \|f-p\|^2 \le \left(\sqrt{\frac{\delta}{\int_a^b w(x)dx}}\right)^2 \int_a^b w(x)dx \le \delta.$$

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

This is true for any $\delta > 0$ and so shows that $\|f - \hat{p}_n\|^2 \to 0$ as $n \to \infty$. Using the identity (2.5) we get the following consequence:

Theorem *(The Parseval identity)* Let $[a, b]$ be finite, and let $p_0, p_1, p_2, \ldots$ be orthogonal polynomials for (2.3). Then for any $f \in C[a, b]$,

$$\sum_{k=0}^{\infty} \frac{\langle p_k, f\rangle^2}{\langle p_k, p_k\rangle} = \langle f, f\rangle. \tag{2.6}$$

2.5 Least-squares fitting to discrete function values

Consider now the following problem: we are given m function values $f(x_1), f(x_2), \ldots, f(x_m)$, where the x_ks are pairwise distinct, and seek $p \in \mathbb{P}_n[x]$ that minimises $\sum_{k=0}^{n}(f(x_k) - p(x_k))^2$. This corresponds to minimising $\langle f - p, f - p\rangle$ with the following inner product:

$$\langle g, h\rangle := \sum_{k=1}^{m} g(x_k)h(x_k). \tag{2.7}$$

The result from the previous subsection extends directly to this situation as long as $n \leq m-1$ (note that (2.7) does not define a valid inner product on polynomials of degree greater than or equal m). So for $n \leq m - 1$ we can solve the problem as follows:

1. Employ the three-term recurrence (2.3) to calculate $p_0, p_1, \ldots, p_n$ (of course, using the scalar product (2.7));

2. Form $p(x) = \sum_{k=0}^{n} \frac{\langle p_k, f\rangle}{\langle p_k, p_k\rangle} p_k(x)$.

2.6 Gaussian quadrature

We are again in $C[a, b]$ and a scalar product is defined as in subsection 2.1, namely $\langle f, g\rangle = \int_a^b w(x)f(x)g(x)\,\mathrm{d}x$, where $w(x) > 0$ for $x \in (a, b)$. Our goal is to approximate integrals by finite sums,

$$\int_a^b w(x)f(x)\,\mathrm{d}x \approx \sum_{k=1}^{\nu} b_k f(c_k), \qquad f \in C[a, b].$$

The above is known as a *quadrature formula.* Here ν is given, whereas the points $b_1, \ldots, b_\nu$ (the *weights*) and $c_1, \ldots, c_\nu$ (the *nodes*) are independent of the choice of f.
A reasonable approach to achieving high accuracy is to require that the approximation is exact for all $f \in \mathbb{P}_m[x]$, where m is as large as possible – this results in *Gaussian quadrature* and we will demonstrate that $m = 2\nu - 1$ can be attained.
Firstly, we claim that $m = 2\nu$ is impossible. To prove this, choose arbitrary nodes $c_1, \ldots, c_\nu$ and note that $p(x) := \prod_{k=1}^{\nu}(x - c_k)^2$ lives in $\mathbb{P}_{2\nu}[x]$. But $\int_a^b w(x)p(x)\,\mathrm{d}x > 0$, while $\sum_{k=1}^{\nu} b_k p(c_k) = 0$ for any choice of weights $b_1, \ldots, b_\nu$. Hence the integral and the quadrature do not match.
Let $p_0, p_1, p_2, \ldots$ denote, as before, the monic polynomials which are orthogonal w.r.t. the underlying scalar product.

Theorem Given $n \geq 1$, p_n has n real distinct zeros in the interval (a, b).

Proof. Let $\xi_1, \ldots, \xi_m$ be the points in (a, b) where p_n changes signs (equivalently these are the zeros of p of odd multiplicity) and let $q(x) = \prod_{i=1}^{m}(x - \xi_i)$. Observe that the polynomial $p_n(x)q(x)$ does not change signs in (a, b): this is because all the roots of $p_n(x)q(x)$ in (a, b) have even multiplicity. It thus follows that

$$|\langle q, p_n\rangle| = \left|\int_a^b w(x)q(x)p_n(x)\,\mathrm{d}x\right| = \int_a^b w(x)|q(x)p_n(x)|\,\mathrm{d}x > 0.$$

Since p_n is orthogonal to all polynomials of degree $\leq n - 1$ it follows that q must be of degree *at least* n, i.e., $m \leq n$. On the other hand, since p_n is of degree n it can have at most n roots. Finally this means that p_n has n distinct real roots in (a, b). □

Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 5[1]

We commence our construction of *Gaussian quadrature* by choosing pairwise-distinct nodes $c_1, c_2, \ldots, c_\nu \in [a, b]$ and define the *interpolatory weights*

$$b_k := \int_a^b w(x) \prod_{\substack{j=1 \\ j\neq k}}^{\nu} \frac{x - c_j}{c_k - c_j} \, \mathrm{d}x, \qquad k = 1, 2, \ldots, \nu.$$

Theorem The quadrature formula with the above choice is exact for all $f \in \mathbb{P}_{\nu-1}[x]$. Moreover, if $c_1, c_2, \ldots, c_\nu$ are the zeros of p_ν then it is exact for all $f \in \mathbb{P}_{2\nu-1}[x]$.

Proof. Every $f \in \mathbb{P}_{\nu-1}[x]$ is its own interpolating polynomial, hence by Lagrange's formula

$$f(x) = \sum_{k=1}^{\nu} f(c_k) \prod_{\substack{j=1 \\ j\neq k}}^{\nu} \frac{x - c_j}{c_k - c_j}. \tag{2.7}$$

The quadrature is exact for all $f \in \mathbb{P}_{\nu-1}[x]$ if $\int_a^b w(x) f(x) \, \mathrm{d}x = \sum_{k=1}^{\nu} b_k f(c_k)$, and this, in tandem with the interpolating-polynomial representation, yields the stipulated form of $b_1, \ldots, b_\nu$.
Let $c_1, \ldots, c_\nu$ be the zeros of p_ν. Given any $f \in \mathbb{P}_{2\nu-1}[x]$, we can represent it uniquely as $f = qp_\nu + r$, where $q, r \in \mathbb{P}_{\nu-1}[x]$. Thus, by orthogonality,

$$\begin{aligned} \int_a^b w(x) f(x) \, \mathrm{d}x &= \int_a^b w(x)[q(x)p_\nu(x) + r(x)] \, \mathrm{d}x = \langle q, p_\nu \rangle + \int_a^b w(x) r(x) \, \mathrm{d}x \\ &= \int_a^b w(x) r(x) \, \mathrm{d}x. \end{aligned}$$

On the other hand, the choice of quadrature knots gives

$$\sum_{k=1}^{\nu} b_k f(c_k) = \sum_{k=1}^{\nu} b_k [q(c_k)p_\nu(c_k) + r(c_k)] = \sum_{k=1}^{\nu} b_k r(c_k).$$

Hence the integral and its approximation coincide, because $r \in \mathbb{P}_{\nu-1}[x]$ and the quadrature is exact for all polynomials in $\mathbb{P}_{\nu-1}[x]$. □

Example Let $[a, b] = [-1, 1]$, $w(x) \equiv 1$. Then the underlying orthogonal polynomials are the *Legendre polynomials:* $P_0 \equiv 1$, $P_1(x) = x$, $P_2(x) = \frac{3}{2}x^2 - \frac{1}{2}$, $P_3(x) = \frac{5}{2}x^3 - \frac{3}{2}x$, $P_4(x) = \frac{35}{8}x^4 - \frac{15}{4}x^2 + \frac{3}{8}$ (it is customary to use this, non-monic, normalisation). The nodes of Gaussian quadrature are
$\nu = 1$: $c_1 = 0$;
$\nu = 2$: $c_1 = -\frac{\sqrt{3}}{3}$, $c_2 = \frac{\sqrt{3}}{3}$;
$\nu = 3$: $c_1 = -\frac{\sqrt{15}}{5}$, $c_2 = 0$, $c_3 = \frac{\sqrt{15}}{5}$;
$\nu = 4$: $c_1 = -\sqrt{\frac{3}{7} + \frac{2}{35}\sqrt{30}}$, $c_2 = -\sqrt{\frac{3}{7} - \frac{2}{35}\sqrt{30}}$, $c_3 = \sqrt{\frac{3}{7} - \frac{2}{35}\sqrt{30}}$, $c_4 = \sqrt{\frac{3}{7} + \frac{2}{35}\sqrt{30}}$.

3 The Peano kernel theorem

In the previous section we looked at quadrature formulae that are exact for polynomials up to certain degree n. The aim of this section is to present a tool that allows us to bound the error if we use the quadrature formula for functions f that are not in $\mathbb{P}_n[x]$. The result we will state is actually quite general, and is not

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

restricted to quadrature formulae, however we will use quadrature formulae as a running example for the sake of motivation. The error function for a quadrature formula is

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

Assume that $f \in \mathrm{C}^{n+1}[a,b]$ and consider Taylor's formula with integral remainder:

$$f(x) = \underbrace{f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \cdots + \frac{(x-a)^n}{n!}f^{(n)}(a)}_{g(x)} + \frac{1}{n!}\int_a^x (x-\theta)^n f^{(n+1)}(\theta)\,\mathrm{d}\theta. \quad (3.1)$$

Since g is a polynomial of degree n, and since our quadrature formula is exact for polynomials up to degree n we have $L(g) = 0$. It thus follows, since L is linear that

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

To make the range of integration independent of x, we introduce the notation

$$(x-\theta)_+^n := \begin{cases} (x-\theta)^n, & x \geq \theta, \\ 0, & x \leq \theta, \end{cases} \qquad \text{whence} \qquad L(f) = \frac{1}{n!}L\left\{x \mapsto \int_a^b (x-\theta)_+^n f^{(n+1)}(\theta)\,\mathrm{d}\theta\right\}.$$

Let $K(\theta) := L[x \mapsto (x-\theta)_+^n]$ for $x \in [a,b]$. *[Note: K is independent of f.]* The function K is called the *Peano kernel* of L. ***Suppose that it is allowed to exchange the order of action of $\int$ and L.*** Because of the linearity of L, we then have

$$L(f) = \frac{1}{n!}\int_a^b K(\theta)f^{(n+1)}(\theta)\,\mathrm{d}\theta. \quad (3.2)$$

The Peano kernel theorem Let L be a linear functional such that $L(f) = 0$ for all $f \in \mathbb{P}_n[x]$. Provided that $f \in \mathrm{C}^{n+1}[a,b]$ and the above exchange of L with the integration sign is valid, the formula (3.2) is true. □

Example Consider Simpson's rule $\int_{-1}^1 f(x)\,\mathrm{d}x \approx \frac{1}{3}(f(-1)+4f(0)+f(1))$. One can verify that the Simpson rule is exact for polynomials up to degree 2 (in fact it is also true for polynomials up to degree 3). Let $L(f) = \int_{-1}^1 f(x)dx - \frac{1}{3}(f(-1)+4f(0)+f(1))$. Peano kernel theorem tells us that for any $f \in \mathrm{C}^3[-1,1]$ we have

$$L(f) = \frac{1}{2}\int_{-1}^1 K(\theta)f'''(\theta)\,\mathrm{d}\theta,$$

where $K(\theta) = L(x \mapsto (x-\theta)_+^2)$. Since $\int_{-1}^1 (x-\theta)_+^2\,\mathrm{d}x = \frac{(1-\theta)^3}{3}$ we can verify that

$$\begin{aligned} K(\theta) &= \begin{cases} \frac{(1-\theta)^3}{3} - \frac{1}{3}(0+4\theta^2+(1-\theta)^2) & -1 \leq \theta \leq 0 \\ \frac{(1-\theta)^3}{3} - \frac{1}{3}(0+4\cdot 0+(1-\theta)^2) & 0 \leq \theta \leq 1 \end{cases} \\ &= \begin{cases} -\frac{1}{3}\theta(1+\theta)^2 & -1 \leq \theta \leq 0 \\ -\frac{1}{3}\theta(1-\theta)^2 & 0 \leq \theta \leq 1. \end{cases} \end{aligned} \quad (3.3)$$

This allows us to bound the approximation error for Simpson's rule. Indeed for any $f \in \mathrm{C}^3[-1,1]$ we get

$$|L(f)| \leq \frac{1}{2}\int_{-1}^1 |K(\theta)||f'''(\theta)|\,\mathrm{d}\theta \leq \frac{1}{36}\|f'''\|_\infty$$

where $\|f'''\|_\infty := \max_{x\in[-1,1]}|f'''(\theta)|$ and where we used the fact $\int_{-1}^1 |K(\theta)|\,\mathrm{d}\theta = \frac{1}{18}$ which can be easily verified from (3.3).

Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 6[1]

We look at another example of application of the Peano kernel theorem.

Example We approximate a derivative by a linear combination of function values, $f'(0) \approx -\frac{3}{2}f(0)+2f(1)-\frac{1}{2}f(2)$. Define $L(f) := f'(0) - [-\frac{3}{2}f(0) + 2f(1) - \frac{1}{2}f(2)]$ and it is easy to check that $L(f) = 0$ for $f \in \mathbb{P}_2[x]$. (Verify by trying $f(x) = 1, x, x^2$ and using linearity of L.) Thus, for $f \in \mathrm{C}^3[0,2]$ we have

$$L(f) = \tfrac{1}{2}\int_0^2 K(\theta) f'''(\theta)\,\mathrm{d}\theta$$

with $K(\theta) = L(x \mapsto (x-\theta)_+^2)$. For fixed θ, let $g(x) := (x-\theta)_+^2$. Then

$$\begin{aligned} K(\theta) = L(g) &= g'(0) - \left[-\tfrac{3}{2}g(0) + 2g(1) - \tfrac{1}{2}g(2)\right] \\ &= 2(0-\theta)_+ - \left[-\tfrac{3}{2}(0-\theta)_+^2 + 2(1-\theta)_+^2 - \tfrac{1}{2}(2-\theta)_+^2\right] \\ &= \begin{cases} 2\theta - \frac{3}{2}\theta^2, & 0 \le \theta \le 1, \\ \frac{1}{2}(2-\theta)^2, & 1 \le \theta \le 2, \\ 0, & \text{else.} \end{cases} \end{aligned}$$

One can verify that $\int_0^2 |K(\theta)|\,\mathrm{d}\theta = \frac{2}{3}$. Consequently for any $f \in \mathrm{C}^3[0,2]$ we have

$$|L(f)| \le \frac{1}{2!}\int_0^2 |K(\theta) f'''(\theta)|\,\mathrm{d}\theta \le \frac{1}{2}\|f'''\|_\infty \int_0^2 |K(\theta)|\,\mathrm{d}\theta = \frac{1}{3}\|f'''\|_\infty,$$

where $\|f'''\|_\infty = \max_{x\in[0,2]} |f'''(x)|$.

4 Ordinary differential equations

We wish to approximate the exact solution of the *ordinary differential equation (ODE)*

$$\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y}), \qquad t \ge 0, \tag{4.1}$$

where $\boldsymbol{y} \in \mathbb{R}^N$ and the function $\boldsymbol{f} : \mathbb{R} \times \mathbb{R}^N \to \mathbb{R}^N$ is sufficiently 'nice'. (In principle, it is enough for $\boldsymbol{f}$ to be Lipschitz to ensure that the solution exists and is unique. Yet, for simplicity, we henceforth assume that $\boldsymbol{f}$ is analytic: in other words, we are always able to expand locally into Taylor series.) The equation (4.1) is accompanied by the initial condition $\boldsymbol{y}(0) = \boldsymbol{y}_0$.
Our purpose is to approximate $\boldsymbol{y}_{n+1} \approx \boldsymbol{y}(t_{n+1})$, $n = 0, 1, \ldots$, where $t_m = mh$ and the *time step* $h > 0$ is small, from $\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ and equation (4.1).

4.1 One-step methods

A one-step method is a map $\boldsymbol{y}_{n+1} = \boldsymbol{\varphi}_h(t_n, \boldsymbol{y}_n)$, i.e. an algorithm which allows $\boldsymbol{y}_{n+1}$ to depend only on t_n, $\boldsymbol{y}_n$, h and the ODE (4.1).

The Euler method: We know $\boldsymbol{y}$ and its slope $\boldsymbol{y}'$ at $t = 0$ and wish to approximate $\boldsymbol{y}$ at $t = h > 0$. The most obvious approach is to truncate $\boldsymbol{y}(h) = \boldsymbol{y}(0) + h\boldsymbol{y}'(0) + \frac{1}{2}h^2\boldsymbol{y}''(0) + \cdots$ at the h^2 term. Since $\boldsymbol{y}'(0) = \boldsymbol{f}(t_0, \boldsymbol{y}_0)$, this procedure approximates $\boldsymbol{y}(h) \approx \boldsymbol{y}_0 + h\boldsymbol{f}(t_0, \boldsymbol{y}_0)$ and we thus set $\boldsymbol{y}_1 = \boldsymbol{y}_0 + h\boldsymbol{f}(t_0, \boldsymbol{y}_0)$. By the same token, we may advance from h to $2h$ by letting $\boldsymbol{y}_2 = \boldsymbol{y}_1 + h\boldsymbol{f}(t_1, \boldsymbol{y}_1)$. In general, we obtain the *Euler method*

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(t_n, \boldsymbol{y}_n), \qquad n = 0, 1, \ldots. \tag{4.2}$$

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

Convergence: Let $t^* > 0$ be given. We say that a method, which for every $h > 0$ produces the solution sequence $\boldsymbol{y}_n = \boldsymbol{y}_n(h)$, $n = 0, 1, \ldots, \lfloor t^*/h \rfloor$, *converges* if

$$\lim_{h \to 0} \max_{n=0,\ldots,\lfloor t^*/h \rfloor} \|\boldsymbol{y}_n(h) - \boldsymbol{y}(nh)\| = 0,$$

where $\boldsymbol{y}(nh)$ is the evaluation at time $t = nh$ of the exact solution of (4.1).

Theorem *Suppose that $\boldsymbol{f}$ satisfies the Lipschitz condition: there exists $\lambda \geq 0$ such that*

$$\|\boldsymbol{f}(t, \boldsymbol{v}) - \boldsymbol{f}(t, \boldsymbol{w})\| \leq \lambda \|\boldsymbol{v} - \boldsymbol{w}\|, \qquad t \in [0, t^*], \quad \boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^N.$$

Then the Euler method (4.2) converges.

Proof. Let $\boldsymbol{e}_n = \boldsymbol{y}_n - \boldsymbol{y}(t_n)$, the error at step n, where $0 \leq n \leq t^*/h$. Thus,

$$\boldsymbol{e}_{n+1} = \boldsymbol{y}_{n+1} - \boldsymbol{y}(t_{n+1}) = [\boldsymbol{y}_n + h\boldsymbol{f}(t_n, \boldsymbol{y}_n)] - [\boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n) + \mathcal{O}\big(h^2\big)].$$

By the Taylor theorem, the $\mathcal{O}\big(h^2\big)$ term can be bounded uniformly for all $[0, t^*]$ in the underlying norm $\|\cdot\|$ by ch^2, where $c > 0$ (Indeed if we take $c = \frac{1}{2}\max_{t \in [0,t^*]} \|\boldsymbol{y}''(t)\|$, then by Taylor's formula with integral remainder we get that for any t, h such that $0 \leq t < t + h \leq t^*$, $\|\boldsymbol{y}(t+h) - (\boldsymbol{y}(t) + h\boldsymbol{y}'(t))\| \leq ch^2$.) Thus, using (4.1) and the triangle inequality,

$$\begin{aligned} \|\boldsymbol{e}_{n+1}\| &\leq \|\boldsymbol{y}_n - \boldsymbol{y}(t_n)\| + h\|\boldsymbol{f}(t_n, \boldsymbol{y}_n) - \boldsymbol{f}(t_n, \boldsymbol{y}(t_n))\| + ch^2 \\ &\leq \|\boldsymbol{y}_n - \boldsymbol{y}(t_n)\| + h\lambda\|\boldsymbol{y}_n - \boldsymbol{y}(t_n)\| + ch^2 = (1 + h\lambda)\|\boldsymbol{e}_n\| + ch^2. \end{aligned}$$

Consequently, by induction,

$$\|\boldsymbol{e}_{n+1}\| \leq (1 + h\lambda)^m \|\boldsymbol{e}_{n+1-m}\| + ch^2 \sum_{j=0}^{m-1} (1 + h\lambda)^j, \qquad m = 0, 1, \ldots, n+1.$$

In particular, letting $m = n + 1$ and bearing in mind that $\boldsymbol{e}_0 = \boldsymbol{0}$, we have

$$\|\boldsymbol{e}_{n+1}\| \leq ch^2 \sum_{j=0}^{n} (1 + h\lambda)^j = ch^2 \frac{(1 + h\lambda)^{n+1} - 1}{(1 + h\lambda) - 1} \leq \frac{ch}{\lambda}(1 + h\lambda)^{n+1}.$$

For small $h > 0$ it is true that $0 < 1 + h\lambda \leq \mathrm{e}^{h\lambda}$. This and $(n+1)h \leq t^*$ imply that $(1 + h\lambda)^{n+1} \leq \mathrm{e}^{t^*\lambda}$, therefore $\|\boldsymbol{e}_n\| \leq \frac{c\mathrm{e}^{t^*\lambda}}{\lambda} h \overset{h \to 0}{\longrightarrow} 0$ uniformly for $0 \leq nh \leq t^*$ and the theorem is true. □

Mathematical Tripos Part IB: Lent 2019
Numerical Analysis – Lecture 7[1]

4.2 Multistep methods

It is often useful to use past solution values in computing a new value to the ODE (4.1). Assuming that $\boldsymbol{y}_n, \boldsymbol{y}_{n+1}, \ldots, \boldsymbol{y}_{n+s-1}$ are available, where $s \geq 1$, we say that

$$\sum_{l=0}^{s} \rho_l \boldsymbol{y}_{n+l} = h \sum_{l=0}^{s} \sigma_l \boldsymbol{f}(t_{n+l}, \boldsymbol{y}_{n+l}), \qquad n = 0, 1, \ldots, \tag{4.4}$$

where $\rho_s = 1$, is an *s-step method*. If $\sigma_s = 0$, the method is *explicit,* otherwise it is *implicit.* If $s \geq 2$, we need to obtain extra *starting values* $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{s-1}$ by different time-stepping method.

Examples: The following are some common multistep methods:

$$\begin{aligned}
\text{Euler:} \quad & \boldsymbol{y}_{n+1} - \boldsymbol{y}_n = h\boldsymbol{f}(t_n, \boldsymbol{y}_n), \\
\text{Implicit Euler:} \quad & \boldsymbol{y}_{n+1} - \boldsymbol{y}_n = h\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}), \\
\text{Trapezoidal rule:} \quad & \boldsymbol{y}_{n+1} - \boldsymbol{y}_n = \tfrac{1}{2}h[\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})] \\
\text{Theta rule:} \quad & \boldsymbol{y}_{n+1} - \boldsymbol{y}_n = h[\theta\boldsymbol{f}(t_n, \boldsymbol{y}_n) + (1-\theta)\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})] \\
\text{2-step Adams-Bashforth:} \quad & \boldsymbol{y}_{n+2} - \boldsymbol{y}_{n+1} = h[\tfrac{3}{2}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) - \tfrac{1}{2}\boldsymbol{f}(t_n, \boldsymbol{y}_n)] \\
\text{2-step Adams-Moulton:} \quad & \boldsymbol{y}_{n+2} - \boldsymbol{y}_{n+1} = h[\tfrac{5}{12}\boldsymbol{f}(t_{n+2}, \boldsymbol{y}_{n+2}) + \tfrac{2}{3}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) - \tfrac{1}{12}f(t_n, \boldsymbol{y}_n)].
\end{aligned}$$

For example Adams-Bashforth is a 2-step method ($s = 2$) with $\rho_2 = 1, \rho_1 = -1, \rho_0 = 0$ and $\sigma_2 = 0, \sigma_1 = \frac{3}{2}$ and $\sigma_0 = -\frac{1}{2}$. The implicit Euler method, trapezoidal rule, theta rule for $0 \leq \theta < 1$, and Adams-Moulton are *implicit* methods. The reason these are called implicit is that $\boldsymbol{y_{n+s}}$ appears in the right-hand side of (4.4) and so one has to solve a (generally nonlinear) algebraic equation to compute the new value $\boldsymbol{y}_{n+s}$ from the recursion rule.

Our goal is to develop some general tools to study the convergence of multistep methods. We first introduce the definition or *order*.

Order: The *order* of the multistep method (4.4) is the largest integer $p \geq 0$ such that

$$\sum_{l=0}^{s} \rho_l \boldsymbol{y}(t_{n+l}) - h \sum_{l=0}^{s} \sigma_l \boldsymbol{y}'(t_{n+l}) = \mathcal{O}\big(h^{p+1}\big) \tag{4.5}$$

for all sufficiently smooth functions $\boldsymbol{y}$. The order is a local measure of accuracy for the method: it measures the error incurred by applying the rule (4.4), assuming that the correct value of $\boldsymbol{y}$ at the previous points is known. Let us evaluate the order of some of the methods given above:

The order of Euler's method: For Euler's method, the left-hand side of (4.5) is

$$\boldsymbol{y}(t_{n+1}) - [\boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n, \boldsymbol{y}(t_n))] = [\boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n) + \tfrac{1}{2}h^2\boldsymbol{y}''(t_n) + \cdots] - [\boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n)] = \mathcal{O}\big(h^2\big)$$

and we deduce that Euler's method is of order 1.

The order of the theta method: From Taylor's theorem we have:

$$\begin{aligned}
& \boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - h[\theta\boldsymbol{y}'(t_n) + (1-\theta)\boldsymbol{y}'(t_{n+1})] \\
&= [\boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n) + \tfrac{1}{2}h^2\boldsymbol{y}''(t_n) + \tfrac{1}{6}h^3\boldsymbol{y}'''(t_n)] - \boldsymbol{y}(t_n) - \theta h\boldsymbol{y}'(t_n) \\
&\quad - (1-\theta)h[\boldsymbol{y}'(t_n) + h\boldsymbol{y}''(t_n) + \tfrac{1}{2}h^2\boldsymbol{y}'''(t_n)] + \mathcal{O}\big(h^4\big) \\
&= (\theta - \tfrac{1}{2})h^2\boldsymbol{y}''(t_n) + (\tfrac{1}{2}\theta - \tfrac{1}{3})h^3\boldsymbol{y}'''(t_n) + \mathcal{O}\big(h^4\big)\,.
\end{aligned}$$

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

Therefore the theta method is of order 1, except that the trapezoidal rule ($\theta = 1/2$) is of order 2.
Let $\rho(w) = \sum_{l=0}^{s} \rho_l w^l$, $\sigma(w) = \sum_{l=0}^{s} \sigma_l w^l$.

Theorem *The multistep method (4.4) is of order $p \geq 1$ iff*

$$\rho(\mathrm{e}^z) - z\sigma(\mathrm{e}^z) = \mathcal{O}\big(z^{p+1}\big), \qquad z \to 0. \tag{4.6}$$

Proof. Substituting the exact solution and expanding into Taylor series about t_n,

$$\begin{aligned}\sum_{l=0}^{s} \rho_l \boldsymbol{y}(t_{n+l}) - h\sum_{l=0}^{s} \sigma_l \boldsymbol{y}'(t_{n+l}) &= \sum_{l=0}^{s} \rho_l \sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{y}^{(k)}(t_n) l^k h^k - h\sum_{l=0}^{s} \sigma_l \sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{y}^{(k+1)}(t_n) l^k h^k \\ &= \left(\sum_{l=0}^{s} \rho_l\right) \boldsymbol{y}(t_n) + \sum_{k=1}^{\infty} \frac{1}{k!} \left(\sum_{l=0}^{s} l^k \rho_l - k\sum_{l=0}^{s} l^{k-1}\sigma_l\right) h^k \boldsymbol{y}^{(k)}(t_n).\end{aligned}$$

Thus, to obtain $\mathcal{O}\big(h^{p+1}\big)$ regardless of the choice of $\boldsymbol{y}$, it is necessary and sufficient that

$$\sum_{l=0}^{s} \rho_l = 0, \qquad \sum_{l=0}^{s} l^k \rho_l = k\sum_{l=0}^{s} l^{k-1}\sigma_l, \qquad k = 1, 2, \ldots, p. \tag{4.7}$$

On the other hand, expanding again into Taylor series,

$$\begin{aligned}\rho(\mathrm{e}^z) - z\sigma(\mathrm{e}^z) &= \sum_{l=0}^{s} \rho_l \mathrm{e}^{lz} - z\sum_{l=0}^{s} \sigma_l \mathrm{e}^{lz} = \sum_{l=0}^{s} \rho_l \left(\sum_{k=0}^{\infty} \frac{1}{k!} l^k z^k\right) - z\sum_{l=0}^{s} \sigma_l \left(\sum_{k=0}^{\infty} \frac{1}{k!} l^k z^k\right) \\ &= \sum_{k=0}^{\infty} \frac{1}{k!} \left(\sum_{l=0}^{s} l^k \rho_l\right) z^k - \sum_{k=1}^{\infty} \frac{1}{(k-1)!} \left(\sum_{l=0}^{s} l^{k-1}\sigma_l\right) z^k \\ &= \left(\sum_{l=0}^{s} \rho_l\right) + \sum_{k=1}^{\infty} \frac{1}{k!} \left(\sum_{l=0}^{s} l^k \rho_l - k\sum_{l=0}^{s} l^{k-1}\sigma_l\right) z^k.\end{aligned}$$

The theorem follows from (4.7). □

Example For the 2-step *Adams–Bashforth method* we have $\rho(w) = w^2 - w$, $\sigma(w) = \frac{3}{2}w - \frac{1}{2}$ and so

$$\rho(\mathrm{e}^z) - z\sigma(\mathrm{e}^z) = [1 + 2z + 2z^2 + \tfrac{4}{3}z^3] - [1 + z + \tfrac{1}{2}z^2 + \tfrac{1}{6}z^3] - \tfrac{3}{2}z[1 + z + \tfrac{1}{2}z^2] + \tfrac{1}{2}z + \mathcal{O}\big(z^4\big) = \tfrac{5}{12}z^3 + \mathcal{O}\big(z^4\big).$$

Hence the method is of order 2.

Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 8[1]

Definition We say that a polynomial obeys the *root condition* if all its zeros reside in $|w| \leq 1$ and all zeros of unit modulus are simple.

Theorem (The Dahlquist equivalence theorem) *The multistep method (4.5) is convergent iff it is of order $p \geq 1$ and the polynomial ρ obeys the root condition.*[2]

Example For the Adams–Bashforth method (see last lecture) we have $\rho(w) = (w-1)w$ and the root condition is obeyed. Also we saw that the Adams–Bashforth has order 2. By the Dahlquist equivalence theorem it is convergent.

Example *(Absence of convergence)* Consider the 2-step method

$$\boldsymbol{y}_{n+2} - 2\boldsymbol{y}_{n+1} + \boldsymbol{y}_n = 0. \tag{4.10}$$

Here $\rho(w) = w^2 - 2w + 1 = (w-1)^2$ and $\sigma(w) = 0$. We have $\rho(\mathrm{e}^z) - z\sigma(\mathrm{e}^z) = (\mathrm{e}^z - 1)^2 = (z + O(z^2))^2 = z^2 + O(z^3)$ and so the method has order 1. However ρ does not obey the root condition since the zero $w = 1$ has multiplicity 2. In fact the method (4.10) is obviously not convergent since it does not use the function $\boldsymbol{f}$ which defines the ODE!

A technique A useful procedure to generate multistep methods which are convergent and of high order is as follows. According to (4.6), order $p \geq 1$ implies $\rho(1) = 0$. Choose an arbitrary s-degree polynomial ρ that obeys the root condition and such that $\rho(1) = 0$. To maximize order, we let σ be the s-degree (alternatively, $(s-1)$-degree for explicit methods) polynomial arising from the truncation of the Taylor expansion of

$$\frac{\rho(w)}{\log w}$$

about the point $w = 1$. Thus, for example, for an *implicit method,*

$$\sigma(w) = \frac{\rho(w)}{\log w} + \mathcal{O}\big(|w-1|^{s+1}\big) \qquad \Rightarrow \qquad \rho(\mathrm{e}^z) - z\sigma(\mathrm{e}^z) = \mathcal{O}\big(z^{s+2}\big)$$

and (4.6) implies order at least $s+1$.

Example The choice $\rho(w) = w^{s-1}(w-1)$ corresponds to *Adams methods:* Adams–Bashforth methods if $\sigma_s = 0$, whence the order is s, otherwise order-$(s+1)$ (but implicit) Adams–Moulton methods. For example, letting $s = 2$ and $\xi = w - 1$, we obtain the 3rd-order Adams–Moulton method by expanding

$$\begin{aligned}\frac{w(w-1)}{\log w} &= \frac{\xi + \xi^2}{\log(1+\xi)} = \frac{\xi + \xi^2}{\xi - \frac{1}{2}\xi^2 + \frac{1}{3}\xi^3 - \cdots} = \frac{1+\xi}{1 - \frac{1}{2}\xi + \frac{1}{3}\xi^2 - \cdots} \\ &= (1+\xi)[1 + (\tfrac{1}{2}\xi - \tfrac{1}{3}\xi^2) + (\tfrac{1}{2}\xi - \tfrac{1}{3}\xi^2)^2 + \mathcal{O}\big(\xi^3\big)] = 1 + \tfrac{3}{2}\xi + \tfrac{5}{12}\xi^2 + \mathcal{O}\big(\xi^3\big) \\ &= 1 + \tfrac{3}{2}(w-1) + \tfrac{5}{12}(w-1)^2 + \mathcal{O}\big(|w-1|^3\big) = -\tfrac{1}{12} + \tfrac{2}{3}w + \tfrac{5}{12}w^2 + \mathcal{O}\big(|w-1|^3\big)\,.\end{aligned}$$

Therefore the 2-step, 3rd-order Adams–Moulton method is

$$\boldsymbol{y}_{n+2} - \boldsymbol{y}_{n+1} = h[-\tfrac{1}{12}\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \tfrac{2}{3}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) + \tfrac{5}{12}\boldsymbol{f}(t_{n+2}, \boldsymbol{y}_{n+2})].$$

BDF methods For reasons that will be made clear in the sequel, we wish to consider s-step, s-order methods s.t. $\sigma(w) = \sigma_s w^s$ for some $\sigma_s \in \mathbb{R} \setminus \{0\}$. In other words,

$$\sum_{l=0}^{s} \rho_l \boldsymbol{y}_{n+l} = h\sigma_s \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}), \qquad n = 0, 1, \ldots.$$

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

[2] If ρ obeys the root condition, the method (4.5) is sometimes said to be *zero-stable:* we will not use this terminology.

Such methods are called *backward differentiation formulae (BDF).*

Theorem *The explicit form of the s-step BDF method is*

$$\rho(w) = \sigma_s \sum_{l=1}^{s} \frac{1}{l} w^{s-l}(w-1)^l, \qquad \text{where} \qquad \sigma_s = \left(\sum_{l=1}^{s} \frac{1}{l} \right)^{-1}. \tag{4.11}$$

Proof We are looking for ρ such that the order condition $\rho(w) = \sigma_s w^s \log w + \mathcal{O}\big(|w-1|^{s+1}\big)$ for $w \to 1$ holds. Note that

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

With the choice of $\rho(w)$ given in (4.11) we get

$$\rho(w) - \sigma_s w^s \log(w) = -\sigma_s \sum_{l=s+1}^{\infty} \frac{1}{l}(w-1)^l w^{s-l} = \mathcal{O}\big(|w-1|^{s+1}\big) \quad (w \to 1)$$

and so the order condition is satisfied. The value of σ_s in (4.11) is such that $\rho_s = 1$. □

Example Let $s = 2$. Substitution in (4.11) yields $\sigma_2 = \frac{2}{3}$ and simple algebra results in $\rho(w) = w^2 - \frac{4}{3}w + \frac{1}{3}$. Hence the 2-step BDF is

$$\boldsymbol{y}_{n+2} - \tfrac{4}{3}\boldsymbol{y}_{n+1} + \tfrac{1}{3}\boldsymbol{y}_n = \tfrac{2}{3} h \boldsymbol{f}(t_{n+2}, \boldsymbol{y}_{n+2}).$$

Remark We cannot take it for granted that BDF methods are convergent. It is possible to prove that they are convergent iff $s \leq 6$. They *must not* be used outside this range!

4.3 Runge–Kutta methods

Recalling quadrature We may approximate

$$\int_0^h f(t)\mathrm{d}t \approx h \sum_{l=1}^{\nu} b_l f(c_l h),$$

where the weights b_l are chosen in accordance with an explicit formula from Lecture 5 (with weight function $w \equiv 1$). This *quadrature formula* is exact for all polynomials of degree $\nu - 1$ and, provided that $\prod_{k=1}^{\nu}(x - c_k)$ is orthogonal w.r.t. the weight function $w(x) \equiv 1$, $0 \leq x \leq 1$, the formula is exact for all polynomials of degree $2\nu - 1$.

Suppose that we wish to solve the 'ODE' $y' = f(t)$, $y(0) = y_0$. The exact solution is $y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t)\mathrm{d}t$ and we can approximate it by quadrature. In general, we obtain the time-stepping scheme

$$y_{n+1} = y_n + h \sum_{l=1}^{\nu} b_l f(t_n + c_l h) \qquad n = 0, 1, \ldots.$$

Here $h = t_{n+1} - t_n$ (the points t_n need not be equispaced). Can we generalize this to genuine ODEs of the form $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$?

Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 9[1]

Formally, $\boldsymbol{y}(t_{n+1}) = \boldsymbol{y}(t_n) + \displaystyle\int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{y}(t))\mathrm{d}t$, and this can be 'approximated' by

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\sum_{l=1}^{\nu} b_l \boldsymbol{f}(t_n + c_l h, \boldsymbol{y}(t_n + c_l h)). \tag{4.11}$$

except that, of course, the vectors $\boldsymbol{y}(t_n + c_l h)$ are unknown! *Runge–Kutta methods* are a means of implementing (4.11) by replacing unknown values of $\boldsymbol{y}$ by suitable linear combinations. The general form of a *ν-stage explicit Runge–Kutta method (RK)* is

$$\begin{aligned}
\boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{y}_n),\\
\boldsymbol{k}_2 &= \boldsymbol{f}(t_n + c_2 h, \boldsymbol{y}_n + h c_2 \boldsymbol{k}_1),\\
\boldsymbol{k}_3 &= \boldsymbol{f}(t_n + c_3 h, \boldsymbol{y}_n + h(a_{3,1}\boldsymbol{k}_1 + a_{3,2}\boldsymbol{k}_2)), \qquad a_{3,1} + a_{3,2} = c_3,\\
&\vdots\\
\boldsymbol{k}_\nu &= \boldsymbol{f}\left(t_n + c_\nu h, \boldsymbol{y}_n + h\sum_{j=1}^{\nu-1} a_{\nu,j}\boldsymbol{k}_j\right), \qquad \sum_{j=1}^{\nu-1} a_{\nu,j} = c_\nu,\\
\boldsymbol{y}_{n+1} &= \boldsymbol{y}_n + h\sum_{l=1}^{\nu} b_l \boldsymbol{k}_l.
\end{aligned}$$

The choice of the *RK coefficients* $a_{l,j}$ is motivated at the first instance by order considerations.

Example Set $\nu = 2$. We have $\boldsymbol{k}_1 = \boldsymbol{f}(t_n, \boldsymbol{y}_n)$ and, Taylor-expanding about $(t_n, \boldsymbol{y}_n)$,

$$\begin{aligned}
\boldsymbol{k}_2 &= \boldsymbol{f}(t_n + c_2 h, \boldsymbol{y}_n + c_2 h \boldsymbol{f}(t_n, \boldsymbol{y}_n))\\
&= \boldsymbol{f}(t_n, \boldsymbol{y}_n) + h c_2 \left[\frac{\partial \boldsymbol{f}(t_n, \boldsymbol{y}_n)}{\partial t} + \frac{\partial \boldsymbol{f}(t_n, \boldsymbol{y}_n)}{\partial \boldsymbol{y}}\boldsymbol{f}(t_n, \boldsymbol{y}_n)\right] + \mathcal{O}\big(h^2\big)\,.
\end{aligned}$$

But

$$\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y}) \qquad \Rightarrow \qquad \boldsymbol{y}'' = \frac{\partial \boldsymbol{f}(t, \boldsymbol{y})}{\partial t} + \frac{\partial \boldsymbol{f}(t, \boldsymbol{y})}{\partial \boldsymbol{y}}\boldsymbol{f}(t, \boldsymbol{y}).$$

Therefore, substituting the exact solution $\boldsymbol{y}_n = \boldsymbol{y}(t_n)$, we obtain $\boldsymbol{k}_1 = \boldsymbol{y}'(t_n)$ and $\boldsymbol{k}_2 = \boldsymbol{y}'(t_n) + h c_2 \boldsymbol{y}''(t_n) + \mathcal{O}\big(h^2\big)$. Consequently, the *local* error is

$$\begin{aligned}
\boldsymbol{y}(t_{n+1}) - (\boldsymbol{y}(t_n) + h b_1 \boldsymbol{k}_1 + h b_2 \boldsymbol{k}_2) = {}& [\boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n) + \tfrac{1}{2}h^2\boldsymbol{y}''(t_n) + \mathcal{O}\big(h^3\big)]\\
&- [\boldsymbol{y}(t_n) + h(b_1 + b_2)\boldsymbol{y}'(t_n) + h^2 b_2 c_2 \boldsymbol{y}''(t_n) + \mathcal{O}\big(h^3\big)].
\end{aligned}$$

We deduce that the RK method is of order 2 if $b_1 + b_2 = 1$ and $b_2 c_2 = \frac{1}{2}$. We can demonstrate that no such method may be of order ≥ 3. To show this consider the ODE $y' = y$ with $y(0) = 1$ whose solution is $y(t) = \mathrm{e}^t$. For this ODE we can write the local error explicitly: indeed we have $k_1 = f(t_n, y(t_n)) = \mathrm{e}^{t_n}$ and $k_2 = f(t_n + c_2 h, y(t_n) + c_2 h k_1) = y(t_n) + c_2 h k_1 = \mathrm{e}^{t_n}(1 + c_2 h)$. Then the local error is

$$\begin{aligned}
y(t_{n+1}) - (y(t_n) + h b_1 k_1 + h b_2 k_2) &= \mathrm{e}^{t_{n+1}} - \mathrm{e}^{t_n} - \mathrm{e}^{t_n}(h b_1 + h b_2 + h^2 b_2 c_2)\\
&= \mathrm{e}^{t_n}(\mathrm{e}^h - 1 - h(b_1 + b_2) - h^2(b_2 c_2))\\
&= \mathrm{e}^{t_n}\left(h(1 - b_1 - b_2) + h^2(1/2 - b_2 c_2) + \frac{h^3}{6} + \mathcal{O}\big(h^4\big)\right).
\end{aligned}$$

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

We see that there is no choice of b_1, b_2, c_2, c_2 that will make the term h^3 vanish, and so the method cannot have order ≥ 3.

General RK methods A general ν-stage *Runge–Kutta method* is

$$\boldsymbol{k}_l = \boldsymbol{f}\left(t_n + c_l h, \boldsymbol{y}_n + h\sum_{j=1}^{\nu} a_{l,j}\boldsymbol{k}_j\right) \quad \text{where} \quad \sum_{j=1}^{\nu} a_{l,j} = c_l, \qquad l = 1, 2, \ldots, \nu,$$

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\sum_{l=1}^{\nu} b_l \boldsymbol{k}_l.$$

Obviously, $a_{l,j} = 0$ for all $l \leq j$ yields the standard *explicit* RK. Otherwise, an RK method is said to be *implicit.*

4.4 Stiff equations

Consider the linear scalar system

$$\begin{cases} y' = \lambda y \\ y(0) = 1 \end{cases}$$

where $\lambda < 0$. The solution is $y(t) = \mathrm{e}^{\lambda t}$ which decays to 0 as $t \to \infty$. If we solve our ODE using a numerical method, we would like our sequence (y_n) to also decay to zero. For example with Euler's method we get $y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)y_n$ whose solution is $y_n = (1 + h\lambda)^n$. Thus the sequence y_n converges to 0 as $n \to \infty$ provided that $|1 + h\lambda| < 1$, i.e., $h < 2/|\lambda|$. For large λ this can be a severe restriction on h: for example for $\lambda = -1000$ this implies $h < 2/1000 = 0.002$.

Consider now the implicit Euler method. Here we have $y_{n+1} = y_n + h\lambda y_{n+1}$ which gives $y_{n+1} = (1 - h\lambda)^{-1} y_n$ and so $y_n = (1 - h\lambda)^{-n}$ which converges to 0 for any choice of $h > 0$ (we assumed $\lambda < 0$)!

Definition Suppose that a numerical method, applied to $y' = \lambda y$, $y(0) = 1$, with constant h, produces the solution sequence $\{y_n\}_{n\in\mathbb{Z}^+}$. We call the set

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

the *linear stability domain* of the method. Noting that the set of $\lambda \in \mathbb{C}$ for which $y(t) \overset{t\to\infty}{\longrightarrow} 0$ is the left half-plane $\mathbb{C}^- = \{z \in \mathbb{C} : \operatorname{Re} z < 0\}$, we say that the method is *A-stable* if $\mathbb{C}^- \subseteq \mathcal{D}$.

Example We have already seen that for the explicit Euler's method $y_n \to 0$ iff $|1 + h\lambda| < 1$, therefore $\mathcal{D} = \{z \in \mathbb{C} : |1 + z| < 1\}$ and the explicit Euler method is not A-stable. Moreover, solving $y' = \lambda y$ with the implicit Euler method we have seen that $y_n \to 0$ iff $|1 - h\lambda|^{-1} < 1$, therefore the linear stability domain is $\mathcal{D} = \{z \in \mathbb{C} : |1 - z| > 1\}$, hence the implicit Euler method is A-stable.

Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 10[1]

Example We have already seen that the linear stability domain of the explicit Euler's method is $\mathcal{D} = \{z \in \mathbb{C} : |1+z| < 1\}$ (not A-stable), and for the implicit Euler's method it is $\mathcal{D} = \{z \in \mathbb{C} : |1-z| > 1\}$ (A-stable). Consider now the trapezoidal rule: $\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{1}{2}h[\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})]$. Applied to $y' = \lambda y$ we get $y_{n+1} = [(1 + \frac{1}{2}h\lambda)/(1 - \frac{1}{2}h\lambda)]y_n$ thus, by induction, $y_n = [(1 + \frac{1}{2}h\lambda)/(1 - \frac{1}{2}h\lambda)]^n y_0$. Therefore

$$z \in \mathcal{D} \qquad \Leftrightarrow \qquad \left|\frac{1+\frac{1}{2}z}{1-\frac{1}{2}z}\right| < 1 \qquad \Leftrightarrow \qquad \operatorname{Re} z < 0$$

and we deduce that $\mathcal{D} = \mathbb{C}^-$. Hence, the method is A-stable.

Discussion A-stability analysis of multistep methods is considerably more complicated. However, according to the *second Dahlquist barrier,* no multistep method of order $p \geq 3$ may be A-stable. Note that the $p = 2$ barrier for A-stability is attained by the trapezoidal rule.

The Dahlquist barrier implies that, in our quest for higher-order methods with good stability properties, we need to pursue one of the following strategies:

- either relax the definition of A-stability
- or consider other methods in place of multistep.

The two courses of action will be considered next.

Stiffness and BDF methods Inasmuch as no multistep method of order $p \geq 3$ may be A-stable, stability properties of BDF, say, are satisfactory for most stiff equations. The point is that in many stiff linear systems in applications the eigenvalues are not just in $\mathbb{C}^-$ but also well away from $\mathrm{i}\mathbb{R}$. *[Analysis of nonlinear stiff equations is difficult and well outside the scope of this course.]* All BDF methods of order $p \leq 6$ (i.e., all convergent BDF methods) share the feature that the linear stability domain $\mathcal{D}$ includes a wedge about $(-\infty, 0)$: such methods are said to be *A_0-stable.*

Stiffness and Runge–Kutta Unlike multistep methods, implicit high-order RK may be A-stable. For example, consider the following 2-stage implicit RK method:

$$\begin{aligned} \boldsymbol{k}_1 &= \boldsymbol{f}\left(t_n, \boldsymbol{y}_n + \tfrac{1}{4}h(\boldsymbol{k}_1 - \boldsymbol{k}_2)\right), \\ \boldsymbol{k}_2 &= \boldsymbol{f}\left(t_n + \tfrac{2}{3}h, \boldsymbol{y}_n + \tfrac{1}{12}h(3\boldsymbol{k}_1 + 5\boldsymbol{k}_2)\right), \\ \boldsymbol{y}_{n+1} &= \boldsymbol{y}_n + \tfrac{1}{4}h(\boldsymbol{k}_1 + 3\boldsymbol{k}_2). \end{aligned}$$

One can show that this method is A-stable and has order 3. We first show that it is A-stable. Applying the method to $y' = \lambda y$, we have

$$\begin{aligned} hk_1 &= h\lambda\left(y_n + \tfrac{1}{4}hk_1 - \tfrac{1}{4}hk_2\right), \\ hk_2 &= h\lambda\left(y_n + \tfrac{1}{4}hk_1 + \tfrac{5}{12}hk_2\right). \end{aligned}$$

This is a linear system, whose solution is

$$\begin{bmatrix} hk_1 \\ hk_2 \end{bmatrix} = \begin{bmatrix} 1 - \frac{1}{4}h\lambda & \frac{1}{4}h\lambda \\ -\frac{1}{4}h\lambda & 1 - \frac{5}{12}h\lambda \end{bmatrix}^{-1} \begin{bmatrix} h\lambda y_n \\ h\lambda y_n \end{bmatrix} = \frac{h\lambda y_n}{1 - \frac{2}{3}h\lambda + \frac{1}{6}(h\lambda)^2} \begin{bmatrix} 1 - \frac{2}{3}h\lambda \\ 1 \end{bmatrix},$$

therefore

$$y_{n+1} = y_n + \tfrac{1}{4}hk_1 + \tfrac{3}{4}hk_2 = \frac{1 + \frac{1}{3}h\lambda}{1 - \frac{2}{3}h\lambda + \frac{1}{6}h^2\lambda^2} y_n.$$

Let

$$r(z) = \frac{1 + \frac{1}{3}z}{1 - \frac{2}{3}z + \frac{1}{6}z^2}.$$

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

Then $y_{n+1} = r(h\lambda)y_n$, therefore, by induction, $y_n = [r(h\lambda)]^n y_0$ and we deduce that

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

We wish to prove that $|r(z)| < 1$ for every $z \in \mathbb{C}^-$, since this is equivalent to A-stability. This will be done by a technique that can be applied to other RK methods. According to the *maximum modulus principle* from Complex Methods, if g is a nonconstant analytic function defined on an open set $\Omega \subset \mathbb{C}$, then $|g|$ has no maximum in Ω. We let $g = r$. This is a rational function, hence its only singularities are the poles $2 \pm \mathrm{i}\sqrt{2}$ and g is analytic in $\Omega = \mathbb{C}^- = \{z \in \mathbb{C} : \operatorname{Re} z < 0\}$. Thus to prove that the method is A-stable, it suffices to check that $\lim_{|z|\to\infty, \operatorname{Re}[z]<0} |r(z)| \le 1$ and that $|r(\mathrm{i}t)| \le 1$ for all $t \in \mathbb{R}$. The first condition is easy to check from the definition of r. For the second condition, we verify that

$$|r(\mathrm{i}t)|^2 \le 1 \qquad \Leftrightarrow \qquad |1 - \tfrac{2}{3}\mathrm{i}t - \tfrac{1}{6}t^2|^2 - |1 + \tfrac{1}{3}\mathrm{i}t|^2 \ge 0.$$

But $|1 - \frac{2}{3}\mathrm{i}t - \frac{1}{6}t^2|^2 - |1 + \frac{1}{3}\mathrm{i}t|^2 = \frac{1}{36}t^4 \ge 0$ and it follows that the method is A-stable.

Let us now show that the method has order 3. To do this we restrict our attention to scalar, autonomous equations of the form $y' = f(y)$. For brevity, we use the convention that all functions are evaluated at $y = y(t_n)$, e.g. $f_y = \mathrm{d}f(y(t_n))/\mathrm{d}y$. Thus,

$$\begin{aligned} k_1 &= f + \tfrac{1}{4}hf_y(k_1 - k_2) + \tfrac{1}{32}h^2 f_{yy}(k_1 - k_2)^2 + \mathcal{O}\big(h^3\big), \\ k_2 &= f + \tfrac{1}{12}hf_y(3k_1 + 5k_2) + \tfrac{1}{288}h^2 f_{yy}(3k_1 + 5k_2)^2 + \mathcal{O}\big(h^3\big). \end{aligned}$$

We have $k_1, k_2 = f + \mathcal{O}(h)$ and substitution in the above equations yields $k_1 = f + \mathcal{O}\big(h^2\big)$, $k_2 = f + \frac{2}{3}hf_y f + \mathcal{O}\big(h^2\big)$. Substituting again, we obtain

$$\begin{aligned} k_1 &= f - \tfrac{1}{6}h^2 f_y^2 f + \mathcal{O}\big(h^3\big), \\ k_2 &= f + \tfrac{2}{3}hf_y f + h^2\left(\tfrac{5}{18}f_y^2 f + \tfrac{2}{9}f_{yy}f^2\right) + \mathcal{O}\big(h^3\big) \\ \Rightarrow \quad y_{n+1} &= y + hf + \tfrac{1}{2}h^2 f_y f + \tfrac{1}{6}h^3(f_y^2 f + f_{yy}f^2) + \mathcal{O}\big(h^4\big). \end{aligned}$$

But $y' = f \Rightarrow y'' = f_y f \Rightarrow y''' = f_y^2 f + f_{yy}f^2$ and we deduce from Taylor's theorem that the method is at least of order 3. (It is easy to verify that it isn't of order 4, for example applying it to the equation $y' = \lambda y$.)

Example It is possible to prove that the 2-stage *Gauss–Legendre method*

$$\begin{aligned} \boldsymbol{k}_1 &= \boldsymbol{f}(t_n + (\tfrac{1}{2} - \tfrac{\sqrt{3}}{6})h, \boldsymbol{y}_n + \tfrac{1}{4}h\boldsymbol{k}_1 + (\tfrac{1}{4} - \tfrac{\sqrt{3}}{6})h\boldsymbol{k}_2), \\ \boldsymbol{k}_2 &= \boldsymbol{f}(t_n + (\tfrac{1}{2} + \tfrac{\sqrt{3}}{6})h, \boldsymbol{y}_n + (\tfrac{1}{4} + \tfrac{\sqrt{3}}{6})h\boldsymbol{k}_1 + \tfrac{1}{4}h\boldsymbol{k}_2), \\ \boldsymbol{y}_{n+1} &= \boldsymbol{y}_n + \tfrac{1}{2}h(\boldsymbol{k}_1 + \boldsymbol{k}_2) \end{aligned}$$

is of order 4. [You can do this for $y' = f(y)$ by expansion, but it becomes messy for $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$.] It can be easily verified that for $y' = \lambda y$ we have $y_n = [r(h\lambda)]^n y_0$, where $r(z) = (1 + \frac{1}{2}z + \frac{1}{12}z^2)/(1 - \frac{1}{2}z + \frac{1}{12}z^2)$. Since the poles of r reside at $3 \pm \mathrm{i}\sqrt{3}$ and $|r(\mathrm{i}t)| \equiv 1$, we can again use the maximum modulus principle to argue that $\mathcal{D} = \mathbb{C}^-$ and the Gauss–Legendre method is A-stable.

Mathematical Tripos Part IB: Lent 2019
Numerical Analysis – Lecture 11[1]

4.6 Implementation of ODE methods

The step size h is not some preordained quantity: it is a parameter of the method (in reality, many parameters, since we may vary it from step to step). The basic input of a well-written computer package for ODEs is not the step size but the *error tolerance:* the level of precision, as required by the user. The choice of $h > 0$ is an important tool at our disposal to keep a local estimate of the error beneath the required tolerance in the solution interval. In other words, we need not just a *time-stepping algorithm,* but also mechanisms for *error control* and for amending the step size.

The Milne device Suppose that we wish to monitor the error of the trapezoidal rule

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \tfrac{1}{2}h[\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})]. \tag{4.12}$$

We already know that the order is 2. Moreover, substituting the true solution we deduce that

$$\boldsymbol{y}(t_{n+1}) - \{\boldsymbol{y}(t_n) + \tfrac{1}{2}h[\boldsymbol{y}'(t_n) + \boldsymbol{y}'(t_{n+1})]\} = -\tfrac{1}{12}h^3\boldsymbol{y}'''(t_n) + \mathcal{O}\big(h^4\big)\,.$$

Therefore, the error in each step is increased roughly by $-\frac{1}{12}h^3\boldsymbol{y}'''(t_n)$. The number $c_{\rm TR} = -\frac{1}{12}$ is called the *error constant* of TR. Similarly, each multistep method (but not RK!) has its own error constant. For example, the 2nd order 2-step Adams–Bashforth method

$$\boldsymbol{y}_{n+1} - \boldsymbol{y}_n = \tfrac{1}{2}h[3\boldsymbol{f}(t_n, \boldsymbol{y}_n) - \boldsymbol{f}(t_{n-1}, \boldsymbol{y}_{n-1})], \tag{4.13}$$

has the error constant $c_{\rm AB} = \frac{5}{12}$.

The idea behind the *Milne device* is to use two multistep methods of the same order, one explicit and the second implicit (e.g., (4.13) and (4.12), respectively), to estimate the local error of the implicit method. For example, *locally,*

$$\begin{aligned}
\boldsymbol{y}^{\rm AB}_{n+1} &\approx \boldsymbol{y}(t_{n+1}) - c_{\rm AB}h^3\boldsymbol{y}'''(t_n) = \boldsymbol{y}(t_{n+1}) - \tfrac{5}{12}h^3\boldsymbol{y}'''(t_n),\\
\boldsymbol{y}^{\rm TR}_{n+1} &\approx \boldsymbol{y}(t_{n+1}) - c_{\rm TR}h^3\boldsymbol{y}'''(t_n) = \boldsymbol{y}(t_{n+1}) + \tfrac{1}{12}h^3\boldsymbol{y}'''(t_n).
\end{aligned}$$

Subtracting, we obtain the estimate $h^3\boldsymbol{y}'''(t_n) \approx -2(\boldsymbol{y}^{\rm AB}_{n+1} - \boldsymbol{y}^{\rm TR}_{n+1})$, therefore

$$\boldsymbol{y}^{\rm TR}_{n+1} - \boldsymbol{y}(t_{n+1}) \approx \tfrac{1}{6}(\boldsymbol{y}^{\rm TR}_{n+1} - \boldsymbol{y}^{\rm AB}_{n+1})$$

and we use the right hand side as an estimate of the local error.

Note that TR is a far better method than AB: it is A-stable, hence its *global* behaviour is superior. We employ AB *solely* to estimate the local error. This adds very little to the overall cost of TR, since AB is an explicit method.

Implementation of the Milne device We work with a *pair* of multistep methods of the same order, one explicit *(predictor)* and the other implicit *(corrector),* e.g.

$$\begin{aligned}
\text{Predictor}: &\quad \boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + h[\tfrac{5}{12}\boldsymbol{f}(t_{n-1}, \boldsymbol{y}_{n-1}) - \tfrac{4}{3}\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \tfrac{23}{12}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})],\\
\text{Corrector}: &\quad \boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + h[-\tfrac{1}{12}\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \tfrac{2}{3}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) + \tfrac{5}{12}\boldsymbol{f}(t_{n+2}, \boldsymbol{y}_{n+2})],
\end{aligned}$$

the third-order Adams–Bashforth and Adams–Moulton methods respectively.
The predictor is employed not just to estimate the error of the corrector, but also to provide *an initial guess in the solution of the implicit corrector equations.* Typically, for nonstiff equations, we iterate correction equations at most twice, while stiff equations require *iteration to convergence*, otherwise the typically superior stability features of the corrector are lost.

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

Let TOL > 0 be a user-specified *tolerance:* the maximal error allowed in approximating the ODE. Having completed a single step and estimated the error, there are three possibilities:

(a) $\frac{1}{10}\text{TOL} \le \|\,\text{error}\,\| \le \text{TOL}$, say: Accept the step, continue to t_{n+2} with the same step size.

(b) $\|\,\text{error}\,\| < \frac{1}{10}\text{TOL}$, say: Accept the step and increase the step length;

(c) $\|\,\text{error}\,\| > \text{TOL}$: Reject the step, recommence integration from t_n with smaller h.

Amending step size can be done easily with polynomial interpolation, although this means that we need to store past values well in excess of what is necessary for simple implementation of both multistep methods.

Error estimation per unit step Let $\boldsymbol{e}$ be our estimate of *local* error. Then $\boldsymbol{e}/h$ is our estimate for the global error in an interval of unit length. It is usual to require the latter quantity not to exceed TOL since good implementations of numerical ODEs should monitor the accumulation of *global* error. This is called *error estimation per unit step.*

Embedded Runge–Kutta methods The situation is more complicated with RK, since no single error constant determines local growth of the error. The approach of *embedded RK* requires, again, two (typically explicit) methods: an RK method of ν stages and order p, say, and another method, of $\nu + l$ stages, $l \ge 1$, and order $p + 1$, such that *the first ν stages of both methods are identical.* (This means that the cost of implementing the higher-order method is marginal, once we have computed the lower-order approximation.) For example, consider (and verify!)

$$\begin{aligned}
\boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{y}_n), \\
\boldsymbol{k}_2 &= \boldsymbol{f}\left(t_n + \tfrac{1}{2}h, \boldsymbol{y}_n + \tfrac{1}{2}h\boldsymbol{k}_1\right), \\
\boldsymbol{y}^{[1]}_{n+1} &= \boldsymbol{y}_n + h\boldsymbol{k}_2 && \Longrightarrow \quad \text{order 2}, \\
\boldsymbol{k}_3 &= \boldsymbol{f}(t_n + h, \boldsymbol{y}_n - h\boldsymbol{k}_1 + 2h\boldsymbol{k}_2), \\
\boldsymbol{y}^{[2]}_{n+1} &= \boldsymbol{y}_n + \tfrac{1}{6}h(\boldsymbol{k}_1 + 4\boldsymbol{k}_2 + \boldsymbol{k}_3) && \Longrightarrow \quad \text{order 3}.
\end{aligned}$$

We thus estimate $\boldsymbol{y}^{[1]}_{n+1} - \boldsymbol{y}(t_{n+1}) \approx \boldsymbol{y}^{[1]}_{n+1} - \boldsymbol{y}^{[2]}_{n+1}$. *[It might look paradoxical, at least at first glance, but the only purpose of the higher-order method is to provide error control for the lower-order one!]*

The Zadunaisky device Suppose that the ODE $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$, $\boldsymbol{y}(0) = \boldsymbol{y}_0$, is solved by an arbitrary numerical method of order p and that we have stored (not necessarily equidistant) past solution values $\boldsymbol{y}_n, \boldsymbol{y}_{n-1}, \ldots, \boldsymbol{y}_{n-p}$. We form an interpolating pth degree polynomial (with vector coefficients) $\boldsymbol{d}$ such that $\boldsymbol{d}(t_{n-i}) = \boldsymbol{y}_{n-i}$, $i = 0, 1, \ldots, p$, and consider the differential equation

$$\boldsymbol{z}' = \boldsymbol{f}(t, \boldsymbol{z}) + \boldsymbol{d}'(t) - \boldsymbol{f}(t, \boldsymbol{d}), \qquad \boldsymbol{z}(t_n) = \boldsymbol{y}_n. \tag{4.14}$$

There are two important observations with regard to (4.14)

(1) Since $\boldsymbol{d}(t) - \boldsymbol{y}(t) = \mathcal{O}\big(h^{p+1}\big)$, the term $\boldsymbol{d}'(t) - \boldsymbol{f}(t, \boldsymbol{d})$ is usually small (because $\boldsymbol{y}'(t) - \boldsymbol{f}(t, \boldsymbol{y}(t)) \equiv \boldsymbol{0}$). Therefore, (4.14) is a small perturbation of the original ODE.

(2) The exact solution of (4.14) is known: $\boldsymbol{z}(t) = \boldsymbol{d}(t)$.

Now, having produced $\boldsymbol{y}_{n+1}$ with our numerical method, we proceed to evaluate $\boldsymbol{z}_{n+1}$ as well, *using exactly the same method and implementation details.* We then evaluate the error in $\boldsymbol{z}_{n+1}$, namely $\boldsymbol{z}_{n+1} - \boldsymbol{d}(t_{n+1})$, and use it as an estimate of the error in $\boldsymbol{y}_{n+1}$.

Solving nonlinear algebraic systems We have already observed that the implementation of an implicit ODE method, whether multistep or RK, requires the solution of (in general, nonlinear) algebraic equations in each step. For example, for an s-step method, we need to solve in each step the algebraic system

$$\boldsymbol{y}_{n+s} = \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}) + \boldsymbol{v}, \tag{4.15}$$

where the vector $\boldsymbol{v}$ can be formed from past (hence known) solution values and their derivatives. The easiest approach is *functional iteration*

$$\boldsymbol{y}^{[j+1]}_{n+s} = \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}^{[j]}_{n+s}) + \boldsymbol{v}, \qquad j = 0, 1, \ldots,$$

where $\boldsymbol{y}^{[0]}_{n+s}$ is typically provided by the predictor scheme. It is very effective for *nonstiff* equations but fails for *stiff ODEs*, since the convergence of this iterative scheme requires similar restriction on h as that we strive to avoid by choosing an implicit method in the first place!

Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 12[1]

If the ODE is stiff, we might prefer a *Newton–Raphson* method. Let $\boldsymbol{\psi}(\boldsymbol{y}) = \boldsymbol{y} - \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}) - \boldsymbol{v}$ so the equation we want to solve is $\boldsymbol{\psi}(\boldsymbol{y}_{n+s}) = 0$. The Newton-Raphson method corresponds to the following iteration rule:

$$\boldsymbol{y}_{n+s}^{[j+1]} = \boldsymbol{y}_{n+s}^{[j]} - \left(\frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{y}} (\boldsymbol{y}_{n+s}^{[j]}) \right)^{-1} \boldsymbol{\psi}(\boldsymbol{y}_{n+s}^{[j]}). \tag{4.16}$$

The justification of the above is as follows: suppose that $\boldsymbol{y}_{n+s}^{[j]}$ is an approximation to the solution. We linearise $\boldsymbol{\psi}$ locally around $\boldsymbol{y}_{n+s}^{[j]}$ to get

$$\boldsymbol{\psi}(\boldsymbol{y}_{n+s}) \approx \boldsymbol{\psi}(\boldsymbol{y}_{n+s}^{[j]}) + \frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{y}} (\boldsymbol{y}_{n+s}^{[j]})(\boldsymbol{y}_{n+s} - \boldsymbol{y}_{n+s}^{[j]}).$$

Setting the right-hand side to zero we get (4.16).

The snag is that repeatedly evaluating and inverting (i.e. LU-factorizing) the Jacobian matrix $\frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{y}}$ in every iteration is *very* expensive. The remedy is to implement the *modified Newton–Raphson method*, namely

$$\boldsymbol{y}_{n+s}^{[j+1]} = \boldsymbol{y}_{n+s}^{[j]} - \left(\frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{y}} (\boldsymbol{y}_{n+s}^{[0]}) \right)^{-1} \boldsymbol{\psi}(\boldsymbol{y}_{n+s}^{[j]}). \tag{4.17}$$

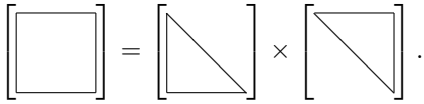
Thus, the Jacobian need be evaluated only *once* a step.

Important observation for future use: Implementation of (4.17) requires repeated solution of linear algebraic systems *with the same matrix.* We will soon study LU factorization of matrices, and there this remark will be appreciated as important and lead to substantial savings. For stiff equations it is much cheaper to solve nonlinear algebraic equations with (4.17) than using a minute step size with a 'bad' (e.g., explicit multistep or explicit RK) method.

5 Numerical linear algebra

5.1 LU factorization and its generalizations

Let A be a real $n \times n$ matrix. We say that the $n \times n$ matrices L and U are an *LU factorization* of A if ***(1)*** L is *unit* lower triangular, i.e., $L_{i,j} = 0$ for $i < j$ and $L_{ii} = 1$ for all i, ***(2)*** U is upper triangular, $U_{i,j} = 0$, $i > j$; and ***(3)*** $A = LU$. Therefore the factorization takes the form



Application 1 Calculation of a determinant: $\det A = (\det L)(\det U) = (\prod_{k=1}^{n} L_{k,k}) \cdot (\prod_{k=1}^{n} U_{k,k})$. This is much faster than the using the formula

$$\det A = \sum_{\sigma} \operatorname{sign}(\sigma) A_{1,\sigma(1)} \dots A_{n,\sigma(n)} \tag{5.1}$$

where the summation is over all permutations σ of $\{1, \dots, n\}$. The number of terms in the sum is $n!$. On a 10^9 flop/sec. computer (flop = floating point operation) evaluating (5.1) would take 4×10^5 years for a matrix of size $n = 30$!

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

Application 2 Testing for nonsingularity: $A = LU$ is nonsingular iff all the diagonal elements of L and U are nonzero.

Application 3 Solution of linear systems: Let $A = LU$ and suppose we wish to solve $A\boldsymbol{x} = \boldsymbol{b}$. This is the same as $L(U\boldsymbol{x}) = \boldsymbol{b}$, which we decompose into $L\boldsymbol{y} = \boldsymbol{b}$, $U\boldsymbol{x} = \boldsymbol{y}$. Both latter systems are triangular and can be calculated easily. Thus, $L_{1,1}y_1 = b_1$ gives y_1, next $L_{2,1}y_1 + L_{2,2}y_2 = b_2$ yields y_2 etc. Having found $\boldsymbol{y}$, we solve for $\boldsymbol{x}$ in reverse order: $U_{n,n}x_n = y_n$ gives x_n, $U_{n-1,n-1}x_{n-1}+U_{n-1,n}x_n = y_{n-1}$ produces x_{n-1} and so on. This requires $\mathcal{O}(n^2)$ computational operations (usually we only bother to count multiplications/divisions).

Application 4 The inverse of A: It is straightforward to devise a direct way of calculating the inverse of triangular matrices, subsequently forming $A^{-1} = U^{-1}L^{-1}$.

The calculation of LU factorization We denote the *columns* of L by $\boldsymbol{l}_1, \boldsymbol{l}_2, \ldots, \boldsymbol{l}_n$ and the *rows* of U by $\boldsymbol{u}_1^\top, \boldsymbol{u}_2^\top, \ldots, \boldsymbol{u}_n^\top$. Hence

$$A = LU = \begin{bmatrix} \boldsymbol{l}_1 & \boldsymbol{l}_2 & \cdots & \boldsymbol{l}_n \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_1^\top \\ \boldsymbol{u}_2^\top \\ \vdots \\ \boldsymbol{u}_n^\top \end{bmatrix} = \sum_{k=1}^{n} \boldsymbol{l}_k \boldsymbol{u}_k^\top. \tag{5.2}$$

Since the first $k-1$ components of $\boldsymbol{l}_k$ and $\boldsymbol{u}_k$ are all zero, each rank-one matrix $\boldsymbol{l}_k\boldsymbol{u}_k^\top$ has zeros in its first $k-1$ rows and columns. We begin our calculation by extracting $\boldsymbol{l}_1$ and $\boldsymbol{u}_1^\top$ from A, and then proceed similarly to extract $\boldsymbol{l}_2$ and $\boldsymbol{u}_2^\top$, etc.

First we note that since the leading $k-1$ elements of $\boldsymbol{l}_k$ and $\boldsymbol{u}_k$ are zero for $k \geq 2$, it follows from (5.2) that $\boldsymbol{u}_1^\top$ is the first row of A and $\boldsymbol{l}_1$ is the first column of A, divided by $A_{1,1}$ (so that $L_{1,1} = 1$).

Next, having found $\boldsymbol{l}_1$ and $\boldsymbol{u}_1$, we form the matrix $A_1 = A - \boldsymbol{l}_1\boldsymbol{u}_1^\top = \sum_{k=2}^{n} \boldsymbol{l}_k\boldsymbol{u}_k^\top$. The first row & column of A_1 are zero and it follows that $\boldsymbol{u}_2^\top$ is the second row of A_1, while $\boldsymbol{l}_2$ is its second column, scaled so that $L_{2,2} = 1$.

We can thus summarize the LU decomposition algorithm as follows: Set $A_0 := A$. For all $k = 1, 2, \ldots, n$ set $\boldsymbol{u}_k^\top$ to the kth row of A_{k-1} and $\boldsymbol{l}_k$ to the kth column of A_{k-1}, scaled so that $L_{k,k} = 1$. Set $A_k := A_{k-1} - \boldsymbol{l}_k\boldsymbol{u}_k^\top$ and increment k.

At each step k, the dominant cost is to form $\boldsymbol{l}_k\boldsymbol{u}_k^\top$. Since the first $k-1$ components of $\boldsymbol{l}_k$ and $\boldsymbol{u}_k$ are zero the cost of forming this rank-one matrix is $(n-k+1)^2$. Thus the total cost of the algorithm is $\sum_{k=1}^{n}(n-k+1)^2 = \sum_{j=1}^{n} j^2 = \mathcal{O}(n^3)$.

Relation to Gaussian elimination In Gaussian elimination, we perform a series of elementary row operations on A to transform it into an upper triangular matrix. Each elementary row operation consists in adding a multiple of the k'th row to the j'th row ($j > k$). One can easily show that such operations can be represented using unit lower triangular matrices. Thus Gaussian elimination can be written concisely as:

$$L_n L_{n-1} \ldots L_1 A = U$$

where each L_k is unit lower triangular and U is upper triangular. This gives $A = LU$ where $L = L_1^{-1} \ldots L_n^{-1}$ is unit lower triangular. In the algorithm described above, the matrix A_k is the matrix obtained after k steps of Gaussian elimination, except for the first $k-1$ rows and columns which are zero in A_k. The only difference between Gaussian elimination and LU is that Gaussian elimination is usually applied to a linear system $A\boldsymbol{x} = \boldsymbol{b}$ and the lower triangular matrices are not stored. One advantage of using LU decomposition is that it can be reused for different right-hand sides: in Gaussian elimination the solution for each new $\boldsymbol{b}$ would require $\mathcal{O}(n^3)$ computational operations, whereas with LU factorization $\mathcal{O}(n^3)$ operations are required for the initial factorization, but then the solution for each new $\boldsymbol{b}$ only requires just $\mathcal{O}(n^2)$ (forward/backward substitution).

Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 13[1]

Pivoting Naive LU factorization fails when, for example, $A_{1,1} = 0$. The remedy is to exchange rows of A, a technique called *pivoting*. Specifically, at the k'th step of the algorithm we look for another row $p \geq k$ such that the entry $(A_{k-1})_{p,k}$ is nonzero. We permute rows p and k and proceed. The algorithm with pivoting can thus be written as follows:

- Let $A_0 = A$.
- For $k = 1, \ldots, n$: find $p \geq k$ such that $(A_{k-1})_{p,k} \neq 0$. Let P_k be the permutation matrix[2] that swaps positions k and p. Let $\boldsymbol{u}_k^\top$ be the k'th row of $P_k A_{k-1}$ and $\boldsymbol{l}_k$ be $\frac{1}{(P_k A_{k-1})_{k,k}} \times$ (k'th column of $P_k A_{k-1}$). Set $A_k = P_k A_{k-1} - \boldsymbol{l}_k \boldsymbol{u}_k^T$.

If we unroll the algorithm we have $A_1 = P_1 A_0 - \boldsymbol{l}_1 \boldsymbol{u}_1^T$, $A_2 = P_2 P_1 A - P_2 \boldsymbol{l}_1 \boldsymbol{u}_1^\top - \boldsymbol{l}_2 \boldsymbol{u}_2^T$, etc. and at the end, since $A_n = 0$ (and P_n the identity matrix):

$$P_{n-1} \cdots P_1 A = \tilde{\boldsymbol{l}}_{\boldsymbol{1}} \boldsymbol{u}_1^\top + \cdots + \tilde{\boldsymbol{l}}_{\boldsymbol{n}} \boldsymbol{u}_n^\top \tag{5.2}$$

where $\tilde{\boldsymbol{l}}_{\boldsymbol{k}} = P_{n-1} \ldots P_{k+1} \boldsymbol{l}_k$. Note that the first $k-1$ components of $\tilde{\boldsymbol{l}}_{\boldsymbol{k}}$ are zero since this is the case for $\boldsymbol{l}_k$ and since the permutations $P_{k+1}, \ldots, P_{n-1}$ only permute components of index $\geq k+1$. Therefore, Equation (5.2) can be rewritten as:

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

where $P = P_{n-1} \ldots P_1$ is a permutation matrix, and $\tilde{L} = [\tilde{\boldsymbol{l}}_{\boldsymbol{1}} \ \ \ldots \ \ \tilde{\boldsymbol{l}}_{\boldsymbol{n}}]$ is unit lower triangular, and U is upper triangular.

There is one situation where the algorithm above can still fail: this if for some k, *all* the entries in the k'th column of A_{k-1} are zero. In this case one can choose $\boldsymbol{l}_k$ to be the vector with a 1 at position k and zero elsewhere, and choose $\boldsymbol{u}_k^\top$ to be the k'th row of A_{k-1}, and $P_k = I$ (identity matrix). With this choice, the first k rows and columns of $A_k = A_{k-1} - \boldsymbol{l}_k \boldsymbol{u}_k^\top$ become zero as desired (this is not the only choice of $P_k, \boldsymbol{l}_k, \boldsymbol{u}_k$ that works in this case; other choices are possible).

We have thus shown that for any matrix A (even singular) one can find a permutation matrix P such that PA has an LU factorization.

Pivoting is not only important to find an element that is nonzero, but also for the overall numerical stability of the algorithm. A common choice of pivot p is to take $p \geq k$ such that $|(A_{k-1})_{p,k}|$ is maximum. This ensures in particular that the entries of $\boldsymbol{l}_k$ are all bounded above by 1 in magnitude.

Symmetric matrices Let A be an $n \times n$ symmetric matrix (i.e., $A_{k,\ell} = A_{\ell,k}$). An analogue of LU factorization that takes advantage of symmetry consists in expressing A in the form of the product $LDL^\top$, where L is $n \times n$ lower triangular, with ones on its diagonal and D is a diagonal matrix. This is a special case of an LU factorization with $U = DL^\top$. If we let $\boldsymbol{l}_1, \ldots, \boldsymbol{l}_n$ be the columns of L then this factorization takes the form $A = \sum_{k=1}^n D_{k,k} \boldsymbol{l}_k \boldsymbol{l}_k^\top$. To compute this factorization, we can use an algorithm very similar to the one for the computation of LU factorization (without pivoting): Set $A_0 = A$ and for $k = 1, 2, \ldots, n$ let $\boldsymbol{l}_k$ be the multiple of the kth column of A_{k-1} such that $L_{k,k} = 1$. Set $D_{k,k} = (A_{k-1})_{k,k}$ and form $A_k = A_{k-1} - D_{k,k} \boldsymbol{l}_k \boldsymbol{l}_k^\top$.

Example Let $A = A_0 = \begin{bmatrix} 2 & 4 \\ 4 & 11 \end{bmatrix}$. Hence $\boldsymbol{l}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $D_{1,1} = 2$ and

$$A_1 = A_0 - D_{1,1} \boldsymbol{l}_1 \boldsymbol{l}_1^\top = \begin{bmatrix} 2 & 4 \\ 4 & 11 \end{bmatrix} - 2 \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 3 \end{bmatrix}.$$

We deduce that $\boldsymbol{l}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $D_{2,2} = 3$ and $A = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$.

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

[2] A permutation matrix is a matrix with exactly one 1 in each row and in each column; the remaining entries being 0. For example $P = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ is a permutation matrix and PA exchanges the two rows of A.

Symmetric positive definite matrices Recall: A is positive definite if $\boldsymbol{x}^\top A\boldsymbol{x} > 0$ for all $\boldsymbol{x} \neq \boldsymbol{0}$.

Theorem Let A be a real $n \times n$ symmetric matrix. It is positive definite if and only if it has an $LDL^\top$ factorization in which the diagonal elements of D are all positive.

Proof. Suppose that $A = LDL^\top$ and let $\boldsymbol{x} \in \mathbb{R}^n \setminus \{\boldsymbol{0}\}$. Since L is nonsingular (it is lower triangular and all diagonal elements are equal to 1), $\boldsymbol{y} := L^\top \boldsymbol{x} \neq \boldsymbol{0}$. Then $\boldsymbol{x}^\top A\boldsymbol{x} = \boldsymbol{y}^\top D\boldsymbol{y} = \sum_{k=1}^n D_{k,k} y_k^2 > 0$, hence A is positive definite.
Conversely, suppose that A is positive definite. We wish to demonstrate that an $LDL^\top$ factorization exists. We denote by $\boldsymbol{e}_k \in \mathbb{R}^n$ the kth unit vector. Hence $\boldsymbol{e}_1^\top A\boldsymbol{e}_1 = A_{1,1} > 0$ and $\boldsymbol{l}_1$ & $D_{1,1}$ are well defined. We now show that $(A_{k-1})_{k,k} > 0$ for $k = 1, 2, \ldots$. This is true for $k = 1$ and we continue by induction, assuming that $A_{k-1} = A - \sum_{j=1}^{k-1} D_{j,j}\boldsymbol{l}_j\boldsymbol{l}_j^\top$ has been computed successfully.

Define $\boldsymbol{x} \in \mathbb{R}^n$ as the solution of the following system of equations: $\boldsymbol{l}_j^\top \boldsymbol{x} = 0$, $j = 1, \ldots, k-1$, $x_k = 1$ and $x_j = 0$ for $j = k+1, \ldots, n$. This is a system of n linear equations in the unknown $\boldsymbol{x} \in \mathbb{R}^n$. The matrix of this system of equations is upper triangular with ones on the diagonal hence it is invertible and our system has a unique solution. Now observe that since the first $k-1$ rows & columns of A_{k-1} vanish, and since $x_k = 1$ and the components $k+1, \ldots, n$ of $\boldsymbol{x}$ vanish we have $(A_{k-1})_{k,k} = \boldsymbol{x}^\top A_{k-1}\boldsymbol{x}$. Thus, from the definition of A_{k-1} and the choice of $\boldsymbol{x}$,

$$(A_{k-1})_{k,k} = \boldsymbol{x}^\top A_{k-1}\boldsymbol{x} = \boldsymbol{x}^\top \left(A - \sum_{j=1}^{k-1} D_{j,j}\boldsymbol{l}_j\boldsymbol{l}_j^\top \right) \boldsymbol{x} = \boldsymbol{x}^\top A\boldsymbol{x} - \sum_{j=1}^{k-1} D_{j,j}(\boldsymbol{l}_j^\top \boldsymbol{x})^2 = \boldsymbol{x}^\top A\boldsymbol{x} > 0,$$

as required. Hence $(A_{k-1})_{k,k} > 0$, $k = 1, 2, \ldots, n$, and the factorization exists. □

Conclusion It is possible to check if a symmetric matrix is positive definite by trying to form its $LDL^\top$ factorization.

Cholesky factorization Define $D^{1/2}$ as the diagonal matrix whose (k,k) element is $D_{k,k}^{1/2}$, hence $D^{1/2}D^{1/2} = D$. Then, A being positive definite, we can write

$$A = (LD^{1/2})(D^{1/2}L^\top) = (LD^{1/2})(LD^{1/2})^\top.$$

In other words, letting $\tilde{L} := LD^{1/2}$, we obtain the *Cholesky factorization* $A = \tilde{L}\tilde{L}^\top$.

Sparse matrices It is often required to solve *very* large systems $A\boldsymbol{x} = \boldsymbol{b}$ ($n = 10^5$ is considered small in this context!) where nearly all the elements of A are zero. Such a matrix is called *sparse* and efficient solution of $A\boldsymbol{x} = \boldsymbol{b}$ should exploit sparsity. In particular, we wish the matrices L and U to inherit as much as possible of the sparsity of A and for the cost of computation to be determined by the number of nonzero entries, rather than by n. The following theorem shows that certain zeros of A are always inherited by an LU factorization.

Theorem Let $A = LU$ be an LU factorization (without pivoting) of a sparse matrix. Then all leading zeros in the rows of A to the left of the diagonal are inherited by L and all the leading zeros in the columns of A above the diagonal are inherited by U.

Proof We assume that $U_{k,k} \neq 0$ for all $k = 1, \ldots, n$ which is the same as saying that $(A_{k-1})_{k,k} \neq 0$ when running the LU factorization algorithm (without pivoting). If $A_{i,1} = 0$ this means that $L_{i,1}U_{1,1} = 0$ and so $L_{i,1} = 0$. If furthermore $A_{i,2} = 0$ we get $L_{i,1}U_{1,2} + L_{i,2}U_{2,2} = 0$ which implies $L_{i,2} = 0$ since $L_{i,1} = 0$. In general we get that if $A_{i,1} = \cdots = A_{i,j} = 0$ where $j < i$ then $L_{i,1} = \cdots = L_{i,j} = 0$. A similar reasoning applies for leading zeros in the columns of A above the diagonal. □

Banded matrices The matrix A is a *banded matrix* if there exists an integer $r < n$ such that $A_{i,j} = 0$ for $|i-j| > r$, $i, j = 1, 2, \ldots, n$. In other words, all the nonzero elements of A reside in a band of width $2r+1$ along the main diagonal. In that case, according to the previous theorem, $A = LU$ implies that $L_{i,j} = U_{i,j} = 0$ $\forall\ |i-j| > r$ and sparsity structure is inherited by the factorization.
In general, the expense of calculating an LU factorization of an $n \times n$ *dense* matrix A is $\mathcal{O}(n^3)$ operations and the expense of solving $A\boldsymbol{x} = \boldsymbol{b}$, provided that the factorization is known, is $\mathcal{O}(n^2)$. However, in the case of a banded A, we need just $\mathcal{O}(r^2 n)$ operations to factorize and $\mathcal{O}(rn)$ operations to solve a linear system. If $r \ll n$ this represents a very substantial saving!

Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 14[1]

Sparse matrices It is often required to solve *very* large systems $A\boldsymbol{x} = \boldsymbol{b}$ ($n = 10^5$ is considered small in this context!) where nearly all the elements of A are zero. Such a matrix is called *sparse* and efficient solution of $A\boldsymbol{x} = \boldsymbol{b}$ should exploit sparsity. In particular, we wish the matrices L and U to inherit as much as possible of the sparsity of A and for the cost of computation to be determined by the number of nonzero entries, rather than by n. The following theorem shows that certain zeros of A are always inherited by an LU factorization.

Theorem Let $A = LU$ be an LU factorization (without pivoting) of a sparse matrix. Then all leading zeros in the rows of A to the left of the diagonal are inherited by L and all the leading zeros in the columns of A above the diagonal are inherited by U.

Proof We assume that $U_{k,k} \neq 0$ for all $k = 1, \ldots, n$ which is the same as saying that $(A_{k-1})_{k,k} \neq 0$ when running the LU factorization algorithm (without pivoting). If $A_{i,1} = 0$ this means that $L_{i,1}U_{1,1} = 0$ and so $L_{i,1} = 0$. If furthermore $A_{i,2} = 0$ we get $L_{i,1}U_{1,2} + L_{i,2}U_{2,2} = 0$ which implies $L_{i,2} = 0$ since $L_{i,1} = 0$. In general we get that if $A_{i,1} = \cdots = A_{i,j} = 0$ where $j < i$ then $L_{i,1} = \cdots = L_{i,j} = 0$. A similar reasoning applies for leading zeros in the columns of A above the diagonal. □

Banded matrices The matrix A is a *banded matrix* if there exists an integer $r < n$ such that $A_{i,j} = 0$ for $|i - j| > r$, $i, j = 1, 2, \ldots, n$. In other words, all the nonzero elements of A reside in a band of width $2r + 1$ along the main diagonal. In that case, according to the previous theorem, $A = LU$ implies that $L_{i,j} = U_{i,j} = 0$ $\forall\ |i - j| > r$ and sparsity structure is inherited by the factorization.
In general, the expense of calculating an LU factorization of an $n \times n$ *dense* matrix A is $\mathcal{O}(n^3)$ operations and the expense of solving $A\boldsymbol{x} = \boldsymbol{b}$, provided that the factorization is known, is $\mathcal{O}(n^2)$. However, in the case of a banded A, we need just $\mathcal{O}(r^2 n)$ operations to factorize and $\mathcal{O}(rn)$ operations to solve a linear system. If $r \ll n$ this represents a very substantial saving!

General sparse matrices feature a wide range of applications, e.g. the solution of partial differential equations, and there exists a wealth of methods for their solution. One approach is efficient factorization, that minimizes *fill-in* (a fill-in is an zero entry of the matrix A that gets *filled in* during the factorization, i.e., $A_{ij} = 0$ and yet $L_{ij} \neq 0$ (if $i > j$) or $U_{ij} \neq 0$ (if $j > i$)). Yet another is to use iterative methods (cf. Part II Numerical Analysis course). There also exists a substantial body of other, highly effective methods, e.g. Fast Fourier Transforms, preconditioned conjugate gradients and multigrid techniques (cf. Part II Numerical Analysis course), fast multipole techniques and much more.

Sparsity and graph theory An exceedingly powerful (and beautiful) methodology of ordering pivots to minimize fill-in of sparse matrices uses graph theory and, like many other cool applications of mathematics in numerical analysis, is alas not in the schedules `:-(`

5.2 QR factorization of matrices

Scalar products, norms and orthogonality We first recall a few definitions. $\mathbb{R}^n$ is the linear space of all real n-tuples.

- For all $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ we define the *scalar product*

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \langle \boldsymbol{v}, \boldsymbol{u} \rangle = \sum_{j=1}^{n} u_j v_j = \boldsymbol{u}^\top \boldsymbol{v} = \boldsymbol{v}^\top \boldsymbol{u}\,.$$

- The vectors $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_m \in \mathbb{R}^n$ are *orthonormal* if

$$\langle \boldsymbol{q}_k, \boldsymbol{q}_\ell \rangle = \begin{cases} 1, & k = \ell, \\ 0, & k \neq \ell, \end{cases} \qquad k, \ell = 1, 2, \ldots, m.$$

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

- An $n \times n$ real matrix Q is *orthogonal* if all its columns are orthonormal. Since $(Q^\top Q)_{k,\ell} = \langle \boldsymbol{q}_k, \boldsymbol{q}_\ell \rangle$, this implies that $Q^\top Q = I$ (I is the *unit matrix*). Hence $Q^{-1} = Q^\top$ and $QQ^\top = QQ^{-1} = I$. We conclude that the rows of an orthogonal matrix are also orthonormal, and that $Q^\top$ is an orthogonal matrix. Further, $1 = \det I = \det(QQ^\top) = \det Q \det Q^\top = (\det Q)^2$, and thus we deduce that $\det Q = \pm 1$, and that an orthogonal matrix is nonsingular.

The QR factorization The QR factorization of an $m \times n$ matrix A has the form $A = QR$, where Q is an $m \times m$ *orthogonal* matrix and R is an $m \times n$ *upper triangular* matrix (i.e., $R_{i,j} = 0$ for $i > j$). When $m \geq n$, a *reduced QR factorization* of A is a factorization $A = QR$ where Q is $m \times n$ with orthonormal columns, and R is $n \times n$ upper triangular.

Application in linear system solving Let $m = n$ and A be nonsingular. We can solve $A\boldsymbol{x} = \boldsymbol{b}$ by calculating the QR factorization of A and solving first $Q\boldsymbol{y} = \boldsymbol{b}$ (hence $\boldsymbol{y} = Q^\top \boldsymbol{b}$) and then $R\boldsymbol{x} = \boldsymbol{y}$ (a triangular system!).

Interpretation of the QR factorization Let $m \geq n$ and denote the columns of A and Q by $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n$ and $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n$ respectively. In a reduced QR factorization:

$$\begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{q}_1 & \boldsymbol{q}_2 & \cdots & \boldsymbol{q}_n \end{bmatrix} \begin{bmatrix} R_{1,1} & R_{1,2} & \cdots & R_{1,n} \\ 0 & R_{2,2} & & \vdots \\ \vdots & \ddots & \ddots & \\ & & 0 & R_{n,n} \end{bmatrix},$$

we have $\boldsymbol{a}_k = \sum_{j=1}^{k} R_{j,k} \boldsymbol{q}_j$, $k = 1, 2, \ldots, n$. In other words, Q has the property that each kth column of A can be expressed as a linear combination of the first k columns of Q.

The Gram–Schmidt algorithm Assume that $m \geq n$ and that the columns of A are linearly independent. We will see how to construct a reduced QR factorization of A, i.e., $Q \in \mathbb{R}^{m \times n}$ having orthonormal columns, $R \in \mathbb{R}^{n \times n}$ upper-triangular and $A = QR$: in other words,

$$\sum_{k=1}^{\ell} R_{k,\ell} \boldsymbol{q}_k = \boldsymbol{a}_\ell, \quad \ell = 1, 2, \ldots, n, \qquad \text{where} \qquad A = \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_n \end{bmatrix}. \tag{5.2}$$

Equation (5.2) for $\ell = 1$ tells us that we must have $\boldsymbol{q}_1 = \boldsymbol{a}_1 / \|\boldsymbol{a}_1\|$ and $R_{1,1} = \|\boldsymbol{a}_1\|$. Next we form the vector $\boldsymbol{b} = \boldsymbol{a}_2 - \langle \boldsymbol{q}_1, \boldsymbol{a}_2 \rangle \boldsymbol{q}_1$. It is orthogonal to $\boldsymbol{q}_1$, since $\langle \boldsymbol{q}_1, \boldsymbol{a}_2 - \langle \boldsymbol{q}_1, \boldsymbol{a}_2 \rangle \boldsymbol{q}_1 \rangle = \langle \boldsymbol{q}_1, \boldsymbol{a}_2 \rangle - \langle \boldsymbol{q}_1, \boldsymbol{a}_2 \rangle \langle \boldsymbol{q}_1, \boldsymbol{q}_1 \rangle = 0$. Since the columns of A are assumed linearly independent, $\boldsymbol{b} \neq \boldsymbol{0}$ and we set $\boldsymbol{q}_2 = \boldsymbol{b} / \|\boldsymbol{b}\|$, hence $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ are orthonormal. Moreover,

$$\langle \boldsymbol{q}_1, \boldsymbol{a}_2 \rangle \boldsymbol{q}_1 + \|\boldsymbol{b}\| \boldsymbol{q}_2 = \langle \boldsymbol{q}_1, \boldsymbol{a}_2 \rangle \boldsymbol{q}_1 + \boldsymbol{b} = \boldsymbol{a}_2,$$

hence, to obey (5.2) for $\ell = 2$, we let $R_{1,2} = \langle \boldsymbol{q}_1, \boldsymbol{a}_2 \rangle$, $R_{2,2} = \|\boldsymbol{b}\|$.

More generally we get the following classical Gram-Schmidt algorithm to compute a QR factorization: Set $\boldsymbol{q}_1 = \boldsymbol{a}_1 / \|\boldsymbol{a_1}\|$ and $R_{11} = \|\boldsymbol{a_1}\|$. For $j = 2, \ldots, n$: Set $R_{ij} = \langle \boldsymbol{q}_i, \boldsymbol{a}_j \rangle$ for $i \leq j - 1$, and $\boldsymbol{b}_j = \boldsymbol{a}_j - \sum_{i=1}^{j-1} R_{ij} \boldsymbol{q}_i$. Set $\boldsymbol{q}_j = \boldsymbol{b}_j / \|\boldsymbol{b}_j\|$ and $R_{jj} = \|\boldsymbol{b}_j\|$.

The total cost of the classical Gram–Schmidt algorithm is $\mathcal{O}\big(n^2 m\big)$, since at each iteration j a total of $\mathcal{O}(mj)$ operations are performed.

The disadvantage of the classical Gram–Schmidt is its *ill-conditioning*: using finite arithmetic, small imprecisions in the calculation of inner products spread rapidly, leading to effective loss of orthogonality. Errors accumulate fast and the computed off-diagonal elements of $Q^\top Q$ may become large.

Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 15[1]

The Gram–Schmidt algorithm Assume that $m \geq n$ and that the columns of $A \in \mathbb{R}^{m\times n}$ are linearly independent. We will see how to construct a reduced QR factorization of A, i.e., $Q \in \mathbb{R}^{m\times n}$ having orthonormal columns, $R \in \mathbb{R}^{n\times n}$ upper-triangular and $A = QR$: in other words,

$$\sum_{k=1}^{\ell} R_{k,\ell}\boldsymbol{q}_k = \boldsymbol{a}_\ell, \quad \ell = 1, 2, \ldots, n, \qquad \text{where} \qquad A = [\ \boldsymbol{a}_1 \quad \boldsymbol{a}_2 \quad \cdots \quad \boldsymbol{a}_n\ ]. \tag{5.2}$$

Equation (5.2) for $\ell = 1$ tells us that we must have $\boldsymbol{q}_1 = \boldsymbol{a}_1/\|\boldsymbol{a}_1\|$ and $R_{1,1} = \|\boldsymbol{a}_1\|$. Next we form the vector $\boldsymbol{b} = \boldsymbol{a}_2 - \langle \boldsymbol{q}_1, \boldsymbol{a}_2\rangle \boldsymbol{q}_1$. It is orthogonal to $\boldsymbol{q}_1$, since $\langle \boldsymbol{q}_1, \boldsymbol{a}_2 - \langle \boldsymbol{q}_1, \boldsymbol{a}_2\rangle \boldsymbol{q}_1\rangle = \langle \boldsymbol{q}_1, \boldsymbol{a}_2\rangle - \langle \boldsymbol{q}_1, \boldsymbol{a}_2\rangle\langle \boldsymbol{q}_1, \boldsymbol{q}_1\rangle = 0$. Since the columns of A are assumed linearly independent, $\boldsymbol{b} \neq \boldsymbol{0}$ and we set $\boldsymbol{q}_2 = \boldsymbol{b}/\|\boldsymbol{b}\|$, hence $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ are orthonormal. Moreover,

$$\langle \boldsymbol{q}_1, \boldsymbol{a}_2\rangle \boldsymbol{q}_1 + \|\boldsymbol{b}\|\boldsymbol{q}_2 = \langle \boldsymbol{q}_1, \boldsymbol{a}_2\rangle \boldsymbol{q}_1 + \boldsymbol{b} = \boldsymbol{a}_2,$$

hence, to obey (5.2) for $\ell = 2$, we let $R_{1,2} = \langle \boldsymbol{q}_1, \boldsymbol{a}_2\rangle$, $R_{2,2} = \|\boldsymbol{b}\|$.

More generally we get the following classical Gram-Schmidt algorithm to compute a QR factorization: Set $\boldsymbol{q}_1 = \boldsymbol{a}_1/\|\boldsymbol{a}_1\|$ and $R_{11} = \|\boldsymbol{a}_1\|$. For $j = 2, \ldots, n$: Set $R_{ij} = \langle \boldsymbol{q}_i, \boldsymbol{a}_j\rangle$ for $i \leq j-1$, and $\boldsymbol{b}_j = \boldsymbol{a}_j - \sum_{i=1}^{j-1} R_{ij}\boldsymbol{q}_i$. Set $\boldsymbol{q}_j = \boldsymbol{b}_j/\|\boldsymbol{b}_j\|$ and $R_{jj} = \|\boldsymbol{b}_j\|$.

The total cost of the classical Gram–Schmidt algorithm is $\mathcal{O}\big(n^2 m\big)$, since at each iteration j a total of $\mathcal{O}(mj)$ operations are performed.

The disadvantage of the classical Gram–Schmidt is its *ill-conditioning*: using finite arithmetic, small imprecisions in the calculation of inner products spread rapidly, leading to effective loss of orthogonality. Errors accumulate fast and the computed off-diagonal elements of $Q^\top Q$ may become large.

The Gram-Schmidt algorithm operates by performing "triangular orthogonalization" on A: triangular operations are applied to A to produce the orthonormal system $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$. We are now going to see two algorithms for QR factorization that are based on "orthogonal triangularization": we will repeatedly apply orthogonal transformations to A to put it into triangular form.

Orthogonal transformations Given real $m \times n$ matrix $A_0 = A$, we seek a sequence $\Omega_1, \Omega_2, \ldots, \Omega_k$ of $m \times m$ orthogonal matrices such that the matrix $A_i := \Omega_i A_{i-1}$ has more zero elements below the main diagonal than A_{i-1} for $i = 1, 2, \ldots, k$ and so that the manner of insertion of such zeros is such that A_k is upper triangular. We then let $R = A_k$, therefore $\Omega_k\Omega_{k-1}\cdots\Omega_2\Omega_1 A = R$ and $Q = (\Omega_k\Omega_{k-1}\cdots\Omega_1)^{-1} = (\Omega_k\Omega_{k-1}\cdots\Omega_1)^\top = \Omega_1^\top\Omega_2^\top\cdots\Omega_k^\top$. Hence $A = QR$, where Q is orthogonal and R upper triangular.

Givens rotations Recall that the matrix associated to clockwise rotation in $\mathbb{R}^2$ by angle θ is $\left[\begin{smallmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{smallmatrix}\right]$. An $m \times m$ *Givens rotation* matrix Ω is an orthogonal matrix specified by two integers $1 \leq p < q \leq m$ and an angle $\theta \in [-\pi, \pi]$, which coincides with the identity matrix except for the 2×2 submatrix associated to rows/columns $\{p, q\}$ which correspond to a 2×2 rotation matrix. Specifically, we use the notation $\Omega^{[p,q]}$, where $1 \leq p < q \leq m$ for a matrix such that

$$\Omega^{[p,q]}_{p,p} = \Omega^{[p,q]}_{q,q} = \cos\theta, \qquad \Omega^{[p,q]}_{p,q} = \sin\theta, \qquad \Omega^{[p,q]}_{q,p} = -\sin\theta$$

for some $\theta \in [-\pi, \pi]$. The remaining elements of $\Omega^{[p,q]}$ are those of an identity matrix. For example,

$$m = 4 \quad \Longrightarrow \quad \Omega^{[1,2]} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \Omega^{[2,4]} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & 0 & \sin\theta \\ 0 & 0 & 1 & 0 \\ 0 & -\sin\theta & 0 & \cos\theta \end{bmatrix}.$$

Geometrically, such matrices correspond to a rotation in the two-dimensional coordinate subspace spanned by $\{\boldsymbol{e}_p, \boldsymbol{e}_q\}$, where $\boldsymbol{e}_i$ is the vector with zeros everywhere except for a 1 in position i.

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

Theorem Let A be an $m \times n$ matrix. Then, for every $1 \le p < q \le m$, $i \in \{p, q\}$ and $1 \le j \le n$, there exists $\theta \in [-\pi, \pi]$ such that $(\Omega^{[p,q]}A)_{i,j} = 0$. Moreover, all the rows of $\Omega^{[p,q]}A$, except for the pth and the qth, are the same as the corresponding rows of A, whereas the pth and the qth rows of $\Omega^{[p,q]}A$ are linear combinations of the pth and qth rows of A.

Proof. Let $i = q$. If $A_{p,j} = A_{q,j} = 0$ then any θ will do, otherwise we let

$$\cos\theta := A_{p,j}/\sqrt{A_{p,j}^2 + A_{q,j}^2}, \qquad \sin\theta := A_{q,j}/\sqrt{A_{p,j}^2 + A_{q,j}^2}.$$

Hence

$$(\Omega^{[p,q]}A)_{q,k} = -(\sin\theta)A_{p,k} + (\cos\theta)A_{q,k}, \quad k = 1, 2, \ldots, n \qquad \Rightarrow \qquad (\Omega^{[p,q]}A)_{q,j} = 0.$$

Likewise, when $i = p$ we let $\cos\theta := A_{q,j}/\sqrt{A_{p,j}^2 + A_{q,j}^2}$, $\sin\theta := -A_{p,j}/\sqrt{A_{p,j}^2 + A_{q,j}^2}$.

The last two statements of the theorem are an immediate consequence of the construction of $\Omega^{[p,q]}$. □

An example: Suppose that A is 3×3. We can force zeros underneath the main diagonal as follows.

1 First pick $\Omega^{[1,2]}$ so that $(\Omega^{[1,2]}A)_{2,1} = 0 \quad \Rightarrow \quad \Omega^{[1,2]}A = \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ \times & \times & \times \end{bmatrix}$.

2 Next pick $\Omega^{[1,3]}$ so that $(\Omega^{[1,3]}\Omega^{[1,2]}A)_{3,1} = 0$. Multiplication by $\Omega^{[1,3]}$ doesn't alter the second row, hence $(\Omega^{[1,3]}\Omega^{[1,2]}A)_{2,1}$ remains zero $\quad \Rightarrow \quad \Omega^{[1,3]}\Omega^{[1,2]}A = \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix}$.

3 Finally, pick $\Omega^{[2,3]}$ so that $(\Omega^{[2,3]}\Omega^{[1,3]}\Omega^{[1,2]}A)_{3,2} = 0$. Since both second and third row of $\Omega^{[1,3]}\Omega^{[1,2]}A$ have a leading zero, $(\Omega^{[2,3]}\Omega^{[1,3]}\Omega^{[1,2]}A)_{2,1} = (\Omega^{[2,3]}\Omega^{[1,3]}\Omega^{[1,2]}A)_{3,1} = 0$. It follows that $\Omega^{[2,3]}\Omega^{[1,3]}\Omega^{[1,2]}A$ is upper triangular. Therefore

$$R = \Omega^{[2,3]}\Omega^{[1,3]}\Omega^{[1,2]}A = \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix}, \qquad Q = (\Omega^{[2,3]}\Omega^{[1,3]}\Omega^{[1,2]})^\top.$$

The Givens algorithm Given $m \times n$ matrix A: For each j from 1 to n and i from $j + 1$ to m, replace A by $\Omega^{[j,i]}A$ where $\Omega^{[j,i]}$ is chosen to annihilate the (i, j) entry.

This algorithm transforms A into an upper triangular matrix by a sequence of orthogonal transformations. The final orthogonal matrix Q however is not computed explicitly in this algorithm. If we want to compute Q explicitly, we commence by letting Ω be the $m \times m$ identity matrix and, each time A is premultiplied by $\Omega^{[j,i]}$, we also premultiply Ω by the same rotation. Hence the final Ω is the product of all the rotations, in correct order, and we let $Q = \Omega^\top$. Note however, in most applications we don't need Q but, instead, just the action of $Q^\top$ on a given vector (recall: solution of linear systems!). This can be accomplished by multiplying the *given vector*, e.g., the right-hand side $\boldsymbol{b}$ if we are solving a linear system, by successive rotations.

The cost For each $j < i$, the cost of computing $\Omega^{[j,i]}A$ is $\mathcal{O}(n)$ since we just have to replace the j'th and i'th rows of A by their appropriate linear combinations. This has to be done less than mn times (the number of pairs (j, i)) and so the total cost is $\mathcal{O}(mn^2)$.

Mathematical Tripos Part IB: Lent 2019

Numerical Analysis – Lecture 16[1]

Householder reflections Let $\boldsymbol{u} \in \mathbb{R}^m \setminus \{\boldsymbol{0}\}$. The $m \times m$ matrix $I - 2\frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2}$ is called a *Householder reflection*. Each such matrix is symmetric and orthogonal, since

$$\left(I - 2\frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2}\right)^\top \left(I - 2\frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2}\right) = \left(I - 2\frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2}\right)^2 = I - 4\frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2} + 4\frac{\boldsymbol{u}(\boldsymbol{u}^\top\boldsymbol{u})\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^4} = I.$$

Householder reflections offer an alternative to Given rotations in the calculation of a QR factorization.

Householder algorithm Our goal is to multiply an $m \times n$ matrix A by a sequence of Householder reflections so that each product induces zeros under the diagonal in an entire column.

At the first step we seek a reflection that transforms the first column $\boldsymbol{a}_1$ of A to a multiple of $\boldsymbol{e}_1$. Since the Householder reflection is orthogonal (it preserves Euclidean norm) the latter has to be $\pm\|\boldsymbol{a}_1\|\boldsymbol{e}_1$ where we are free to choose the sign. The Householder reflection that does this operation is given by the choice of vector $\boldsymbol{u} = \boldsymbol{a}_1 - (\pm\|\boldsymbol{a}_1\|\boldsymbol{e}_1)$. For numerical stability the sign is usually chosen to be $-\text{sign}(A_{11})$.

More generally, at the beginning of the k'th step of the algorithm, the columns 1 to $k-1$ have been processed and have zeros under their diagonal element. Our goal is to find a Householder reflection that will induce zeros under the diagonal element of the k'th column. To do so we use a block orthogonal matrix $\left[\begin{smallmatrix} I & 0 \\ 0 & H \end{smallmatrix}\right]$ where I is a $(k-1) \times (k-1)$ identity matrix, and H is a $(m-k+1) \times (m-k+1)$ Householder reflection associated with the choice $\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{a}}_k + \text{sign}(A_{kk})\|\tilde{\boldsymbol{a}}_k\|\tilde{\boldsymbol{e}}_1$, where $\tilde{\boldsymbol{a}}_k$ is the vector of size $m-k+1$ consisting of the entries of A under the diagonal in the k'th column, and $\tilde{\boldsymbol{e}}_1$ is the vector of size $m-k+1$ with a 1 in the first position and zero elsewhere.

To summarize it is convenient to use the (Matlab-style) notation where $A_{k:m,j}$ indicates the vector of size $m-k+1$ obtained from rows $k, \ldots, m$ of column j of A. Then the algorithm can be written as follows:

Given $A \in \mathbb{R}^{m \times n}$ with $m \geq n$. For $k = 1$ to n:

- Let $\tilde{\boldsymbol{a}}_k = A_{k:m,k} \in \mathbb{R}^{m-k+1}$
- Let $\tilde{\boldsymbol{e}}_1$ be the vector of size $m-k+1$ with a 1 in the first position and zero elsewhere.
- Let $\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{a}}_k + \text{sign}(A_{kk})\|\tilde{\boldsymbol{a}}_k\|\tilde{\boldsymbol{e}}_1$
- For each column $j = k, \ldots, n$ update $A_{k:m,j} = A_{k:m,j} - 2(\tilde{\boldsymbol{u}}^T A_{k:m,j})\tilde{\boldsymbol{u}}/\|\tilde{\boldsymbol{u}}\|^2$.

Example ($k = 3$, assuming the first two columns have already been processed)

$$A = \begin{bmatrix} 2 & 4 & 7 \\ 0 & 3 & -1 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & -2 \end{bmatrix} \quad \rightarrow \quad \tilde{\boldsymbol{a}}_3 = \begin{bmatrix} 2 \\ 1 \\ -2 \end{bmatrix}, \ \tilde{\boldsymbol{u}} = \begin{bmatrix} 5 \\ 1 \\ -2 \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} 2 & 4 & 7 \\ 0 & 3 & -1 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Calculation of Q Like for the case of Givens algorithm, the matrix Q is not explicitly formed. To form Q explicitly we start with $\Omega = I$ initially and, for each step we replace Ω, by $\left(I - 2\frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2}\right)\Omega = \Omega - \frac{2}{\|\boldsymbol{u}\|^2}\boldsymbol{u}(\boldsymbol{u}^\top\Omega)$ where $\boldsymbol{u} = \begin{bmatrix} \boldsymbol{0} \\ \tilde{\boldsymbol{u}} \end{bmatrix}$ is obtained from $\tilde{\boldsymbol{u}}$ by adding $k-1$ zeros above it[2]. However, if we require just the vector $\boldsymbol{c} = Q^\top\boldsymbol{b}$, say, rather than the matrix Q, then we set initially $\boldsymbol{c} = \boldsymbol{b}$ and in each stage replace $\boldsymbol{c}$ by $\left(I - 2\frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2}\right)\boldsymbol{c} = \boldsymbol{c} - 2\frac{\boldsymbol{u}^\top\boldsymbol{c}}{\|\boldsymbol{u}\|^2}\boldsymbol{u}$.

[1] Corrections and suggestions to these notes should be emailed to `h.fawzi@damtp.cam.ac.uk`.

[2] Indeed, note that the reflection $I - 2\boldsymbol{u}\boldsymbol{u}^\top/\|\boldsymbol{u}\|^2$ is the same as the block orthogonal matrix $\left[\begin{smallmatrix} I & 0 \\ 0 & H \end{smallmatrix}\right]$ where H is the Householder reflection corresponding to $\tilde{\boldsymbol{u}}$.

Givens or Householder? If A is dense, it is in general more convenient to use Householder reflections. Givens rotations come into their own, however, when A has many leading zeros in its rows. E.g., if an $n \times n$ matrix A consists of zeros underneath the first subdiagonal, they can be 'rotated away' in just $n-1$ Givens rotations, at the cost of $\mathcal{O}(n^2)$ operations!

5.3 Linear least squares

Statement of the problem Suppose that an $m \times n$ matrix A and a vector $\boldsymbol{b} \in \mathbb{R}^m$ are given. The equation $A\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{x} \in \mathbb{R}^n$ is unknown, has in general no solution (if $m > n$) or an infinity of solutions (if $m < n$). Problems of this form occur frequently when we collect m observations (which, typically, are prone to measurement error) and wish to exploit them to form an n-variable linear model, where $n \ll m$. (In statistics, this is known as *linear regression.*) Bearing in mind the likely presence of errors in A and $\boldsymbol{b}$, we seek $\boldsymbol{x} \in \mathbb{R}^n$ that minimises the Euclidean length $\|A\boldsymbol{x} - \boldsymbol{b}\|$. This is the *least squares problem.*

Theorem $\boldsymbol{x} \in \mathbb{R}^n$ is a solution of the least squares problem iff $A^\top(A\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{0}$.

Proof. If $\boldsymbol{x}$ is a solution then it minimises

$$f(\boldsymbol{x}) := \|A\boldsymbol{x} - \boldsymbol{b}\|^2 = \langle A\boldsymbol{x} - \boldsymbol{b}, A\boldsymbol{x} - \boldsymbol{b}\rangle = \boldsymbol{x}^\top A^\top A\boldsymbol{x} - 2\boldsymbol{x}^\top A^\top \boldsymbol{b} + \boldsymbol{b}^\top \boldsymbol{b}.$$

Hence $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$. But $\frac{1}{2}\nabla f(\boldsymbol{x}) = A^\top A\boldsymbol{x} - A^\top \boldsymbol{b}$, hence $A^\top(A\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{0}$.
Conversely, suppose that $A^\top(A\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{0}$ and let $\boldsymbol{u} \in \mathbb{R}^n$. Hence, letting $\boldsymbol{y} = \boldsymbol{u} - \boldsymbol{x}$,

$$\begin{aligned}\|A\boldsymbol{u} - \boldsymbol{b}\|^2 = \langle A\boldsymbol{x} + A\boldsymbol{y} - \boldsymbol{b}, A\boldsymbol{x} + A\boldsymbol{y} - \boldsymbol{b}\rangle = \langle A\boldsymbol{x} - \boldsymbol{b}, A\boldsymbol{x} - \boldsymbol{b}\rangle + 2\boldsymbol{y}^\top A^\top(A\boldsymbol{x} - \boldsymbol{b}) \\ + \langle A\boldsymbol{y}, A\boldsymbol{y}\rangle = \|A\boldsymbol{x} - \boldsymbol{b}\|^2 + \|A\boldsymbol{y}\|^2 \geq \|A\boldsymbol{x} - \boldsymbol{b}\|^2\end{aligned}$$

and $\boldsymbol{x}$ is indeed optimal. □

Corollary Optimality of $\boldsymbol{x} \Leftrightarrow$ the vector $A\boldsymbol{x} - \boldsymbol{b}$ is orthogonal to all columns of A.

Normal equations One way of finding optimal $\boldsymbol{x}$ is by solving the $n \times n$ linear system $A^\top A\boldsymbol{x} = A^\top \boldsymbol{b}$; this is the method of *normal equations.* This approach is popular in many applications. However, there are three disadvantages. Firstly, $A^\top A$ might be singular, secondly sparse A might be replaced by a dense $A^\top A$ and, finally, forming $A^\top A$ might lead to loss of accuracy. Thus, suppose that our computer works in the IEEE arithmetic standard (≈ 15 significant digits) and let

$$A = \begin{bmatrix} 10^8 & -10^8 \\ 1 & 1 \end{bmatrix} \qquad \Longrightarrow \qquad A^\top A = \begin{bmatrix} 10^{16}+1 & -10^{16}+1 \\ -10^{16}+1 & 10^{16}+1 \end{bmatrix} \approx 10^{16}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

Given $\boldsymbol{b} = [0, 2]^\top$ the solution of $A\boldsymbol{x} = \boldsymbol{b}$ is $[1, 1]^\top$, as can be easily found by Gaussian elimination. However, our computer 'believes' that $A^\top A$ is singular!

QR and least squares
Let A be an $m \times n$ matrix with $m \geq n$, and let $A = QR$ be a *reduced* QR factorization where Q is $m \times n$ has orthonormal columns and R is $n \times n$ upper triangular. We know that $\boldsymbol{x}$ is a solution to the least squares problem iff $A\boldsymbol{x} - \boldsymbol{b}$ is orthogonal to all columns of A. Since the columns of Q span the same space as the columns of A this is equivalent to saying that $Q^\top(A\boldsymbol{x} - \boldsymbol{b}) = 0$. Since the columns of Q form an orthonormal system we have[3] $Q^\top Q = I_n$, and so this leads to the equation $R\boldsymbol{x} = Q^\top \boldsymbol{b}$. The latter can be solved using backsubstitution.

[3] Note however that $QQ^\top$ is not equal to the identity matrix! (Q is a rectangular matrix here)