Numerical Analysis *

Zhiyuan Bai

Compiled on June 15, 2021

This document serves as a set of revision materials for the Cambridge Mathematical Tripos Part IB course *Numerical Analysis* in Lent 2021. However, despite its primary focus, readers should note that it is NOT a verbatim recall of the lectures, since the author might have made further amendments in the content. Therefore, there should always be provisions for errors and typos while this material is being used.

Contents

*Based on the lectures under the same name taught by Prof. C. B. Schönlieb in Lent 2021.

0 What is Numerical Analysis?

Numerical analysis is the stufy of algorithms for problems in continuous mathematics, i.e. to devise algorithms to approximate solution of continuous models. In theory and in practice, what we want are finite dimensional procedures that produces sufficiently close approximation to the solution of continuous (i.e. infinite dimensional) problems. Of course, we want to algorithms to be fast, stable, accurate ("consistent with the continuous model"). The study of the construction of these algorithms and, of course, explore the precise meanings of these criteria, is the central focus of this subject.
The example of the problems we are interested in are algebraic equations, differential equations and optimisation problems.

1 Polynomial Interpolation

Let $\mathbb{P}_n[x]$ be the (linear) space of all real polynomials of degree at most n.

1.1 The Interpolation Problem

Given $n+1$ distinct real points $x_0, \ldots, x_n$ and real numbers $f_0, \ldots, f_n$, we seek a polynomial $p \in \mathbb{P}_n[x]$ such that $p(x_i) = f_i$ for all i. Such a polynomial is called an interpolant of the points $\{(x_i, f_i) : 0 \leq i \leq n\}$. A polynomial has $n+1$ degrees of freedom in terms of its coefficients, and there are $n+1$ data points we are using, which justifies why we are interested in this problem. Of course, we can simply solve an $n+1$ dimensional linear equation to find the coefficients, but we don't have to go through all that mess.

1.2 The Lagrange Formula

What's better than having a nice, closed-form solution. So here goes:

$$p(x) = \sum_{k=0}^{n} f_k L_k(x), L_k(x) = \prod_{l \neq k} \frac{x - x_l}{x_k - x_l}$$

This is called the Lagrange formula. It is obvious that $L_k(x_j) = \delta_{jk}$, so it is a interpolant of the points. Also, it is the interpolant, since the interpolent has to be unique as a polynomial of degree n has at most n zeros unless it is identically zero.
What is the complexity of computing the Lagrange formula? By just staring we can conclude that L_k requires $O(n)$ to compute, so p requires $O(n^2)$.

1.3 Error Size

Let $[a, b]$ be a closed interval, and let $C[a, b]$ denote the space of all continuous functions $[a, b] \to \mathbb{R}$ and let $C^s[a, b]$ denote the space of all functions $[a, b] \to \mathbb{R}$ that has continuous derivatives up to order s.

Theorem 1.1. *Suppose we have $f \in C^{n+1}[a, b]$ and $x_0, \ldots, x_n \in [a, b]$ distinct. Let $p \in \mathbb{P}_n[x]$ be the interpolant of $(x_i, f(x_i))$, then $\forall x \in [a, b], \exists \xi \in [a, b]$ such that*

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

Remark. From the formula, we can see the error is smaller for large n if we fix everything else, which is nice. Also, it can also depends on the "quality" of the data (i.e. the precise position of the x_i's).

Proof. The formula is trivially true when $x = x_i$ for some i. Fix any other $x \in [a, b]$ and define

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

Observe now that ϕ vanishes at all x_i and x, which constitute $n + 2$ distinct points. So by Rolle's theorem, ϕ' vanishes at a minimum of $n+1$ distinct points, and thus ϕ'' vanishes at n distint points, etc.. Continuing this and we arrive at the conclusion that $\phi^{(n+1)}$ has a root in $[a, b]$. Say the root is ξ, then

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

Rearranging gives the result. □

How does sampling affect the quality of the interpolation though?

Example 1.1 (Runge's Example)**.** If we interpolate $f(x) = (1 + x^2)^{-1}, x \in [-5, 5]$ at the equally-spaced points $x_j = -5+10j/n$, then the term $\prod_{i=0}^{n} (x-x_i)$ explodes near ± 5 and caused a huge error there which does not get solved for large n.
A cleverer sampling is to take $x_j = 5 \cos((n-j)\pi/n)$ (the "Chebyshev points"), which, as one can prove, minimises the maximum magnitude of that term.

1.4 Divided Differences

Definition 1.1. Given pairwise distinct points $x_0, \ldots, x_n \in [a, b]$, we let $p \in \mathbb{P}_n[x]$ interpolate $f \in C[a, b]$ at these points. The coefficient of x^n in p is called the divided difference $f[x_0, \ldots, x_n]$ of degree n.

Explicitly,

$$f[x_0, \ldots, x_n] = \sum_{k=0}^{n} f(x_k) \prod_{l \neq k} \frac{1}{x_k - x_l}$$

by Lagrange's formula. It is easy to verify that $f[x_0] = f(x_0)$ and $f[x_0, x_1] = (f(x_0) - f(x_1))/(x_0 - x_1)$, which should convince you of the terminology.

Theorem 1.2. *Suppose $x_0, \dots, x_{n+1}$ are pairwise distinct, then*

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

Proof. Suppose p interpolates f in $x_0, \dots, x_n$ and q interpolates $x_1, \dots, x_{n+1}$. Then

$$\frac{(x - x_0)q(x) + (x_{n+1} - x)p(x)}{x_{n+1} - x_0}$$

interpolates f in $x_0, \dots, x_{n+1}$. The theorem follows. □

So, naturally, we expect derivatives to play some role here.

Theorem 1.3. *Let $[a, b]$ be the smallest interval that contains the distinct points $x_0, \dots, x_n$ and $f \in C^n[a, b]$. Then there exists $\xi \in [a, b]$ such that*

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

Proof. Suppose p interpolates f in $x_0, \dots, x_n$. Then $f - p$ has at least $n + 1$ zeros, so by Rolle's theorem $f^{(n)} - p^{(n)} = f^{(n)} - n! f[x_0, \dots, x_n]$ vanishes at some $\xi \in [a, b]$. Plugging in ξ gives the formula. □

1.5 Newton Interpolation

Theorem 1.4. *Suppose $x_0, \dots, x_n$ are pairwise distinct, then*

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

interpolates f at $x_0, \dots, x_n$.

Proof. Induction. This is obviously true for $n = 0$. Suppose it is true up to n, let p_{n+1} interpolates f at $x_0, \dots, x_{n+1}$. Then $p_{n+1} - p_n$ has degree $n + 1$, leading coefficient $f[x_0, \dots, x_{n+1}]$, and vanishes at $x_0, \dots, x_n$, therefore equals

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

which gives the formula. □

Computing all the divided differences up to degree n costs $O(n^2)$ if we use the recursion formula. Knowing the divided differences, we can then compute the interpolant in $O(n)$ by using the Horner scheme, i.e. by writing

$$p_n(x) = f[x_0] + (x - x_0)\left(f[x_0, x_1] + (x - x_1)\left(f[x_0, x_1, x_2] + \cdots\right)\right)$$

So in total, this algorithm has complexity $O(n^2)$, which is the same as Lagrange's formula. But each has advantages and disadvantages.
For Newton's formula, having computed $p_n(x)$ for $x_0, \dots, x_n$, we can easily add in x_{n+1} without having to recompute the interpolant. This comes in handy when we want to refine our estimation.
On the other hand, since we have a simple formula for Lagrange's method, it is better when we wish to manipulate the interpolation polynomial as part of a larger expression, which comes in handy when we are dealing with numerical differentiation and numerical integration.

2 Orthogonal Polynomials

2.1 Orthogonality in General

Definition 2.1. In general, an inner product is on a real vector space V is any $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$ satisfying:
1. Symmetry: $\forall x, y \in V, \langle x, y \rangle = \langle y, x \rangle$.
2. Nonnegativity: $\forall x \in V, \langle x, x \rangle \geq 0$ and equality holds iff $x = 0$.
3. Linearity: $\langle \cdot, \cdot \rangle$ is linear.

Definition 2.2. Given an inner product $\langle \cdot, \cdot \rangle$ on V, we say $x, y \in V$ are orthogonal if $\langle x, y \rangle = 0$.

The vector space we are interested in is usually $C[a, b]$ equipped with an inner product of the form

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

where $w : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is a fixed nonnegative continuous function with isolated zeros.

2.2 Orthogonal Polynomials

Given an inner product $\langle \cdot, \cdot \rangle$ in $V = \mathbb{P}[x]$ (the vector space of polynomials in x), we seek a sequence $p_0, p_1, \ldots$ such that:
1. $\deg(p_n) = n$.
2. $\langle p_n, p_m \rangle = 0$ for all $n < m$.

Definition 2.3. This sequence is called orthogonal polynomials, and p_n is called the n^{th} orthogonal polynomial.

If we can find $p_0, \ldots, p_n$, then they form an orthogonal basis for $\mathbb{P}_n[x]$.

Theorem 2.1. *There exists a unique monic orthogonal polynomial p_n of degree n.*

Proof. Induction. Obviously the only degree 0 monic orthogonal polynomial is $p_0(x) = 1$. Suppose we have found $p_0, \ldots, p_n$, then choose any monic polynomial q of degree $n + 1$ and cast the Gram-Schmidt process

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

which is monic, has degree $n + 1$, and is orthogonal to $p_0, \ldots, p_n$.
To see the uniqueness, if we have another, say $\tilde{p}_{n+1}$, then $p_{n+1} - \tilde{p}_{n+1}$ is orthogonal to $p_0, \ldots, p_n$. But its degree is at most n, so it has to be 0, hence the uniqueness. □

The proof also shows a way to construct orthogonal polynomials.

Example 2.1. Using the inner product

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

$[a,b] = [-1,1]$ and $w \equiv 1$ gives the Legendre polynomials

$$p_0(x) = 1, p_1(x) = x, p_2(x) = x^2 - \frac{1}{3}, p_3(x) = x^3 - \frac{3}{5}x, p_4(x) = x^4 - \frac{6}{7}x^2 + \frac{3}{35}$$

There are other examples, of course, like Chebyshev polynomials (with $[a,b] = [-1,1], w(x) = (1-x^2)^{-1/2}$), Laguerre polynomials ($[a,b] = [0,\infty), w(x) = e^{-x}$), Hermite polynomials $[a,b] = (-\infty,\infty), w(x) = e^{-x^2}$, etc..

2.3 A Three-Step Recurrence

Of course, we can use the Gram-Schmidt process shown earlier to compute the orthogonal polynomials. But this is not a very good method, as it is not very quick and the error can accumulate since the formula depends on all previously computed polynomials. For certain scalar product, we have a nicer way.

Theorem 2.2. *Suppose the scalar product on $\mathbb{P}[x]$ we defined satisfies $\langle xp, q\rangle = \langle p, xq\rangle$, then the monic orthogonal polynomials are given by the recurrence*

$$p_{-1}(x) = 0, p_0(x) = 1, p_{n+1}(x) = (x - \alpha_n)p_n(x) - \beta_n p_{n-1}(x)$$

where

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

Proof. Just plugging in. □

Example 2.2. Consider the Chebyshev polynomials T_n which arises from the inner product

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

They are, in fact, primarily defined by the relation $T_n(\cos\theta) = \cos(n\theta)$, which calculates to

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

They are, indeed, orthogonal polynomials to the inner product defined above as whenever $m \neq n$,

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

We also have the recurrence relation $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$. They are not monic, but we can make them monic by dividing T_n by 2^{n-1}.

2.4 Least-Square Polynomial Fitting

We are working in $C[a,b]$ equipped with an inner product $\langle\cdot,\cdot\rangle$. Given $f \in C[a,b]$, we want to find $p \in \mathbb{P}_n[x]$ so that $\|f-p\|^2 = \langle f-p, f-p\rangle$ is minimised. We call such a minimiser $p = \hat{p}_n$.

Theorem 2.3. *Suppose $p_0, p_1, \ldots$ are the orthogonal polynomials of $\langle\cdot,\cdot\rangle$, then*

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

Proof. For any $p \in \mathbb{P}_n[x]$, we can write it in terms of the orthogonal polynomials $p = \sum_k c_k p_k$. So

$$\begin{aligned}\|f - p\|^2 &= \left\langle f - \sum_{k=0}^{n} c_k p_k, f - \sum_{k=0}^{n} c_k p_k \right\rangle \\ &= \langle f, f\rangle - 2\sum_{k=0}^{n} c_k \langle p_k, f\rangle + \sum_{k=0}^{n} c_k^2 \langle p_k, p_k\rangle\end{aligned}$$

We are essentially optimising this expression with respect to $c_0, \ldots, c_n$. By calculus or whatever, we find that $c_k = \langle f, p_k\rangle / \langle p_k, p_k\rangle$ is the minimiser. □

The error of this approximation is then

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

which kind of looks like Pythagoras' theorem (well, duh!). A rerrangement of this also shows that $\langle f - \hat{p}_n, \hat{p}_n\rangle = 0$. Clearly, increasing n brings the error down. But will it go to zero?

Theorem 2.4 (Weierstrass' Theorem)**.** *Suppose $f \in C[a, b]$, then for any $\epsilon > 0$ there exists a polynomial p such that $|f(x) - p(x)| < \epsilon$ for all $x \in [a, b]$.*

Proof. Haha no. □

Corollary 2.5. *If the inner product is defined by*

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

where $w : [a, b] \to \mathbb{R}_{\geq 0}$ is continuous and has isolated zeros, then $\|f - \hat{p}_n\|^2 \to 0$ as $n \to \infty$.

Proof. For a polynomial p,

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

which can be arbitrarily small by Weierstrass' theorem. The corollary follows by the definition of $\hat{p}_n$. □

Theorem 2.6 (Parseval's Identity)**.** *Suppose $p_0, p_1, \ldots$ are orthogonal polynomials of the inner product $\langle \cdot, \cdot\rangle$ in the form as in the preceding corollary, then*

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

Proof. We already know

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

sending $n \to \infty$ gives the result (as we all know, the corollary of a corollary is a theorem). □

2.5 Least-Square Fitting to Discrete Function Values

Sometimes (or in real-life practice, always) we only know finitely many values of a function $f(x_1), \ldots, f(x_m)$ at distinct $x_1, \ldots, x_m \in [a, b]$. In that case, we of course want $p \in \mathbb{P}_n[x]$ that minimises $\sum_k (f(x_k) - p(x_k))^2$. As long as $n < m$, we can take the inner product

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

which naturally inherits most of the results from the continuous case. Of course, the orthogonal polynomials can be calculated from the three-term recurrence relation we found earlier.

2.6 Gaussian Quadrature

We again work in $C[a, b]$ with inner product defined by the integral

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

where $w : [a, b] \to \mathbb{R}_{\geq 0}$ is continuous and has isolated zeros. The aim is to approximate integrals by finite sums

$$\int_a^b w(x) f(x) \, \mathrm{d}x \approx \sum_{k=1}^{\nu} b_k f(c_k)$$

where ν is a given number and we want to find $b_1, \ldots, b_\nu$ (the "weights") and $c_1, \ldots, c_\nu$ (the "nodes") that can ensure high accuracy. This is known as a quadrature formula.

There are many ways to make a quadrature formula. One of them is obtained by requiring the approximation to be exact for any $f \in \mathbb{P}_m[x]$ where m is as large as possible. The result with the maximal m is called Gaussian quadrature. It is obvious that $m = 2\nu$ doesn't work, since the formula fails to be exact if we take $p(x) = \prod_{k=1}^{\nu} (x - c_k)^2$. We shall show that the maximal m is $2\nu - 1$. Let $p_0, p_1, \ldots$ be the monic orthogonal polynomials.

Theorem 2.7. *p_n has n real distinct zeros in (a, b).*

Proof. $n = 0$ is trivial. For $n \geq 1$,

$$\int_a^b w(x) p_n(x) \, \mathrm{d}x = \langle p_0, p_n \rangle = 0$$

So p_n has to change sign in (a, b) at least once. Suppose p_n changes sign at $\phi_1, \ldots, \phi_m \in (a, b)$, then in particular it vanishes there. So $m \leq n$. We shall show that $m = n$ which implies the theorem.

Define $q(x) = \prod_{j=1}^{m} (x - \phi_j)$. Then $q(x) p_n(x) \geq 0$ for all $x \in [a, b]$ by definition, so $w(x) q(x) p_n(x) \geq 0$ for all $x \in [a, b]$ and it is not identically zero, hence $\langle q, p_n \rangle > 0 \implies m = \deg q \geq n \implies m = n$. □

For distinct $c_1, \ldots, c_\nu \in [a, b]$, we define the interpolatory weights

$$b_k = \int_a^b w(x) \prod_{j \neq k} \frac{x - c_j}{c_k - c_j} \, \mathrm{d}x$$

Theorem 2.8. *The quadrature formula with the interpolatory weight is exact for all $f \in \mathbb{P}_{\nu-1}[x]$. Moreover, if $c_1, \ldots, c_\nu$ are chosen to be the zeros of p_ν, then it is exact for all $f \in \mathbb{P}_{2\nu-1}[x]$.*

Proof. For $f \in \mathbb{P}_{\nu-1}[x]$, we have

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

by Lagrange's formula. The claim follows directly.
Now assume $c_1, \ldots, c_\nu$ are the zeros of p_ν. For $f \in \mathbb{P}_{2\nu-1}[x]$, we write $f = qp_\nu + r$ where $q, r \in \mathbb{P}_{\nu-1}[x]$, then

$$\int_a^b w(x) f(x) \, \mathrm{d}x = \langle q, p_\nu \rangle + \int_a^b w(x) r(x) \, \mathrm{d}x = \int_a^b w(x) r(x) \, \mathrm{d}x$$

On the other hand,

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

The two expressions are equal by the first part of the theorem. □

Example 2.3. Let $[a, b] = [-1, 1]$ and $w \equiv 1$. Then the underlying orthogonal polynomials are the Legendre polynomials. The nodes of the Gaussian quadratures are

$$\begin{aligned}
&\nu = 1 : c_1 = 0 \\
&\nu = 2 : c_1 = -\sqrt{3}/3, c_2 = \sqrt{3}/3 \\
&\nu = 3 : c_1 = -\sqrt{15}/5, c_2 = 0, c_3 = \sqrt{15}/5
\end{aligned}$$

and so on.

We've got the full Gaussian quadrature formula now, which is exact on $\mathbb{P}_{2\nu-1}[x]$. Naturally, the next question is, how do we analyse the error

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

that was committed via Gaussian quadrature or other kind of quadrature formulas?

3 The Peano Kernel Theorem

The result that we shall discuss, which is somehow more general, provides a way to estimate the error arised from numerical integration (or, actually, from a wide class of linear functionals) via Gaussian quadrature or other quadrature formulas that are exact on certain polynomials. Assume $f \in C^{n+1}[a,b]$ for some n such that $\mathbb{P}_n[x] \subset \ker L$ (e.g. $n \leq 2\nu - 1$ in Gaussian quadrature). By Taylor's theorem

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

Then $L(f) = L(r)$ where r is the remainder term

$$r(x) = \frac{1}{n!}\int_a^x (x-\theta)^n f^{(n+1)}(\theta)\,\mathrm{d}\theta = \frac{1}{n!}\int_a^b (x-\theta)_+^n f^{(n+1)}(\theta)\,\mathrm{d}\theta$$

with

$$(x-\theta)_+^n = \begin{cases} (x-\theta)^n, \text{ for } x \geq \theta \\ 0, \text{ for } x \leq \theta \end{cases}$$

Definition 3.1. The Peano kernel is $K(\theta) = L(x \mapsto (x-\theta)_+^n)$.

Assuming everything is nice enough so that the integral over $[a,b]$ commutes with L, then essentially

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

This is known as the Peano Kernel Theorem, which allows us to obtain a bound on $L(f)$ given useful bounds on $f^{(n+1)}$.

Example 3.1. Consider the quadrature formula

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx \frac{f(-1) + 4f(0) + f(1)}{3}$$

which is exact for polynomials up to degree 2. Then, by brutal calculation,

$$K(\theta) = L(x \mapsto (x-\theta)_+^2) = -\frac{1}{3}\theta \begin{cases} (1+\theta)^2, \text{ for } \theta \in [-1,0] \\ (1-\theta)^2, \text{ for } \theta \in [0,1] \end{cases}$$

So for $f \in C^3[-1,1]$,

$$|L(f)| \leq \frac{1}{2}\int_{-1}^{1} |K(\theta)||f'''(\theta)|\,\mathrm{d}\theta \leq \frac{1}{36}\sup_{\theta\in[-1,1]} |f'''(\theta)|$$

Note that the specific form of L is not of concern in our derivation of the Peano Kernel Theorem. Indeed, all we need is that it is linear, that $\mathbb{P}_n[x] \subset \ker L$, and that it is nice enough to commute with the integral. So we naturally want to use it on estimations to other linear functionals, like $\mathrm{d}/\mathrm{d}x$.

Example 3.2. We approximate a derivative $f'(0)$ by the formula $-(3/2)f(0)+2f(1)-(1/2)f(2)$. Let $L(f) = f'(0) - (-(3/2)f(0) + 2f(1) - (1/2)f(2))$ be the error arised from the formula which is zero over $\mathbb{P}_2[x]$. Therefore, for $f \in C^3[0,2]$, we have

$$L(f) = \frac{1}{2}\int_0^2 K(\theta) f'''(\theta)\,\mathrm{d}\theta \implies |L(f)| \le \frac{1}{3}\sup_{\theta\in[0,2]}|f'''(\theta)|$$

4 Ordinary Differential Equations

We wish to approximate the solution $y : \mathbb{R}_{\ge 0} \to \mathbb{R}^N$ to the initial value problem $y' = f(t,y)$ subject to $y(0) = y_0$. We will assume that f is sufficiently nice, which in most situations means Lipschitz (for existence and uniqueness of an exact solution) and analytic (since we love to Taylor expand). Our aim is to approximate a solution $y(t)$ that is continuous in t by some data y_n of a discrete form, so that $y_n \approx y(t_n)$ for some $0 = t_0 < t_1 < \cdots$. Naturally, we would like the error (either pointwise or uniform) to converge quickly to 0 as $h = \sup_n(t_{n+1} - t_n) \to 0$. We will also want the algorithm to obtain these y_n's to be as efficient as possible. During our discussions in this section, we will always take $t_{n+1} - t_n$ to be constant over n unless otherwise specified, so $t_{n+1} - t_n = h$ for all n.

4.1 One-Step Methods

A one-step method is a recurrence $y_{n+1} = \phi_h(t_n, y_n)$ which allows y_{n+1} to depend only on t_n, y_n, h and the problem. An example of this is Euler's method. Suppose we know $y(0) = y_0$ and wish to approximate y at $t = h > 0$, then an obvious approach is to truncate its Taylor series $y(h) \approx y(0) + hy'(0) = y_0 + hf(t_0, y_0)$. So (taking $t_n = nh$), we can use the recurrence $y_{n+1} = y_n + hf(x_n, y_n)$.
The first thing we want to know is whether it converges. That is, given $t^* > 0$, we want to know if

$$\lim_{h\to 0}\left(\max_{n\in\{0,\dots,\lfloor t^*/h\rfloor\}}\|y_n(h) - y(nh)\|\right) = 0$$

where y is the exact solution (assuming existence and uniqueness).

Theorem 4.1. *Suppose there exists $\lambda \ge 0$ such that*

$$\forall t \in [0,t^*], v, w \in \mathbb{R}^N, \|f(t,v) - f(t,w)\| \le \lambda|v - w|$$

then the Euler method converges to the analytic solution.

Proof. The condition also implies the existence and uniqueness of a solution y by Picard-Lindelöf, so the notion of an analytic solution is well-defined.
Let $e_n = y_n - y(t_n)$ for $0 \le n \le t^*/h$. Then we have

$$\begin{aligned}\|e_{n+1}\| &= \|y_{n+1} - y(t_{n+1})\| \\ &= \|(y_n + hf(t_n, y_n)) - (y(t_n) + hy'(t_n) + O(h^2))\| \\ &\le \|e_n\| + h\|f(t_n, y_n) - f(t_n, y(t_n))\| + \|O(h^2)\| \\ &\le (1 + h\lambda)\|e_n\| + \|O(h^2)\|\end{aligned}$$

Consequently there exists $C \geq 0$ such that

$$\begin{aligned}\|e_{n+1}\| &\leq Ch^2 \sum_{j=0}^{n} (1+h\lambda)^j = Ch^2 \frac{(1+h\lambda)^{n+1}-1}{(1+h\lambda)-1} \leq \frac{Ch}{\lambda}(1+h\lambda)^{n+1} \\ &\leq \frac{Ch}{\lambda}\exp((n+1)h\lambda) = \frac{Ch}{\lambda}\exp(t_{n+1}\lambda) \\ &\leq h\frac{C}{\lambda}\exp(t^*\lambda) \to 0\end{aligned}$$

as $h \to 0$. □

4.2 Multi-Step Methods

Sometimes, we can get a better result by using multiple previous values, i.e. assuming $y_n, y_{n+1}, \ldots, y_{n+s-1}$ are computed, the iteration scheme

$$\sum_{l=0}^{s} \rho_l y_{n+l} = h \sum_{l=0}^{s} \sigma_l f(t_{n+l}, y_{n+l})$$

(where $\rho_s = 1$) is called an s-step method. We say the method is explicit if $\sigma_s = 0$, otherwise it is called implicit. Of course, explicit iterations are computationally cheap, but implicit methods can be more accurate. For an s-step method, we need s initial values $y_0, \ldots, y_{s-1}$.

Example 4.1. We have seen the Euler method $y_{n+1} - y_n = hf(t_n, y_n)$, and we can also have the implicit Euler $y_{n+1} - y_n = hf(t_{n+1}, y_{n+1})$. We can combine them to get the trapezoidal rule $y_{n+1} - y_n = (h/2)(f(t_n, y_n) + f(t_{n+1}, y_{n+1}))$, or in general the theta rule $y_{n+1} - y_n = h(\theta f(t_n, y_n) + (1-\theta)f(t_{n+1}, y_{n+1}))$. There are also 2-steps rules, for example the 2-step Adams-Bashforth

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

or the 2-step Adams-Moulton

$$y_{n+2} - y_{n+1} = h\left(\frac{5}{12}f(t_{n+2}, t_{n+2}) + \frac{2}{3}f(t_{n+1}, y_{n+1}) - \frac{1}{12}f(t_n, y_n)\right)$$

Our goal is of course to develop general theories about the convergence of such schemes. We first introduce a measure of local accuracy for the method.

Definition 4.1. The order of the multistep method as defined above is the largest integer $p \geq 0$ such that

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

for all sufficiently smooth y as $h \to 0$.

Almost always, like we did when dealing with Euler method, we'll assume that $t_{i+1} - t_i$ is constant over i.

Example 4.2. 1. For Euler's method, we have

$$y(t_{n+1}) - (y(t_n) + hy'(t_n)) = \frac{1}{2}h^2y''(t_n) + O(h^3) = O(h^2)$$

So it has order 1.
2. For theta method, we again use Taylor expansion to estimate the local error term and get

$$\left(\theta - \frac{1}{2}\right)h^2y''(t_n) + \left(\frac{1}{2}\theta - \frac{1}{3}\right)h^3y'''(t_n) + O(h^4)$$

So it has order 1 except when $\theta = 1/2$ (trapezoidal rule), in which case it has order 2.

All these look quite messy yet repetitive, so naturally we are motivated to do something like

Theorem 4.2. *Let $\rho(w) = \sum_l p_l w^l, \sigma(w) = \sum_l \sigma_l w^l$, then the multistep method has order p iff $\rho(e^z) - z\sigma(e^z) = O(z^{p+1})$ as $z \to 0$.*

Proof. Good ol' Taylor expansion allow us to write the local error term in the form

$$\left(\sum_{l=0}^{s}\rho_l\right)y(t_n) + \sum_{k=1}^{\infty}\frac{1}{k!}\left(\sum_{l=0}^{s}l^k\rho_l - k\sum_{l=0}^{s}l^{k-1}\sigma_l\right)h^ky^{(k)}(t_n)$$

which is $O(h^{p+1})$ iff $\sum_l \rho_l = 0$ and $\sum_l l^k\rho_l = k\sum_l l^{k-1}\sigma_l$ for $k = 1, \ldots, p$. On the other hand,

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

which is $O(h^{p+1})$ iff the condition described earlier is satisfied. The theorem follows. □

Example 4.3. The 2-step Adams-Bashforth has $\rho(w) = w^2 - w, \sigma(w) = (3/2)w - (1/2)$, then easily $\rho(e^z) - z\sigma(e^z) = (5/12)z^3 + O(z^4)$, so it has order 2.

Definition 4.2. 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 4.3 (The Dahlquist Equivalence Theorem)**.** *The multistep method is convergent if and only if it has order $p \geq 1$ and the polynomial ρ obeys the root condition.*

Proof. Omitted. □

Example 4.4. For the Adams-Bashforth method

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

We have $\rho(w) = (w-1)w$ which satisfies the root condition and the recurrence has order 2. Therefore it converges.

Example 4.5. Consider the 2-step method

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

which has order 1 but $\rho(w) = w^2 - 2w + 1 = (w-1)^2$ so the root condition is not fulfilled, so it does not converge (which is pretty obvious since it does not even take f into account).

Of course, it would be nice if we have a way to generate multistep methods which are convergent and has high order. Recall that the method has order p if $\rho(e^z) - z\sigma(e^z) = O(z^{p+1})$ as $z \to 0$. In particular, $\rho(1) = 0$. Suppose we have chosen our favourite ρ with $\rho(1) = 0$ and ρ fulfills the root condition, how do we choose our σ? Let $s = \deg \rho$. A natural choice of σ is then the truncation of the Taylor series of $\rho(w)/\log w$ about $w = 1$ to the appropriate degree, i.e. $s-1$ for explicit methods and s for implicit methods. Say we are dealing with an implicit method, then $\rho(e^z) - z\sigma(x^z) = -zO(|e^z - 1|^{s+1}) = O(z^{s+2})$, which means it has order $s+1$.

Example 4.6. The choice $\rho(w) = w^{s-1}(w-1)$ corresponds to the Adams methods. The explicit methods produced like described above are called Adams-Bashforth methods and have order s. The implicit methods are Adams-Moulton methods, which have order $s+1$. In this case, we can verify that $s = 2$ yields the same method we stated before

$$y_{n+2} - y_{n+1} = h\left(\frac{5}{12}f(t_{n+2}, t_{n+2}) + \frac{2}{3}f(t_{n+1}, y_{n+1}) - \frac{1}{12}f(t_n, y_n)\right)$$

which has order 3.

An interesting special case of numerical approximation methods to ODEs are called the BDF (backward differentiation formulae) methods, i.e. methods of the form

$$\sum_{l=0}^{s} \rho_l y_{n+l} = h\sigma_s f(t_{n+s}, y_{n+s}), n = 0, 1, \dots$$

We will discuss why we like to consider this class of methods later.

Theorem 4.4. *If we want the BDF method to have order s, then*

$$\rho(w) = \sigma_s \sum_{l=1}^{s} \frac{1}{l} w^{s-l}(w-1)^l, \sigma_s = \left(\sum_{l=1}^{s} \frac{1}{l}\right)^{-1}$$

Proof. Set $w = e^z$, then we want

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

as $w \to 1$. Taylor expansion then shows that it is equivalent to say

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

The conclusion follows. □

Example 4.7. BDF of order 1 is just implicit Euler. BDF of order 2 is, by computation,

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

Remark. BDF methods are convergent if and only if $s \leq 6$.

4.3 Runge-Kutta Methods

Recall that we have developed quadrature formulas

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{k=1}^{\nu} b_k f(c_k)$$

that can approximate integrals pretty well. Set $[a,b] = [0,h], w \equiv 1, b_k \to hb_k, c_k \to hc_k$ transforms the appximation problem to

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

of which the corresponding Gaussian quadrature has the form

$$hb_l = \int_0^h \prod_{j\neq l}^{\nu} \frac{t - hc_j}{hc_l - hc_j}\,\mathrm{d}t$$

where $\{hc_l\}$ are the roots of the orthogonal polynomial of degree ν with respect to $w \equiv 1$. Now suppose we want to solve $y' = f(t), y(0) = y_0$, then the exact solution at each step can be written in terms of the recursion

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t)\,\mathrm{d}t$$

If we approximate the integral in question using a quadrature formula, then we can obtain a time-stepping scheme of the form

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

where $h = t_{n+1} - t_n$ does not have to be constant.
What if f depends on y as well? The analytic solution should satisfy

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))\,\mathrm{d}t$$

If we plug in the quadrature directly, we end up with a recursion of the form

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

except that it doesn't quite work since we don't have information about $y(t_n + c_l h)$. The idea of the Runge-Kutta methods is to replace these unknown values by suitable linear combinations of known data. That is, we are after a recursion of the form

$$k_l = f\left(t_n + c_\nu h, y_n + h\sum_{j=1}^{l-1} a_{l,j}k_j\right), \sum_{j=1}^{l-1} a_{l,j} = c_l$$

$$y_{n+1} = y_n + h\sum_{l=1}^{\nu} b_l k_l$$

These are known as the ν-stage explicit Runge-Kutta (RK) methods. $a_{l,j}$ are called RK coefficients, the choice of which depends on order considerations.

Example 4.8. For $\nu = 2$, the method reduces to

$$k_1 = f(t_n, y_n), k_2 = f(t_n + c_2 h, y_n + c_2 h f(t_n, y_n))$$

Taylor expanding k_2 around (t_n, y_n) gives

$$k_2 = f(t_n, y_n) + hc_2 \left(\frac{\partial f}{\partial t}(t_n, y_n) + \frac{\partial f}{\partial y}(t_n, y_n) f(t_n, y_n) \right) + O(h^2)$$

The 2-stage RK is $y_{n+1} = y_n + hb_1 k_1 + hb_2 k_2$. Observe that $y'' = f_t(t, y) + f_y(t, y)y' = f_t(t, y) + f_y(t, y)f(t, y)$, so indeed $E = y(t_{n+1}) - (y(t_n) + hb_1 k_1 + hb_2 k_2)$ has

$$E = h(1 - b_1 - b_2)y'(t_n) + \frac{h^2}{2}(1 - 2b_2 c_2)y''(t_n) + O(h^3)$$

Choosing $b_1 + b_2 = 1, 2b_2 c_2 = 1$ can then yield a method of order 2. In fact, this is the best we can ensure with 2-stage RK. Consider $y' = y, y(0) = 1$ which has exact solution $y(t) = e^t$. Then $k_1 = e^{t_n}, k_2 = e^{t_n}(1 + c_2 h)$, so

$$\begin{aligned} E &= e^{t_{n+1}} - e^{t_n} - e^{t_n}(hb_1 + hb_2 + h^2 b_2 c_2) \\ &= e^{t_n}(e^h - 1 - h(b_1 + b_2) - h^2 b_2 c_2) \\ &= e^{t_n} \left(h(1 - b_1 + b_2) + \frac{h^2}{2}(1 - 2b_2 c_2) + \frac{h^3}{6} + O(h^4) \right) \end{aligned}$$

which cannot be $o(h^3)$ however we choose b_l, c_l.

A general ν-stage Runge-Kutta method looks like

$$k_l = f\left(t_n + c_l h, y_n + h \sum_{j=1}^{\nu} a_{l,j} k_j \right), \sum_{j=1}^{\nu} a_{l,j} = c_l$$

$$y_{n+1} = y_n + h \sum_{l=1}^{\nu} b_l k_l$$

If $a_{l,j} = 0$ for $l \leq j$, then it gives the explicit RK as we stated before. Otherwise, it is called an implicit RK.

4.4 Stiff Equations

Definition 4.3. We say that the ODE $y' = f(t, y)$ is stiff if we need to surpress the value of h in some numerical method in order to maintain stability.

Consider the linear scalar system $y' = \lambda y, y(0) = 1$ for some $\lambda < 0$ which has solution $y(t) = e^{\lambda t}$. This decays to 0 as $t \to \infty$. If we cooked up a numerical method intending to solve this ODE, we of course also expect the sequence y_n tends to 0 as well. For example, Euler's method have $y_n = (1 + h\lambda)^n$ which converges to 0 as $n \to \infty$ if $|1 + h\lambda| < 1$, i.e. $h < 2/|\lambda|$. This is a quite harsh restriction on the value of h.
However, if we consider the implicit Euler method, then the numerical solutions are $y_n = (1 - h\lambda)^{-n}$ which is always stable for $h > 0$.

Definition 4.4. Suppose we want to solve $y' = \lambda y, y(0) = 1$ with constant h by a numerical method that produces the solution sequence $\{y_n\}_{n\in\mathbb{N}}$. We call the set $\mathcal{D} = \{h\lambda \in \mathbb{C} : y_n \to 0 \text{ as } n \to \infty\}$ the linear stability domain of the method. We say the method is A-stable if $\mathbb{C}^- = \{z \in \mathbb{C} : \operatorname{Re} z < 0\} \subset \mathcal{D}$.

Example 4.9. For Euler's method, $\mathcal{D} = \{h\lambda \in \mathbb{C} : |1 + h\lambda| < 1\} = \{z \in \mathbb{C} : |1 + z| < 1\}$ which does not contain $\mathbb{C}^-$, hence Euler's method is not A-stable. For Implicit Euler, $\mathcal{D} = \{h\lambda \in \mathbb{C} : |1-h\lambda|^{-1} < 1\} = \{z \in \mathbb{C} : |1-z| > 1\}$ which contains $\mathbb{C}^-$, hence is A-stable.

Example 4.10. Consider the trapezoidal rule

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

The recursion for $y' = \lambda y$ gives

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

So indeed we exactly have $D = \mathbb{C}^-$ and hence the trapezoidal rule is A-stable.

The A-stability analysis of multistep methods is considerably more complicated. However, according to the second Dahlquist barrier, no multistep method of order $p \geq 3$ can be A-stable, which is quite sad. Naturally, we want to get around this by either choosing other definitions of stability or consider other kind of methods. Indeed, stability properties of BDF methods, say, are satisfactory for most stiff equations even though they might not be A-stable. Indeed, the eigenvalues in many linear stiff systems are not just in $\mathbb{C}^-$ but also very far away from the imaginary axis $i\mathbb{R}$. In fact, all BDF methods of order at most 6 share the feature that D includes a wedge about $-\infty + 0i$. Such methods are said to be A_0-stable.
Runge-Kutta methods, however, can potentially be A-stable in addition to having high orders.

Example 4.11. Consider the 2-stage implicit RK

$$\begin{aligned} k_1 &= f\left(t_n, y_n + \frac{1}{4}h(k_1 - k_2)\right) \\ k_2 &= f\left(t_n + \frac{2}{3}h, y_n + \frac{1}{12}h(3k_1 + 5k_2)\right) \\ y_{n+1} &= y_n + \frac{1}{4}h(k_1 + 3k_2) \end{aligned}$$

We shall show that this method is A-stable in addition to having order 3. Applying the method to $y' = \lambda y$ gives $y_{n+1} = r(h\lambda)y_n$ (so $y_n = (r(h\lambda))^n y_0$) where

$$r(z) = \frac{6 + 2z}{6 - 4z + z^2}$$

This can be verified manually, best done on that night when you found out your crush had caught feeling for somebody else. Anyhow, we have $D = \{z \in \mathbb{C} : |r(z)| < 1\}$. Indeed, we shall show that $|r(z)| < 1$ implies $z \in \mathbb{C}^-$. By the maximum modulus principle, it suffices to show that $|r(ix)| \leq 1$ for any

$x \in \mathbb{R} \cup \{\infty\}$, which is easy enough to verify.
We shall then show that this method has order 3. The general case is a bit complicated, so we restrict our attention to equations of the form $y' = f(y)$. Assuming the method is exact at step y_n, then

$$k_1 = f + \frac{h}{4}(k_1 - k_2)f_y + \frac{h^2}{32}(k_1 - k_2)^2 f_{yy} + O(h^3)$$
$$k_2 = f + \frac{h}{12}(3k_1 + 5k_2)f_y + \frac{h^2}{288}(3k_1 + 5k_2)^2 f_{yy} + O(h^3)$$

(For brevity, all functions quoted above are assumed to have been evaluated at $y = y(t_n)$.) So $k_1, k_2 = f + O(h)$. Inserting this estimation back shows that in fact $k_1 = f + O(h^2), k_2 = f + (2h/3)f_y f + O(h^2)$. Substituting it again gives

$$k_1 = f - \frac{1}{6}h^2 f_y^2 f + O(h^3)$$
$$k_2 = f + \frac{2}{3}h f_y f + h^2\left(\frac{5}{18}f_y^2 f + \frac{2}{9}f_{yy}f^2\right) + O(h^3)$$
$$\implies y_{n+1} = y + hf + \frac{1}{2}h^2 f_y f + \frac{1}{6}h^3(f_y^2 f + f_{yy}f^2) + O(h^4)$$

But we have $y' = f, y'' = f_y f, y''' = f_y^2 f + f_{yy}f^2$, so the method has order 3 by Taylor's theorem.

Example 4.12. It is possible to prove that the 2-stage Gauss-Legendre method

$$k_1 = f\left(t_n + \left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right)h, y_n + \frac{1}{4}hk_1 + \left(\frac{1}{4} - \frac{\sqrt{3}}{6}\right)hk_2\right)$$
$$k_2 = f\left(t_n + \left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right)h, y_n + \left(\frac{1}{4} + \frac{\sqrt{3}}{6}\right)hk_1 + \frac{1}{4}hk_2\right)$$
$$y_{n+1} = y_n + \frac{1}{2}h(k_1 + k_2)$$

has order 4. It can be easily verified that for $y' = \lambda y$ we have $y_n = (r(h\lambda))^n y_0$ where

$$r(z) = \frac{12 + 6z + z^2}{12 - 6z + z^2}$$

which allows us to prove that $D \supset \mathbb{C}^-$. So Gauss-Legendre method is A-stable as well.

4.5 Implementation of ODE Methods

It all comes down to this: We have to write an algorithm to apply what we have learnt mathematically about numerical methods in ODEs. But when we do this, we can't make the step size as an input as that wouldn't be much of a blackboxing. Instead, what we should working on is the error tolerance: Given an error bound we are after, can we find a corresponding step size so as to control the error?
The first investigation of ours is known as the Milne device. Suppose we wish to monitor the error of the trapezoidal rule

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

This method has order 2. More precisely, we have

$$y(t_{n+1}) - \left(y(t_n) + \frac{h}{2}(y'(t_n) + y'(t_{n+1}))\right) = -\frac{1}{12}h^3 y^{(3)}(t_n) + O(h^4)$$

The value $c_{\mathrm{TR}} = -1/12$ is known as the error constant of trapezoidal tule. Similarly, each multistep methods has its own error constant in an analogous way. For example, the second order 2-step Adams-Bashforth method

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

has error constant $c_{\mathrm{AB}} = 5/12$. Notably, the error constant of a multistep method is exactly the leading coefficient in the Taylor expansion of the expression in Theorem 4.2.
The idea behind the Milne device is to use two multistep methods of the same order, one explicit (predictor) and the other implicit (corrector), to estimate the local error of the implicit method. How do we do this? In this example (second-order 2-step Adams-Bashforth as predictor and trapezoidal rule as corrector), we have

$$\begin{cases} y_{n+1}^{\mathrm{AB}} \approx y(t_{n+1}) - c_{\mathrm{AB}} h^3 y^{(3)}(t_n) \\ y_{n+1}^{\mathrm{TR}} \approx y(t_{n+1}) - c_{\mathrm{TR}} h^3 y^{(3)}(t_n) \end{cases}$$

Combining these two estimations gives

$$y_{n+1}^{\mathrm{AB}} - y_{n+1}^{\mathrm{TR}} \approx (c_{\mathrm{TR}} - c_{\mathrm{AB}}) h^3 y^{(3)}(t_n) \implies h^3 y^{(3)}(t_n) \approx \frac{y_{n+1}^{\mathrm{AB}} - y_{n+1}^{\mathrm{TR}}}{c_{\mathrm{TR}} - c_{\mathrm{AB}}}$$

Therefore,

$$y_{n+1}^{\mathrm{TR}} - y(t_{n+1}) \approx -c_{\mathrm{TR}} h^3 y^{(3)}(t_n) \approx c_{\mathrm{TR}} \frac{y_{n+1}^{\mathrm{AB}} - y_{n+1}^{\mathrm{TR}}}{c_{\mathrm{AB}} - c_{\mathrm{TR}}} = -\frac{1}{6}(y_{n+1}^{\mathrm{AB}} - y_{n+1}^{\mathrm{TR}})$$

which allows us to estimate the local error with no knowledge about the exact solution. This (or rather the absolute value of this) is called the Milne device. Note that since Adams-Bashforth is explicit, doing error control in this fashion does not add a big computational cost. It is also clear why we want to use the trapezoidal rule as chief output: It is A-stable.
We can, of course, use this method in other examples as well, e.g. the third-order Adams-Bashforth

$$y_{n+2} = y_{n+1} + h\left(\frac{5}{12}f(t_{n-1}, y_{n-1}) - \frac{4}{3}f(t_n, y_n) + \frac{23}{12}f(t_{n+1}, y_{n+1})\right)$$

as predictor and third-order Adams-Moulton

$$y_{n+2} = y_{n+1} + h\left(-\frac{1}{12}f(t_n, y_n) + \frac{2}{3}f(t_{n+1}, y_{n+1}) + \frac{5}{12}f(t_{n+2}, y_{n+2})\right)$$

as corrector. In fact, the predictor is useful not just in the estimation of error, but also as an initial guess when solving implicit corrector equations. Typically, we only iterate the solving algorithm for the corrector equations at most twice for nonstiff equations; But for stiff equations, we often require iteration to convergence since we want to exploit the often superior stability of the corrector.

Now we know how to approximate the local error, the natural question next is then the details of the implementation. Usually, it is done as follows: Suppose $\epsilon > 0$ is a tolerence of error (which usually depends on the step size h and the prescribed tolerence of global error). If the estimated error (obtained from the Milne device) is above ϵ, then we reject the step and recommence the method from the last step with a smaller h. If it is below ϵ but above, say, $\epsilon/10$ (or some other cleverly chosen quantities in $(0, \epsilon)$), then we simply continue the iteration. If however it is below it, then we still accept the step (of course!) but increase the step size, since we still want maximal efficiency under the constraint on error.

For Runga-Kutta methods, the situation is, of course, much more complicated as there is no single error constant that determines local growth of error. The same philosophy, however, still applies and is known as embedded RK. As expected, we need two (typically explicit) RK methods, one of ν stages and order p and another of $\nu + l$ stages for $l \geq 1$ and order $p + 1$ such that the first ν stages of both methods are identical (so that the cost of computation is low). For example, one can pick

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + h/2, y_n + hk_1/2) \\ k_3 &= f(t_n + h, y_n - hk_1 + 2hk_2) \end{aligned}$$

where the two methods are

$$y_{n+1}^{[1]} = y_n + hk_2, y_{n+1}^{[2]} = y_n + \frac{h}{6}(k_1 + 4k_2 + k_3)$$

As one can verify (haha I won't), the first method has order 2 and the second has order 3. The estimation we can use is then $y_{n+1}^{[1]} - y(t_{n+1}) \approx y_{n+1}^{[1]} - y_{n+1}^{[2]}$.

Another way to do error control is the Zadunaisky device. Suppose we aim to approximate the solution to the ODE $y' = f(t, y), y(0) = y_0$ by a numerical method of order p and that we have stored some past values $y_n, \ldots, y_{n-p}$. We form an interpolating polynomial (possibly with vector coefficients) $d \in \mathbb{P}_d[x]$ with $d(t_{n-i}) = p_{n-i}$ for $i = 0, \ldots, p$ and consider the differential equation $z'(t) = f(t, z) + d'(t) - f(t, d), z(t_n) = y_n$. This can be viewed as a perturbation of the original ODE since $d(t) - y(t) = O(h^{p+1})$ (thus $d'(t) - f(t, d)$ is expected to be small). However, we can actually solve this equation: $z(t) = d(t)$. Therefore, when we compute y_{n+1} with our numerical method, we can also use the same numerical method but on this new ODE to obtain a numerical approximation z_{n+1} to z. The error $z_{n+1} - d(t_{n+1})$ is then a reasonable estimation of the error $y_{n+1} - y(t_{n+1})$.

4.6 Solving Nonlinear Equations

If we want to use an implicit numerical method, we often have to solve a nonlinear equation. For example, in an s-step method, we have to solve an equation in the form

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

where v is a function of known values from preceding iterations. There are a few ways to do this. One approach is functional iteration

$$y_{n+s}^{[j+1]} = \sigma_s h f(t_{n+s}, y_{n+s}^{[j]}) + v$$

where $y_{n+s}^{[0]}$ is provided by the predictor scheme. This is effective for non-stiff equations but fails miserably for stiff ones since the convergence of this scheme requires unpleasant restriction on h. For stiff equations, we often use the Newton-Rhapson method

$$y_{n+s}^{[j+1]} = y_{n+s}^{[j]} - (D\psi(y_{n+s}^{[j]}))^{-1}\psi(y_{n+s}^{[j]})$$

where $\psi(y) = y - \sigma_s h f(t_{n+s}, y) - v$ (so the solution we want to solve now has the form $\psi(y_{n+s}) = 0$). Why does this work? Well, it doesn't always do. However, in some nice cases, this does converge pretty fast to the actual value. This can, of course, be justified from a Taylor expansion (welp) of ψ around $y_{n+s}^{[j]}$ which gives

$$0 = \psi(y_{n+s}) \approx \psi(y_{n+s}^{[j]}) + D\psi(y_{n+s}^{[j]})(y_{n+s} - y_{n+s}^{[j]})$$

Rearranging this gives the recursion. We also need to choose an initial value $y_{n+s}^{[0]}$ that is suitably close to y_{n+s} so that everything is nice. In particular, we will want $D\phi$ is invertible in a suitable region containing y_{n+s} and $y_{n+s}^{[0]}$.
There is another problem: Evaluating and inverting the derivative of ψ at each step is hugely expensive. So sometimes we modify the iteration to the form

$$y_{n+s}^{[j+1]} = y_{n+s}^{[j]} - (D\psi(y_{n+s}^{[0]}))^{-1}\psi(y_{n+s}^{[j]})$$

i.e. fixing the derivative term. This discussion also suggest that we might also need to explore cheap methods to solve linear equations, so that is what we shall do.

5 Numerical Linear Algebra

5.1 LU Factorisation and its Generalisations

Definition 5.1. Let A be be a real square matrix. We say the matrices L, U form an LU (Upper-Lower) factorisation of A if L is unit lower triangular, i.e. it is lower triangular and $L_{ii} = 1$ for all i, U is upper triangular, and $A = LU$.

If an LU factorisation $A = LU$ is given, then we can calculate many things with very low computational cost. For example, $\det A = (\det L)(\det U)$ is just the product of diagonal entries of U, which is obviously a much faster method than evaluating $\det A$ by Laplace expansion. Consequently, to check if A is singular, we only have to look for zero entries in the diagonal of U. In addition, if we want to solve the linear system $Ax = b \iff LUx = b$, we can factor it through the linear systems $Ly = b, Ux = y$, both are very easy to calculate since they are triangular. By the same philosophy we can calculate $A^{-1} = U^{-1}L^{-1}$ very cheaply as well.
So, how do we calculate the LU factorisation? Let's work backwards: Suppose we already have L with columns $l_1, \dots, l_n$ and U with rows $u_1^\top, \dots, u_n^\top$, then

$$A = LU = \begin{pmatrix} l_1 & \cdots & l_n \end{pmatrix} \begin{pmatrix} u_1^\top \\ \vdots \\ u_n^\top \end{pmatrix} = \sum_{k=1}^{n} l_k u_k^\top$$

where for each k, $l_k u_k^\top$ is a rank one (or zero in the degenerate case) matrix having zeros in its first $k-1$ rows and columns. This decomposition gives an obvious way to work from A to l_k, u_k. $u_1^\top$ is simply the first row of A and l_1 the first column of A divided by $A_{1,1}$. Having found $u_1^\top$ and l_1, we can form the matrix $A_1 = A - l_1 u_1^\top$ which has zeros in the first row and first column. It then follows that $u_2^\top$ can be taken as the second row of A_1 and l_2 is the second column scaled so that $L_{2,2} = 1$.
In general, we set $A_0 = A$, $A_k = A_{k-1} - l_k u_k^\top$, $u_k^\top$ the k^{th} row of A_{k-1} and l_k the k^{th} column of A_{k-1} scaled appropriately so that $L_{k,k} = 1$. At each step, the dominant cost is to evaluate $l_k u_k^\top$ which has complexity $(n-k+1)^2$, so the total cost of the algorithm is $O(n^3)$.
The LU factorisation has a canonical relation to the Gaussian elimination. Indeed, since elementary row operations are essentially the operation of left-multiplying with a unit lower triangular matrix, what we are doing is essentially finding an upper triangular U and a bunch of unit lower triangular $L_1, \ldots, L_m$ such that $L_m L_{m-1} \cdots L_1 A = U$, which can be written as $A = LU$ where $L = L_1^{-1} \cdots L_m^{-1}$ is unit lower triangular. Gaussian elimination also has complexity $O(n^3)$, but we prefer LU factorisation in many situations in numerical analysis. This is because Gaussian elimination only solves one linear system at a time while, for LU factorisation, we can repeatedly take advantage of the simplified subsequent steps once we have obtained L and U.
The algorithm, however, does not always work: It fails precisely when normalising l_k is not possible, i.e. $(A_{k-1})_{k,k} = 0$. The solution to this is to employ a method called pivoting, which modifies the algorithm as follows:
In each step, we find some $p \geq k$ such that $(A_{k-1})_{p,k} \neq 0$. Let P_k be the permutation matrix that swaps k and p, i.e. $P_k = I - E_{k,k} - E_{p,p} + E_{p,k} + E_{k,p}$ (where as usual $E_{i,j}$ is the matrix with entry 1 at i, j and 0 anywhere else). Then we can take $u_k^\top$ to be the k^{th} row of $P_k A_{k-1}$,

$$l_k = \frac{1}{(P_k A_{k-1})_{k,k}} (k^{th} \text{ column of } P_k A_{k-1})$$

and $A_k = P_k A_{k-1} - l_k u_k^\top$.
What would happen when we unroll the algorithm? Indeed,

$$(P_{n-1} \cdots P_1) A = \tilde{l}_1 u_1^\top + \cdots + \tilde{l}_n u_n^\top, \tilde{l}_k = (P_{n-1} \cdots P_{k+1}) l_k$$

Note that the first $k-1$ columns of $\tilde{l}_k$ are still zero, therefore if we let $\tilde{L}$ be the matrix whose columns are $\{\tilde{l}_k\}$, then we obtain an LU factorisation $PA = \tilde{L}U$ of PA where $P = P_{n-1} \cdots P_1$ is a permutation matrix.
This new algorithm can still fail, since there can be some k where all entries in the k^{th} column are zero. But in this case, we can just choose $l_k = e_k$ (where e_k is, as usual, the column vector with 1 at the k^{th} entry and 0 elsewhere), $u_k^\top$ the k^{th} row of A_{k-1} and $P_k = I$. Easy to see that it works as expected.
In conclusion, for any matrix A, there exists a permutation matrix such that PA has an LU factorisation.
Note that we are free to choose $p \geq k$ when doing pivoting in each step. A common choice is such that $|(A_{k-1})_{p,k}|$ is maximum, which ensures in particular that the entries of l_k are bounded above by 1 in magnitude. This gives a certain numerical stability to the algorithm.
These are for general matrices, but when the matrix is nice, we can make things even nicer.

Definition 5.2. A matrix A is symmetric if $A_{k,l} = A_{l,k}$.

In the case where A is symmetric, since we know from linear algebra that A can be diagonalised nicely, we certainly won't be satisfied with just LU factorisation. We want a factorisation of the form $A = LDL^\top$ where L is unit lower triangular and D is diagonal. $U = DL^\top$ reduces it to LU factorisation.
Again, if $l_1, \ldots, l_k$ are the columns of L, then the factorisation takes the form

$$A = \sum_{k=1}^{n} D_{k,k} l_k l_k^\top$$

A very similar algorithm works: Set $A_0 = A$ and simply take l_k to be the k^{th} column of A_{k-1} scaled appropriately, $D_{k,k} = (A_{k-1})_{k,k}$ and $A_k = A_{k-1} - D_{k,k} l_k l_k^\top$.

Example 5.1. Let's work with

$$A = A_0 = \begin{pmatrix} 2 & 4 \\ 4 & 11 \end{pmatrix}$$

Casting the algorithm gives $l_1 = (1, 2)^\top, D_{1,1} = 2$ and

$$A_1 = A_0 - D_{1,1} l_1 l_1^\top = \begin{pmatrix} 0 & 0 \\ 0 & 3 \end{pmatrix}$$

So indeed $l_2 = (0, 1)^\top, D_{2,2} = 3$ and

$$A = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$$

is the factorisation we want.

Definition 5.3. A symmetric matrix A is positive definite if $x^\top Ax > 0$ for all $x \neq 0$.

Theorem 5.1. *A real symmetric matrix A is positive definite if and only if it has an $LDL^\top$ factorisation such the diagonal elements of D are all positive.*

Proof. The "if" part is trivial. Casting the algorithm as described earlier gives the converse. □

If $A = LDL^\top$ is positive definite, we can define $D^{1/2}$ by $(D^{1/2})_{ij} = \sqrt{D_{ij}}$. This allows us to write $A = \tilde{L}\tilde{L}^\top$ where $\tilde{L} = LD^{1/2}$. This is known as the Cholesky factorisation.
Another class of matrices of interest consists of the sparse matrices, i.e. systems with very large dimension yet most entries vanish. This is significant in the sense that we often need to solve sparse linear systems in numerical methods for ODEs and PDEs.
Naturally, we will want the LU factorisation of a sparse matrix to inherit as much of the sparsity as possible.

Theorem 5.2. *Suppose $A = LU$ is an LU factorisation, 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. Say $U_{k,k} \neq 0$ for all k. If $A_{i,1} = 0$, then $L_{i,1}U_{1,1} = 0 \implies L_{i,1} = 0$. If $A_{i,2} = 0$, then $L_{i,1}U_{1,2} + L_{i,2}U_{2,2} = 0 \implies L_{i,2} = 0$. Continuing this process shows the theorem. □

Definition 5.4. A matrix A is a banded matrix if there exists an integer $r < n$ such that $A_{i,j} = 0$ for $|i - j| > r$.

That is, all nonzero entries of A are contained in a band along the diagonal. Then, if $A = LU$ is an LU factorisation of a banded matrix, then both L, U inherit the same banded structure. Knowing this, if we want to calculate the LU factorisation of an r-banded matrix, the complexity of the algorithm would be just $O(r^2 n)$ instead of $O(n^3)$, which is very helpful when r is small compared to n (i.e. when the matrix is sparse enough).
How about general sparse matrices? There are a lot of methods in this direction, like efficient factorisation (which minimises the fill-in of zero entries), some iterative methods that will be covered in Part II Numerical Analysis, fast fourier transforms, preconditioned conjugate gradients, multigrid techniques, and much more.
A hugely powerful method of ordering pivots to minimise fill-in of sparse matrices uses graph theory and many other cool stuff, but those are out of the scope of this course.

5.2 QR Factorisation

We work in $\mathbb{R}^n$ with the usual scalar product $\langle u, v\rangle = u^\top v$.

Definition 5.5. A set of vectors $\{q_i\}$ form an orthonormal set if $\langle q_i, q_j\rangle = \delta_{ij}$. A square matrix Q is orthogonal if its columns form an orthonormal set, i.e. $Q^\top Q = I$.

In particular, if Q is orthogonal then it has determinant ± 1, and is invertible with inverse $Q^\top$.

Definition 5.6. For an $m \times n$ matrix A, a QR factorisation of A is a factorisation $A = QR$ where Q is $m \times m$ and orthogonal and R is $m \times n$ and upper triangular. When $m \geq n$, a reduced QR factorisation is a factorisation $A = QR$ where Q is $m \times n$ with orthonormal columns and R is $n \times n$ and upper triangular.

If a QR factorisation $A = QR$ is available, then we can solve $Ax = b$ by $Qy = b, Rx = y$, both extremely easy and computationally efficient.
How do we interpret a (reduced) QR factorisation? Suppose $m \geq n$, $a_1, \ldots, a_n$ are columns of A and $q_1, \ldots, q_n$ columns of Q, then

$$a_k = \sum_{j=1}^{k} R_{jk} q_j$$

So the k^{th} column of A is a linear combination of the first k columns of Q. This hints that we can conversely obtain a reduced QR factorisation from an orthogonalisation process of the columns of A, in the case where they are linearly independent.
Assume $m \geq n$ and that the columns of A are linearly independent, we shall describe an algorithm to compute a reduced QR factorisation of A. We can do

this just by looking at the formula $a_k = \sum_{j=1}^{k} R_{jk} q_j$: It immediately gives $q_1 = a_1/\|a_1\|$ and $R_{1,1} = \|a_1\|$. Next, we find that $b = a_2 - \langle q_1, a_2\rangle q_1$ is orthogonal to q_1. $b \neq 0$ as we assumed the columns of A to be linearly independent. So we can set $q_2 = b/\|b\|$ and $R_{1,2} = \langle q_1, a_2\rangle, R_{2,2} = \|b\|$. More generally, we use the classic Gram-Schmidt algorithm: Set $q_1 = a_2/\|a_1\|$ and $R_{1,1} = \|a_1\|$. Having computed $q_1, \ldots, q_{j-1}$, set $R_{i,j} = \langle q_i, a_j\rangle$ for $i \leq j-1$, $b_j = a_j - \sum_{i=1}^{j-1} R_{i,j} q_i$, $q_j = b_j/\|b_j\|$ and $R_{j,j} = \|b_j\|$.
The total cost of this algorithm is $O(n^2 m)$. However, there is a problem: When we compute everything at finite arithmetic, small imprecisions in the calculation of inner product can have a great influence on the numerical accuracy.
Another approach to view the factorisation problem is to take it as orthogonal triangularisation: Repeatly multiply the original matrix by orthogonal matrices until we get an upper triangular product. This certainly will produce a QR transformation given that this process does terminate. So our goal is to find a nice set of orthogonal matrices that does the trick efficiently.

Definition 5.7. An $m \times m$ Givens rotation matrix $\Omega = \Omega^{[p,q]} = \Omega^{[p,q]}(\theta)$ (where $1 \leq p < q \leq n, \theta \in (-\pi, \pi]$) with $\Omega^{[p,q]}_{p,p} = \Omega^{[p,q]}_{q,q} = \cos\theta, \Omega^{[p,q]}_{p,q} = \sin\theta, \Omega^{[p,q]}_{q,p} = -\sin\theta$ and $\Omega^{[p,q]}_{i,j} = \delta_{ij}$ whenever $\{i,j\} \not\subseteq \{p,q\}$.

So a Givens rotation $\Omega^{[p,q]}$ is just the natural extension of a rotation in the 2-dimension subspace of $\mathbb{R}^m$ spanned by $\{e_p, e_q\}$.

Example 5.2. For $m = 4$,

$$\Omega^{[1,2]} = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \Omega^{[2,4]} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & 0 & \sin\theta \\ 0 & 0 & 1 & 0 \\ 0 & -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

Theorem 5.3. *Let A be an $m \times n$ matrix, then for every $1 \leq p < q \leq m, i \in \{p,q\}$ and $1 \leq j \leq n$, there exists $\theta \in (-\pi, \pi]$ such that $\Omega^{[p,q]} = \Omega^{[p,q]}(\theta)$ has:*
1. $(\Omega^{[p,q]} A)_{i,j} = 0$.
2. The rows of $\Omega^{[p,q]} A$ coincides with the corresponding rows of A except for the p^{th} or q^{th} row.
3. The p^{th} and q^{th} rows of $\Omega^{[p,q]} A$ are linear combinations of the p^{th} and q^{th} rows of A.

Proof. The second and third statements are obviously true for any θ. Suppose $i = q$. If $A_{p,j} = A_{q,j} = 0$ then any θ would do. Otherwise, take θ such that

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

which works. For $i = p$, we do a similar thing

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

That's it. □

This provides an algorithm to triangularise a matrix, which is known as the Given algorithm. In the procedure, we will always take $(i, j) = (q, p)$.

Example 5.3. Suppose A is 3×3, then $\Omega^{[2,3]}\Omega^{[1,3]}\Omega^{[1,2]}A$ is upper triangular by certain suitable choices of θ.

The order in which we apply the Givens rotations is important since we don't want to destroy the existing zeros. One way of achieving this would be to follow the lexicographic order of $[p, q]$ due to part 3 of the theorem. This shall yield a QR factorisation of the form $A = QR$ where Q is a product of $(\Omega^{[p,q]})^\top$. We don't always have to multiply it out – knowing Q in the form of a product of orthogonal matrix already helps a lot in most applications.
The total cost of the algorithm is $O(n^2 m)$ which is the same as Gram-Schmidt, but it can be applied to more general settings.
Another useful set of orthogonal matrices is Householder reflections.

Definition 5.8. Let $u \in \mathbb{R}^m \setminus \{0\}$. The $m \times m$ matrix $I - 2uu^\top/\|u\|^2$ is called a Householder reflection.

One can easily verify that this is indeed orthogonal. Also, if H is the Householder reflection wrt u, then $Hu = -u$; If $v \in (\text{span}\{u\})^\perp$, then $Hv = v$. In addition, if $a, b \in \mathbb{R}^m$ has $\|a\| = \|b\|$ and $u = a - b$, then $Ha = b$. In particular, if we choose $b = \|a\|e_1$ (so $u = a - \|a\|e_1$), then we have $Ha = \|a\|e_1$. This will be the basis of the algorithm.
The Householder algorithm multiplies A by a sequence of Householder reflections so that each product induces zeros under the diagonal in a column. At the start, we week a reflection that transforms the first column a_1 of A to a multiple of e_1. Since the Householder reflection is orthogonal, this multiple must be $\pm\|a_1\|e_1$. Previous discussions gives us the idea of choosing $u = a_1 \pm \|a_1\|e_1$ which indeed gives the desired result. For numerical stability, we usually choose the sign to be the sign of A_{11}.
More generally, after the columns 1 to $k-1$ have been processed and have zeros under the diagonal, we would want to find a Householder reflection that cleans up the sub-diagonal elements of the k^{th} column as well. This is given by a block-diagonal matrix $\left(\begin{smallmatrix} I & 0 \\ 0 & H \end{smallmatrix}\right)$ where I is the $(k-1) \times (k-1)$ identity matrix and H is a $(m-k+1) \times (m-k+1)$ Householder reflection (in the subspace generated by the last $m - k + 1$ coordinates) associated with $\tilde{u} = \tilde{a}_k + \text{sign}(A_{kk})\|\tilde{a}_k\|e_k$. In practice, we don't always have to write out H explicitly at each step if we only wanted R: Let $A_{k:m,j}$ denote the vector of size $m - k + 1$ obtained from rows $k, \ldots, m$ of column j of A, then in each step what we do reduces to the recursion $A_{k:m,j} = A_{k:m,j} - 2(\tilde{u}^\top A_{k:m,j})\tilde{u}/\|\tilde{u}\|^2$ for $j \geq k$.

Example 5.4. Suppose we have

$$A = \begin{pmatrix} 2 & 4 & 7 \\ 0 & 3 & -1 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & -2 \end{pmatrix}$$

where the first two columns have already been processed. Then

$$\tilde{a}_3 = \begin{pmatrix} 2 \\ 1 \\ -2 \end{pmatrix} \implies \tilde{u} = \begin{pmatrix} 5 \\ 1 \\ -2 \end{pmatrix}$$

So we get the final result

$$\begin{pmatrix} 2 & 4 & 7 \\ 0 & 3 & -1 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

To recover Q, we start with $\Omega = I$ and for each step we replace Ω by $(I - 2uu^\top/\|u\|^2)\Omega$ where $u = \binom{0}{\tilde{u}}$. Then we can say $Q = \Omega^\top$. If we only need $Q^\top b = \Omega b$ for some b, then the iteration can be simplified by initialising $c = b$ and, in each stage, replacing c by $(I - 2uu^\top/\|u\|^2)c$. The whole algorithm, again, costs $O(n^2 m)$.
So, is Givens or Householder better? For non-sparse matrices, it is obviously more convenient to use Householder reflections. But if A has many leading zeros in its rows, then Givens rotations can be more efficient. For example, if an $n \times n$ matrix A consisting of zeros underneath the first subdiagonal, then we only need $n - 1$ Givens rotations to get what we want, reducing the cost to $O(n^2)$.

5.3 Linear Least Squares

Suppose we have an $m \times n$ matrix A and a vector $b \in \mathbb{R}^m$. The equation $Ax = b$ for $x \in \mathbb{R}^n$ has, in general, no solution for $n < m$. In real life, we often see problems in this form where $n << m$, e.g. in an n-variable linear model with m observations. Since there is no exact solution, our focus turns to optimising certain related quantities. In this section, we want to investigate the least squares problem: Minimise $\|Ax - b\|$.

Theorem 5.4. *$x \in \mathbb{R}^n$ is a solution to the least squares problem iff $A^\top(Ax - b) = 0$.*

Proof. The "only if" direction follows from calculus. For the "if" part, suppose $A^\top(Ax - b) = 0$ and $u \in \mathbb{R}^n$, then

$$\begin{aligned} \|Au - b\|^2 &= \|Ax - b\|^2 + 2(u - x)^\top A^\top (Ax - b) + \|A(u - x)\|^2 \\ &= \|Ax - b\|^2 + \|A(u - x)\|^2 \geq \|Ax - b\|^2 \end{aligned}$$

As desired. □

How do we solve the equation $A^\top(Ax - b) = 0$? Of course, we can attempt to solve the equation $A^\top Ax = A^\top b$. However, there are a few problems from a numerical analyst's point of view: Apart from the obvious issues that arise from a singular $A^\top A$, we can also lose sparsity of A from forming $A^\top A$ which largely reduces efficiency. Also, calculating $A^\top A$ can lose a lot of accuracy especially when the entries have too large or too small absolute values.
How to do it in a numerically efficient way? We make use of the QR factorisation. Suppose A is $m \times n$ with $m \geq n$ and let $A = QR$ be a reduced QR factorisation where Q is $m \times n$ has orthogonal columns and R is $n \times n$ upper triangular. We know that x is a solution to the least squares problem iff $Ax - b$ is orthogonal to all columns of A, so we only have to solve $Rx = Q^\top b$ which can be done easily.