Numerical Analysis Hilary Term 2026
Lecture 1: Lagrange Interpolation

Numerical analysis is the study of computational algorithms for solving problems in scientific computing. It combines mathematical beauty, rigor and numerous applications; we hope you'll enjoy it! In this course we will cover the basics of three key fields in the subject:

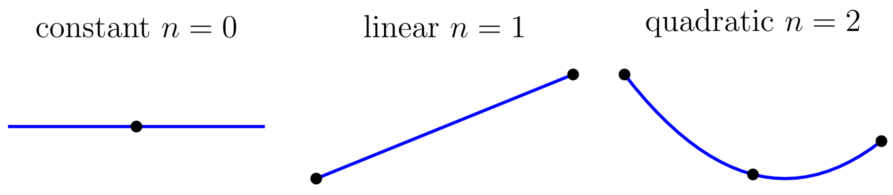
- Approximation Theory (lectures 1, 9–11); recommended reading: L. N. Trefethen, Approximation Theory and Approximation Practice, and E. Süli and D. F. Mayers, An Introduction to Numerical Analysis.
- Numerical Linear Algebra (lectures 2–8); recommended reading: L. N. Trefethen and D. Bau, Numerical Linear Algebra.
- Numerical Solution of Differential Equations (lectures 12–16); recommended reading: E. Süli and D. F. Mayers, An Introduction to Numerical Analysis.

This first lecture comes from Chapter 6 of Süli and Mayers, and Chapter 10+15 of Trefethen.

Notation: $\Pi_n = \{\text{real polynomials of degree} \le n\}$

Setup: Given data f_i at distinct x_i, $i = 0, 1, \dots, n$, with $x_0 < x_1 < \cdots < x_n$, can we find a polynomial p_n such that $p_n(x_i) = f_i$? Such a polynomial is said to **interpolate** the data, and (as we shall see) can approximate f at other values of x if f is smooth enough. This is the most basic question in approximation theory.

E.g.:



Theorem. Let $x_0 < x_1 < \cdots < x_n$, and $f_i \in \mathbb{R}$ for $i = 0, \dots, n$. There exists a polynomial interpolant $p_n \in \Pi_n$ such that $p_n(x_i) = f_i$ for $i = 0, 1, \dots, n$.

Proof. Consider, for $k = 0, 1, \dots, n$, the "cardinal polynomial" (aka Lagrange basis polynomial)

$$L_{n,k}(x) = \frac{(x - x_0)\cdots(x - x_{k-1})(x - x_{k+1})\cdots(x - x_n)}{(x_k - x_0)\cdots(x_k - x_{k-1})(x_k - x_{k+1})\cdots(x_k - x_n)} \in \Pi_n. \tag{1}$$

Then $L_{n,k}(x_i) = \delta_{ik}$, that is,

$$L_{n,k}(x_i) = 0 \;\text{ for }\; i = 0, \dots, k-1, k+1, \dots, n \;\text{ and }\; L_{n,k}(x_k) = 1.$$

So now define

$$p_n(x) = \sum_{k=0}^{n} f_k L_{n,k}(x) \in \Pi_n \tag{2}$$

$\Longrightarrow$

$$p_n(x_i) = \sum_{k=0}^{n} f_k L_{n,k}(x_i) = f_i \ \text{ for } \ i = 0, 1, \ldots, n. \qquad \square$$

The polynomial (2) is the **Lagrange interpolating polynomial**.

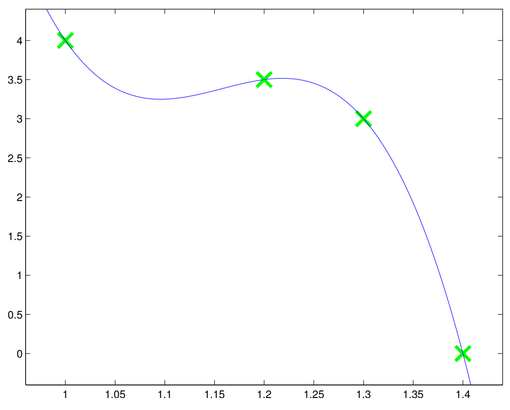
Theorem. The interpolating polynomial p_n of degree $\leq n$ is unique.

Proof. Consider two interpolating polynomials $p_n, q_n \in \Pi_n$. Their difference $d_n = p_n - q_n \in \Pi_n$ satisfies $d_n(x_k) = 0$ for $k = 0, 1, \ldots, n$. i.e., d_n is a polynomial of degree at most n but has at least $n+1$ distinct roots. Fundamental theorem of algebra $\Longrightarrow$ $d_n \equiv 0 \Longrightarrow$ $p_n = q_n$. $\square$

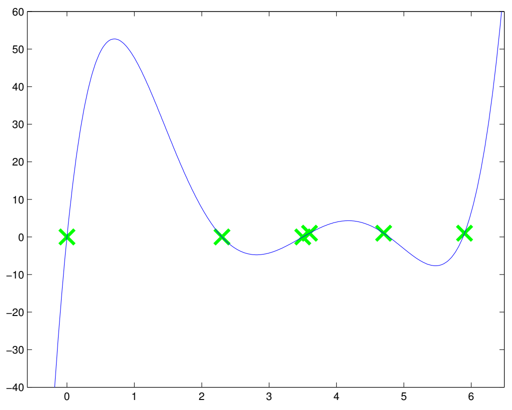
Matlab:

```
>> help lagrange
 LAGRANGE  Plots the Lagrange polynomial interpolant for the
 given DATA at the given NODES/interpolation points

>> lagrange([1,1.2,1.3,1.4],[4,3.5,3,0]);
```



```
>> lagrange([0,2.3,3.5,3.6,4.7,5.9],[0,0,0,1,1,1]);
```



Data from an underlying smooth function: Suppose that $f(x)$ has at least $n+1$ smooth derivatives in the interval (x_0, x_n). Let $f_k = f(x_k)$ for $k = 0, 1, \ldots, n$, and let p_n be the Lagrange interpolating polynomial for the data (x_k, f_k), $k = 0, 1, \ldots, n$.

Error: How large can the error $f(x) - p_n(x)$ be on the interval $[x_0, x_n]$?

Theorem. Suppose that f is $(n+1)$-times continuously differentiable. Let p_n be the polynomial interpolant at $\{x_i\}_{i=0}^n$. For every $x \in [x_0, x_n]$ there exists $\xi = \xi(x) \in (x_0, x_n)$ such that

$$e(x) \overset{\text{def}}{=} f(x) - p_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n) \frac{f^{(n+1)}(\xi)}{(n+1)!}, \tag{3}$$

where $f^{(n+1)}$ is the $(n+1)$-st derivative of f.

Proof. Trivial for $x = x_k$, $k = 0, 1, \ldots, n$ as $e(x) = 0$ by construction. So suppose $x \neq x_k$. Let

$$\phi(t) \overset{\text{def}}{=} e(t) - \frac{e(x)}{\pi(x)} \pi(t),$$

where

$$\begin{aligned} \pi(t) &\overset{\text{def}}{=} (t - x_0)(t - x_1) \cdots (t - x_n) \\ &= t^{n+1} - \left(\sum_{i=0}^{n} x_i \right) t^n + \cdots (-1)^{n+1} x_0 x_1 \cdots x_n \\ &\in \Pi_{n+1}. \end{aligned}$$

Now note that ϕ vanishes at $n+2$ points x and x_k, $k = 0, 1, \ldots, n$. $\implies$ ϕ' vanishes at $n+1$ points $\xi_0, \ldots, \xi_n$ between these points $\implies$ ϕ'' vanishes at n points between these new points, and so on until $\phi^{(n+1)}$ vanishes at an (unknown) point ξ in (x_0, x_n). But

$$\phi^{(n+1)}(t) = e^{(n+1)}(t) - \frac{e(x)}{\pi(x)} \pi^{(n+1)}(t) = f^{(n+1)}(t) - \frac{e(x)}{\pi(x)} (n+1)!$$

since $p_n^{(n+1)}(t) \equiv 0$ and because $\pi(t)$ is a monic polynomial of degree $n+1$. The result then follows immediately from this identity since $\phi^{(n+1)}(\xi) = 0$.

$\square$

The above proof may seem ingenious/mysterious. It is perhaps helpful to observe the connections and similarity to Taylor's theorem with remainder, and its proof. Indeed the latter can be seen as a special case of the theorem when x_i all tend to a single point $x_i \to x_*$ and one interpolates high-order derivatives at a single point.

Example: $f(x) = \log(1+x)$ on $[0, 1]$. Here, $|f^{(n+1)}(\xi)| = n!/(1+\xi)^{n+1} < n!$ on $(0, 1)$. So $|e(x)| < |\pi(x)| n!/(n+1)! \leq 1/(n+1)$ since $|x - x_k| \leq 1$ for each x, x_k, $k = 0, 1, \ldots, n$, in $[0, 1] \implies |\pi(x)| \leq 1$. This is probably pessimistic for many x, e.g. for $x = \frac{1}{2}$, $\pi(\frac{1}{2}) \leq 2^{-(n+1)}$ as $|\frac{1}{2} - x_k| \leq \frac{1}{2}$.

This shows the important fact that the error can be large at the end points when the interpolation points $\{x_k\}$ are equispaced, an effect known as the "Runge phenomena" (Carl Runge, 1901), which we see below and return to in lecture 4. More generally, as the expression (3) suggests, the location of the samples $\{x_k\}$ is very important, as we see at the end.

Generalisation: Given data f_i and g_i at distinct x_i, $i = 0, 1, \dots, n$, with $x_0 < x_1 < \dots < x_n$, can we find a polynomial p such that $p(x_i) = f_i$ and $p'(x_i) = g_i$? (i.e., interpolate derivatives in addition to values)

Theorem. There is a unique polynomial $p_{2n+1} \in \Pi_{2n+1}$ such that $p_{2n+1}(x_i) = f_i$ and $p'_{2n+1}(x_i) = g_i$ for $i = 0, 1, \dots, n$.

Construction: Given $L_{n,k}(x)$ in (1), let

$$\begin{aligned} H_{n,k}(x) &= [L_{n,k}(x)]^2(1 - 2(x - x_k)L'_{n,k}(x_k)) \\ \text{and} \quad K_{n,k}(x) &= [L_{n,k}(x)]^2(x - x_k). \end{aligned}$$

Then

$$p_{2n+1}(x) = \sum_{k=0}^{n}[f_k H_{n,k}(x) + g_k K_{n,k}(x)] \tag{4}$$

interpolates the data as required. The polynomial (4) is called the **Hermite interpolating polynomial**. Note that $H_{n,k}(x_i) = \delta_{ik}$ and $H'_{n,k}(x_i) = 0$, and $K_{n,k}(x_i) = 0$, $K'_{n,k}(x_i) = \delta_{ik}$.

Theorem. Let p_{2n+1} be the Hermite interpolating polynomial in the case where $f_i = f(x_i)$ and $g_i = f'(x_i)$ and f has at least $2n+2$ smooth derivatives. Then, for every $x \in [x_0, x_n]$,

$$f(x) - p_{2n+1}(x) = [(x - x_0)(x - x_1) \cdots (x - x_n)]^2 \frac{f^{(2n+2)}(\xi)}{(2n+2)!},$$

where $\xi \in (x_0, x_n)$ and $f^{(2n+2)}$ is the $(2n+2)$nd derivative of f.

Proof (non-examinable): see Süli and Mayers, Theorem 6.4. □

Chebyshev interpolation (New 2026:) Let us return to interpolating the function values f_i and not the derivative. The error expression (3) is exact, but does not tell us how best (or near-best) to choose the interpolation points $\{x_i\}_{i=0}^n$ (assuming we get to choose them). One might expect the choice depends strongly on f in a complicated fashion. Fortunately, this is far from the truth! We now introduce Chebyshev interpolation, a cornerstone tool in numerical analysis. In brief, taking $\{x_i\}_{i=0}^n$ to be the *Chebyshev points*[1]

$$x_i = \cos\theta_i, \quad \theta_i = \frac{(2i+1)\pi}{2(n+1)}, \qquad i = 0, \dots, n, \tag{5}$$

results in p_n being a near-optimal approximation to f in the sense of (8) with $\Lambda_n = O(\log n)$, *regardless* of f. The Chebyshev points x_i are displayed below for $n = 50$. Note the points cluster near the endpoints ± 1.

[1]These are the roots of T_{n+1}, the $(n+1)$st Chebyshev polynomial, to be introduced in the lecture on orthogonal polynomials. Another common choice is $\theta_i = \frac{i\pi}{n}$, also called Chebyshev points. This set gives essentially the same Lebesgue constant (7), i.e., $\Lambda_n \approx \log n$. The crucial point is that they cluster x_i near the endpoints ± 1 following a cos-distribution; we will see this effect again with the nodes in Gauss quadrature, Lecture 11.

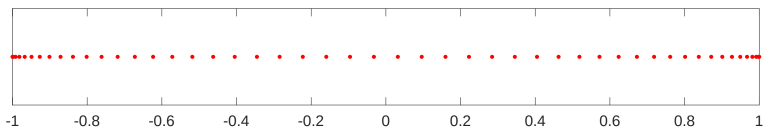


Figure 1: Chebyshev points (5) for $n = 50$.

To make this precise, we prepare two definitions.

Best polynomial approximation: Given a continuous function[2] $f \in C([-1,1])$, a natural question is: what is the optimal approximation $p \in \Pi_n$ to f ? The answer depends on the norm (or metric) used. In the L_∞ norm $\|\cdot\|_\infty$ defined by $\|f\|_\infty = \sup_{x\in[-1,1]} |f|$, the solution exists and is unique (see Trefethen Thm. 10.1) for any $f \in C([-1,1])$. We denote it by p_n^*, that is,

$$\|f - p_n^*\|_\infty = \min_{p_n\in\Pi_n} \|f - p_n\|_\infty. \tag{6}$$

In Sheet 3 you'll explore some pretty properties of p_n^*. For now, note that $\|f - p_n^*\|_\infty \to 0$ as $n \to \infty$ for any continuous f, and be aware that the smoother the f is, the faster the convergence; e.g. if f is holomorphic=analytic on $[-1,1]$, the convergence is exponential $\|f - p_n^*\|_\infty \le C \exp(-cn)$ for some positive constants c and C.

Lebesgue constant: Let $\mathcal{L}_n$ be the interpolation operator that outputs the interpolant p_n given f; that is, $\mathcal{L}_n f = p_n$. Note that $\mathcal{L}_n$ is well-defined once $\{x_i\}_{i=0}^n$ are chosen.

The Lebesgue constant Λ_n is defined[3] as the ∞-norm of the operator $\mathcal{L}_n$:

$$\Lambda_n = \sup_{f\in C([-1,1])} \frac{\|\mathcal{L}_n f\|_\infty}{\|f\|_\infty} \left(= \sup_{f\in C([-1,1])} \frac{\|p_n\|_\infty}{\|f\|_\infty} \right) \tag{7}$$

Here is the interpretation: Given data values on an $(n+1)$-point grid coming from sampling a function with $\|f\|_\infty = 1$, Λ_n is the largest possible value of the polynomial interpolant.

The significance of Λ_n is the following:

Theorem. Let f be continuous $f \in C([-1,1])$[4], and let p_n be the polynomial interpolant of f at $\{x_i\}_{i=0}^n$. With Λ_n as defined in (7),

$$\|f - p_n\|_\infty \le (\Lambda_n + 1)\|f - p_n^*\|_\infty. \tag{8}$$

Proof. We have $\|f - p_n\|_\infty \le \|f - p_n^*\|_\infty + \|p_n^* - p_n\|_\infty$. The key step is to note that $p_n^* - p_n = \mathcal{L}_n(p_n^* - p_n) = \mathcal{L}_n(p_n^* - f)$, because $p_n^* - p_n \in \Pi_n$, so $\mathcal{L}_n(p_n^* - p_n) = p_n^* - p_n$. Also $\mathcal{L}_n f = p_n$ by definition. Hence $\|p_n^* - p_n\|_\infty \le \|\mathcal{L}_n(p_n^* - f)\|_\infty \le \Lambda_n \|f - p_n^*\|_\infty$ by the definition of Λ_n.

[2]We work with the interval $[-1,1]$ for simplicity. One can use an affine transformation to deal with a general interval $[a,b]$.

[3]Let us repeat that Λ_n depends on $\{x_i\}_{i=0}^n$, not just n.

[4]We actually don't even need f to be continuous for the proof to hold, but then $f - p_n^*$ wouldn't go to 0 as $n \to \infty$, so this is a sensible assumption.

□

Also, there exist functions $f \in C[-1,1]$ such that $\|f - p_n\|_\infty > (\Lambda_n - \epsilon - 1)\|f - p_n^*\|_\infty$ for any $\epsilon > 0$ (consider an example where $\|p_n\|_\infty > (\Lambda_n - \epsilon)\|f\|_\infty$). The crux is that iff the Lebesgue constant is modest (say $O(1)$; note that $\Lambda_n \geq 1$ trivially), the interpolant p_n is guaranteed to be near best.

The Lebesgue constant can be characterized using Lagrange basis polynomials $L_{n,k}(x)$ as (Sheet 1)

$$\Lambda_n = \max_{x \in [-1,1]} \sum_{k=0}^{n} |L_{n,k}(x)|. \tag{9}$$

$\sum_{k=0}^{n} |L_{n,k}(x)|$ is called the Lebesgue function associated with the set of interpolation points $\{x_i\}_{i=0}^{n}$.

The remainder is non-examinable.

Equispaced points Figure 2 shows the Lebesgue function $\sum_{k=0}^{n} |L_{n,k}(x)|$ for $n = 30$.

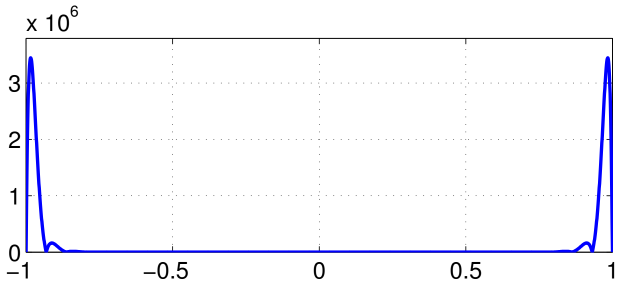


Figure 2: Lebesgue function for 30 equispaced points. $\Lambda_n \approx 2^n$.

Observe how large it is near $x = \pm 1$; in fact Λ_n grows like 2^n (more precisely $\frac{2^n}{n \log n}$). A famous example where interpolation in equispaced points results in $\|p_n - f\|_\infty$ diverging as $n \to \infty$, even though f is analytic, is $f(x) = \frac{1}{1+25x^2}$. This is Runge's phenomenon; see Figure 5.

Chebyshev points Now with Chebyshev points, the Lebesgue constant is fantastically small with $\Lambda_n = O(\log(n))$. Here is $\sum_{k=0}^{n} |L_{n,k}(x)|$ for $n = 30$.

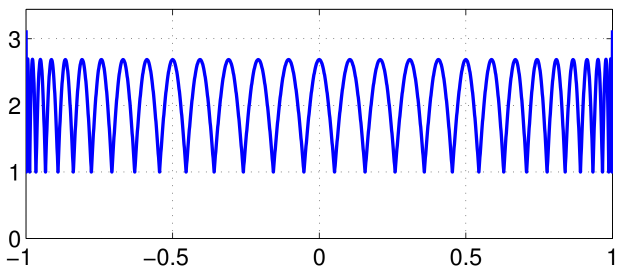


Figure 3: Lebesgue function for $n = 30$ Chebyshev points. $\Lambda_n \approx \frac{2}{\pi} \log n$.

Observe how small Λ_n is with Chebyshev interpolation and the equioscillation-like behavior of the Lebesgue function. Note that $\sum_{k=0}^{n} |L_{n,k}(x)| = 1$ at $x = x_i$.

Examples We compare four methods for polynomial approximation: (i) piecewise-linear approximation, (ii) equispaced interpolation, (iii) Chebyshev interpolation, and (iv) best polynomial approximation[5]. We take two representative functions: the exponential function $f(x) = \exp(x)$ (which is entire), and the Runge function $f(x) = \frac{1}{1+25x^2}$.

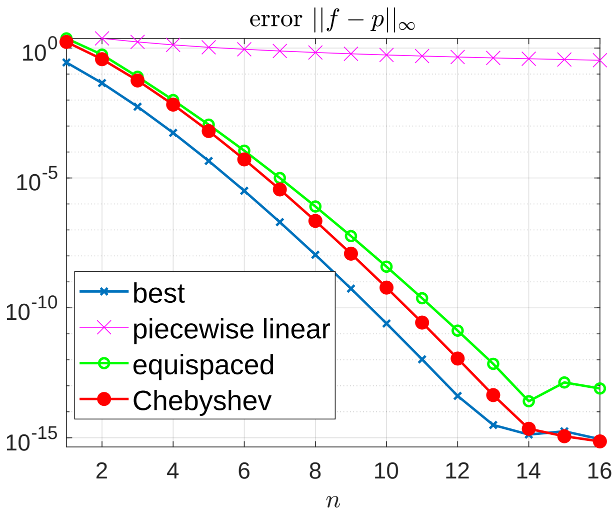


Figure 4: Exponential function $f(x) = \exp(x)$.

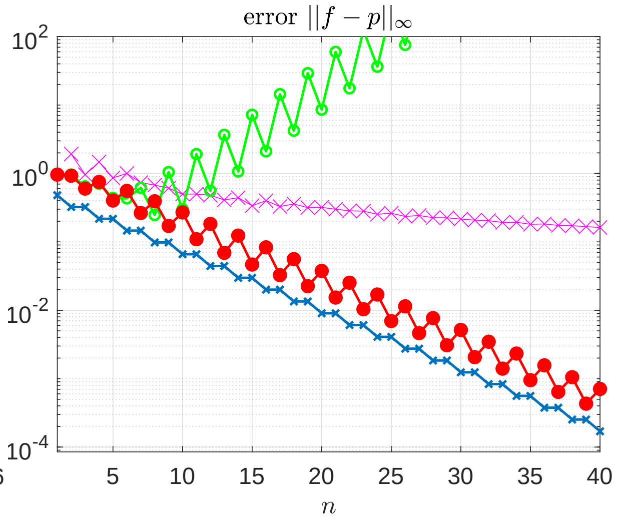


Figure 5: Runge function $f(x) = \frac{1}{1+25x^2}$.

Unlike the best approximation p_n^*, Chebyshev interpolation is linear in f (as is any interpolation once $\{x_i\}_{i=0}^{n}$ are fixed), and can be computed quickly in $O(n \log n)$ operations, using the FFT. The combination of near-optimal speed and convergence (and numerical stability) makes Chebyshev interpolation a truly remarkable algorithm.

Chebyshev interpolation (where one represents p_n in the Chebyshev polynomial basis) is actually equivalent to Fourier series for functions that are even and periodic on $[-\pi, \pi]$ under the change of variables $x = \cos(\theta)$.

[5]These are easily computed in MATLAB using the Chebfun toolbox `https://www.chebfun.org/`, namely `p=chebfun(f,n+1)` for Chebyshev interpolation, and `p=minimax(f,n)` for best approximation.

Numerical Analysis Hilary Term 2026
Lecture 2: Gaussian Elimination and LU factorisation

In lecture 1 we treated Lagrange interpolation. A traditional, more straightforward approach (worse for computation!) would be to express the interpolating polynomial as $p_n(x) = \sum_{i=0}^{n} c_i x^i$ and find the coefficients c_i by a linear system of equations:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix}.$$

(The matrix here is known as the *Vandermonde* matrix, and nonsingular iff $\{x_i\}$ are distinct.) This is a linear algebra problem, which is the subject we will discuss in the next lectures. We start with solving linear systems.

Setup: Given a square n by n matrix A and vector with n components b, find x such that

$$Ax = b.$$

Equivalently find $x = (x_1, x_2, \dots, x_n)^{\mathrm{T}}$ for which

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n. \end{aligned} \tag{1}$$

Lower-triangular matrices: the matrix A is **lower triangular** if $a_{ij} = 0$ for all $1 \le i < j \le n$. The system (1) is easy to solve if A is lower triangular.

$$\begin{array}{llll} a_{11}x_1 & = b_1 \implies & x_1 = \dfrac{b_1}{a_{11}} & \Downarrow \\ a_{21}x_1 + a_{22}x_2 & = b_2 \implies & x_2 = \dfrac{b_2 - a_{21}x_1}{a_{22}} & \Downarrow \\ \quad\vdots & & & \Downarrow \\ a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ii}x_i & = b_i \implies & x_i = \dfrac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j}{a_{ii}} & \Downarrow \\ \quad\vdots & & & \Downarrow \end{array}$$

This works if, and only if, $a_{ii} \neq 0$ for each i. The procedure is known as **forward substitution**.

Computational work estimate: one floating-point operation (flop) is one scalar multiply/division/addition/subtraction as in $y = a * x$ where a, x and y are computer representations of real scalars.[1]

[1] This is an abstraction: e.g., some hardware can do $y = a * x + b$ in one FMA flop ("Fused Multiply and Add") but then needs several FMA flops for a single division. For a trip down this sort of rabbit hole, look up the "Fast inverse square root" as used in the source code of the video game "Quake III Arena".

Hence the work in forward substitution is 1 flop to compute x_1 plus 3 flops to compute x_2 plus ... plus $2i-1$ flops to compute x_i plus ... plus $2n-1$ flops to compute x_n, or in total

$$\sum_{i=1}^{n}(2i-1) = 2\left(\sum_{i=1}^{n} i\right) - n = 2\left(\tfrac{1}{2}n(n+1)\right) - n = n^2 + \text{lower order terms}$$

flops. We sometimes write this as $n^2 + O(n)$ flops or more crudely $O(n^2)$ flops.

Upper-triangular matrices: the matrix A is **upper triangular** if $a_{ij} = 0$ for all $1 \le j < i \le n$. Once again, the system (1) is easy to solve if A is upper triangular.

$$\begin{array}{rcccl}
 & \vdots & & & \Uparrow \\
a_{ii}x_i + \cdots + a_{in-1}x_{n-1} + a_{1n}x_n & = b_i & \Longrightarrow & x_i = \dfrac{b_i - \displaystyle\sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}} & \Uparrow \\
 & \vdots & & & \Uparrow \\
a_{n-1n-1}x_{n-1} + a_{n-1n}x_n & = b_{n-1} & \Longrightarrow & x_{n-1} = \dfrac{b_{n-1} - a_{n-1n}x_n}{a_{n-1n-1}} & \Uparrow \\
a_{nn}x_n & = b_n & \Longrightarrow & x_n = \dfrac{b_n}{a_{nn}}. & \Uparrow
\end{array}$$

Again, this works if, and only if, $a_{ii} \neq 0$ for each i. The procedure is known as **backward** or **back substitution**. This also takes approximately n^2 flops.
For computation, we need a reliable, systematic technique for reducing $Ax = b$ to $Ux = c$ with the same solution x but with U (upper) triangular $\Longrightarrow$ Gauss elimination.

Example

$$\begin{bmatrix} 3 & -1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 12 \\ 11 \end{bmatrix}.$$

Multiply first equation by 1/3 and subtract from the second $\Longrightarrow$

$$\begin{bmatrix} 3 & -1 \\ 0 & \frac{7}{3} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 12 \\ 7 \end{bmatrix}.$$

Gauss(ian) Elimination (GE): this is most easily described in terms of overwriting the matrix $A = \{a_{ij}\}$ and vector b. At each stage, it is a systematic way of introducing zeros into the lower triangular part of A by subtracting multiples of previous equations (i.e., rows); such (elementary row) operations do not change the solution.

```
for columns $j = 1, 2, \ldots, n-1$
  for rows $i = j+1, j+2, \ldots, n$
      $\text{row } i \leftarrow \text{row } i - \frac{a_{ij}}{a_{jj}} * \text{row } j$
      $b_i \leftarrow b_i - \frac{a_{ij}}{a_{jj}} * b_j$
  end
end
```

Example.

$$\begin{bmatrix} 3 & -1 & 2 \\ 1 & 2 & 3 \\ 2 & -2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ 11 \\ 2 \end{bmatrix} : \text{ represent as } \left[\begin{array}{rrr|r} 3 & -1 & 2 & 12 \\ 1 & 2 & 3 & 11 \\ 2 & -2 & -1 & 2 \end{array}\right]$$

$$\Longrightarrow \quad \begin{array}{l} \text{row } 2 \leftarrow \text{row } 2 - \tfrac{1}{3}\text{row } 1 \\ \text{row } 3 \leftarrow \text{row } 3 - \tfrac{2}{3}\text{row } 1 \end{array} \left[\begin{array}{rrr|r} 3 & -1 & 2 & 12 \\ 0 & \frac{7}{3} & \frac{7}{3} & 7 \\ 0 & -\frac{4}{3} & -\frac{7}{3} & -6 \end{array}\right]$$

$$\Longrightarrow \quad \text{row } 3 \leftarrow \text{row } 3 + \tfrac{4}{7}\text{row } 2 \left[\begin{array}{rrr|r} 3 & -1 & 2 & 12 \\ 0 & \frac{7}{3} & \frac{7}{3} & 7 \\ 0 & 0 & -1 & -2 \end{array}\right]$$

Back substitution:

$$\begin{aligned} x_3 &= 2 \\ x_2 &= \frac{7 - \frac{7}{3}(2)}{\frac{7}{3}} = 1 \\ x_1 &= \frac{12 - (-1)(1) - 2(2)}{3} = 3. \end{aligned}$$

Cost of Gaussian Elimination: note, row $i \leftarrow$ row $i - \frac{a_{ij}}{a_{jj}} *$ row j is

```
  for columns $k = j+1, j+2, \ldots, n$
      $a_{ik} \leftarrow a_{ik} - \frac{a_{ij}}{a_{jj}} a_{jk}$
  end
```

This is approximately $2(n-j)$ flops as the **multiplier** a_{ij}/a_{jj} is calculated with just one flop; a_{jj} is called the **pivot**. Overall therefore, the cost of GE is approximately

$$\sum_{j=1}^{n-1} 2(n-j)^2 = 2\sum_{l=1}^{n-1} l^2 = 2\frac{n(n-1)(2n-1)}{6} = \frac{2}{3}n^3 + O(n^2)$$

flops. The calculations involving b are

$$\sum_{j=1}^{n-1} 2(n-j) = 2\sum_{l=1}^{n-1} l = 2\frac{n(n-1)}{2} = n^2 + O(n)$$

flops, just as for the triangular substitution.

LU factorization:

The basic operation of Gaussian Elimination, row $i \leftarrow$ row $i + \lambda *$row j, can be achieved by pre-multiplication by a special lower-triangular matrix

$$M(i,j,\lambda) = I + \begin{array}{cc} \left[\begin{array}{ccc} 0 & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & 0 \end{array}\right] & \begin{array}{l} \\ \leftarrow i \\ \\ \end{array} \\ \begin{array}{ccc} & \uparrow & \\ & j & \end{array} & \end{array}$$

where I is the identity matrix.

Example: $n = 4$,

$$M(3,2,\lambda) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \lambda & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } M(3,2,\lambda)\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} a \\ b \\ \lambda b + c \\ d \end{bmatrix},$$

i.e., $M(3,2,\lambda)A$ performs: row 3 of $A \leftarrow$ row 3 of $A + \lambda *$ row 2 of A and similarly $M(i,j,\lambda)A$ performs: row i of $A \leftarrow$ row i of $A + \lambda *$ row j of A.

So GE for e.g., $n = 3$ is

$$\begin{array}{ccccccc} M(3,2,-l_{32}) & \cdot & M(3,1,-l_{31}) & \cdot & M(2,1,-l_{21}) & \cdot & A = U = (\urcorner). \\ l_{32} = \dfrac{a_{32}}{a_{22}} & & l_{31} = \dfrac{a_{31}}{a_{11}} & & l_{21} = \dfrac{a_{21}}{a_{11}} & & \text{(upper triangular)} \end{array}$$

The l_{ij} are called the **multipliers**.

Be careful: each multiplier l_{ij} uses the data a_{ij} and a_{ii} that *results from the transformations already applied*, not data from the original matrix. So l_{32} uses a_{32} and a_{22} that result from the previous transformations $M(2,1,-l_{21})$ and $M(3,1,-l_{31})$.

Lemma. If $i \neq j$, $(M(i,j,\lambda))^{-1} = M(i,j,-\lambda)$.

Proof. Exercise.

Outcome: for $n = 3$, $A = M(2,1,l_{21}) \cdot M(3,1,l_{31}) \cdot M(3,2,l_{32}) \cdot U$, where

$$M(2,1,l_{21}) \cdot M(3,1,l_{31}) \cdot M(3,2,l_{32}) = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} = L = (\llcorner). \quad \text{(lower triangular)}$$

This is true for general n:

Fact: For any dimension n, GE can be expressed as $A = LU$, where $U = (\urcorner)$ is upper triangular resulting from GE, and $L = (\llcorner)$ is unit lower triangular (lower triangular with ones on the diagonal) with l_{ij} = multiplier used to create the zero in the (i,j)th position.

Most implementations of GE therefore, rather than doing GE as above, perform

$$\begin{array}{rll} \text{factorize} & A = LU & (\approx \frac{1}{3}n^3 \text{ adds} + \approx \frac{1}{3}n^3 \text{ mults}) \\ \text{and then solve} & Ax = b & \\ \text{by solving} & Ly = b & \text{(forward substitution)} \\ \text{and then} & Ux = y & \text{(back substitution)} \end{array}$$

Note: this is much more efficient if we have many different right-hand sides b but the same A.

Pivoting: GE or LU can fail if the pivot $a_{ii} = 0$. For example, if

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

GE fails at the first step. However, we are free to reorder the equations (i.e., the rows) into any order we like. For example, the equations

$$\begin{array}{c} 0 \cdot x_1 + 1 \cdot x_2 = 1 \\ 1 \cdot x_1 + 0 \cdot x_2 = 2 \end{array} \quad \text{and} \quad \begin{array}{c} 1 \cdot x_1 + 0 \cdot x_2 = 2 \\ 0 \cdot x_1 + 1 \cdot x_2 = 1 \end{array}$$

are the same, but their matrices

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

have had their rows reordered: GE fails for the first but succeeds for the second $\Longrightarrow$ better to interchange the rows and then apply GE.

Partial pivoting: when creating the zeros in the jth column, find

$$|a_{kj}| = \max(|a_{jj}|, |a_{j+1j}|, \ldots, |a_{nj}|),$$

then swap (interchange) rows j and k.

For example,

$$\begin{bmatrix} a_{11} & \cdot & a_{1j-1} & a_{1j} & \cdot & \cdot & \cdot & a_{1n} \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & a_{j-1j-1} & a_{j-1j} & \cdot & \cdot & \cdot & a_{j-1n} \\ 0 & \cdot & 0 & a_{jj} & \cdot & \cdot & \cdot & a_{jn} \\ 0 & \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & 0 & a_{kj} & \cdot & \cdot & \cdot & a_{kn} \\ 0 & \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & 0 & a_{nj} & \cdot & \cdot & \cdot & a_{nn} \end{bmatrix} \rightarrow \begin{bmatrix} a_{11} & \cdot & a_{1j-1} & a_{1j} & \cdot & \cdot & \cdot & a_{1n} \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & a_{j-1j-1} & a_{j-1j} & \cdot & \cdot & \cdot & a_{j-1n} \\ 0 & \cdot & 0 & a_{kj} & \cdot & \cdot & \cdot & a_{kn} \\ 0 & \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & 0 & a_{jj} & \cdot & \cdot & \cdot & a_{jn} \\ 0 & \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & 0 & a_{nj} & \cdot & \cdot & \cdot & a_{nn} \end{bmatrix}$$

Theorem: GE with partial pivoting cannot fail if A is nonsingular.

Proof. If A is the first matrix above at the jth stage,

$$\det[A] = a_{11} \cdots a_{j-1j-1} \cdot \det \begin{bmatrix} a_{jj} & \cdot & \cdot & \cdot & a_{jn} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{kj} & \cdot & \cdot & \cdot & a_{kn} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{nj} & \cdot & \cdot & \cdot & a_{nn} \end{bmatrix}.$$

Hence $\det[A] = 0$ if $a_{jj} = \cdots = a_{kj} = \cdots = a_{nj} = 0$. Thus if the pivot $a_{k,j}$ is zero, A is singular. So if A is nonsingular, all of the pivots are nonzero. (Note: actually a_{nn} can be zero and an LU factorization still exist.) $\square$

The effect of pivoting is just a permutation (reordering) of the rows, and hence can be represented by a permutation matrix P.

Permutation matrix: P has the same rows as the identity matrix, but in the pivoted order. So

$$PA = LU$$

represents the factorization—equivalent to GE with partial pivoting. E.g.,

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} A$$

has the 2nd row of A first, the 3rd row of A second and the 1st row of A last.

Matlab example:

```
>> A = rand(5,5)
A =
      0.69483      0.38156      0.44559       0.6797      0.95974
       0.3171      0.76552      0.64631       0.6551      0.34039
      0.95022       0.7952      0.70936      0.16261      0.58527
     0.034446      0.18687      0.75469        0.119      0.22381
      0.43874      0.48976      0.27603      0.49836      0.75127
>> exactx = ones(5,1);  b = A*exactx;
>> [LL, UU] = lu(A)  % note "psychologically lower triangular" LL
LL =
      0.73123     -0.39971      0.15111            1            0
      0.33371            1            0            0            0
            1            0            0            0            0
     0.036251        0.316            1            0            0
      0.46173      0.24512     -0.25337      0.31574            1
UU =
      0.95022       0.7952      0.70936      0.16261      0.58527
            0      0.50015      0.40959      0.60083      0.14508
            0            0      0.59954    -0.076759      0.15675
            0            0            0      0.81255      0.56608
```

```
                0             0             0             0       0.30645

>> [L, U, P] = lu(A)
L =
                1             0             0             0             0
          0.33371             1             0             0             0
         0.036251         0.316             1             0             0
          0.73123      -0.39971       0.15111             1             0
          0.46173       0.24512      -0.25337       0.31574             1
U =
          0.95022        0.7952       0.70936       0.16261       0.58527
                0       0.50015       0.40959       0.60083       0.14508
                0             0       0.59954     -0.076759       0.15675
                0             0             0       0.81255       0.56608
                0             0             0             0       0.30645
P =
     0     0     1     0     0
     0     1     0     0     0
     0     0     0     1     0
     1     0     0     0     0
     0     0     0     0     1

>> max(max(P'*L - LL))  % we see LL is P'*L
ans =
     0

>> y = L \ (P*b);  % now to solve Ax = b...
>> x = U \ y
x =
                1
                1
                1
                1
                1

>> norm(x - exactx, 2)  % within roundoff error of exact soln
ans =
   3.5786e-15
```

Numerical Analysis Hilary Term 2026
Lecture 3: QR Factorization

Definition: a square real matrix Q is **orthogonal** if $Q^{\mathrm{T}} = Q^{-1}$. This is true if, and only if, $Q^{\mathrm{T}}Q = I = QQ^{\mathrm{T}}$.

Example: the permutation matrices P in LU factorization with partial pivoting are orthogonal.

Proposition. The product of orthogonal matrices is an orthogonal matrix.

Proof. If S and T are orthogonal, $(ST)^{\mathrm{T}} = T^{\mathrm{T}}S^{\mathrm{T}}$ so

$$(ST)^{\mathrm{T}}(ST) = T^{\mathrm{T}}S^{\mathrm{T}}ST = T^{\mathrm{T}}(S^{\mathrm{T}}S)T = T^{\mathrm{T}}T = I.$$

□

Definition: The **scalar (dot)(inner) product** of two vectors

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{and} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

in $\mathbb{R}^n$ is

$$x^{\mathrm{T}}y = y^{\mathrm{T}}x = \sum_{i=1}^{n} x_i y_i \in \mathbb{R}$$

Definition: Two vectors $x,\ y \in \mathbb{R}^n$ are **orthogonal** if $x^{\mathrm{T}}y = 0$. A set of vectors $\{u_1, u_2, \ldots, u_r\}$ is an **orthogonal set** if $u_i^{\mathrm{T}}u_j = 0$ for all $i, j \in \{1, 2, \ldots, r\}$ such that $i \neq j$.

Lemma. The columns of an orthogonal matrix Q form an orthogonal set, which is moreover an orthonormal basis for $\mathbb{R}^n$.

Proof. Suppose that $Q = [q_1 \ \ q_2 \ \ \cdots \ \ q_n]$, i.e., q_j is the jth column of Q. Then

$$Q^{\mathrm{T}}Q = I = \begin{bmatrix} q_1^{\mathrm{T}} \\ q_2^{\mathrm{T}} \\ \vdots \\ q_n^{\mathrm{T}} \end{bmatrix} [q_1 \ \ q_2 \ \ \cdots \ \ q_n] = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Comparing the (i, j)th entries yields

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

Note that the columns of an orthogonal matrix are of length 1 as $q_i^{\mathrm{T}}q_i = 1$, so they form an orthonormal set. □

To see that it forms a basis, let $x \in \mathbb{R}^n$ be any vector. One has $x = QQ^T x = Qc$ where

$c = Q^T x$, so $x = \sum_{i=1}^n c_i q_i$.

Lemma. If $u \in \mathbb{R}^n$, P is n-by-n orthogonal and $v = Pu$, then $u^{\mathrm{T}}u = v^{\mathrm{T}}v$.

Proof. $v^{\mathrm{T}}v = (Pu)^{\mathrm{T}}(Pu) = (u^{\mathrm{T}}P^{\mathrm{T}})(Pu) = u^{\mathrm{T}}(P^{\mathrm{T}}P)u = u^{\mathrm{T}}u$. □

Definition: The **outer product** of two vectors x and $y \in \mathbb{R}^n$ is

$$xy^{\mathrm{T}} = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_ny_1 & x_ny_2 & \cdots & x_ny_n \end{bmatrix},$$

an n-by-n matrix (notation: $xy^{\mathrm{T}} \in \mathbb{R}^{n\times n}$). More usefully, if $z \in \mathbb{R}^n$, then

$$(xy^{\mathrm{T}})z = xy^{\mathrm{T}}z = x(y^{\mathrm{T}}z) = \left(\sum_{i=1}^n y_i z_i\right) x.$$

Definition: For $w \in \mathbb{R}^n$, $w \neq 0$, the **Householder** reflector $H(w) \in \mathbb{R}^{n\times n}$ is the matrix

$$H(w) = I - \frac{2}{w^{\mathrm{T}}w} ww^{\mathrm{T}}.$$

Proposition. $H(w)$ is a symmetric orthogonal matrix.

Proof.

Symmetry is straightforward to verify. For orthogonality,

$$\begin{aligned} H(w)H(w)^{\mathrm{T}} = & \left(I - \frac{2}{w^{\mathrm{T}}w}ww^{\mathrm{T}}\right)\left(I - \frac{2}{w^{\mathrm{T}}w}ww^{\mathrm{T}}\right) \\ = & \; I - \frac{4}{w^{\mathrm{T}}w}ww^{\mathrm{T}} + \frac{4}{(w^{\mathrm{T}}w)^2}w(w^{\mathrm{T}}w)w^{\mathrm{T}} \\ = & \; I. \end{aligned}$$

□

Lemma. Given $u \in \mathbb{R}^n$, there exists a $w \in \mathbb{R}^n$ such that

$$H(w)u = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix} \equiv v,$$

say, where $\alpha = \pm\sqrt{u^{\mathrm{T}}u}$.

Remark: Since $H(w)$ is an orthogonal matrix for any $w \in \mathbb{R}$, $w \neq 0$, it is necessary for the validity of the equality $H(w)u = v$ that $v^{\mathrm{T}}v = u^{\mathrm{T}}u$, i.e., $\alpha^2 = u^{\mathrm{T}}u$; hence our choice of $\alpha = \pm\sqrt{u^{\mathrm{T}}u}$.

Proof. Take $w = \gamma(u - v)$, where $\gamma \neq 0$. Recall that $u^{\mathrm{T}}u = v^{\mathrm{T}}v$. Thus,

$$\begin{aligned} w^{\mathrm{T}}w = & \; \gamma^2(u-v)^{\mathrm{T}}(u-v) = \gamma^2(u^{\mathrm{T}}u - 2u^{\mathrm{T}}v + v^{\mathrm{T}}v) \\ = & \; \gamma^2(u^{\mathrm{T}}u - 2u^{\mathrm{T}}v + u^{\mathrm{T}}u) = 2\gamma u^{\mathrm{T}}(\gamma(u-v)) \\ = & \; 2\gamma w^{\mathrm{T}}u. \end{aligned}$$

So

$$H(w)u = \left(I - \frac{2}{w^{\mathrm{T}}w} ww^{\mathrm{T}}\right) u = u - \frac{2w^{\mathrm{T}}u}{w^{\mathrm{T}}w} w = u - \frac{1}{\gamma} w = u - (u - v) = v.$$

$\square$

Now if u is the first column of the n-by-n matrix A,

$$H(w)A = \left[\begin{array}{c|ccc} \alpha & \times & \cdots & \times \\ \hline 0 & & & \\ \vdots & & B & \\ 0 & & & \end{array}\right], \text{ where } \times = \text{general entry.}$$

Similarly for B, we can find $\hat{w} \in \mathbb{R}^{n-1}$ such that

$$H(\hat{w})B = \left[\begin{array}{c|ccc} \beta & \times & \cdots & \times \\ \hline 0 & & & \\ \vdots & & C & \\ 0 & & & \end{array}\right]$$

and then

$$\left[\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & H(\hat{w}) & \\ 0 & & & \end{array}\right] H(w)A = \left[\begin{array}{cc|ccc} \alpha & \times & \times & \cdots & \times \\ 0 & \beta & \times & \cdots & \times \\ \hline 0 & 0 & & & \\ 0 & 0 & & & \\ \vdots & \vdots & & C & \\ 0 & 0 & & & \end{array}\right].$$

Note

$$\begin{bmatrix} 1 & 0 \\ 0 & H(\hat{w}) \end{bmatrix} = H(w_2), \quad \text{where } \; w_2 = \begin{bmatrix} 0 \\ \hat{w} \end{bmatrix}.$$

Thus if we continue in this manner for the $n-1$ steps, we obtain

$$\underbrace{H(w_{n-1}) \cdots H(w_3) H(w_2) H(w)}_{Q^{\mathrm{T}}} A = \begin{bmatrix} \alpha & \times & \cdots & \times \\ 0 & \beta & \cdots & \times \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \gamma \end{bmatrix} = (\urcorner).$$

The matrix Q^{T} is orthogonal as it is the product of orthogonal (Householder) matrices, so we have constructively proved the existence of the QR factorisation $A = QR$. Such factorisation *always* exists.

Theorem. Given any $m \times n$ matrix A, there exists an orthogonal matrix Q and an upper triangular matrix R such that

$$A = QR$$

.

Notes: 1. This could also be established using the Gram–Schmidt Process.

2. If u is already of the form $(\alpha,\ 0,\ \cdots,\ 0)^{\mathrm{T}}$, we just take $H = I$.

3. Householder reflectors can be applied to a vector in $O(n^2)$ flops; $4n^2$ to be precise. To see this, note that $Hv = (I - 2ww^T)v = v - 2w(w^Tv)$. Using this, the QR factorisation can be computed in $O(n^3)$ flops.
4. It is not necessary that A is square: if $A \in \mathbb{R}^{m\times n}$, then we need the product of (a) $m-1$ Householder matrices if $m \leq n \implies$

$$\left(\,\Box\,\right) = A = QR = \left(\,\Box\,\right)\left(\,\urcorner\,\right)$$

or (b) n Householder matrices if $m > n \implies$

$$\left(\,\Box\,\right) = A = QR = \left(\,\Box\,\right)\left(\,\urcorner\,\right). \tag{1}$$

This $m > n$ case is particular important, and we note that one can also write

$$\left(\,\Box\,\right) = A = QR = \left(\,\Box\,\right)\left(\,\urcorner\,\right).$$

That is, writing (1) as $A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\begin{bmatrix} R_1 \\ 0 \end{bmatrix}$, we have $A = Q_1R_1$. This is called the *thin* QR factorization, wherein $Q \in \mathbb{R}^{m\times n}$ has orthonormal columns and has the same size as A; by contrast, in (1) Q is square orthogonal, and (1) is called the *full* QR.

Numerical Analysis Hilary Term 2026
Lecture 4: Least-squares problem

So far the linear systems we treated had the same number of equations as unknowns (variables), so the problem was $Ax = b$ for a square matrix A. Very often in practice, we have more equations that we would like to satisfy than variables to fit them. It is then usually impossible to obtain $Ax = b$; a common approach is then to try minimise the difference between Ax and b. If we choose to minimise the Euclidean length of the vector, this leads to a *least-squares problem*:

$$\min_x \|Ax - b\|, \qquad A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, m \geq n. \tag{1}$$

Here $\|y\| := \sqrt{y_1^2 + y_2^2 + \cdots + y_m^2} = \sqrt{y^Ty}$.

Least-squares problems (also known as *overdetermined* systems) are ubiquitous in applied mathematics and data science; linear regression is a basic example.

Solution of least-squares by the QR factorisation:

Let $A = [Q\ Q_\perp]\left[\begin{smallmatrix}R\\0\end{smallmatrix}\right] = Q_F\left[\begin{smallmatrix}R\\0\end{smallmatrix}\right]$ be a 'full' QR factorization, computed e.g. via the Householder QR factorization. We assume R is nonsingular (i.e., A has full column rank); this is a generic condition.

Theorem. The least-squares problem (1) has solution $x = R^{-1}Q^Tb$.

Proof. Noting that $\|Q_F^Ty\| = \sqrt{y^TQ_FQ_F^Ty} = \sqrt{y^Ty} = \|y\|$ for any vector y, we have

$$\|Ax - b\| = \|Q_F^T(Ax - b)\| = \left\| \begin{bmatrix} R \\ 0 \end{bmatrix} x - \begin{bmatrix} Q^Tb \\ Q_\perp^Tb \end{bmatrix} \right\|.$$

The bottom part is $-Q_\perp^Tb$, no matter what x is. The top part can be made 0 by taking $x = R^{-1}Q^Tb$—this is therefore the solution. □

The argument also suggests an algorithm: compute the "thin" QR factorization $A = QR$, then solve $Rx = Q^Tb$ for x, which is obtained by backward substitution as R is triangular. Note that while we used the full QR for the derivation, we only need the thin QR for the solution of (1).

Later we will see that a general linear least-squares problem has solution characterised by the orthogonality condition, which in our context reduces to $A^T(Ax - b) = 0$, so $x = (A^TA)^{-1}A^Tb$; one can verify this is the same as $R^{-1}Q^Tb$ obtained above.

Illustration of least-squares for polynomial approximation: We treated Lagrange interpolation in Lecture 1. While Lagrange polynomials give a clean expression for the interpolating polynomial, the interpolating polynomial is not always a good approximation to the original underlying function f. For example, suppose $f(x) = 1/(25x^2 + 1)$ (this is a famous function called the *Runge function*), and take a degree-n polynomial interpolant p_n at $n + 1$ equispaced points in $[-1, 1]$. The interpolating polynomials for varying n are shown in Figure 1.

As we increase n, we hope that $p_n \to f$—but this is far from the truth! p_n is diverging as n grows near the endpoints ± 1, and the divergence is actually exponential (very bad); note the vertical scales of the final plots! This is called Runge's phenomenon.

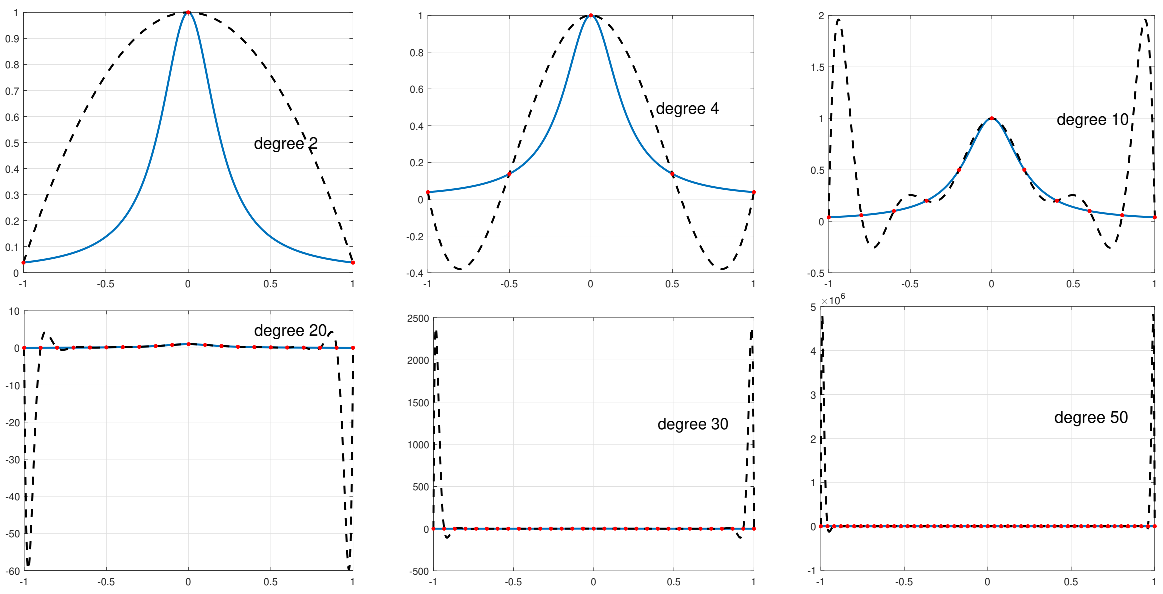


Figure 1: Polynomial interpolants (dashed black curves) of $f(x) = 1/(25x^2 + 1)$ (blue). The red dots are the interpolation points.

How can we avoid the divergence, and get $p_n \to f$ as we hope? One approach is to *oversample*: take (many) more points than the degree n. With $m(> n + 1)$ data points $x_1, \ldots, x_m$, this will lead to the least-squares problem $\min_c \|Ac - b\|$, wherein $c = [c_0, c_1, \ldots, c_n]^T$ represents the coefficients of the polynomial $p_n(x) = \sum_{j=0}^n c_j x^j$, $A \in \mathbb{R}^{m\times(n+1)}$ with $A_{ij} = (x_i)^{j-1}$ and $b = [f(x_1), \ldots, f(x_m)]^T$.

We illustrate this in Figure 2 with the example above, but now fixing $n = 20$ and varying the number of data points m. This time, for large enough m the polynomial p_n is close to f across the whole interval $[-1, 1]$.

Extensions and related facts (Non-examinable):

- Instead of $p_n(x) = \sum_{j=0}^n c_j x^j$, it is actually much better to use a different polynomial basis involving *orthogonal polynomials* $\{\phi_i\}_{i=0}^n$ such as the Chebyshev polynomials, a topic discussed later. Then we would express $p_n(x) = \sum_{j=0}^n c_j \phi_j(x)$ and $A_{ij} = (\phi_{j-1}(x_i))$, and the least-squares problem will be beter-conditioned (easier to solve accurately). However, Runge's phenomenon still persists unless $m \gg n$.
- Note that we do not have $p_n \to f$ in Figure 2 as $m \to \infty$ because the polynomial degree $n = 20$ is fixed; to get $p_n \to f$ one needs to increase n together with m. It can be shown that if one takes $m = n^2$, we do have $p_n \to f$ for any analytic function f (the convergence is exponential in n).
- Another—more elegant—solution to overcome the instability in Figure 1 is to change the interpolation points, as we saw at the end of lecture 1! But sometimes it is inevitable that the data acquired is equispaced or come from other distribution, e.g. random. In such cases, least-squares is often the best or only way forward. This is

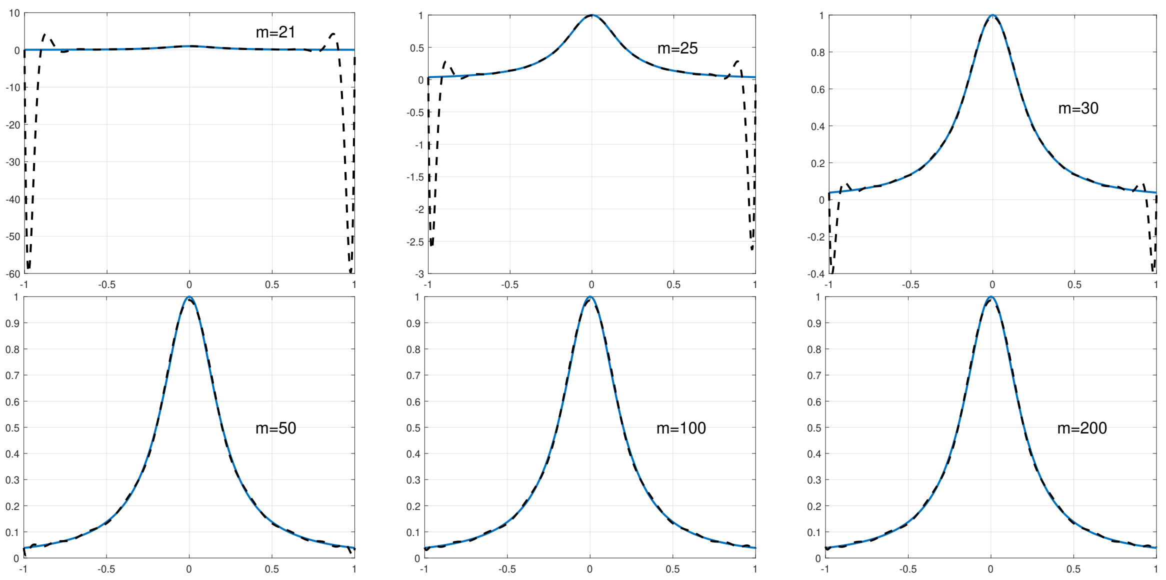


Figure 2: Least-squares polynomial fits of degree 20 (black dashed curves) of $f(x) = 1/(25x^2+1)$ (blue).

a fundamental fact in approximation theory; for a rigorous and extended discussions (including an explanation of Runge's phenomenon), check out the Part C course Approximation of Functions.

Underdetermined case (Non-examinable): One might wonder, what if we have *fewer* equations than variables? That is, if we have $Ax = b$ with $A \in \mathbb{R}^{m\times n}$, $m < n$. This *underdetermined* system of equations has infinitely many solutions (if there is one). The natural question becomes, which one should we look for? One possibility is to find the minimum-norm solution minimize $\|x\|$ subject to $Ax = b$; the solution can be computed again via the QR factorization (of A^T). This problem has connections to the hot topic of *deep learning*. Another fascinating approach that has had enormous impact is to minimise the 1-norm $\|x\|_1$ subject to $Ax = b$, where $\|x\|_1 = \sum_{i=1}^n |x_i|$. It turns out that the solution x then tends to be sparse, i.e., most of its entries are 0. This is the basis of the exciting field of *compressed sensing*.

Numerical Analysis Hilary Term 2026
Lecture 5: Singular Value Decomposition

We now introduce the Singular Value Decomposition (SVD), an extremely important matrix decomposition applicable to any matrix, including nonsymmetric and rectangular ones.

Theorem. (SVD) Every matrix $A \in \mathbb{R}^{m\times n}$ with $m \geq n$ can be written as

$$A = U\Sigma V^T, \tag{1}$$

where $U \in \mathbb{R}^{m\times n}$ and $V \in \mathbb{R}^{n\times n}$ are matrices with orthonormal columns, i.e., $U^TU = I_n$ and $V^TV = I_n = VV^T$ (V is square orthogonal; note that $UU^T \neq I_m$), and

$$\Sigma = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix} \quad (= \operatorname{diag}(\sigma_1, \ldots, \sigma_n))$$

is a diagonal matrix with nonnegative diagonal entries. In short, the SVD is a decomposition of A into a product of 'orthonormal-diagonal-orthogonal' matrices; when A is square $m = n$, 'orthogonal-diagonal-orthogonal'.

Note: One can think of orthogonal matrices as a length-preserving rotation, so the SVD indicates that applying a matrix performs a rotation, followed by shrinkage or amplification of the elements, followed by another (different) rotation.

σ_i are called the *singular values* and usually arranged in decreasing order $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. The columns of U, V are called the (left and right) *singular vectors* of A. The *rank* of a matrix A is the number of its positive singular values (this is equivalent e.g. to the number of linearly independent columns or rows).

Proof. Let's prove the existence of the SVD (1) by the following steps.

1. The matrix $A^TA \in \mathbb{R}^{n\times n}$ is symmetric. This is straightforward to verify, either by direct calculations or from the general identity $(XY)^T = Y^TX^T$.

2. The eigenvalues of A^TA are all real and nonnegative (such matrices are called symmetric positive definite). To see this, suppose $A^TAx = \lambda x$, $x \neq 0$. Then $x^TA^TAx = \lambda x^Tx$, so $\lambda = \frac{x^TA^TAx}{x^Tx} = \frac{y^Ty}{x^Tx} \geq 0$, where $y = Ax$.

3. Let $A^TA = VD^2V^T$ be the symmetric eigenvalue decomposition, with $V \in \mathbb{R}^{n\times n}$ orthogonal and D diagonal. Then let $B = AV$. Now $B^TB = D^2$ is a diagonal matrix, implying that the columns of B are pairwise orthogonal.

4. Let's write $B^TB = D^2 = \operatorname{diag}(\lambda_1, \ldots, \lambda_\ell, 0, \ldots, 0)$, where $\lambda_\ell > 0$.

   (a) It is possible that $\ell = n$, and this is an important case (happens iff $\operatorname{rank}(A) = n$, more generally ℓ is equal to $\operatorname{rank}(A)$) where there is no 0 diagonal entry in D^2. We then have $D^{-1} = \operatorname{diag}(1/\sqrt{\lambda_1}, \ldots, 1/\sqrt{\lambda_\ell})$. Take $U := BD^{-1} = AVD^{-1}$, which has orthonormal columns $U^TU = I_n$. We are then done, as taking $\Sigma = D$, $A = U\Sigma V^T$.

(b) When $\ell < n$ (the rank-deficient case), B has columns that are 0. Let $D_\ell = \mathrm{diag}(\lambda_1, \dots, \lambda_\ell)$. We still have $B \begin{bmatrix} D_\ell^{-1} & \\ & I_{n-\ell} \end{bmatrix} = [U_1,\ 0]$, and so

$$A = [U_1,\ 0] \begin{bmatrix} D_r & \\ & I_{n-\ell} \end{bmatrix} V^T = [U_1, U_2] \begin{bmatrix} D_\ell & \\ & 0 \end{bmatrix} V^T$$

for any U_2; we take it to be orthonormal $U_2^T U_2 = I_{n-\ell}$ and $U_2^T U_1 = 0$ (U_2 is any orthonormal matrix in the orthogonal complement of U_1; its existence can be verified e.g. using Householder reflectors). Taking $U = [U_1, U_2]$ completes the proof, again with $\Sigma = D$.

□

Some comments:

- Analogous to the full QR factorisation, there is a 'full SVD' $A = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, where $\tilde{U} = [U\ U_\perp] \in \mathbb{R}^{m\times m}$ is orthogonal and $\tilde{\Sigma} \in \mathbb{R}^{m\times n} = \begin{bmatrix} \Sigma \\ 0_{(m-n)\times n} \end{bmatrix}$ and $\tilde{V} = V$. This can be obtained by starting from (1) and finding an orthogonal complement $U_\perp$ of U.
- Fat matrices: the assumption $m \geq n$ is just for convenience; if $m < n$, one still has $A = U\Sigma V^T$ where $\Sigma \in \mathbb{R}^{m\times m}$ is diagonal, $U \in \mathbb{R}^{m\times m}$ is orthogonal, and $V \in \mathbb{R}^{n\times m}$ has orthonormal columns. Below we continue with the assumption $m \geq n$.
- The SVD extends directly to matrices with nonreal entries: $A = U\Sigma V^*$, where U, V are unitary matrices and * denotes the conjugate transpose.

Matrix spectral norm Let us briefly introduce the *spectral norm*[1] for matrices $A \in \mathbb{R}^{m\times n}$: $\|A\|_2 = \sigma_1(A)$, i.e., the largest singular value. It is a nonnegative scalar that measures 'how large' the matrix is. It has the equivalent characterisation $\|A\|_2 = \max_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2}$, where the norms in the right-hand side are the standard Euclidean norm (length) for vectors $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$.

Low-rank approximation The SVD is useful for theoretical purposes, as it identifies e.g. the range (column space), null space, rank, and many more. In applications, the primary reason SVD is so important is that it gives the optimal low-rank approximation.

Let $A = U\Sigma V^T$ be the SVD and write $U = [u_1, \dots, u_n], V = [v_1, \dots, v_n]$. Let r be any integer $r \leq n$, and define the "tall-skinny matrices" $U_r = [u_1, \dots, u_r]$, $V_r = [v_1, \dots, v_r]$, and $\Sigma_r = \mathrm{diag}(\sigma_1, \dots, \sigma_r)$. Then set

$$A_r = U_r \Sigma_r V_r^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T.$$

Note that $\mathrm{rank}(A_r) = r$. Also note that $A = \sum_{i=1}^{n} \sigma_i u_i v_i^T$, which is another way of expressing the SVD. A_r is called the (rank-r) *truncated SVD* of A, as A_r is obtained by truncating the trailing components of the SVD of A.

[1]Also known as the 2-norm or the operator norm. We return to the topic of norms later in the course.

We are now ready to state the result.

Theorem. Let $r \le n$ be an integer. For any $B \in \mathbb{C}^{m\times n}$ with $\operatorname{rank}(B) \le r$,

$$\|A - A_r\|_2 = \sigma_{r+1} \le \|A - B\|_2. \tag{2}$$

In other words, the truncated SVD A_r *is the best rank-r approximant to* A in the spectral norm.

Proof. The first equality $\|A - A_r\|_2 = \sigma_{r+1}$ can be seen by noting that $A - A_r = \sum_{i=r+1}^{n} \sigma_i u_i v_i^T$ with singular values $\sigma_{r+1}, \dots, \sigma_n$, along with r 0's. For the inequality:

1. Since $\operatorname{rank}(B) \le r$, we can write $B = B_1 B_2^T$ where B_1, B_2 have r columns. Therefore, there exists an orthonormal null space $W \in \mathbb{C}^{n\times(n-r)}$ s.t. $BW = 0$.

2. Then $\|A - B\|_2 \ge \|(A - B)W\|_2 = \|AW\|_2 = \|U\Sigma(V^T W)\|_2$. Now since W is $(n-r)$-dimensional, there is an interesection between W and $[v_1, \dots, v_{r+1}]$, the $(r+1)$-dimensional subspace spanned by the leading $r+1$ left singular vectors ($[W, v_1, \dots, v_{r+1}]\left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right] = 0$ has a nonzero solution; then Wx_1 is such a vector).

3. Scale x_1 to have unit norm, and by orthogonal invariance $\|U\Sigma V^T W x_1\|_2 = \|\Sigma V^T W x_1\|_2 = \|\Sigma_{r+1} y_1\|_2$, where $\|y_1\|_2 = 1$ (b.c. Wx_1 lies in $\operatorname{span}[v_1, \dots, v_{r+1}]$) and Σ_{r+1} is the leading $r+1$ part of Σ.

4. Then $\|U\Sigma_{r+1} y_1\|_2 \ge \sigma_{r+1}$ can be verified by direct calculations.

$\square$

In fact, more generally it is known that

$$\|A - A_r\| \le \|A - B\| \tag{3}$$

for any so-called unitarily invariant norm $\|\cdot\|$ (non-examinable).

In many applications $\sigma_{r+1} \ll \sigma_1$ for some $r \ll n$, in which case $A \approx U_r \Sigma_r V_r^T$. Now, storing U_r, Σ_r, V_r requires $\approx (m + n + 1)r$ memory, as opposed to mn for the full A, so when $r \ll \min(m, n)$, this can be used for data compression; this fact is used everywhere e.g. in data science!

Illustration of low-rank approximation: A traditional example to illustrate low-rank approximation via the truncated SVD is image compression. A grayscale image can be represented by a matrix A, with each entry representing the intensity of a pixel. One can then approximate A by a truncated SVD, and use that to get a compressed image that hopefully looks similar to the original image to human eyes. Images tend to have structure that lends A to be approximately low-rank.

Below we take an image of the Oxford logo, represent it as a matrix $A \in \mathbb{R}^{589\times 589}$ and compute its SVD (just `[U,S,V] = svd(A)` in MATLAB). Using the truncated SVD we then compute a rank-r approximation for different values of r. With a rank-1 matrix the rows (and columns) are all parallel so the image is uninformative; but as r increases the image becomes clear, and with rank 50 the image is almost indistinguishable from the original, while still giving some data compression. For larger images, such savings can be significant. (This is however not how images are usually compressed in practice; e.g. the algorithm behind the jpg format is completely different).

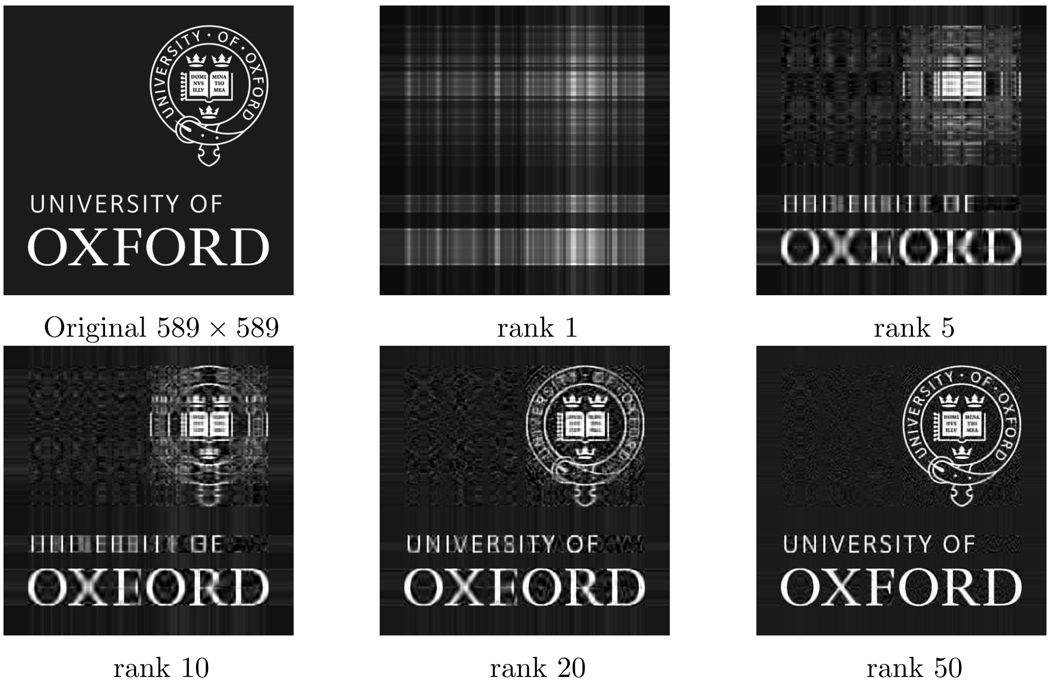


Figure 1: The Oxford logo and its low-rank approximations via the truncated SVD.

The SVD $A = U\Sigma V^T$ and symmetric eigenvalue decomposition $A = V\Lambda V^T$ have many properties and results in common (e.g. Courant-Fisher min-max theorem; nonexaminable), stemming from the fact that they are both decompositions of the form "orthogonal-diagonal-orthogonal". In fact the SVD proof given above suggests an algorithm for computing the SVD via a symmetric eigenvalue decomposition of A^TA (this is not exactly how the SVD is computed in practice, but that is outside the scope); we now turn to eigenvalue problems $Ax = \lambda x$ and describe an algorithm for solving them.

Numerical Analysis Hilary Term 2026
Lecture 6: Matrix Eigenvalues

We now turn to eigenvalue problems $Ax = \lambda x$, where $A \in \mathbb{R}^{n\times n}$ or $A \in \mathbb{C}^{n\times n}$, $\lambda \in \mathbb{C}$, and $x (\neq 0) \in \mathbb{C}^n$. Recall that there are n eigenvalues in $\mathbb{C}$ (nonreal λ possible even if A is real). There are usually, but not always, n linearly independent eigenvectors (e.g. Jordan block $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ has only one eigenvector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$).

Background: An important result from analysis (not proved or examinable!), which will be useful.

Theorem. (Ostrowski) The eigenvalues of a matrix are continuously dependent on the entries. That is, suppose that $\{\lambda_i, i = 1, \dots, n\}$ and $\{\mu_i, i = 1, \dots, n\}$ are the eigenvalues of $A \in \mathbb{R}^{n\times n}$ and $A + B \in \mathbb{R}^{n\times n}$ respectively. Given any $\varepsilon > 0$, there is a $\delta > 0$ such that $|\lambda_i - \mu_i| < \varepsilon$ whenever $\max_{i,j} |b_{ij}| < \delta$, where $B = \{b_{ij}\}_{1\le i,j\le n}$.

Noteworthy properties and facts related to eigenvalues:

- A has n eigenvalues (counting multiplicities), equal to the roots of the **characteristic polynomial** $p_A(\lambda) = \det(\lambda I - A)$.
- If $Ax_i = \lambda_i x_i$ for $i = 1, \dots, n$ and x_i are linearly independent so that $[x_1, x_2, \dots, x_n] =: X$ is nonsingular, then A has the **eigenvalue decomposition** $A = X\Lambda X^{-1}$. This usually, but not always, exist. The most general form is the Jordan canonical form (which we don't treat much in this course).
- Any square matrix has a **Schur decomposition** $A = QTQ^*$ where Q is unitary $QQ^* = Q^*Q = I_n$, and T triangular. The superscript $*$ denotes the (complex) conjugate transpose, $(Q^*)_{ij} = \overline{Q_{ji}}$.
- For a **normal matrix** s.t. $A^*A = AA^*$, the Schur decomposition shows T is diagonal (proof: examine diagonal elements of A^*A and AA^*), i.e., A can be diagonalized by a unitary similarity transformation: $A = Q\Lambda Q^*$, where $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$. Most of the structured matrices we treat are normal, including symmetric ($\lambda \in \mathbb{R}$), orthogonal ($|\lambda| = 1$), and skew-symmetric ($\lambda \in i\mathbb{R}$).

Aim: estimate the eigenvalues of a matrix.

Theorem. Gerschgorin's theorem: Suppose that $A = \{a_{ij}\}_{1\le i,j\le n} \in \mathbb{R}^{n\times n}$, and λ is an eigenvalue of A. Then, λ lies in the union of the **Gerschgorin discs**

$$D_i = \left\{ z \in \mathbb{C} \;\middle|\; |a_{ii} - z| \le \sum_{\substack{j\neq i \\ j=1}}^{n} |a_{ij}| \right\}, \quad i = 1, \dots, n.$$

Proof. If λ is an eigenvalue of $A \in \mathbb{R}^{n\times n}$, then there exists an eigenvector $x \in \mathbb{R}^n$ with $Ax = \lambda x$, $x \neq 0$, i.e.,

$$\sum_{j=1}^{n} a_{ij} x_j = \lambda x_i, \quad i = 1, \dots, n.$$

Suppose that $|x_k| \geq |x_\ell|$, $\ell = 1, \ldots, n$, i.e.,

$$\text{“}x_k \text{ is the largest entry”}. \tag{1}$$

Then the kth row of $Ax = \lambda x$ gives $\sum_{j=1}^{n} a_{kj}x_j = \lambda x_k$, or

$$(a_{kk} - \lambda)x_k = -\sum_{\substack{j\neq k\\ j=1}}^{n} a_{kj}x_j.$$

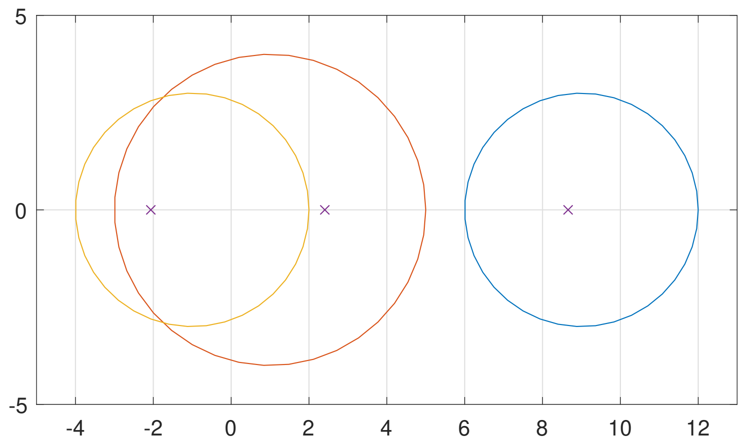
Dividing by x_k, (which, we know, is $\neq 0$) and taking absolute values,

$$|a_{kk} - \lambda| = \left|\sum_{\substack{j\neq k\\ j=1}}^{n} a_{kj}\frac{x_j}{x_k}\right| \leq \sum_{\substack{j\neq k\\ j=1}}^{n} |a_{kj}|\left|\frac{x_j}{x_k}\right| \leq \sum_{\substack{j\neq k\\ j=1}}^{n} |a_{kj}|$$

by (1). □

Example.

$$A = \begin{bmatrix} 9 & 1 & 2 \\ -3 & 1 & 1 \\ 1 & 2 & -1 \end{bmatrix}$$



With Matlab calculate `>> eig(A) = 8.6573, -2.0639, 2.4066`

Theorem. Gerschgorin's 2nd theorem: If any union of ℓ (say) discs is disjoint from the other discs, then it contains exactly ℓ eigenvalues.

Proof. Consider $B(\theta) = \theta A + (1-\theta)D$, where $D = \text{diag}(A)$, the diagonal matrix whose diagonal entries are those from A. As θ varies from 0 to 1, $B(\theta)$ has entries that vary continuously from $B(0) = D$ to $B(1) = A$. Hence the eigenvalues $\lambda(\theta)$ vary continuously by Ostrowski's theorem. The Gerschgorin discs of $B(0) = D$ are points (the diagonal entries), which are clearly the eigenvalues of D. As θ increases the Gerschgorin discs of $B(\theta)$ increase in radius about these same points as centres. Thus if $A = B(1)$ has a disjoint set of ℓ Gerschgorin discs by continuity of the eigenvalues it must contain exactly ℓ eigenvalues (as they can't jump!). □

Numerical Analysis Hilary Term 2026
Lectures 7–8: Computing eigenvalues: The Symmetric QR Algorithm

Direct vs. Iterative Methods: methods such as LU or QR factorisations and solving $Ax = b$ using them are *direct*: they compute a certain number of operations and then finish with "the answer". Another class of methods are **iterative**:

- construct a sequence;
- truncate that sequence "after convergence";
- typically concerned with fast convergence rate (rather than operation count).

Note that unlike LU, QR or linear systems $Ax = b$, algorithms for eigenvalues are necessarily iterative: By Galois theory, no finite algorithm can compute eigenvalues of $n \times n (\geq 5)$ matrices exactly in a finite number of operations. We still have an incredibly reliable algorithm to compute them, essentially to full accuracy (for symmetric matrices; for nonsymmetric matrices, in a "backward stable" manner; this is outside the scope).

Notation: for $x \in \mathbb{R}^n$, $\|x\| = \sqrt{x^{\mathrm{T}}x}$ is the (Euclidean) length of x.

Notation: in iterative methods, x_k usually means the vector x at the kth iteration (rather than kth entry of vector x). Some sources use x^k or $x^{(k)}$ instead.

Power Iteration: a simple method for calculating a single (largest) eigenvalue of a square matrix A (and its associated eigenvector). For arbitrary $y \in \mathbb{R}^n$, set $x_0 = y/\|y\|$ to calculate an initial vector, and then for $k = 0, 1, \dots$

Compute $y_k = Ax_k$
and set $x_{k+1} = y_k/\|y_k\|$.

This is the **Power Method** or **Power Iteration**, and computes unit vectors in the direction of $x_0, Ax_0, A^2x_0, A^3x_0, \dots, A^kx_0$.
Suppose that A is diagonalizable so that there is a basis of eigenvectors of A:

$$\{v_1, v_2, \dots, v_n\}$$

with $Av_i = \lambda_i v_i$ and $\|v_i\| = 1$, $i = 1, 2, \dots, n$, and assume that

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|.$$

Then we can write

$$x_0 = \sum_{i=1}^{n} \alpha_i v_i$$

for some $\alpha_i \in \mathbb{R}$, $i = 1, 2, \dots, n$, so

$$A^k x_0 = A^k \sum_{i=1}^{n} \alpha_i v_i = \sum_{i=1}^{n} \alpha_i A^k v_i.$$

However, since $Av_i = \lambda_i v_i \implies A^2 v_i = A(Av_i) = \lambda_i A v_i = \lambda_i^2 v_i$, inductively $A^k v_i = \lambda_i^k v_i$. So

$$A^k x_0 = \sum_{i=1}^{n} \alpha_i \lambda_i^k v_i = \lambda_1^k \left[\alpha_1 v_1 + \sum_{i=2}^{n} \alpha_i \left(\frac{\lambda_i}{\lambda_1} \right)^k v_i \right].$$

Since $(\lambda_i/\lambda_1)^k \to 0$ as $k \to \infty$, $A^k x_0$ tends to look like $\lambda_1^k \alpha_1 v_1$ as k gets large. The result is that by normalizing to be a unit vector

$$\frac{A^k x_0}{\|A^k x_0\|} \to \pm v_1 \ \text{ and } \ \frac{\|A^k x_0\|}{\|A^{k-1} x_0\|} \approx \left| \frac{\lambda_1^k \alpha_1}{\lambda_1^{k-1} \alpha_1} \right| = |\lambda_1|$$

as $k \to \infty$, and the sign of λ_1 is identified by looking at, e.g., $(A^k x_0)_1/(A^{k-1}x_0)_1$.

Essentially the same argument works when we normalize at each step: the Power Iteration may be seen to compute $y_k = \beta_k A^k x_0$ for some β_k. Then, from the above,

$$x_{k+1} = \frac{y_k}{\|y_k\|} = \frac{\beta_k}{|\beta_k|} \cdot \frac{A^k x_0}{\|A^k x_0\|} \to \pm v_1.$$

Similarly, $y_{k-1} = \beta_{k-1} A^{k-1} x_0$ for some β_{k-1}. Thus

$$x_k = \frac{\beta_{k-1}}{|\beta_{k-1}|} \cdot \frac{A^{k-1} x_0}{\|A^{k-1} x_0\|} \quad \text{and hence} \quad y_k = Ax_k = \frac{\beta_{k-1}}{|\beta_{k-1}|} \cdot \frac{A^k x_0}{\|A^{k-1} x_0\|}.$$

Therefore, as above,

$$\|y_k\| = \frac{\|A^k x_0\|}{\|A^{k-1} x_0\|} \approx |\lambda_1|,$$

and the sign of λ_1 may be identified by looking at, e.g., $(x_{k+1})_1/(x_k)_1$.

Hence the largest eigenvalue (and its eigenvector) can be found.
Note: it is unlikely but possible for a chosen vector x_0 that $\alpha_1 = 0$, but rounding errors in the computation generally introduce a small component in v_1, so that in practice this is not a concern!
This simplified method for eigenvalue computation is the basis for effective methods, but the current state of the art is the **QR Algorithm** which was invented by John Francis in London in 1959/60. As we shall see, the mechanics of QR algorithm is very much related to the power method.

The QR algorithm: We now describe the QR algorithm, a magical algorithm that can solve eigenvalue problems $Ax = \lambda x$ and finds all eigenvalues (and eigenvectors).

For simplicity we consider the algorithm only in the case when A is symmetric, but it is applicable also to nonsymmetric matrices with minor modifications.

Recall: a symmetric matrix A is similar to B if there is a nonsingular matrix P for which $A = P^{-1}BP$. Similar matrices have the same eigenvalues, since if $A = P^{-1}BP$,

$$0 = \det(A - \lambda I) = \det(P^{-1}(B - \lambda I)P) = \det(P^{-1})\det(P)\det(B - \lambda I),$$

so $\det(A - \lambda I) = 0$ if, and only if, $\det(B - \lambda I) = 0$.

The basic **QR algorithm** is:

```
Set $A_1 = A$.
for $k = 1, 2, \ldots$
  form the QR factorization $A_k = Q_k R_k$
  and set $A_{k+1} = R_k Q_k$
```

```
end
```

Proposition. The matrices $A_1, A_2, \dots, A_k, \dots$ are all symmetric and similar, and thus have the same eigenvalues.

Proof. Since

$$A_{k+1} = R_k Q_k = (Q_k^{\mathrm{T}} Q_k) R_k Q_k = Q_k^{\mathrm{T}} (Q_k R_k) Q_k = Q_k^{\mathrm{T}} A_k Q_k = Q_k^{-1} A_k Q_k,$$

A_{k+1} is symmetric if A_k is, and is similar to A_k. □

At least when A has eigenvalues of distinct modulus $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, this basic QR algorithm can be shown to work (A_k converges to a diagonal matrix as $k \to \infty$, the diagonal entries of which are the eigenvalues). To see this, we make the following observations.

Lemma. The iterates A_k of the QR algorithm satisfy (defining $Q^{(k)} := Q_1 \cdots Q_k$ and $R^{(k)} := R_k \cdots R_1$)

$$A_{k+1} = (Q^{(k)})^T A Q^{(k)}, \tag{1}$$

and

$$A^k = (Q_1 \cdots Q_k)(R_k \cdots R_1) = Q^{(k)} R^{(k)} \tag{2}$$

is the QR factorization of A^k.

Proof. (1) follows from a repeated application of the above proposition.

We use induction for (2): $k = 1$ trivial. Suppose $A^{k-1} = Q^{(k-1)} R^{(k-1)}$. Then $A_k = R_{k-1} Q_{k-1} = (Q^{(k-1)})^T A Q^{(k-1)}$, and

$$(Q^{(k-1)})^T A Q^{(k-1)} = Q_k R_k.$$

Then $AQ^{(k-1)} = Q^{(k-1)} Q_k R_k$, and so

$$A^k = A Q^{(k-1)} R^{(k-1)} = Q^{(k-1)} Q_k R_k R^{(k-1)} = Q^{(k)} R^{(k)},$$

giving (2). □

Let us now connect the above lemma with the power method.

Lemma. With $Q^{(k)}$ as in (2), let q_1 be its first column, and let $e_1 = [1, 0, \dots, 0]^T$. Then q_1 is equal to either $\frac{A^k e_1}{\|A^k e_1\|_2}$ or $-\frac{A^k e_1}{\|A^k e_1\|_2}$.

Proof. Right-multiplying e_1 to (2) yields $A^k e_1 = Q^{(k)} R^{(k)} e_1$. Since $R^{(k)}$ is upper triangular $R^{(k)} e_1 = [R^{(k)}_{1,1}, 0, \dots, 0]^T$, and so $Q^{(k)} R^{(k)} e_1$ is parallel to q_1, which has unit norm. □

The results show in particular that the first column q_1 of $Q^{(k)}$ is the result of power method applied k times to the initial vector $e_1 = [1, 0, \dots, 0]^T$. It then follows that q_1 converges to the dominant eigenvector. The second vector then starts converging to the 2nd dominant eigenvector, and so on. Once the columns of $Q^{(k)}$ converge to eigenvectors (note that they are orthogonal by design), (1) shows that A_k converge to a diagonal matrix of eigenvalues.

However, a really practical, fast algorithm is based on some refinements.

Reduction to tridiagonal form: the idea is to apply explicit similarity transformations $QAQ^{-1} = QAQ^{\mathrm{T}}$, with Q orthogonal, so that QAQ^{T} is tridiagonal.

Note: direct reduction to triangular form would reveal the eigenvalues, but is not possible. If

$$H(w)A = \begin{bmatrix} \times & \times & \cdots & \times \\ 0 & \times & \cdots & \times \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \times & \cdots & \times \end{bmatrix}$$

then $H(w)AH(w)^{\mathrm{T}}$ is generally full, i.e., all zeros created by pre-multiplication are destroyed by the post-multiplication. However, if

$$A = \begin{bmatrix} \gamma & u^{\mathrm{T}} \\ u & C \end{bmatrix}$$

(as $A = A^{\mathrm{T}}$) and

$$w = \begin{bmatrix} 0 \\ \hat{w} \end{bmatrix} \text{ where } H(\hat{w})u = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

it follows that

$$H(w)A = \begin{bmatrix} \gamma & & u^{\mathrm{T}} & \\ \alpha & \times & \vdots & \times \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \times & \vdots & \times \end{bmatrix},$$

i.e., the u^{T} part of the first row of A is unchanged. However, then

$$H(w)AH(w)^{-1} = H(w)AH(w)^{\mathrm{T}} = H(w)AH(w) = \left[\begin{array}{c|c} \gamma & \alpha \ 0 \ \cdots \ 0 \\ \hline \alpha & \\ 0 & \\ \vdots & B \\ 0 & \end{array}\right],$$

where $B = H(\hat{w})CH^{\mathrm{T}}(\hat{w})$, as $u^{\mathrm{T}}H(\hat{w})^{\mathrm{T}} = (\alpha, \quad 0, \quad \cdots, \quad 0)$; note that $H(w)AH(w)^{\mathrm{T}}$ is symmetric as A is.
Now we inductively apply this to the smaller matrix B, as described for the QR factorization but using post- as well as pre-multiplications. The result of $n-2$ such Householder similarity transformations is the matrix

$$H(w_{n-2}) \cdots H(w_2)H(w)AH(w)H(w_2) \cdots H(w_{n-2}),$$

which is tridiagonal.
The QR factorization of a tridiagonal matrix can now easily be achieved with $n-1$ *Givens* rotations $J(i,j)$; these are orthogonal matrices that are I except for the four elements: the $(i,i),(i,j),(j,i),(j,j)$ entries with values $c, s, -s, c$ respectively, where $c^2 + s^2 = 1$

(cosine and sine); one can choose c s.t. $\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sqrt{a^2+b^2} \\ 0 \end{bmatrix}$. (The operations below can be done with Householder matrices too, but Givens rotations are more straightforward).

Now if A is tridiagonal

$$\underbrace{J(n-1,n)\cdots J(2,3)J(1,2)}_{Q^{\mathrm{T}}}A = R, \quad \text{upper triangular.}$$

Precisely, R has a diagonal and 2 super-diagonals,

$$R = \begin{bmatrix} \times & \times & \times & 0 & 0 & 0 & \cdots & 0 \\ 0 & \times & \times & \times & 0 & 0 & \cdots & 0 \\ 0 & 0 & \times & \times & \times & 0 & \cdots & 0 \\ \vdots & \vdots & & & & & & \vdots \\ 0 & 0 & 0 & 0 & \times & \times & \times & 0 \\ 0 & 0 & 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix}$$

(exercise: check!). In the QR algorithm, the next matrix in the sequence is RQ.

Lemma. In the QR algorithm applied to a symmetric tridiagonal matrix, the symmetry and tridiagonal form are preserved, that is, A_k is symmetric tridiagonal for all k.

Proof. We have already shown that if $A_k = QR$ is symmetric, then so is $A_{k+1} = RQ$. If $A_k = QR = J(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}}\cdots J(n-1,n)^{\mathrm{T}}R$ is tridiagonal, then $A_{k+1} = RQ = RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}}\cdots J(n-1,n)^{\mathrm{T}}$. Recall that post-multiplication of a matrix by $J(i,i+1)^{\mathrm{T}}$ replaces columns i and $i+1$ by linear combinations of the pair of columns, while leaving columns $j = 1,2,\ldots,i-1,i+2,\ldots,n$ alone. Thus, since R is upper triangular, the only subdiagonal entry in $RJ(1,2)^{\mathrm{T}}$ is in position $(2,1)$. Similarly, the only subdiagonal entries in $RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}} = (RJ(1,2)^{\mathrm{T}})J(2,3)^{\mathrm{T}}$ are in positions $(2,1)$ and $(3,2)$. Inductively, the only subdiagonal entries in

$$\begin{aligned} &RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}}\cdots J(i-2,i-1)^{\mathrm{T}}J(i-1,i)^{\mathrm{T}} \\ &\quad = (RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}}\cdots J(i-2,i-1)^{\mathrm{T}})J(i-1,i)^{\mathrm{T}} \end{aligned}$$

are in positions $(j,j-1)$, $j = 2,\ldots i$. So, the lower triangular part of A_{k+1} only has nonzeros on its first subdiagonal. However, then since A_{k+1} is symmetric, it must be tridiagonal. □

Using shifts. One further and final step in making an efficient algorithm is the use of **shifts**:

```
for $k = 1,2,\ldots$
  form the QR factorization of $A_k - \mu_k I = Q_k R_k$
  and set $A_{k+1} = R_k Q_k + \mu_k I$
```

```
end
```

For any chosen sequence of values of $\mu_k \in \mathbb{R}$, $\{A_k\}_{k=1}^{\infty}$ are symmetric and tridiagonal if A_1 has these properties, and similar to A_1.

The simplest shift to use is $a_{n,n}$, which leads rapidly in almost all cases to

$$A_k = \left[\begin{array}{c|c} T_k & 0 \\ \hline 0^{\mathrm{T}} & \lambda \end{array}\right],$$

where T_k is $n-1$ by $n-1$ and tridiagonal, and λ is an eigenvalue of A_1. Inductively, once this form has been found, the QR algorithm with shift $a_{n-1,n-1}$ can be concentrated only on the $n-1$ by $n-1$ leading submatrix T_k. This process is called **deflation**.

Why does introducing shifts help? To understand this we establish a connection between QR and the power method applied to the *inverse* (known as the *inverse power method*).

Lemma. With $Q^{(k)}$ as in (2), denote by q_n its last column, and let $e_n = [0, 0, \dots, 1]^T$. Then q_n is equal to either $\frac{A^{-k}e_n}{\|A^{-k}e_1\|_2}$ or $-\frac{A^{-k}e_n}{\|A^{-k}e_1\|_2}$.

Proof. Recall (2), and take the inverse:

$$A^{-k} = (R^{(k)})^{-1}(Q^{(k)})^T,$$

and take the transpose:

$$(A^{-k})^T (= A^{-k}) = Q^{(k)}(R^{(k)})^{-T}.$$

Now multiplying e_n gives

$$A^{-k}e_n = Q^{(k)}(R^{(k)})^{-T}e_n.$$

Since $(R^{(k)})^{-T}$ is lower triangular, it follows that $Q^{(k)}(R^{(k)})^{-T}e_n$ is parallel to q_n. □

This shows that the **final** column of $Q^{(k)}$ is the result of power method applied to e_n now with the **inverse** A^{-1}. Thus the last column of $Q^{(k)}$ is converging to the eigenvector for the smallest eigenvalue λ_n, with convergence factor $|\frac{\lambda_n}{\lambda_{n-1}}|$; $Q^{(k)}$ is converging not only from the first, but (more significantly) from the last column(s).

Now we see how the introduction of shift has a drastic effect on the convergence: it changes the factor to $|\frac{\lambda_{\sigma(n)}-\mu}{\lambda_{\sigma(n-1)}-\mu}|$, where σ is a permutation such that $|\lambda_{\sigma(1)} - \mu| \geq |\lambda_{\sigma(2)} - \mu| \geq \cdots \geq |\lambda_{\sigma(n)} - \mu|$. If μ is close to an eigenvalue, this implies (potentially extremely) fast convergence; in fact by choosing the shift $\mu_k = a_{n,n}$, it can be shown that (proof omitted and non-examinable) $a_{m,m-1}$ converges *cubically*: $|a_{m,m-1,k+1}| = O(|a_{m,m-1,k}|^3)$.

The overall algorithm for calculating the eigenvalues of an n by n symmetric matrix:

```
reduce $A$ to tridiagonal form by orthogonal
(Householder) similarity transformations.

for $m = n, n-1, \dots 2$
  while $a_{m-1,m} >$ tol
    $[Q, R] =$ qr$(A - a_{m,m}I)$
    $A = RQ + a_{m,m}I$
  end while

  record eigenvalue $\lambda_m = a_{m,m}$
  $A \leftarrow$ leading $m-1$ by $m-1$ submatrix of $A$
end
record eigenvalue $\lambda_1 = a_{1,1}$
```

Computing roots of polynomials via eigenvalues Let us describe a nice application of computing eigenvalues (by the QR algorithm). Let $p(x) = \sum_{i=0}^{n} c_i x^i$ be a degree-n polynomial so that $c_n \neq 0$, and suppose we want to find its roots, i.e., values of λ for which $p(\lambda) = 0$; there are n of them in $\mathbb{C}$. For example, $p(x)$ might be an approximant to data, obtained by Lagrange interpolation from the first lecture. Why roots? For example, you might be interested in the minimum of p; this can be obtained by differentiating and setting to zero $p'(x) = 0$, which is again a polynomial rootfinding problem (for p').

How do we solve $p(x) = 0$? Recall that eigenvalues of A are the roots of its characteristic polynomial. Here we take the opposite direction—construct a matrix whose characteristic polynomial is p.

Consider the following matrix, which is called the **companion matrix** (the blank elements are all 0) for the polynomial $p(x) = \sum_{i=0}^{n} c_i x^i$:

$$C = \begin{bmatrix} -\frac{c_{n-1}}{c_n} & -\frac{c_{n-2}}{c_n} & \cdots & -\frac{c_1}{c_n} & -\frac{c_0}{c_n} \\ 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & 0 \end{bmatrix}. \tag{3}$$

Then direct calculation shows that if $p(\lambda) = 0$ then $Cx = \lambda x$ with $x = [\lambda^{n-1}, \lambda^{n-2}, \ldots, \lambda, 1]^T$. Indeed one can show that the characteristic polynomial is $\det(\lambda I - C) = p(\lambda)/c_n$ (nonexaminable), so this implication is necessary and sufficient, so the eigenvalues of C are precisely the roots of p, counting multiplicities.

Thus to compute roots of polynomials, one can compute eigenvalues of the companion matrix via the QR algorithm—this turns out to be a very powerful idea!

Numerical Analysis Hilary Term 2020
Lectures 8–9: The Symmetric QR Algorithm

We consider only the case where A is symmetric.

Recall: a symmetric matrix A is similar to B if there is a nonsingular matrix P for which $A = P^{-1}BP$. Similar matrices have the same eigenvalues, since if $A = P^{-1}BP$,

$$0 = \det(A - \lambda I) = \det(P^{-1}(B - \lambda I)P) = \det(P^{-1})\det(P)\det(B - \lambda I),$$

so $\det(A - \lambda I) = 0$ if, and only if, $\det(B - \lambda I) = 0$.

The basic **QR algorithm** is:

```
Set $A_1 = A$.
for $k = 1, 2, \ldots$
  form the QR factorization $A_k = Q_k R_k$
  and set $A_{k+1} = R_k Q_k$
end
```

Proposition. The symmetric matrices $A_1, A_2, \ldots, A_k, \ldots$ are all similar and thus have the same eigenvalues.

Proof. Since

$$A_{k+1} = R_k Q_k = (Q_k^{\mathrm{T}} Q_k) R_k Q_k = Q_k^{\mathrm{T}}(Q_k R_k) Q_k = Q_k^{\mathrm{T}} A_k Q_k = Q_k^{-1} A_k Q_k,$$

A_{k+1} is symmetric if A_k is, and is similar to A_k. □

At least when A has eigenvalues of distinct modulus $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, this basic QR algorithm can be shown to work (A_k converges to a diagonal matrix as $k \to \infty$, the diagonal entries of which are the eigenvalues). To see this, we make the following observations.

Lemma.

$$A^k = (Q_1 \cdots Q_k)(R_k \cdots R_1) = Q^{(k)} R^{(k)} \tag{1}$$

is the QR factorization of A^k, and

$$A_k = (Q^{(k)})^T A Q^{(k)}. \tag{2}$$

Proof. (2) follows from a repeated application of the above proposition.

We use induction for (1): $k = 1$ trivial. Suppose $A^{k-1} = Q^{(k-1)} R^{(k-1)}$. Then $A_k = R_{k-1} Q_{k-1} = (Q^{(k-1)})^T A Q^{(k-1)}$, and

$$(Q^{(k-1)})^T A Q^{(k-1)} = Q_k R_k.$$

Then $AQ^{(k-1)} = Q^{(k-1)} Q_k R_k$, and so

$$A^k = AQ^{(k-1)} R^{(k-1)} = Q^{(k-1)} Q_k R_k R^{(k-1)} = Q^{(k)} R^{(k)},$$

giving (1). □

The lemma shows in particular that the first column q_1 of $Q^{(k)}$ is the result of power method applied k times to the initial vector $e_1 = [1, 0, \ldots, 0]^T$ (verify). It then follows that

q_1 converges to the dominant eigenvector. The second vector then starts converging to the 2nd dominant eigenvector, and so on. Once the columns of $Q^{(k)}$ converge to eigenvectors (note that they are orthogonal by design), (2) shows that A_k converge to a diagonal matrix of eigenvalues.

However, a really practical, fast algorithm is based on some refinements.

Reduction to tridiagonal form: the idea is to apply explicit similarity transformations $QAQ^{-1} = QAQ^{\mathrm{T}}$, with Q orthogonal, so that QAQ^{T} is tridiagonal.
Note: direct reduction to triangular form would reveal the eigenvalues, but is not possible. If

$$H(w)A = \begin{bmatrix} \times & \times & \cdots & \times \\ 0 & \times & \cdots & \times \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \times & \cdots & \times \end{bmatrix}$$

then $H(w)AH(w)^{\mathrm{T}}$ is generally full, i.e., all zeros created by pre-multiplication are destroyed by the post-multiplication. However, if

$$A = \begin{bmatrix} \gamma & u^{\mathrm{T}} \\ u & C \end{bmatrix}$$

(as $A = A^{\mathrm{T}}$) and

$$w = \begin{bmatrix} 0 \\ \hat{w} \end{bmatrix} \quad \text{where} \quad H(\hat{w})u = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

it follows that

$$H(w)A = \begin{bmatrix} \gamma & & u^{\mathrm{T}} & \\ \alpha & \times & \vdots & \times \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \times & \vdots & \times \end{bmatrix},$$

i.e., the u^{T} part of the first row of A is unchanged. However, then

$$H(w)AH(w)^{-1} = H(w)AH(w)^{\mathrm{T}} = H(w)AH(w) = \left[\begin{array}{c|c} \gamma & \alpha \;\; 0 \;\; \cdots \;\; 0 \\ \hline \alpha & \\ 0 & \\ \vdots & B \\ 0 & \end{array}\right],$$

where $B = H(\hat{w})CH^{\mathrm{T}}(\hat{w})$, as $u^{\mathrm{T}}H(\hat{w})^{\mathrm{T}} = (\alpha, \;\; 0, \;\; \cdots, \;\; 0)$; note that $H(w)AH(w)^{\mathrm{T}}$ is symmetric as A is.
Now we inductively apply this to the smaller matrix B, as described for the QR factorization but using post- as well as pre-multiplications. The result of $n-2$ such Householder similarity transformations is the matrix

$$H(w_{n-2}) \cdots H(w_2)H(w)AH(w)H(w_2) \cdots H(w_{n-2}),$$

which is tridiagonal.
The QR factorization of a tridiagonal matrix can now easily be achieved with $n-1$ Givens rotations: if A is tridiagonal

$$\underbrace{J(n-1,n)\cdots J(2,3)J(1,2)}_{Q^{\mathrm{T}}}A = R, \quad \text{upper triangular.}$$

Precisely, R has a diagonal and 2 super-diagonals,

$$R = \begin{bmatrix} \times & \times & \times & 0 & 0 & 0 & \cdots & 0 \\ 0 & \times & \times & \times & 0 & 0 & \cdots & 0 \\ 0 & 0 & \times & \times & \times & 0 & \cdots & 0 \\ \vdots & \vdots & & & & & & \vdots \\ 0 & 0 & 0 & 0 & \times & \times & \times & 0 \\ 0 & 0 & 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix}$$

(exercise: check!). In the QR algorithm, the next matrix in the sequence is RQ.

Lemma. In the QR algorithm applied to a symmetric tridiagonal matrix, the symmetry and tridiagonal form are preserved when Givens rotations are used.

Proof. We have already shown that if $A_k = QR$ is symmetric, then so is $A_{k+1} = RQ$. If $A_k = QR = J(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}}\cdots J(n-1,n)^{\mathrm{T}}R$ is tridiagonal, then $A_{k+1} = RQ = RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}}\cdots J(n-1,n)^{\mathrm{T}}$. Recall that post-multiplication of a matrix by $J(i,i+1)^{\mathrm{T}}$ replaces columns i and $i+1$ by linear combinations of the pair of columns, while leaving columns $j = 1,2,\ldots,i-1,i+2,\ldots,n$ alone. Thus, since R is upper triangular, the only subdiagonal entry in $RJ(1,2)^{\mathrm{T}}$ is in position $(2,1)$. Similarly, the only subdiagonal entries in $RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}} = (RJ(1,2)^{\mathrm{T}})J(2,3)^{\mathrm{T}}$ are in positions $(2,1)$ and $(3,2)$. Inductively, the only subdiagonal entries in

$$\begin{aligned} &RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}}\cdots J(i-2,i-1)^{\mathrm{T}}J(i-1,i)^{\mathrm{T}} \\ &= (RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}}\cdots J(i-2,i-1)^{\mathrm{T}})J(i-1,i)^{\mathrm{T}} \end{aligned}$$

are in positions $(j, j-1)$, $j = 2,\ldots i$. So, the lower triangular part of A_{k+1} only has nonzeros on its first subdiagonal. However, then since A_{k+1} is symmetric, it must be tridiagonal. $\square$

Using shifts. One further and final step in making an efficient algorithm is the use of **shifts**:

```
for $k = 1, 2, \ldots$
  form the QR factorization of $A_k - \mu_k I = Q_k R_k$
  and set $A_{k+1} = R_k Q_k + \mu_k I$
end
```

For any chosen sequence of values of $\mu_k \in \mathbb{R}$, $\{A_k\}_{k=1}^{\infty}$ are symmetric and tridiagonal if A_1 has these properties, and similar to A_1.

The simplest shift to use is $a_{n,n}$, which leads rapidly in almost all cases to

$$A_k = \left[\begin{array}{c|c} T_k & 0 \\ \hline 0^{\mathrm{T}} & \lambda \end{array}\right],$$

where T_k is $n-1$ by $n-1$ and tridiagonal, and λ is an eigenvalue of A_1. Inductively, once this form has been found, the QR algorithm with shift $a_{n-1,n-1}$ can be concentrated only on the $n-1$ by $n-1$ leading submatrix T_k. This process is called **deflation**.

Why does introducing shifts help? To understand this, we recall (1), and take the inverse:

$$A^{-k} = (R^{(k)})^{-1}(Q^{(k)})^T,$$

and take the transpose:

$$(A^{-k})^T (= A^{-k}) = Q^{(k)}(R^{(k)})^{-T}.$$

Noting that $(R^{(k)})^{-T}$ is lower triangular, this shows that the **final** column of $Q^{(k)}$ is the result of power method applied to $e_n = [0,0,\ldots,0,1]^T$ now with the **inverse** A^{-1}. Thus the last column $Q^{(k)}$ is converging to the eigenvector for the smallest eigenvalue λ_n, with convergence factor $|\frac{\lambda_n}{\lambda_{n-1}}|$; $Q^{(k)}$ is converging not only from the first, but (more significantly) from the last column(s).

Finally, the introduction of shift changes the factor to $|\frac{\lambda_{\sigma(n)}-\mu}{\lambda_{\sigma(n-1)}-\mu}|$, where σ is a permutation such that $|\lambda_{\sigma(1)}-\mu| \geq |\lambda_{\sigma(2)}-\mu| \geq \cdots \geq |\lambda_{\sigma(n)}-\mu|$. If μ is close to an eigenvalue, this implies (potentially very) fast convergence; in fact it can be shown that (proof omitted and non-examinable) rather than linear convergence, $a_{m,m-1}$ converges cubically: $|a_{m,m-1,k+1}| = O(|a_{m,m-1,k}|^3)$.

The overall algorithm for calculating the eigenvalues of an n by n symmetric matrix:

```
reduce $A$ to tridiagonal form by orthogonal
(Householder) similarity transformations.
for $m = n, n-1, \ldots 2$
  while $a_{m-1,m} >$ tol
    $[Q,R] =$ qr$(A - a_{m,m} * I)$
    $A = R * Q + a_{m,m} * I$
  end while
  record eigenvalue $\lambda_m = a_{m,m}$
  $A \leftarrow$ leading $m-1$ by $m-1$ submatrix of $A$
end
record eigenvalue $\lambda_1 = a_{1,1}$
```

Computing roots of polynomials via eigenvalues Let us describe a nice application of computing eigenvalues (by the QR algorithm). Let $p(x) = \sum_{i=0}^{n} c_i x^i$ be a degree-n polynomial so that $c_n \neq 0$, and suppose we want to find its roots, i.e., values of λ for which $p(\lambda) = 0$; there are n of them in $\mathbb{C}$. For example, $p(x)$ might be an approximant to data, obtained by Lagrange interpolation from the first lecture. Why roots? For example, you might be interested in the minimum of p; this can be obtained by differentiating and setting to zero $p'(x) = 0$, which is again a polynomial rootfinding problem (for p').

How do we solve $p(x) = 0$? Recall that eigenvalues of A are the roots of its characteristic polynomial. Here we take the opposite direction—construct a matrix whose characteristic polynomial is p.

Consider the following matrix, which is called the **companion matrix** (the blank elements are all 0) for the polynomial $p(x) = \sum_{i=0}^{n} c_i x^i$:

$$C = \begin{bmatrix} -\frac{c_{n-1}}{c_n} & -\frac{c_{n-2}}{c_n} & \cdots & -\frac{c_1}{c_n} & -\frac{c_0}{c_n} \\ 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & 0 \end{bmatrix}. \tag{3}$$

Then direct calculation shows that if $p(\lambda) = 0$ then $Cx = \lambda x$ with $x = [\lambda^{n-1}, \lambda^{n-2}, \ldots, \lambda, 1]^T$. Indeed one can show that the characteristic polynomial is $\det(\lambda I - C) = p(\lambda)/c_n$ (nonexaminable), so this implication is necessary and sufficient, so the eigenvalues of C are precisely the roots of p, counting multiplicities.

Thus to compute roots of polynomials, one can compute eigenvalues of the companion matrix via the QR algorithm—this turns out to be a very powerful idea!

Numerical Analysis Hilary Term 2026
Lecture 9: Best Approximation in Inner-Product Spaces

Best approximation of functions: given a function f on $[a, b]$, find the "closest" polynomial/piecewise polynomial (see later sections)/ trigonometric polynomial (truncated Fourier series).

Norms: are used to measure the size of/distance between elements of a vector space. Given a vector space V over the field $\mathbb{R}$ of real numbers, the mapping $\|\cdot\| : V \to \mathbb{R}$ is a **norm** on V if it satisfies the following axioms:
(i) $\|f\| \geq 0$ for all $f \in V$, with $\|f\| = 0$ if, and only if, $f = 0 \in V$;
(ii) $\|\lambda f\| = |\lambda|\|f\|$ for all $\lambda \in \mathbb{R}$ and all $f \in V$; and
(iii) $\|f + g\| \leq \|f\| + \|g\|$ for all $f, g \in V$ (the **triangle inequality**).

Examples: **1.** For vectors $x \in \mathbb{R}^n$, with $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$,

$$\|x\| \equiv \|x\|_2 = (x_1^2 + x_2^2 + \cdots + x_n^2)^{\frac{1}{2}} = \sqrt{x^{\mathrm{T}} x}$$

is the ℓ^2- or vector two-norm.
2. For continuous functions on $[a, b]$,

$$\|f\| \equiv \|f\|_\infty = \max_{x \in [a,b]} |f(x)|$$

is the L^∞- or ∞-norm.
3. For integrable functions on (a, b),

$$\|f\| \equiv \|f\|_1 = \int_a^b |f(x)| \,\mathrm{d}x$$

is the L^1- or one-norm.
4. For functions in

$$V = \mathrm{L}^2_w(a, b) \equiv \{f : [a, b] \to \mathbb{R} \mid \int_a^b w(x)[f(x)]^2 \,\mathrm{d}x < \infty\}$$

for some given **weight** function $w(x) > 0$ (this certainly includes continuous functions on $[a, b]$, and piecewise continuous functions on $[a, b]$ with a finite number of jump-discontinuities),

$$\|f\| \equiv \|f\|_2 = \left(\int_a^b w(x)[f(x)]^2 \,\mathrm{d}x\right)^{\frac{1}{2}}$$

is the L^2- or two-norm—the space $\mathrm{L}^2(a, b)$ is a common abbreviation for $\mathrm{L}^2_w(a, b)$ for the case $w(x) \equiv 1$.
Note: $\|f\|_2 = 0 \implies f = 0$ almost everywhere on $[a, b]$. We say that a certain property P holds *almost everywhere* (a.e.) on $[a, b]$ if property P holds at each point of $[a, b]$ except perhaps on a subset $S \subset [a, b]$ of zero measure. We say that a set $S \subset \mathbb{R}$ has *zero measure* (or that it is of *measure zero*) if for any $\varepsilon > 0$ there exists a sequence $\{(\alpha_i, \beta_i)\}_{i=1}^\infty$ of subintervals of $\mathbb{R}$ such that

$S \subset \cup_{i=1}^{\infty}(\alpha_i, \beta_i)$ and $\sum_{i=1}^{\infty}(\beta_i - \alpha_i) < \varepsilon$. Trivially, the empty set $\emptyset(\subset \mathbb{R})$ has zero measure. Any finite subset of $\mathbb{R}$ has zero measure. Any countable subset of $\mathbb{R}$, such as the set of all natural numbers $\mathbb{N}$, the set of all integers $\mathbb{Z}$, or the set of all rational numbers $\mathbb{Q}$, is of measure zero.

Least-squares polynomial approximation: aim to find the best polynomial approximation to $f \in \mathrm{L}^2_w(a,b)$, i.e., find $p_n \in \Pi_n$ for which

$$\|f - p_n\|_2 \leq \|f - q\|_2 \qquad \forall q \in \Pi_n.$$

Seeking p_n in the form $p_n(x) = \displaystyle\sum_{k=0}^{n} \alpha_k x^k$ then results in the minimization problem

$$\min_{(\alpha_0,\ldots,\alpha_n)} \int_a^b w(x) \left[f(x) - \sum_{k=0}^{n} \alpha_k x^k \right]^2 \mathrm{d}x.$$

The unique minimizer can be found from the (linear) system

$$\frac{\partial}{\partial \alpha_j} \int_a^b w(x) \left[f(x) - \sum_{k=0}^{n} \alpha_k x^k \right]^2 \mathrm{d}x = 0 \ \text{ for each } \ j = 0, 1, \ldots, n,$$

but there is important additional structure here.

Inner-product spaces: a real **inner-product space** is a vector space V over $\mathbb{R}$ with a mapping $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$ (the **inner product**) for which

(i) $\langle v, v \rangle \geq 0$ for all $v \in V$ and $\langle v, v \rangle = 0$ if, and only if $v = 0$;

(ii) $\langle u, v \rangle = \langle v, u \rangle$ for all $u, v \in V$; and

(iii) $\langle \alpha u + \beta v, z \rangle = \alpha \langle u, z \rangle + \beta \langle v, z \rangle$ for all $u, v, z \in V$ and all $\alpha, \beta \in \mathbb{R}$.

Examples: 1. $V = \mathbb{R}^n$,

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

where $x = (x_1, \ldots, x_n)^{\mathrm{T}}$ and $y = (y_1, \ldots, y_n)^{\mathrm{T}}$.

2. $V = \mathrm{L}^2_w(a,b) = \{f : (a,b) \to \mathbb{R} \mid \displaystyle\int_a^b w(x)[f(x)]^2 \,\mathrm{d}x < \infty\}$,

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

where $f, g \in \mathrm{L}^2_w(a,b)$ and w is a weight-function, defined, positive and integrable on (a,b).

Notes: 1. Suppose that V is an inner product space, with inner product $\langle \cdot, \cdot \rangle$. Then $\langle v, v \rangle^{\frac{1}{2}}$ defines a norm on V (see the final paragraph on the last page for a proof). In Example 2 above, the norm defined by the inner product is the (weighted) L^2-norm.

2. Suppose that V is an inner product space, with inner product $\langle \cdot, \cdot \rangle$, and let $\| \cdot \|$ denote the norm defined by the inner product via $\|v\| = \langle v, v \rangle^{\frac{1}{2}}$, for $v \in V$. The angle θ between $u, v \in V$ is

$$\theta = \cos^{-1}\left(\frac{\langle u, v \rangle}{\|u\| \|v\|} \right).$$

Thus u and v are orthogonal in $V \iff \langle u, v\rangle = 0$.

E.g., x^2 and $\frac{3}{4} - x$ are orthogonal in $\mathrm{L}^2(0,1)$ with inner product $\langle f, g\rangle = \int_0^1 f(x)g(x)\,\mathrm{d}x$ as

$$\int_0^1 x^2\left(\tfrac{3}{4} - x\right)\,\mathrm{d}x = \tfrac{1}{4} - \tfrac{1}{4} = 0.$$

3. Pythagoras Theorem: Suppose that V is an inner-product space with inner product $\langle\cdot,\cdot\rangle$ and norm $\|\cdot\|$ defined by this inner product. For any $u, v \in V$ such that $\langle u, v\rangle = 0$ we have

$$\|u \pm v\|^2 = \|u\|^2 + \|v\|^2.$$

Proof.

$$\begin{aligned}
\|u \pm v\|^2 &= \langle u \pm v, u \pm v\rangle = \langle u, u \pm v\rangle \pm \langle v, u \pm v\rangle && \text{[axiom (iii)]}\\
&= \langle u, u \pm v\rangle \pm \langle u \pm v, v\rangle && \text{[axiom (ii)]}\\
&= \langle u, u\rangle \pm \langle u, v\rangle \pm \langle u, v\rangle + \langle v, v\rangle\\
&= \langle u, u\rangle + \langle v, v\rangle && \text{[orthogonality]}\\
&= \|u\|^2 + \|v\|^2.
\end{aligned}$$

4. The **Cauchy–Schwarz inequality**: Suppose that V is an inner-product space with inner product $\langle\cdot,\cdot\rangle$ and norm $\|\cdot\|$ defined by this inner product. For any $u, v \in V$,

$$|\langle u, v\rangle| \leq \|u\|\|v\|.$$

Proof. For every $\lambda \in \mathbb{R}$,

$$0 \leq \langle u - \lambda v, u - \lambda v\rangle = \|u\|^2 - 2\lambda\langle u, v\rangle + \lambda^2\|v\|^2 = \phi(\lambda),$$

which is a quadratic in λ. The minimizer of ϕ is at $\lambda_* = \langle u, v\rangle/\|v\|^2$, and thus since $\phi(\lambda_*) \geq 0$, $\|u\|^2 - \langle u, v\rangle^2/\|v\|^2 \geq 0$, which gives the required inequality. □

5. The **triangle inequality**: Suppose that V is an inner-product space with inner product $\langle\cdot,\cdot\rangle$ and norm $\|\cdot\|$ defined by this inner product. For any $u, v \in V$,

$$\|u + v\| \leq \|u\| + \|v\|.$$

Proof. Note that

$$\|u + v\|^2 = \langle u + v, u + v\rangle = \|u\|^2 + 2\langle u, v\rangle + \|v\|^2.$$

Hence, by the Cauchy–Schwarz inequality,

$$\|u + v\|^2 \leq \|u\|^2 + 2\|u\|\|v\| + \|v\|^2 = \left(\|u\| + \|v\|\right)^2.$$

Taking square-roots yields

$$\|u + v\| \leq \|u\| + \|v\|.$$

□

Note: The function $\|\cdot\| : V \to \mathbb{R}$ defined by $\|v\| := \langle v, v\rangle^{\frac{1}{2}}$ on the inner-product space V, with inner product $\langle\cdot,\cdot\rangle$, trivially satisfies the first two axioms of norm on V; this is a consequence of $\langle\cdot,\cdot\rangle$ being an inner product on V. Result 5 above implies that $\|\cdot\|$ also satisfies the third axiom of norm, the triangle inequality.

Least-Squares Approximation

For the problem of least-squares approximation, $\langle f,g\rangle = \displaystyle\int_a^b w(x)f(x)g(x)\,\mathrm{d}x$ and $\|f\|_2^2 = \langle f,f\rangle$ where $w(x)>0$ on (a,b).

Theorem. If $f \in \mathrm{L}^2_w(a,b)$ and $p_n \in \Pi_n$ is such that

$$\langle f-p_n, r\rangle = 0 \qquad \forall r \in \Pi_n, \tag{1}$$

then

$$\|f-p_n\|_2 \le \|f-r\|_2 \qquad \forall r\in\Pi_n,$$

i.e., p_n is a best (weighted) least-squares approximation to f on $[a,b]$.

Proof.

$$\begin{aligned}\|f-p_n\|_2^2 &= \langle f-p_n, f-p_n\rangle\\ &= \langle f-p_n, f-r\rangle + \langle f-p_n, r-p_n\rangle \qquad \forall r\in\Pi_n\\ &\ \text{Since } r-p_n\in\Pi_n \text{ the assumption (1) implies that}\\ &= \langle f-p_n, f-r\rangle\\ &\le \|f-p_n\|_2\|f-r\|_2 \ \text{ by the Cauchy–Schwarz inequality.}\end{aligned}$$

Dividing both sides by $\|f-p_n\|_2$ gives the required result. □

Remark: the converse is true too (see problem sheet 3).

This gives a direct way to calculate a best approximation: we want to find $p_n(x) = \displaystyle\sum_{k=0}^n \alpha_k x^k$ such that

$$\int_a^b w(x)\left(f - \sum_{k=0}^n \alpha_k x^k\right) x^i\,\mathrm{d}x = 0 \ \text{ for } \ i=0,1,\ldots,n. \tag{2}$$

[Note that (2) holds if, and only if,

$$\int_a^b w(x)\left(f-\sum_{k=0}^n \alpha_k x^k\right)\left(\sum_{i=0}^n \beta_i x^i\right)\mathrm{d}x = 0 \qquad \forall q = \sum_{i=0}^n \beta_i x^i \in \Pi_n.]$$

However, (2) implies that

$$\sum_{k=0}^n \left(\int_a^b w(x)x^{k+i}\,\mathrm{d}x\right)\alpha_k = \int_a^b w(x)f(x)x^i\,\mathrm{d}x \ \text{ for } \ i=0,1,\ldots,n$$

which is the component-wise statement of a matrix equation

$$A\alpha = \varphi, \tag{3}$$

to determine the coefficients $\alpha = (\alpha_0,\alpha_1,\ldots,\alpha_n)^{\mathrm{T}}$, where $A = \{a_{i,k}, i,k = 0,1,\ldots,n\}$, $\varphi = (f_0,f_1,\ldots,f_n)^{\mathrm{T}}$,

$$a_{i,k} = \int_a^b w(x)x^{k+i}\,\mathrm{d}x \ \text{ and } \ f_i = \int_a^b w(x)f(x)x^i\,\mathrm{d}x.$$

The system (3) are called the **normal equations**.

Example: the best least-squares approximation to e^x on $[0,1]$ from Π_1 in $\langle f,g\rangle = \int_a^b f(x)g(x)\,\mathrm{d}x$. We want

$$\int_0^1 [\mathrm{e}^x - (\alpha_0 1 + \alpha_1 x)]1\,\mathrm{d}x = 0 \text{ and } \int_0^1 [\mathrm{e}^x - (\alpha_0 1 + \alpha_1 x)]x\,\mathrm{d}x = 0.$$

$\iff$

$$\begin{aligned} \alpha_0 \int_0^1 \mathrm{d}x + \alpha_1 \int_0^1 x\,\mathrm{d}x &= \int_0^1 \mathrm{e}^x\,\mathrm{d}x \\ \alpha_0 \int_0^1 x\,\mathrm{d}x + \alpha_1 \int_0^1 x^2\,\mathrm{d}x &= \int_0^1 \mathrm{e}^x x\,\mathrm{d}x \end{aligned}$$

i.e.,

$$\begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} \mathrm{e}-1 \\ 1 \end{bmatrix}$$

$\implies \alpha_0 = 4\mathrm{e} - 10$ and $\alpha_1 = 18 - 6\mathrm{e}$, so $p_1(x) := (18 - 6\mathrm{e})x + (4\mathrm{e} - 10)$ is the best approximation.

Proof that the coefficient matrix A is nonsingular will now establish existence and uniqueness of (weighted) $\|\cdot\|_2$ best-approximation.

Theorem. The coefficient matrix A in (3) is nonsingular.

Proof. Suppose not $\implies \exists \alpha \neq 0$ with $A\alpha = 0 \implies \alpha^{\mathrm{T}} A \alpha = 0$

$$\iff \sum_{i=0}^n \alpha_i (A\alpha)_i = 0 \iff \sum_{i=0}^n \alpha_i \sum_{k=0}^n a_{ik}\alpha_k = 0,$$

and using the definition $a_{ik} = \int_a^b w(x) x^k x^i\,\mathrm{d}x$,

$$\iff \sum_{i=0}^n \alpha_i \sum_{k=0}^n \left(\int_a^b w(x) x^k x^i\,\mathrm{d}x \right) \alpha_k = 0.$$

Rearranging gives

$$\int_a^b w(x) \left(\sum_{i=0}^n \alpha_i x^i \right) \left(\sum_{k=0}^n \alpha_k x^k \right) \mathrm{d}x = 0 \text{ or } \int_a^b w(x) \left(\sum_{i=0}^n \alpha_i x^i \right)^2 \mathrm{d}x = 0$$

which implies that $\sum_{i=0}^n \alpha_i x^i = 0$ and thus $\alpha_i = 0$ for $i = 0, 1, \ldots, n$. This contradicts the initial supposition, and thus A is nonsingular. □

Remark:

- Note in the simplest least-squares approximation problem $\min_x \|Ax - b\|_2$ that we dealt with in lecture 4, the theorem gives the solution $A^T(Ax - b) = 0$, that is, $x = (A^T A)^{-1} A^T b$. This coincides with the QR-based solution derived in lecture 4.
- The above theorem does not imply that the normal equations are usable in practice: the method would need to be stable with respect to small perturbations. In fact, difficulties arise from the "ill-conditioning" of the matrix A as n increases. The next lecture looks at a fix.

Numerical Analysis Hilary Term 2026
Lecture 10: Orthogonal Polynomials

Gram–Schmidt orthogonalization procedure: the solution of the normal equations $A\alpha = \varphi$ for best least-squares polynomial approximation would be easy if A were diagonal. Instead of $\{1, x, x^2, \ldots, x^n\}$ as a basis for Π_n, suppose we have a basis $\{\phi_0, \phi_1, \ldots, \phi_n\}$. Then $p_n(x) = \sum_{k=0}^{n} \beta_k \phi_k(x)$, and the normal equations become

$$\int_a^b w(x) \left(f(x) - \sum_{k=0}^{n} \beta_k \phi_k(x) \right) \phi_i(x) \, \mathrm{d}x = 0 \quad \text{for} \quad i = 0, 1, \ldots, n,$$

or equivalently

$$\sum_{k=0}^{n} \left(\int_a^b w(x)\phi_k(x)\phi_i(x) \, \mathrm{d}x \right) \beta_k = \int_a^b w(x) f(x) \phi_i(x) \, \mathrm{d}x, \quad i = 0, \ldots, n, \quad \text{i.e.,}$$

$$A\beta = \varphi, \tag{1}$$

where $\beta = (\beta_0, \beta_1, \ldots, \beta_n)^{\mathrm{T}}$, $\varphi = (f_1, f_2, \ldots, f_n)^{\mathrm{T}}$ and now

$$a_{i,k} = \int_a^b w(x)\phi_k(x)\phi_i(x) \, \mathrm{d}x \quad \text{and} \quad f_i = \int_a^b w(x) f(x) \phi_i(x) \, \mathrm{d}x.$$

So A is diagonal if

$$\langle \phi_i, \phi_k \rangle = \int_a^b w(x)\phi_i(x)\phi_k(x) \, \mathrm{d}x \quad \begin{cases} = 0 & i \neq k \text{ and} \\ \neq 0 & i = k. \end{cases}$$

We can create such a set of **orthogonal polynomials**

$$\{\phi_0, \phi_1, \ldots, \phi_n, \ldots\},$$

with $\phi_i \in \Pi_i$ for each i, by the Gram–Schmidt procedure, which is based on the following lemma.

Lemma. Suppose that $\phi_0, \ldots, \phi_k$, with $\phi_i \in \Pi_i$ for each i, are orthogonal with respect to the inner product $\langle f, g \rangle = \int_a^b w(x) f(x) g(x) \, \mathrm{d}x$. Then,

$$\phi_{k+1}(x) = x^{k+1} - \sum_{i=0}^{k} \lambda_i \phi_i(x)$$

satisfies

$$\langle \phi_{k+1}, \phi_j \rangle = \int_a^b w(x)\phi_{k+1}(x)\phi_j(x) \, \mathrm{d}x = 0, \quad j = 0, 1, \ldots, k, \quad \text{with}$$

$$\lambda_j = \frac{\langle x^{k+1}, \phi_j \rangle}{\langle \phi_j, \phi_j \rangle}, \quad j = 0, 1, \ldots, k.$$

Proof. For any j, $0 \le j \le k$,

$$\begin{aligned}
\langle \phi_{k+1}, \phi_j \rangle &= \langle x^{k+1}, \phi_j \rangle - \sum_{i=0}^{k} \lambda_i \langle \phi_i, \phi_j \rangle \\
&= \langle x^{k+1}, \phi_j \rangle - \lambda_j \langle \phi_j, \phi_j \rangle \\
&\qquad \text{by the orthogonality of } \phi_i \text{ and } \phi_j,\ i \neq j, \\
&= 0 \qquad \text{by definition of } \lambda_j.
\end{aligned}$$

□

Notes: 1. The G–S procedure does this successively for $k = 0, 1, \ldots, n$.
2. ϕ_k is always of exact degree k, so $\{\phi_0, \ldots, \phi_\ell\}$ is a basis for Π_ℓ $\forall \ell \ge 0$.
3. ϕ_k can be normalised to satisfy $\langle \phi_k, \phi_k \rangle = 1$ or to be monic, or ...

Examples: 1. The inner product $\langle f, g \rangle = \int_{-1}^{1} f(x) g(x) \, \mathrm{d}x$

gives orthogonal polynomials called the **Legendre polynomials**,

$$\phi_0(x) \equiv 1, \ \ \phi_1(x) = x, \ \ \phi_2(x) = x^2 - \tfrac{1}{3}, \ \ \phi_3(x) = x^3 - \tfrac{3}{5}x, \ldots$$

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

gives orthogonal polynomials called the **Chebyshev polynomials**,

$$\phi_0(x) \equiv 1, \ \ \phi_1(x) = x, \ \ \phi_2(x) = 2x^2 - 1, \ \ \phi_3(x) = 4x^3 - 3x, \ldots$$

3. The inner product $\langle f, g \rangle = \int_{0}^{\infty} \mathrm{e}^{-x} f(x) g(x) \, \mathrm{d}x$

gives orthogonal polynomials called the **Laguerre polynomials**,

$$\phi_0(x) \equiv 1, \ \ \phi_1(x) = 1 - x, \ \ \phi_2(x) = 2 - 4x + x^2,$$
$$\phi_3(x) = 6 - 18x + 9x^2 - x^3, \ldots$$

Lemma. Suppose that $\{\phi_0, \phi_1, \ldots, \phi_k, \ldots\}$ are orthogonal polynomials for a given inner product $\langle \cdot, \cdot \rangle$. Then, $\langle \phi_k, q \rangle = 0$ whenever $q \in \Pi_{k-1}$.

Proof. This follows since if $q \in \Pi_{k-1}$, then $q(x) = \sum_{i=0}^{k-1} \sigma_i \phi_i(x)$ for some $\sigma_i \in \mathbb{R}$, $i = 0, 1, \ldots, k-1$, so

$$\langle \phi_k, q \rangle = \sum_{i=0}^{k-1} \sigma_i \langle \phi_k, \phi_i \rangle = 0.$$

□

Remark: note from the above argument that if $q(x) = \sum_{i=0}^{k} \sigma_i \phi_i(x)$ is of exact degree k (so $\sigma_k \neq 0$), then $\langle \phi_k, q \rangle = \sigma_k \langle \phi_k, \phi_k \rangle \neq 0$.

Theorem. Suppose that $\{\phi_0, \phi_1, \ldots, \phi_n, \ldots\}$ is a set of orthogonal polynomials. Then, there exist sequences of real numbers $(\alpha_k)_{k=1}^{\infty}$, $(\beta_k)_{k=1}^{\infty}$, $(\gamma_k)_{k=1}^{\infty}$ such that a three-term recurrence relation holds of the form

$$\phi_{k+1}(x) = \alpha_k (x - \beta_k) \phi_k(x) - \gamma_k \phi_{k-1}(x), \qquad k = 1, 2, \ldots.$$

Proof. The polynomial $x\phi_k \in \Pi_{k+1}$, so there exist real numbers

$$\sigma_{k,0}, \sigma_{k,1}, \ldots, \sigma_{k,k+1}$$

such that

$$x\phi_k(x) = \sum_{i=0}^{k+1} \sigma_{k,i}\phi_i(x)$$

as $\{\phi_0, \phi_1, \ldots, \phi_{k+1}\}$ is a basis for Π_{k+1}. Now take the inner product on both sides with ϕ_j where $j \leq k-2$. On the left-hand side, note $x\phi_j \in \Pi_{k-1}$ and thus

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

by the above lemma for $j \leq k-2$. On the right-hand side

$$\left\langle \sum_{i=0}^{k+1} \sigma_{k,i}\phi_i, \phi_j \right\rangle = \sum_{i=0}^{k+1} \sigma_{k,i}\langle \phi_i, \phi_j\rangle = \sigma_{k,j}\langle \phi_j, \phi_j\rangle$$

by the linearity of $\langle \cdot, \cdot \rangle$ and orthogonality of ϕ_i and ϕ_j for $i \neq j$. Hence $\sigma_{k,j} = 0$ for $j \leq k-2$, and so

$$x\phi_k(x) = \sigma_{k,k+1}\phi_{k+1}(x) + \sigma_{k,k}\phi_k(x) + \sigma_{k,k-1}\phi_{k-1}(x).$$

Almost there: taking the inner product with ϕ_{k+1} reveals that

$$\langle x\phi_k, \phi_{k+1}\rangle = \sigma_{k,k+1}\langle \phi_{k+1}, \phi_{k+1}\rangle,$$

so $\sigma_{k,k+1} \neq 0$ by the above remark as $x\phi_k$ is of exact degree $k+1$ (e.g., from above Gram–Schmidt notes). Thus,

$$\phi_{k+1}(x) = \frac{1}{\sigma_{k,k+1}}(x - \sigma_{k,k})\phi_k(x) - \frac{\sigma_{k,k-1}}{\sigma_{k,k+1}}\phi_{k-1}(x),$$

which is of the given form, with

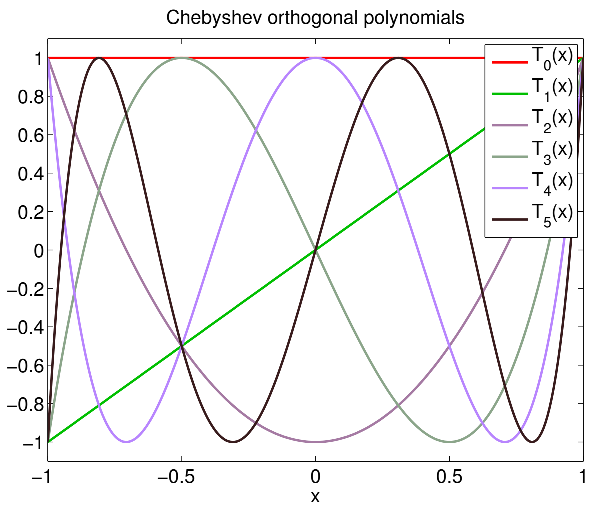
$$\alpha_k = \frac{1}{\sigma_{k,k+1}}, \qquad \beta_k = \sigma_{k,k}, \qquad \gamma_k = \frac{\sigma_{k,k-1}}{\sigma_{k,k+1}}, \qquad k = 1, 2, \ldots.$$

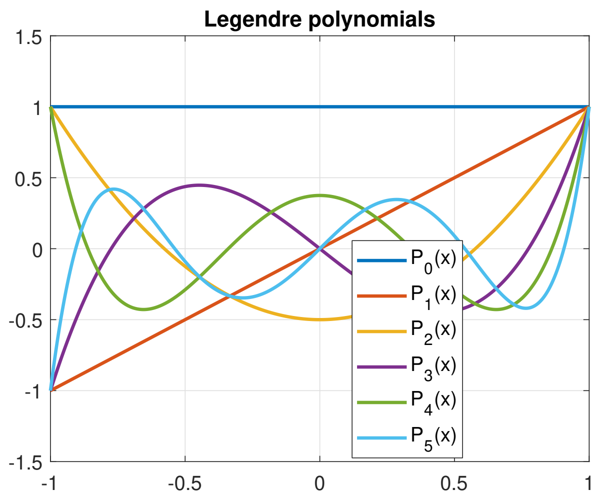
That completes the proof. $\square$

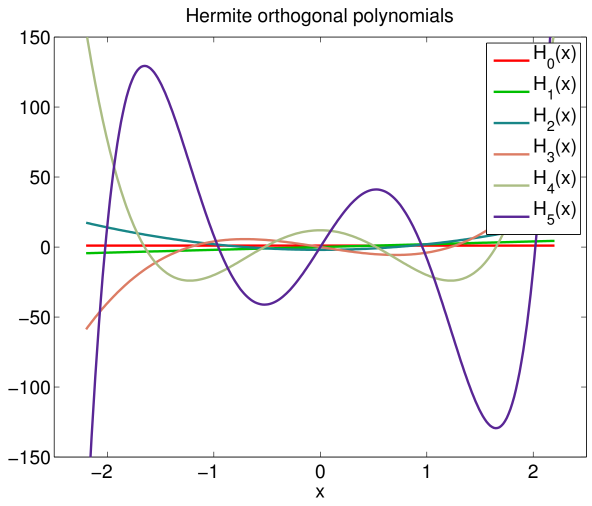
Example. The inner product $\langle f, g\rangle = \displaystyle\int_{-\infty}^{\infty} \mathrm{e}^{-x^2} f(x)g(x)\,\mathrm{d}x$

gives orthogonal polynomials called the **Hermite polynomials**,

$$\phi_0(x) \equiv 1, \;\; \phi_1(x) = 2x, \;\; \phi_{k+1}(x) = 2x\phi_k(x) - 2k\phi_{k-1}(x) \;\text{ for }\; k \geq 1.$$







While all seems good, we note in (1) that to obtain the right-hand side we need to compute the integrals $f_i = \int_a^b w(x)f(x)\phi_i(x)\,\mathrm{d}x$. How do we do this accurately? This is the subject of the next lecture.

Numerical Analysis Hilary Term 2026
Lecture 11: Gauss quadrature

Terminology: Quadrature $\equiv$ numerical integration

Goal: given a (continuous) function $f : [a, b] \to \mathbb{R}$, find its integral $I = \int_a^b f(x)dx$, as accurately as possible.

Idea: Approximate and Integrate. Find a polynomial p_n from data $\{(x_k, f(x_k))\}_{k=0}^n$ by Lagrange interpolation (lecture 1), and integrate $\int_{x_0}^{x_n} p_n(x)\,\mathrm{d}x =: I_n$. Ideally, $I_n = I$ or at least $I_n \approx I$. Is this true?

If we choose x_k to be equispaced points in $[a, b]$, the resulting I_n is known as the Newton-Cotes quadrature. This method is actually quite unstable and inaccurate, and a much more accurate and elegant quadrature rule exists: Gauss quadrature. In this lecture we cover this beautiful result involving orthogonal polynomials.

Preparations: Suppose that w is a weight function, defined, positive and integrable on the open interval (a, b) of $\mathbb{R}$.

Lemma. Let $\{\phi_0, \phi_1, \ldots, \phi_n, \ldots\}$ be orthogonal polynomials for the inner product $\langle f, g\rangle = \displaystyle\int_a^b w(x)f(x)g(x)\,\mathrm{d}x$. Then, for each $k = 0, 1, \ldots,$ ϕ_k has k distinct roots in the interval (a, b).

Proof. Since $\phi_0(x) \equiv$ const. $\neq 0$, the result is trivially true for $k = 0$. Suppose that $k \geq 1$: $\langle\phi_k, \phi_0\rangle = \displaystyle\int_a^b w(x)\phi_k(x)\phi_0(x)\,\mathrm{d}x = 0$ with ϕ_0 constant implies that $\displaystyle\int_a^b w(x)\phi_k(x)\,\mathrm{d}x = 0$ with $w(x) > 0$, $x \in (a, b)$. Thus $\phi_k(x)$ must change sign in (a, b), i.e., ϕ_k has at least one root in (a, b).

Suppose that there are ℓ points $a < r_1 < r_2 < \cdots < r_\ell < b$ where ϕ_k changes sign for some $1 \leq \ell \leq k$. Then

$$q(x) = \prod_{j=1}^{\ell}(x - r_j) \times \text{ the sign of } \phi_k \text{ on } (r_\ell, b)$$

has the same sign as ϕ_k on (a, b). Hence

$$\langle\phi_k, q\rangle = \int_a^b w(x)\phi_k(x)q(x)\,\mathrm{d}x > 0,$$

and thus it follows from the previous lemma (cf. Lecture 12) that q, (which is of degree ℓ) must be of degree $\geq k$, i.e., $\ell \geq k$. However, ϕ_k is of exact degree k, and therefore the number of its distinct roots, ℓ, must be $\leq k$. Hence $\ell = k$, and ϕ_k has k distinct roots in (a, b). □

Application to quadrature. The above lemma leads to very efficient quadrature rules since it answers the question: how should we choose the quadrature points $x_0, x_1, \ldots, x_n$ in the quadrature rule

$$\int_a^b w(x)f(x)\,\mathrm{d}x \approx \sum_{j=0}^{n} w_j f(x_j) \tag{1}$$

so that the rule is exact for polynomials of degree as high as possible? (The case $w(x) \equiv 1$ is the most common.)

Recall: the Lagrange interpolating polynomial

$$p_n = \sum_{j=0}^{n} f(x_j) L_{n,j} \in \Pi_n$$

is unique, so $f \in \Pi_n \implies p_n \equiv f$ whatever interpolation points are used, and moreover

$$\int_a^b w(x) f(x)\,\mathrm{d}x = \int_a^b w(x) p_n(x)\,\mathrm{d}x = \sum_{j=0}^{n} w_j f(x_j),$$

exactly, where

$$w_j = \int_a^b w(x) L_{n,j}(x)\,\mathrm{d}x. \qquad (2)$$

Theorem. Suppose that $x_0 < x_1 < \cdots < x_n$ are the roots of the $n+1$-st degree orthogonal polynomial ϕ_{n+1} with respect to the inner product

$$\langle g, h \rangle = \int_a^b w(x) g(x) h(x)\,\mathrm{d}x. \qquad (3)$$

Then, the quadrature formula (1) with weights (2) is exact whenever $f \in \Pi_{2n+1}$.

Proof. Let $p \in \Pi_{2n+1}$. Then by the Division Algorithm $p(x) = q(x)\phi_{n+1}(x) + r(x)$ with $q, r \in \Pi_n$. So

$$\int_a^b w(x) p(x)\,\mathrm{d}x = \int_a^b w(x) q(x) \phi_{n+1}(x)\,\mathrm{d}x + \int_a^b w(x) r(x)\,\mathrm{d}x = \sum_{j=0}^{n} w_j r(x_j) \qquad (4)$$

since the integral involving $q \in \Pi_n$ is zero by the lemma above and the other is integrated exactly since $r \in \Pi_n$. Finally $p(x_j) = q(x_j)\phi_{n+1}(x_j) + r(x_j) = r(x_j)$ for $j = 0, 1, \ldots, n$ as the x_j are the roots of ϕ_{n+1}. So (4) gives

$$\int_a^b w(x) p(x)\,\mathrm{d}x = \sum_{j=0}^{n} w_j p(x_j),$$

where w_j is given by (2) whenever $p \in \Pi_{2n+1}$. □

These quadrature rules are called **Gauss quadratures**; namely, given a nonnegative weight function w, the $(n+1)$-point Gauss quadrature approximates $\int_a^b w(x)f(x)dx$ by $\int_a^b w(x)p_n(x)dx$, where p_n is the polynomial interpolant to f at $\{x_i\}_{i=0}^n$ where x_i are the distinct roots of the $(n+1)$th orthogonal polynomial in the inner product (3).

- $w(x) \equiv 1$, $(a, b) = (-1, 1)$: Gauss–Legendre quadrature.
- $w(x) = (1 - x^2)^{-1/2}$ and $(a, b) = (-1, 1)$: Gauss–Chebyshev quadrature.
- $w(x) = \mathrm{e}^{-x}$ and $(a, b) = (0, \infty)$: Gauss–Laguerre quadrature.

- $w(x) = \mathrm{e}^{-x^2}$ and $(a, b) = (-\infty, \infty)$: Gauss–Hermite quadrature.

They give better accuracy than Newton–Cotes quadrature for the same number of function evaluations.
Note when using quadrature on unbounded intervals, the integral should be of the form $\int_0^\infty \mathrm{e}^{-x} f(x)\,\mathrm{d}x$ and only f is sampled at the nodes.
Note that by the linear change of variable $t = (2x - a - b)/(b - a)$, which maps $[a, b] \to [-1, 1]$, we can evaluate for example

$$\int_a^b f(x)\,\mathrm{d}x = \int_{-1}^1 f\left(\frac{(b-a)t + b + a}{2}\right)\frac{b-a}{2}\,\mathrm{d}t \simeq \frac{b-a}{2}\sum_{j=0}^n w_j f\left(\frac{b-a}{2}t_j + \frac{b+a}{2}\right),$$

where $\simeq$ denotes "quadrature" and the t_j, $j = 0, 1, \ldots, n$, are the roots of the $n + 1$-st degree Legendre polynomial.

Example. 2-point Gauss–Legendre quadrature: $\phi_2(t) = t^2 - \frac{1}{3} \implies t_0 = -\frac{1}{\sqrt{3}}$, $t_1 = \frac{1}{\sqrt{3}}$ and

$$w_0 = \int_{-1}^1 \frac{t - \frac{1}{\sqrt{3}}}{-\frac{1}{\sqrt{3}} - \frac{1}{\sqrt{3}}}\,\mathrm{d}t = -\int_{-1}^1 \left(\tfrac{\sqrt{3}}{2}t - \tfrac{1}{2}\right)\,\mathrm{d}t = 1,$$

with $w_1 = 1$, similarly. So e.g., changing variables $x = (t + 3)/2$,

$$\int_1^2 \frac{1}{x}\,\mathrm{d}x = \frac{1}{2}\int_{-1}^1 \frac{2}{t+3}\,\mathrm{d}t \simeq \frac{1}{3 + \frac{1}{\sqrt{3}}} + \frac{1}{3 - \frac{1}{\sqrt{3}}} = 0.6923077\ldots\ .$$

Note that the trapezium rule (also two evaluations of the integrand) gives

$$\int_1^2 \frac{1}{x}\,\mathrm{d}x \simeq \frac{1}{2}\left[\frac{1}{2} + 1\right] = 0.75,$$

whereas $\displaystyle\int_1^2 \frac{1}{x}\,\mathrm{d}x = \ln 2 = 0.6931472\ldots\ .$

Theorem. Error in Gauss quadrature: suppose that $f^{(2n+2)}$ is continuous on (a, b). Then

$$\int_a^b w(x) f(x)\,\mathrm{d}x = \sum_{j=0}^n w_j f(x_j) + \frac{f^{(2n+2)}(\eta)}{(2n+2)!}\int_a^b w(x)\prod_{j=0}^n (x - x_j)^2\,\mathrm{d}x,$$

for some $\eta \in (a, b)$.

Proof. The proof is based on the Hermite interpolating polynomial H_{2n+1} to f on $x_0, x_1, \ldots, x_n$. [Recall that $H_{2n+1}(x_j) = f(x_j)$ and $H'_{2n+1}(x_j) = f'(x_j)$ for $j = 0, 1, \ldots, n$.] The error in Hermite interpolation is

$$f(x) - H_{2n+1}(x) = \frac{1}{(2n+2)!} f^{(2n+2)}(\eta(x))\prod_{j=0}^n (x - x_j)^2$$

for some $\eta = \eta(x) \in (a, b)$. Now $H_{2n+1} \in \Pi_{2n+1}$, so

$$\int_a^b w(x) H_{2n+1}(x)\,\mathrm{d}x = \sum_{j=0}^n w_j H_{2n+1}(x_j) = \sum_{j=0}^n w_j f(x_j),$$

the first identity because Gauss quadrature is exact for polynomials of this degree and the second by interpolation. Thus

$$\begin{aligned}\int_a^b w(x)f(x)\,\mathrm{d}x - \sum_{j=0}^{n} w_j f(x_j) = & \int_a^b w(x)[f(x) - H_{2n+1}(x)]\,\mathrm{d}x \\ = & \frac{1}{(2n+2)!}\int_a^b f^{(2n+2)}(\eta(x))w(x)\prod_{j=0}^{n}(x-x_j)^2\,\mathrm{d}x,\end{aligned}$$

and hence the required result follows from the integral mean value theorem as $w(x)\prod_{j=0}^{n}(x-x_j)^2 \geq 0$. □

Remark: the "direct" approach of finding Gauss quadrature formulae sometimes works for small n, but more sophisticated algorithms are used for large n.[1]

Example. To find the two-point Gauss–Legendre rule $w_0f(x_0)+w_1f(x_1)$ on $(-1,1)$ with weight function $w(x) \equiv 1$, we need to be able to integrate any cubic polynomial exactly, so

$$2 = \int_{-1}^{1} 1\,\mathrm{d}x = w_0 + w_1 \tag{5}$$

$$0 = \int_{-1}^{1} x\,\mathrm{d}x = w_0x_0 + w_1x_1 \tag{6}$$

$$\tfrac{2}{3} = \int_{-1}^{1} x^2\,\mathrm{d}x = w_0x_0^2 + w_1x_1^2 \tag{7}$$

$$0 = \int_{-1}^{1} x^3\,\mathrm{d}x = w_0x_0^3 + w_1x_1^3. \tag{8}$$

These are four nonlinear equations in four unknowns w_0, w_1, x_0 and x_1. Equations (6) and (8) give

$$\begin{bmatrix} x_0 & x_1 \\ x_0^3 & x_1^3 \end{bmatrix}\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

which implies that

$$x_0x_1^3 - x_1x_0^3 = 0$$

for w_0, $w_1 \neq 0$, i.e.,

$$x_0x_1(x_1 - x_0)(x_1 + x_0) = 0.$$

If $x_0 = 0$, this implies $w_1 = 0$ or $x_1 = 0$ by (6), either of which contradicts (7). Thus $x_0 \neq 0$, and similarly $x_1 \neq 0$. If $x_1 = x_0$, (6) implies $w_1 = -w_0$, which contradicts (5). So $x_1 = -x_0$, and hence (6) implies $w_1 = w_0$. But then (5) implies that $w_0 = w_1 = 1$ and (7) gives

$$x_0 = -\tfrac{1}{\sqrt{3}} \qquad \text{and} \qquad x_1 = \tfrac{1}{\sqrt{3}},$$

[1]See e.g., the research paper by Hale and Townsend, "Fast and accurate computation of Guass–Legendre and Gauss–Jacobi quadrature nodes and weights" SIAM J. Sci. Comput. 2013.

which are the roots of the Legendre polynomial $x^2 - \frac{1}{3}$.

Convergence: Gauss quadrature converges astonishingly fast. It can be shown that if f is analytic on $[a,b]$, the convergence is geometric (exponential) in the number of samples. This is in contrast to other (more straightforward) quadrature rules:

- Newton-Cotes: Find interpolant in n equispaced points, and integrate interpolant. Convergence: (often) Divergent!
- (Composite) trapezium rule: Find piecewise-linear interpolant in n equispaced points, and integrate interpolant. Convergence: $O(1/n^2)$ (assumes f'' exists)
- (Composite) Simpson's rule: Find piecewise-quadratic interpolant in n equispaced points (each subinterval containing three points), and integrate interpolant. Convergence: $O(1/n^4)$ (assumes f'''' exists)

The figure below illustrates the performance on integrating the Runge function.

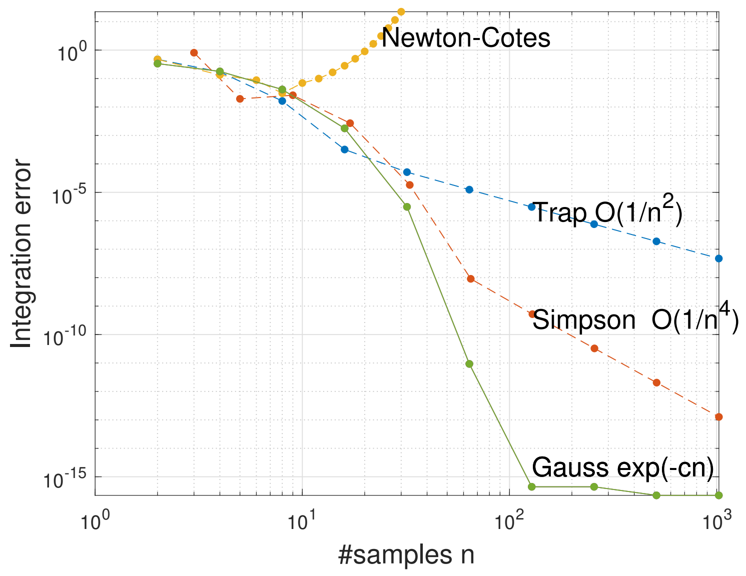
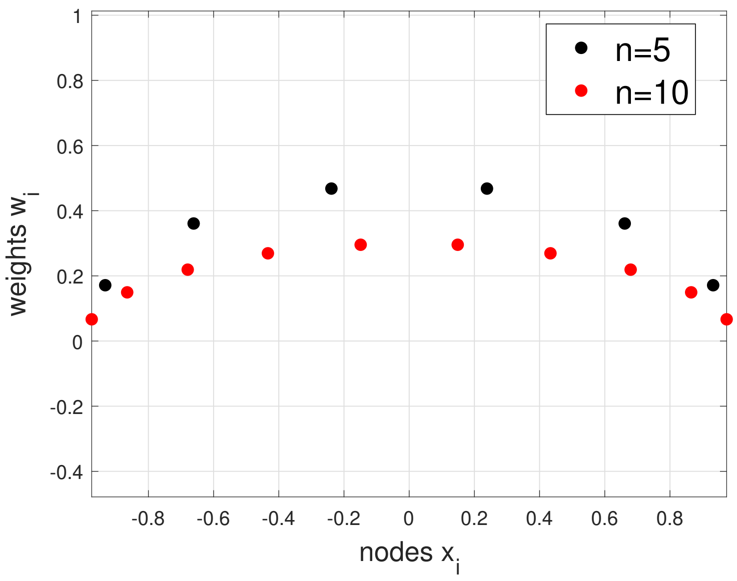


Figure 1: Convergence of quadrature rules for $\int_{-1}^{1} \frac{1}{25x^2+1} dx$ (Runge function)

Nodes and weights for Gauss(-Legendre) quadrature The figure below shows the nodes (interpolation points) and the corrsponding weights with the standard Gauss-Legendre quadrature rule, i.e., when $w(x) = 1$ and $[a,b] = [-1,1]$. In Chebfun these are computed conveniently by `[x,w] = legpts(n+1)`



Note that the nodes/interpolation points cluster near endpoints (and sparser in the middle); this is a general phenomenon, and very analogous to the Chebyshev interpolation points mentioned in the least-squares lecture (Gauss and Chebyshev points have asymptotically the same distribution of points). Note also that the weights are all positive and shrink as n grows; they have to because they sum to 2 (why?).

Numerical Analysis Hilary Term 2026
Lecture 12–13: Introduction to numerical methods for initial-value problems

Initial value problems (IVP)[1]**:** These arise everywhere in mathematics where we wish to model the evolution in time of a given system.

Definition 1. *Let $I = [x_0, X] \subset \mathbb{R}$ be a (time) interval and $D \subset \mathbb{R}^d$ be an open subset, where $d \in \mathbb{N}^+$ denotes the space dimension.*

- *A* first-order ordinary differential equation *(ODE) is an equation of the form*
$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}),$$
where the righthand side is a function $\mathbf{f} : I \times D \to \mathbb{R}^d$.
- *An* initial value problem *(IVP) is an ODE with an initial condition, that is,*
$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}), \qquad \mathbf{y}(x_0) = \mathbf{y}_0 .$$
The goal is to find $\mathbf{y}(x)$, either at the final (time) value $x = X$, or any $x \in [x, X]$.

We'll focus on IVPs in this course, but another important class of ODEs is boundary value problems (BVP), where the condition(s) are imposed at e.g. both the initial and final values $x = x$ and X (and more intricate conditions in domains $I \subseteq \mathbb{R}^2$ and $I \subseteq \mathbb{R}^3$).

Picard's Theorem gives sufficient conditions to ensure that the IVP admits a unique solution[2].

Picard's Theorem. Suppose that $\mathbf{f}$ is continuous in a neighborhood $U \subset \mathbb{R}^{1+d}$ of $(x_0, \mathbf{y}_0)$ that contains the (closed) cylinder
$$R = \{(x, \mathbf{y}) : x_0 \le x \le X_M, \|\mathbf{y} - \mathbf{y}_0\| \le Y_M\},$$
where $X_M > x_0$ and $Y_M > 0$ are constants. Suppose also that there exists a positive constant L such that
$$\|\mathbf{f}(x, \mathbf{y}) - \mathbf{f}(x, \mathbf{z})\| \le L\|\mathbf{y} - \mathbf{z}\|$$
holds whenever $(x, \mathbf{y})$ and $(x, \mathbf{z})$ lie in R. Finally, letting
$$M := \max\{\|\mathbf{f}(x, \mathbf{y})\| : (x, \mathbf{y}) \in R\},$$
suppose that $M(X_M - x_0) \le Y_M$. Then, there exists a unique continuously differentiable function
$$[x_0, X_M] \ni x \mapsto \mathbf{y}(x) \in \mathbb{R}^d$$
that is the solution to our IVP.

Remarks about Picard's theorem.

[1]The remaining lecture notes on IVP are based on notes by Patrick Farrell, Yuji Nakatsukasa, Alberto Paganini, and Endre Süli. Many of the figures presented are courtesy of Maike Meier.

[2]In the remainder, we only deal with problems that satisfy the conditions in Picard's theorem, and assume $\mathbf{y}$ is differentiable as many times as we take derivatives. For more details about Picard's theorem, including the proof, we refer to the lecture notes of A1 Differential Equations. Another source is chapter 11 of Nick Trefethen's book *Exploring ODEs*, which is freely available at http://people.maths.ox.ac.uk/trefethen/Exploring.pdf

- The solution, if it exists, can be expressed by the integral equation
$$\mathbf{y}(x) = \mathbf{y}(x_0) + \int_{x_0}^{x} \mathbf{f}(t, \mathbf{y}(t))\mathrm{d}t\,, \tag{1}$$
which is obtained by integrating the IVP, and where the integration is to be understood componentwise.

- Picard's theorem guarantees the existence of a solution only up to a finite time X_M; consider : $y' = y^2$, $y(0) = 1$, which has solution $y(x) = (1-x)^{-1}$ and blows up at $x = 1$.

- If an IVP satisfies the assumptions of Picard's theorem, its solution is stable on the bounded interval $[x_0, X]$. This means that if $\mathbf{y} : [x_0, X] \to D$ solves the IVP
$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y})\,, \qquad \mathbf{y}(x_0) = \mathbf{y}_0\,, \tag{2}$$
and $\tilde{\mathbf{y}} : [x_0, X] \to D$ solves the same ODE with a perturbed initial condition $\tilde{\mathbf{y}}_0$, that is,
$$\tilde{\mathbf{y}}'(x) = \mathbf{f}(x, \tilde{\mathbf{y}})\,, \qquad \tilde{\mathbf{y}}(x_0) = \tilde{\mathbf{y}}_0\,,$$
then
$$\|\mathbf{y}(x) - \tilde{\mathbf{y}}(x)\| \le e^{L(X-x_0)}\|\mathbf{y}_0 - \tilde{\mathbf{y}}_0\| \quad \forall\, x \in [x_0, X]\,.$$
(We'll prove a related result in the theorem below.) This implies that a small error in the initial condition does not compromise dramatically the solution of the IVP. This is an important property—small errors in $\mathbf{y}$, either at x_0 or some other $x \in [x_0, X)$, would not stop us from getting an accurate $\mathbf{y}$ at $x = X$. However, note that the constant $e^{L(X-x_0)}$ in the above bound grows exponentially as the final time X increases.

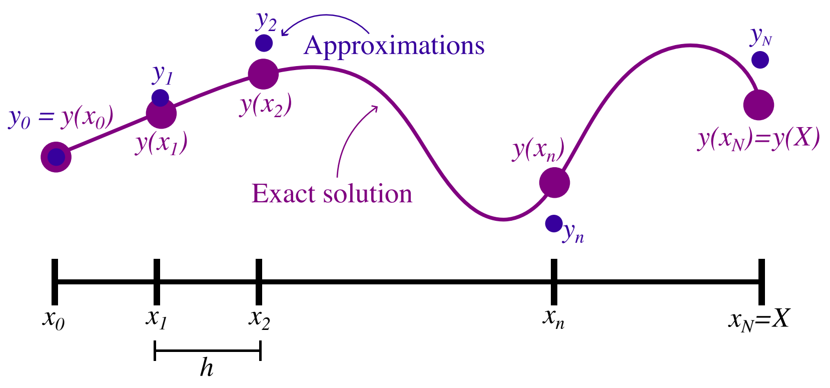
Note. Any higher-order IVP can be reformulated as a larger first-order IVP. The simplest way to do so is to define $\hat{\mathbf{y}} := \left[\ y, y', y'', \dots, y^{(k)}\ \right]^T$ and solve the IVP with respect to $\hat{\mathbf{y}}$. We therefore mainly consider numerical methods for first-order problems. It is also possible to reformulate nonautonomous problems as larger autonomous ones (wherein $\mathbf{y}' = \mathbf{f}(\mathbf{y})$, by working with $\hat{\mathbf{y}} := \begin{bmatrix} x \\ \mathbf{y} \end{bmatrix}$), so sometimes we will restrict ourselves to autonomous ones when it is convenient to do so.

Numerical methods for IVPs: basic idea (Explicit and Implicit methods)

An inconvenient truth is that most IVPs (and most differential equations) cannot be solved analytically (i.e., exactly, to obtain closed-form solutions). It therefore becomes necessary to find approximate solutions with a numerical algorithm. Fortunately, a number of reliable and efficient methods are available for such solution. The remainder of this course is devoted to these methods and their analysis.

Assume that the IVP
$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y})\,, \qquad \mathbf{y}(x_0) = \mathbf{y}_0\,,$$
admits a unique stable solution $\mathbf{y} : [x_0, X] \to D$ that is defined on the bounded interval $[x_0, X]$. How can we compute a numerical approximation of $\mathbf{y}$ that can be made arbitrarily

accurate? A simple idea is to first divide the interval $[x_0, X]$ into $N \in \mathbb{N}^+$ subintervals defined by the equidistant points $x_n = x_0 + nh$, $n = 0, \ldots, N$, where the *step size* h is $h = (X - x_0)/N$. To each time step x_n, we associate an approximation $\mathbf{y}_n$ of $\mathbf{y}(x_n)$.



To define how to compute these approximations, we take inspiration from a variant of the integral equation (1)

$$\mathbf{y}(x_{n+1}) = \mathbf{y}(x_n) + \int_{x_n}^{x_{n+1}} \mathbf{f}(x, \mathbf{y}(x))\,\mathrm{d}x\,,$$

This equality suggests that, if we have already computed an approximation $\mathbf{y}_n$ of $\mathbf{y}(x_n)$, we could compute $\mathbf{y}_{n+1}$ by adding to $\mathbf{y}_n$ an approximation of the integral appearing on the right-hand side. There is therefore a deep connection between quadrature and the solution of IVPs. Indeed if $\mathbf{f}(x, \mathbf{y}) = \mathbf{f}(x)$, i.e., $\mathbf{f}$ does not depend on $\mathbf{y}$, then computing $\mathbf{y}$ is a standard quadrature problem, and can be solved by e.g. Gauss quadrature. Starting with $n = 0$, we could iterate such a strategy to compute the entire sequence $\{\mathbf{y}_n\}_{n=0}^N$. In what follows, we construct three different schemes based on three different (and still very similar) approximations of the integral and investigate the impact that this choice has on the properties of the resulting numerical method.

To construct an approximation of the integral, we recall that by the mean value theorem there is a $\xi \in [x_n, x_{n+1}]$ such that

$$\int_{x_n}^{x_{n+1}} \mathbf{f}(x, \mathbf{y}(x))\,\mathrm{d}x = h\mathbf{f}(\xi, \mathbf{y}(\xi))\,.$$

Therefore, we can construct an approximation by replacing ξ with a value s we like. The resulting numerical approximation rule is called a *rectangle rule.* A consequence is that, for any $s \in [x_n, x_{n+1}]$, the approximation error is at most

$$\int_{x_n}^{x_{n+1}} \mathbf{f}(x, \mathbf{y}(x))\,\mathrm{d}x - h\mathbf{f}(s, \mathbf{y}(s)) = h(\mathbf{f}(\xi, \mathbf{y}(\xi)) - \mathbf{f}(s, \mathbf{y}(s))) \leq h \max_{r \in [x_n, x_{n+1}]} |\mathbf{f}(r, \mathbf{y}(r)) - \mathbf{f}(s, \mathbf{y}(s))|.$$

For instance, we can choose $s = x_n$, so that

$$\int_{x_n}^{x_{n+1}} \mathbf{f}(x, \mathbf{y}(x))\,\mathrm{d}x \approx h\mathbf{f}(x_n, \mathbf{y}(x_n))\,.$$

Inserting this gives

$$\mathbf{y}(x_{n+1}) \approx \mathbf{y}(x_n) + h\mathbf{f}(x_n, \mathbf{y}(x_n))\,,$$

which motivates the definition of the *explicit Euler method*[3]

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(x_n, \mathbf{y}_n)\,.$$

Two other interesting choices are $\xi = x_{n+1}$ and $\xi = (x_n + x_{n+1})/2$, which give rise to the *implicit Euler method*

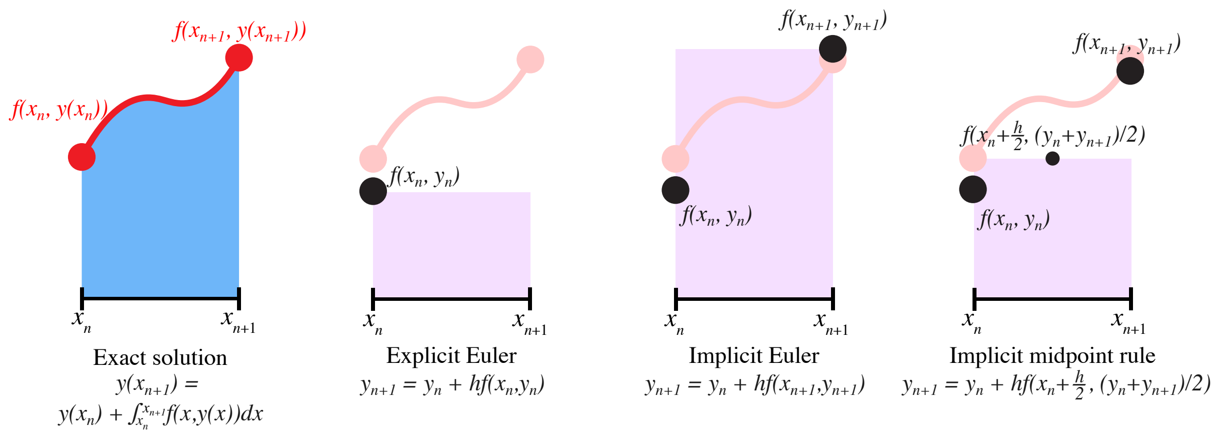
$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(x_{n+1}, \mathbf{y}_{n+1})$$

and the *implicit midpoint rule*

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(x_n + h/2, (\mathbf{y}_n + \mathbf{y}_{n+1})/2)\,,$$

respectively.

Here is an illustration of the three methods.



Note the occurrence of $\mathbf{y}_{n+1}$ on the right-hand side of these last two methods. These numerical methods are called *implicit*, because computing $\mathbf{y}_{n+1}$ requires solving a (generally nonlinear) system, which makes them more computationally expensive than explicit Euler. The arising equations are typically solved with Newton's method, which you saw in M4 Constructive Mathematics. Explicit methods are faster per timestep, but as we will see often suffer from severe timestep restrictions to retain stability, and implicit methods are usually faster for such problems.

Examples. We test these methods on two different examples. First, we consider the linear test case

$$y' = \lambda y\,, \quad y_0 = 1\,, \quad x \in [0,1]\,. \tag{3}$$

For $\lambda = 3$ and $N = 10$, we observe that all three methods compute a qualitatively correct solution, although the one computed with the implicit midpoint rule is way more accurate. Doubling the value of N, we see that the accuracy of the Euler methods improves, although they are not nearly as accurate as the implicit midpoint rule.

[3]The explicit and the implicit Euler methods have been known since 1768! Due, of course, to the great Leonhard Euler.

Next, we investigate what happens for negative values of λ. This case is interesting because the exact solution converges to 0 exponentially as $x \to \infty$. We fix $N = 10$ and investigate different values of λ. For $\lambda \in [-1, -10]$, we see that all methods provide a qualitatively correct solution. For $\lambda < -10$, we see that the explicit Euler solution start oscillating, becoming equioscillatory for $\lambda = -20$, and diverging to $\pm\infty$ for $\lambda < -20$.

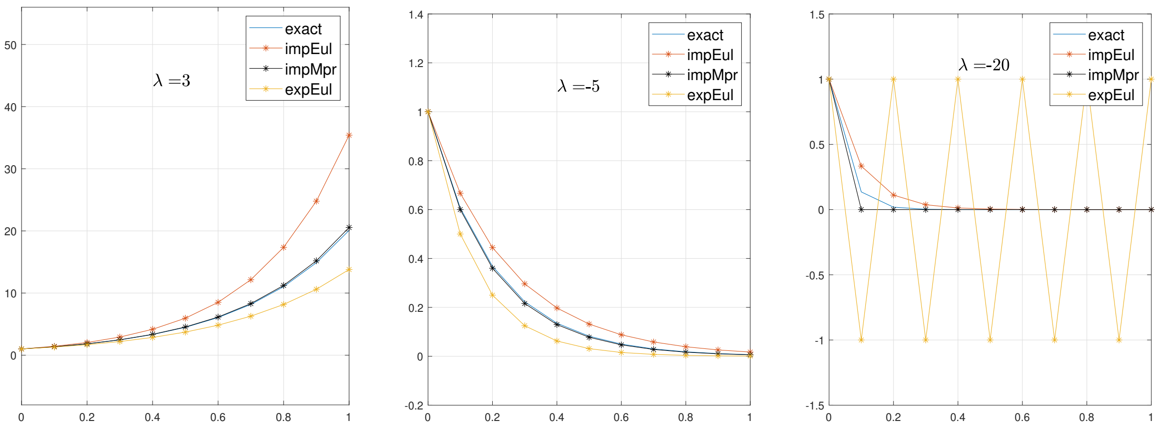


Figure 1: Solving (3) using explicit Euler, midpoint and implicit Euler methods.

For $\lambda < -20$, the solution computed with the implicit midpoint rule also starts to oscillate, although the level of these oscillations cannot be compared with the ones of the explicit Euler method, and the method does not diverge (not even for $\lambda = -9000$). On the other hand, it is surprising to see that the implicit Euler method provides excellent solutions for any negative number of λ. This example shows that the stability of a numerical method can vary drastically. We will explore this further in the upcoming lectures.

The second test case we consider is the following IVP:

$$\mathbf{y}' = \begin{pmatrix} y_2 \\ -y_1 \end{pmatrix}, \quad \mathbf{y}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad x \in [0, 2\pi]\,, \tag{4}$$

whose analytical (exact) solution is $\mathbf{y}(x) = \binom{\cos(x)}{\sin(x)}$. This case is interesting because the quantity $Q(\mathbf{y}) := \|\mathbf{y}(x)\|$ is constant in time. We fix $N = 40$ and plot the orbit (that is, the curve $t \mapsto \mathbf{y}(x)$) of the numerical solutions computed with the three methods above and the evolution of their quantity Q.

The numerical solutions are illustrated in Figure 2. We see that the implicit midpoint rule is the only method that preserves Q, and that it does it up to machine precision! The laws of physics are typically formulated by considering the conservation laws of energy, mass, momentum, etc., and numerical methods that exactly preserve key structural properties of the underyling models are now of prime importance. This line of thinking has led to a beautiful confluence of numerical analysis with geometry and topology, leading to the field called geometric numerical integration (which is off-syllabus).

Convergence and consistency of a one-step method To discuss the convergence and convergence speed of methods, we formally define the class of one-step methods.

Definition 2. *A* one-step method *is a function* $\boldsymbol{\Psi}$ *that takes the triplet* $(s, \mathbf{y}, h) \subset \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}$ *and a function* $\mathbf{f}$, *and computes an approximation* $\boldsymbol{\Psi}(s, \mathbf{y}, h, \mathbf{f})$ *of* $\mathbf{y}(s + h)$, *which*

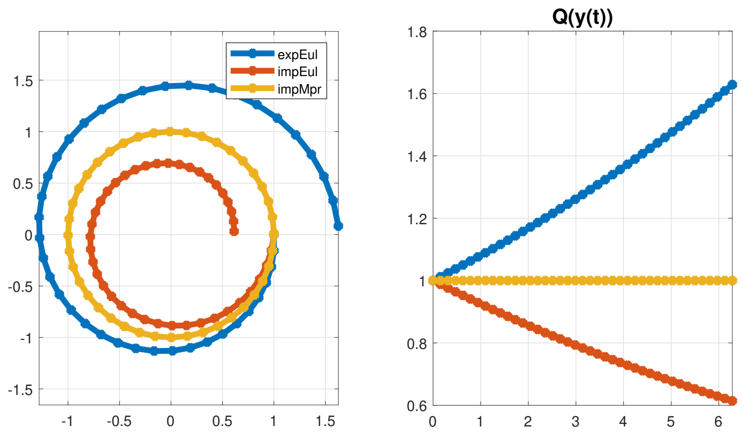


Figure 2: Numerical solutions for (4).

is the solution at $s+h$ of the IVP

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}), \quad \mathbf{y}(s) = \mathbf{y}.$$

Here, we tacitly assume that $\mathbf{y}(s+h)$ exists. Additionally, the timestep h may need to be sufficiently small for $\boldsymbol{\Psi}$ to be well defined.

Definition 3. *A one-step method $\boldsymbol{\Psi}$ is said to be* consistent *if*

$$\boldsymbol{\Psi}(s, \mathbf{y}, 0, \mathbf{f}) = \mathbf{y}$$

and

$$\frac{\mathrm{d}}{\mathrm{d}h}\boldsymbol{\Psi}(s, \mathbf{y}, h, \mathbf{f})|_{h=0} = \mathbf{f}(s, \mathbf{y}).$$

Informally, consistency imposes that the derivative is computed sensibly. It can be seen as a minimum requirement for a good method—the more interesting question is the consistency order, which indicates how quickly the consistency error goes to 0 as $h \to 0$. To examine this, we define

Definition 4. *The* consistency error *(also known as the local trunction error) $\boldsymbol{\tau}$ is defined as*

$$\boldsymbol{\tau}(s, \mathbf{y}, h, \mathbf{f}) := \frac{\mathbf{y}(s+h) - \mathbf{y}}{h} - \frac{\boldsymbol{\Psi}(s, \mathbf{y}, h, \mathbf{f}) - \mathbf{y}}{h} = \frac{\mathbf{y}(s+h) - \boldsymbol{\Psi}(s, \mathbf{y}, h, \mathbf{f})}{h}, \tag{5}$$

where $\mathbf{y}(s+h)$ is the solution at $s+h$ of the IVP.

The following lemma gives additional insight about the definition of consistent one-step method. The gist of it is that a one-step method is consistent if the consistency error can be made arbitrarily small by reducing h.

Lemma. Assume that $h \mapsto \boldsymbol{\Psi}(s, \mathbf{y}, h, \mathbf{f})$ is continuously differentiable in a neighborhood of 0. Then, $\boldsymbol{\Psi}$ is consistent if and only if, for any fixed $\mathbf{f}$,

$$\|\boldsymbol{\tau}(\tilde{s}, \tilde{\mathbf{y}}, h, \mathbf{f})\| \to 0 \quad \text{as } h \to 0$$

locally uniformly in $(\tilde{s}, \tilde{\mathbf{y}}) \in R$, where R is the cylinder from Picard's Theorem.

It is sometimes convenient to represent an abstract one-step method via its *increment function.*

Lemma. Assume that $h \mapsto \mathbf{\Psi}(s, \mathbf{y}, h, \mathbf{f})$ is continuously differentiable in a neighborhood of 0. Then, $\mathbf{\Psi}$ is consistent if and only if there is a continuous increment function $h \mapsto \boldsymbol{\psi}(s, \mathbf{y}, h, \mathbf{f})$ such that

$$\mathbf{\Psi}(s, \mathbf{y}, h, \mathbf{f}) = \mathbf{y} + h\boldsymbol{\psi}(s, \mathbf{y}, h, \mathbf{f}), \quad \boldsymbol{\psi}(s, \mathbf{y}, 0, \mathbf{f}) = \mathbf{f}(s, \mathbf{y}).$$

We also define the *global error*

$$e_n := \mathbf{y}(x_n) - \mathbf{y}_n.$$

Then $e := e_N = \mathbf{y}(x_N) - \mathbf{y}_N$ is the (global) error in the solution at $x = X$. With a good method, we hope $e \to 0$ as the step size $h \to 0$ (and hence $\boldsymbol{\tau} \to 0$).

Theorem. Let $\mathbf{\Psi}$ be a consistent one-step method and assume that its increment function $\boldsymbol{\psi}$ is Lipschitz continuous with respect to $\mathbf{y}$, that is, that there exists a positive constant $L_{\boldsymbol{\psi}}$ such that, for $0 \le h \le h_0$ and for the same region R of Picard's theorem,

$$\|\boldsymbol{\psi}(x, \mathbf{y}, h, \mathbf{f}) - \boldsymbol{\psi}(x, \mathbf{z}, h, \mathbf{f})\| \le L_{\boldsymbol{\psi}} \|\mathbf{y} - \mathbf{z}\| \quad \text{for} (x, \mathbf{y}), (x, \mathbf{z}) \text{ in } R.$$

Then, assuming that $(x_n, \mathbf{y}_n)$ remains in R, it follows that

$$e \le \left(\frac{\exp\left(L_{\boldsymbol{\psi}}(x_N - x_0)\right) - 1}{L_{\boldsymbol{\psi}}} \right) \max_{n=0,\dots,N-1} \|\boldsymbol{\tau}(x_n, \mathbf{y}(x_n), h, \mathbf{f})\|.$$

Proof. For a generic $n \in \{1, \dots, N-1\}$,

$$\begin{aligned}
e_{n+1} &= \|\mathbf{y}(x_{n+1}) - \mathbf{y}_{n+1}\|, \\
&= \|\mathbf{y}(x_{n+1}) - \mathbf{\Psi}(x_n, \mathbf{y}_n, h, \mathbf{f})\|, \\
&= \|\mathbf{y}(x_{n+1}) - \mathbf{\Psi}(x_n, \mathbf{y}(x_n), h, \mathbf{f}) + \mathbf{\Psi}(x_n, \mathbf{y}(x_n), h, \mathbf{f}) - \mathbf{\Psi}(x_n, \mathbf{y}_n, h, \mathbf{f})\|, \\
&\le \|\mathbf{y}(x_{n+1}) - \mathbf{\Psi}(x_n, \mathbf{y}(x_n), h, \mathbf{f})\| + \|\mathbf{\Psi}(x_n, \mathbf{y}(x_n), h, \mathbf{f}) - \mathbf{\Psi}(x_n, \mathbf{y}_n, h, \mathbf{f})\|, \\
&= h\|\boldsymbol{\tau}(x_n, \mathbf{y}(x_n), h, \mathbf{f})\| + \|\left(\mathbf{y}(x_n) + h\boldsymbol{\psi}(x, \mathbf{y}(x_n), h, \mathbf{f})\right) - \left(\mathbf{y}_n + h\boldsymbol{\psi}(x, \mathbf{y}_n, h, \mathbf{f})\right)\|, \\
&\le h\|\boldsymbol{\tau}(x_n, \mathbf{y}(x_n), h, \mathbf{f})\| + \|\mathbf{y}(x_n) - \mathbf{y}_n\| + h\|\boldsymbol{\psi}(x, \mathbf{y}(x_n), h, \mathbf{f}) - \boldsymbol{\psi}(x, \mathbf{y}_n, h, \mathbf{f})\|, \\
&= h\|\boldsymbol{\tau}(x_n, \mathbf{y}(x_n), h, \mathbf{f})\| + e_n + h\|\boldsymbol{\psi}(x, \mathbf{y}(x_n), h, \mathbf{f}) - \boldsymbol{\psi}(x, \mathbf{y}_n, h, \mathbf{f})\|, \\
&\le h\|\boldsymbol{\tau}(x_n, \mathbf{y}(x_n), h, \mathbf{f})\| + e_n + hL_{\boldsymbol{\psi}}\|\mathbf{y}(x_n) - \mathbf{y}_n\|, \\
&= h\|\boldsymbol{\tau}(x_n, \mathbf{y}(x_n), h, \mathbf{f})\| + (1 + hL_{\boldsymbol{\psi}})e_n.
\end{aligned}$$

Iterating recursively, this implies that (note that $e_0 = 0$)

$$\begin{aligned}
e_{n+1} &\le (1 + hL_{\boldsymbol{\psi}})^{n+1} e_0 + h \textstyle\sum_{k=0}^{n} (1 + hL_{\boldsymbol{\psi}})^k \max_{m=0,\dots,n} \|\boldsymbol{\tau}(x_m, \mathbf{y}(x_m), h, \mathbf{f})\| \\
&= \tfrac{(1+hL_{\boldsymbol{\psi}})^{n+1} - 1}{L_{\boldsymbol{\psi}}} \max_{m=0,\dots,n} \|\boldsymbol{\tau}(x_m, \mathbf{y}(x_m), h, \mathbf{f})\|.
\end{aligned}$$

To conclude the proof, note that $1 + hL_{\boldsymbol{\psi}} \le \exp hL_{\boldsymbol{\psi}}$. $\square$

Note that the global error accounts for the accumulation of errors over all steps, wheres the consistency error measures the error in a single step.

Order of accuracy and consistency order. In the following lectures we will examine methods that can give higher accuracy than Euler's methods. To explore these, let us state a few definitions. A method is said to have the *order of accuracy* (or just *order*) p if $e \le Ch^p$ for some constant C; $p > 0$ is usually an integer. The related notion of *consistency order* is the largest $\tilde{p}$ such that the consistency error $\|\boldsymbol{\tau}(s, \mathbf{y}, h, \mathbf{f})\| \le \tilde{C}h^{\tilde{p}}$ for some $\tilde{C}$. The consistency order $\tilde{p}$ measures the local error, whereas p does the global error; they are usually the same, as the above theorem suggests. Informally, at each step the computation incurs an error of $O(h^{\tilde{p}+1})$ (note the $+1$ from the definition (5)), and we perform $O(h^{-1})$ steps, so the global error is $O(h^{\tilde{p}}) = O(h^p)$; the theorem above makes this precise.

In the next lectures we will study two classes of methods—Runge-Kutta and multistep methods—that can achieve order of accuracy higher than 1. These usually result in more accurate solutions for a fixed computational budget. In studying these methods, the two overarching questions we shall address are:

(i) *Convergence*: Does the method converge to the exact solution as $h \to 0$?

(ii) *Stability*: How small does h need to be for the computed solution to start resembling the exact solution?

The theorem above shows that the answer to (i) is straightforward for one-step methods (including Runge-Kutta methods); roughly, convergence holds as $h \to 0$ if the method is consistent. The second question (ii) is intricate (as we saw already in the discussion of the test problem (3)), and we'll explore this question for Runge-Kutta methods in the next lecture.

For multistep methods, even (i) is not obvious; we'll treat both questions in the final lectures.

Listing 1: l11_ivp1.m

```
clear, set(0,'DefaultFigureWindowStyle','docked')
N = 10; h = 1/N; lambda = -20; %modify these parameters to experiment
expEul = nan(1, N+1); impEul = nan(1, N+1); impMpr = nan(1, N+1);
y0 = 1; expEul(1) = y0; impEul(1) = y0; impMpr(1) = y0;

for ii = 1:N
    expEul(ii+1) = expEul(ii)*(1+h*lambda);
    impEul(ii+1) = impEul(ii)/(1-h*lambda);
    impMpr(ii+1) = impMpr(ii)*(1+h*lambda/2)/(1-h*lambda/2);
end

t = linspace(0, 1, N+1);figure(1);
plot(t, exp(lambda*t), t, impEul, '*-', t, impMpr, 'k*-',t, expEul, '*-')
legend({'exact', 'impEul', 'impMpr','expEul'})
```

Listing 2: l11_ivp2.m

```
clear, set(0,'DefaultFigureWindowStyle','docked')
N = 40; T = 2*pi; h = T/N; A = [0 1; -1 0];
```

```
expEul = nan(2, N+1); impEul = nan(2, N+1); impMpr = nan(2, N+1);
y0 = [1; 0]; expEul(:,1) = y0; impEul(:,1) = y0; impMpr(:,1) = y0;

for ii = 1:N
    expEul(:,ii+1) = (eye(2)+h*A)*expEul(:,ii);
    impEul(:,ii+1) = (eye(2)-h*A)\impEul(:,ii);
    impMpr(:,ii+1) = (eye(2)-h*A/2)\((eye(2)+h*A/2)*impMpr(:,ii));
end
figure(2); subplot(1,2,1);
plot(expEul(1,:), expEul(2,:), '*-', 'linewidth', 4);
plot(impEul(1,:), impEul(2,:), '*-', 'linewidth', 4);
plot(impMpr(1,:), impMpr(2,:), '*-', 'linewidth', 4);
axis equal
legend({'expEul', 'impEul', 'impMpr'})
subplot(1,2,2), t = linspace(0, T, N+1); Q = @(y) sqrt(sum(y.^2, 1));
plot(t, Q(expEul), '*-', t, Q(impEul), '*-', t, Q(impMpr), '*-','linewidth', 4)
title('Q(y(t))','FontSize',24);
fprintf('max(abs(Q(impMpr)-1)) = %e\n', max(abs(Q(impMpr)-1)))
```

Numerical Analysis Hilary Term 2026
Lecture 14–15: Runge–Kutta methods

Runge–Kutta methods: Runge–Kutta (RK) methods form a broad class of algorithms for the numerical solution of IVPs. The idea is to achieve a higher order of accuracy than 1 (as in Euler's method) by approximately finding the derivative f at 'intermediate values' (called *stages*) between x_n and x_{n+1} (and similarly $\mathbf{y}_n$ and $\mathbf{y}_{n+1}$).

The class includes both explicit and implicit schemes. When applications call for an integrator with some kind of stability or conservation property, there usually exists a suitable RK method. In particular, RK methods can be made arbitrarily high-order without the loss of stability.

Definition 1. *The family of* s-stage Runge–Kutta methods *is defined by*

$$\mathbf{\Psi}(x, \mathbf{y}, h, \mathbf{f}) = \mathbf{y} + h \sum_{i=1}^{s} b_i \mathbf{k}_i \,, \tag{1}$$

where the stages $\mathbf{k}_i$ *(recall that* $\mathbf{y} \in \mathbb{R}^d$*, and also* $\mathbf{k}_i \in \mathbb{R}^d$*) are the solutions of the coupled system of (generally nonlinear) equations*

$$\mathbf{k}_i := \mathbf{f}(x + c_i h, \mathbf{y} + h \sum_{j=1}^{s} a_{ij} \mathbf{k}_j) \,, \quad i = 1, \ldots, s \,. \tag{2}$$

The coefficients $\{c_i\}_{i=1}^{s}$ *are always given by*

$$c_i := \sum_{j=1}^{s} a_{ij} \quad i = 1, \ldots, s \,.$$

Definition 2. *The coefficients of a Runge–Kutta method are commonly summarized in a* Butcher tableau[1]

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^\top \end{array} \,.$$

Example 3. *The explicit Euler method, the implicit Euler method, and the implicit midpoint rule are Runge–Kutta methods. Their Butcher tables are*

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}\,, \quad \begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}\,, \quad \text{and} \quad \begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array}\,, \quad \textit{respectively.}$$

Example 4. *Let us derive two famous RK methods using Taylor expansions. Consider the following family of methods (where for simplicity we assume* $d = 1$*, i.e.,* $\mathbf{y}_n \in \mathbb{R}$ *is a scalar and write* y_n, k_i *etc):*

$$y_{n+1} = y_n + h(b_1 k_1 + b_2 k_2), \tag{3}$$

[1] The use of this tableau was introduced by J. C. Butcher in 1963 with the article *Coefficients for the study of Runge–Kutta integration processes.*

where

$$k_1 = f(x_n, y_n), \tag{4}$$
$$k_2 = f(x_n + c_2 h, y_n + a_{21} h k_1), \tag{5}$$

and where the parameters b_1, b_2, c_2 and a_{21} are to be determined.[2] *The method is consistent iff $b_1 + b_2 = 1$. Further conditions on the parameters are obtained by attempting to maximise the order of accuracy of the method. Indeed, expanding the consistency error τ_n of (3)–(5) in powers of h, after some algebra we obtain*

$$\begin{aligned}\tau_n &= \frac{1}{2}hy''(x_n) + \frac{1}{6}h^2 y'''(x_n)\\ &\quad -b_2 h[c_2 f_x + a_{21} f_y f] - b_2 h^2\left[\frac{1}{2}c_2^2 f_{xx} + c_2 a_{21} f_{xy} f + \frac{1}{2}a_{21}^2 f_{yy} f^2\right] + \mathcal{O}(h^3).\end{aligned}$$

Here we have used the abbreviations $f = f(x_n, y(x_n))$, $f_x = \frac{\partial f}{\partial x}(x_n, y(x_n))$, etc. On noting that $y'' = f_x + f_y f$, it follows that $\tau_n = \mathcal{O}(h^2)$ for any f provided that

$$c_2 b_2 = a_{21} b_2 = \frac{1}{2},$$

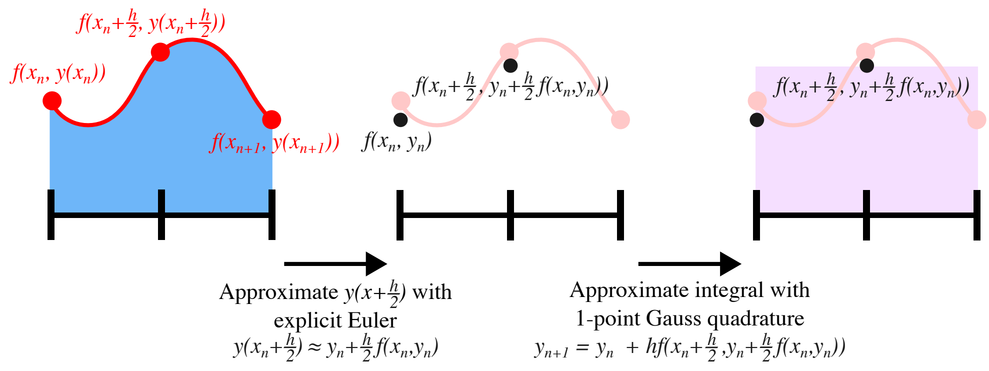
which implies that if $a_{21} = c_2$, $b_2 = 1/(2a_2)$ and $b_1 = 1 - 1/(2c_2)$ then the method is second-order accurate; while this still leaves one free parameter, c_2, it is easy to see that no choice of the parameters will make the method generally third-order accurate. There are two well-known examples of second-order explicit Runge–Kutta methods of the form (3), (5):

*a) **The modified Euler method:** In this case we take $c_2 = \frac{1}{2}$ to obtain*

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\,f\left(x_n + \frac{1}{2}h, \mathbf{y}_n + \frac{1}{2}hf(x_n, \mathbf{y}_n)\right);$$

$$\mathbf{\Psi}(x, \mathbf{y}, h, f) = x + hf\left(x + \frac{h}{2}, \mathbf{y} + \frac{h}{2}f(x, \mathbf{y})\right). \tag{6}$$

The figure below illustrates this method.


$f(x_n+\frac{h}{2}, y(x_n+\frac{h}{2}))$
$f(x_n, y(x_n))$
$f(x_{n+1}, y(x_{n+1}))$
$f(x_n+\frac{h}{2}, y_n+\frac{h}{2}f(x_n,y_n))$
$f(x_n, y_n)$
$f(x_n+\frac{h}{2}, y_n+\frac{h}{2}f(x_n,y_n))$
Approximate $y(x+\frac{h}{2})$ with explicit Euler
$y(x_n+\frac{h}{2}) \approx y_n+\frac{h}{2}f(x_n,y_n)$
Approximate integral with 1-point Gauss quadrature
$y_{n+1} = y_n + hf(x_n+\frac{h}{2}, y_n+\frac{h}{2}f(x_n,y_n))$

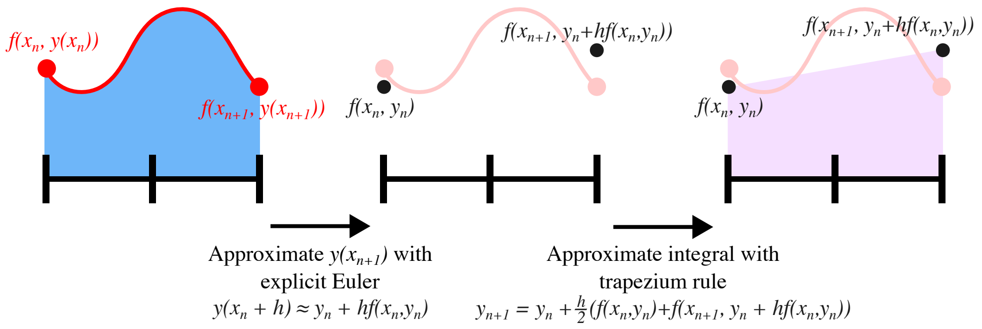
[2] We note in passing that Euler's explicit method is a member of this family of methods, corresponding to $b_1 = 1$ and $b_2 = 0$. However we are now seeking methods that are at least second-order accurate.

*b) **The improved Euler method:** This is obtained by choosing $c_2 = 1$, which gives*

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{1}{2}h\left[f(x_n, \mathbf{y}_n) + f(x_n + h, \mathbf{y}_n + hf(x_n, \mathbf{y}_n))\right].$$

$$\mathbf{\Psi}(x, \mathbf{y}, h, \mathbf{f}) = \mathbf{y} + \frac{h}{2}\mathbf{f}(x, \mathbf{y}) + \frac{h}{2}\mathbf{f}\left(x + h, \mathbf{y} + h\mathbf{f}(x, \mathbf{y})\right)\,. \tag{7}$$

Here's the corresponding figure.



$f(x_n, y(x_n))$
$f(x_{n+1}, y(x_{n+1}))$
$f(x_n, y_n)$
$f(x_{n+1}, y_n+hf(x_n,y_n))$
$f(x_n, y_n)$
$f(x_{n+1}, y_n+hf(x_n,y_n))$
Approximate $y(x_{n+1})$ with explicit Euler
$y(x_n + h) \approx y_n + hf(x_n,y_n)$
Approximate integral with trapezium rule
$y_{n+1} = y_n + \frac{h}{2}(f(x_n,y_n)+f(x_{n+1}, y_n + hf(x_n,y_n))$

For these two methods it is easily verified by Taylor series expansion that the consistency error is of the form, respectively,

$$\begin{aligned}\tau_n &= \frac{1}{6}h^2\left[f_y F_1 + \frac{1}{4}F_2\right] + \mathcal{O}(h^3),\\ \tau_n &= \frac{1}{6}h^2\left[f_y F_1 - \frac{1}{2}F_2\right] + \mathcal{O}(h^3),\end{aligned}$$

where

$$F_1 = f_x + ff_y \quad \textit{and} \quad F_2 = f_{xx} + 2ff_{xy} + f^2 f_{yy}.$$

The family (3)–(5) is referred to as the class of explicit two-stage explicit Runge–Kutta methods.

Their Butcher tables read

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ \hline & 0 & 1 \end{array} \quad \textit{and} \quad \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & 1/2 & 1/2 \end{array},$$

respectively.

Example 5. *Last but not least, RK4, a 4-stage 4th-order explicit Runge–Kutta method is a famous and very popular choice.*

$$\mathbf{\Psi}(x, \mathbf{y}, h, \mathbf{f}) = \mathbf{y} + \frac{h}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right),$$

where

$$\begin{aligned}\mathbf{k}_1 &= f(x,\mathbf{y}), \qquad \mathbf{k}_2 = f\left(x+\frac{1}{2}h, \mathbf{y}+\frac{1}{2}h\mathbf{k}_1\right),\\ \mathbf{k}_3 &= f\left(x+\frac{1}{2}h, \mathbf{y}+\frac{1}{2}h\mathbf{k}_2\right), \quad \mathbf{k}_4 = f(x+h, \mathbf{y}+h\mathbf{k}_3).\end{aligned}$$

Its Butcher table reads

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0\\ 1/2 & 1/2 & 0 & 0 & 0\\ 1/2 & 0 & 1/2 & 0 & 0\\ 1 & 0 & 0 & 1 & 0\\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array}.$$

We now turn to the convergence properties of RK methods. As mentioned at the end of the last lecture, convergence in the sense that $\mathbf{y}$ converges to the exact solution as $h \to 0$ holds under mild assumptions[3]. We shall thus look at two questions:

- What is the order of accuracy? (Which, as last lecture's theorem showed, is essentially the same as the consistency error)
- Stability: How small does h need to be?

We treat these questions separately.

Order of accuracy/consistency order of RK methods.

It is convenient at this point to restrict our attention to autonomous IVPs. (Recall that a nonautonomous system can always be made autonomous by increasing its dimension.) The process of making an IVP autonomous commutes with Runge–Kutta discretisation if and only if

$$\sum_{i=1}^{s} b_i = 1, \quad c_k = \sum_{j=1}^{s} a_{kj} \quad k = 1,\ldots,s\,,$$

which we assume henceforth. (In other words, if these conditions hold, the RK discretisation of the autonomised system is the autonomisation of the RK discretisation of the original problem.)

By computing appropriate Taylor expansions (as in Example 4), it is possible to derive algebraic conditions the Runge–Kutta coefficients must satisfy for the method to have a targeted consistency order. For example:

Lemma 6. *A Runge–Kutta method is consistent if and only if $\sum_{i=1}^{s} b_i = 1$. If the condition*

$$\sum_{i=1}^{s} b_i c_i = \frac{1}{2}$$

[3]Strictly speaking, it is not trivial to apply that theorem to general RK methods, which cannot be written easily, as writing a step as $\mathbf{\Psi}(s,\mathbf{y},h,\mathbf{f})$ is not trivial, especially for implicit methods. Nonetheless an analogous result still holds; we refer to Section II.3 of the book by Hairer, Norsett, and Wanner for details.

is also satisfied, the Runge–Kutta method has consistency order 2, and if the conditions

$$\sum_{i=1}^{s} b_i c_i^2 = \frac{1}{3} \quad \textit{and} \quad \sum_{i=1}^{s} b_i \sum_{j=1}^{s} a_{ij} c_j = \frac{1}{6}$$

are also satisfied, the Runge–Kutta method has consistency order 3.

Nonexaminable: The following table indicates the number of conditions as described above that a Runge–Kutta method must satisfy to have order p:

p	1	2	3	4	5	6	7	8	9	10	20
#conditions	1	2	4	8	17	37	85	200	486	1205	20247374

.

The number of stages of a Runge–Kutta method provides an interesting upper bound on its consistency order.

Lemma 7. *The (consistency) order p of an s-stage Runge–Kutta method is bounded by $p \leq 2s$. If the Runge–Kutta method is explicit, then $p \leq s$.*

To evolve a numerical solution from x_n to x_{n+1} with a Runge–Kutta method, one needs to compute the stages $\mathbf{k}_i$. If the Runge–Kutta method is explicit, these stages can be computed sequentially (and at a low-cost) starting from $\mathbf{k}_1$ (a Runge–Kutta method is explicit if $a_{ij} = 0$ whenever $j \geq i$, i.e. the matrix $\mathbf{A}$ is strictly lower-triangular). An example of this is the explicit Euler method. If $\mathbf{A}$ is lower-triangular (i.e. possibly $a_{ii} \neq 0$), then the scheme is said to be *diagonally-implicit*; one can compute the stages $\mathbf{k}_i$ sequentially, solving a sequence of nonlinear problems. The implicit Euler and implicit midpoint rules are examples of diagonally-implicit RK methods. Finally, if $\mathbf{A}$ enjoys neither of these structures, the RK method is said to be fully implicit; one must solve a large coupled nonlinear system for all stages simultaneously.

It is possible to construct Runge–Kutta methods that achieve maximal order. So-called *Butcher barriers* quantify the minimal amount of stages that an explicit Runge–Kutta method of order p requires (nonexaminable):

p	1	2	3	4	5	6	7	8	≥ 9
minimal value of s	1	2	3	4	6	7	9	11	$\geq p+3$

.

This implies that a Runge–Kutta method that has maximal order must be implicit.

Stability of Runge–Kutta methods

As discussed above, if the step size h is sufficiently small the computed solution with a RK method tends to the exact solution. This does not tell us how small h needs to be. Indeed, we have seen that numerical methods for IVPs may encounter stability issues in the last lecture. We now study the appropriate values of h; taking h too small means we'll do many $(O(h^{-1}))$ steps to find $y(X)$, which is undesirable. For simplicity, we only consider autonomous ODEs.

Definition 8. *A* fixed point *of* $\mathbf{y}' = \mathbf{f}(\mathbf{y})$ *is a point* $\mathbf{y}^*$ *such that* $\mathbf{f}(\mathbf{y}^*) = \mathbf{0}$. *A fixed point* $\mathbf{y}^*$ *is* asymptotically stable *(or attractive) if there exists a ball* $B_\delta(\mathbf{y}^*)$ *(of radius* $\delta > 0$ *and centered at* $\mathbf{y}^*$*) such that, whenever* $\mathbf{y}_0 \in B_\delta(\mathbf{y}^*)$, *the solution to* $\mathbf{y}' = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$ *satisfies* $\lim_{x\to\infty} \mathbf{y}(x) = \mathbf{y}^*$.

Theorem 9. *A fixed point $\mathbf{y}^*$ of an autonomous ODE is asymptotically stable if*

$$\sigma\left(\mathbf{Df}(\mathbf{y}^*)\right) \subset \mathbb{C}^- := \{z \in \mathbb{C} : \mathrm{Re} z < 0\},$$

where $\sigma\left(\mathbf{Df}(\mathbf{y}^*)\right)$ denotes the set of eigenvalues of the matrix $\mathbf{Df}(\mathbf{y}^*)$.

This theorem implies that, to study the asymptotic stability of $\mathbf{y}^*$, we can restrict our considerations to the linearised ODE $\mathbf{y}' = \mathbf{Df}(\mathbf{y}^*)(\mathbf{y} - \mathbf{y}^*)$, that is, we can restrict our attention to linear ODEs. To further simplify the analysis, we restrict our attention to a single eigenvalue, yielding the *Dahlquist test equation*

$$y' = zy\,, \quad y(0) = 1\,, \quad \text{and} \quad \mathrm{Re} z < 0\,. \tag{8}$$

Clearly, the solution of the Dahlquist test equation is $y(x) = \exp(zx)$, which satisfies $\lim_{x\to\infty} y(x) = 0$. Therefore, $y^* = 0$ is an attractive fixed point.

In what follows we ask the question: Is the fact that $y \to 0$ respected by the computed solution obtained by a RK method? This can be seen as a 'minimum requirement' for a method to be reasonable; if the exact solution is tending to 0 but the computed one isn't, that surely cannot be a good method. Conversely, if a method (with a given step size h) does satisfy $y_n \to 0$, then it tends to work well also for more general problems with similar h, in addition to respecting the asymptocic behavior of fixed points as discussed above; this is why we look specifically at the Dahlquist test equation (8).

The solution of the Dahlquist test equation obtained with a Runge–Kutta method has a special structure:

Definition 10. *Let $\mathbf{\Psi}$ be a Runge–Kutta method. The function*

$$S : \mathbb{C} \to \mathbb{C}\,, \quad z \mapsto S(z) := \mathbf{\Psi}(0, 1, 1, f : y \mapsto zy)\,,$$

is called the stability function *of $\mathbf{\Psi}$ (defined in (1)). To shorten the notation, we henceforth write $\mathbf{\Psi}(0, 1, 1, z)$ instead of $\mathbf{\Psi}(0, 1, 1, f : y \mapsto zy)$.*

Lemma 11. *If $\mathbf{\Psi}$ is a Runge–Kutta method, then $\mathbf{\Psi}(0, \ell, h, z) = \mathbf{\Psi}(0, 1, 1, zh)\ell$.*

Theorem 12. *Let $\{y_k\}_{k\in\mathbb{N}}$ be the Runge–Kutta solution to the Dahlquist test equation obtained with a time step $h > 0$. Then, $y_k = S(zh)^k$.*

Proof. By direct computation, we can see that

$$y_1 = \mathbf{\Psi}(0, 1, h, z) = \mathbf{\Psi}(0, 1, 1, zh) = S(zh)$$

and that

$$y_2 = \mathbf{\Psi}(0, y_1, h, z) = \mathbf{\Psi}(0, 1, 1, zh)y_1 = S(zh)y_1 = S(zh)^2\,.$$

Therefore, we conclude that $y_k = S(h\lambda)^k$. □

As discussed, it is desirable that the discrete solution $\{y_k\}_{k\in\mathbb{N}}$ satisfies $\lim_{k\to\infty} y_k = 0$, mimicking the behavior of the exact solution to the Dahlquist test equation. When this happens, we say that $\{y_k\}_{k\in\mathbb{N}}$ is *asymptotically stable*.

Definition 13. *The region in the complex plane*

$$S_{\mathbf{\Psi}} := \{z \in \mathbb{C} : |S(z)| < 1\}$$

is called the stability region *of the Runge–Kutta method. Clearly,* $\{y_k\}_{k\in\mathbb{N}}$ *is asymptotically stable if* $zh \in S_\Psi$.

It is not so difficult to see that the stability function of an explicit Runge–Kutta method is a polynomial, which implies that $S_{\mathbf{\Psi}}$ is bounded. Therefore, the numerical approximation computed with an explicit Runge–Kutta method cannot be asymptotically stable if the time step h is too large. This is what we saw in our numerical experiments in the last lecture. However, the stability function of an implicit Runge–Kutta method is a rational function, and hence may not suffer from this limitation.

Definition 14. *A Runge–Kutta method is said to be* A-stable[4] *if* $\mathbb{C}^- \subset S_{\mathbf{\Psi}}$.

A-stability guarantees that $\{y_k\}_{k\in\mathbb{N}}$ will eventually converge to zero. However, the decay can be very slow compared to that of the exact solution.

Example 15. *Let* $\{y_k\}$ *be the approximate solution to the Dalhquist test equation obtained with the implicit midpoint rule and a fixed step size* h*. By direct computation, we can see that stability function of the implicit midpoint rule is*

$$S(z) = \frac{1 + z/2}{1 - z/2}.$$

The exact solution converges exponentially to zero with rate $\mathrm{Re}z$*; the smaller (more negative) the* $\mathrm{Re}z$*, the quicker the convergence. On the other hand,* $\{y_k\}$ *is a geometric sequence with ratio* $S(zh)$*. This also converges to zero, but the more negative the* $\mathrm{Re}z$*, the closer* $|S(zh)|$ *to* 1*, and the slower the decay of* $\{y_k\}$*. This implies that, if* $\mathrm{Re}z \ll 0$*, the qualitative behavior of* $\{y_k\}$ *can be very different from the one of the exact solution.*

Therefore, if the initial value problem has a strongly attractive fixed point, it is advisable to further ensure that $\lim_{\mathrm{Re}z\to-\infty} |S(z)| = 0$.

Definition 16. *An A-stable method that further satisfies* $\lim_{\mathrm{Re}z\to-\infty} |S(z)| = 0$ *is said to be* L-stable.

One can verify that the implicit Euler method is L-stable, but it is not the only one.

The remainder is nonexaminable.

Construction of explicit/implicit RK methods To construct explicit Runge–Kutta methods, we start by recalling that the analytic solution of

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y})\,, \qquad \mathbf{y}(x_0) = \mathbf{y}_0\,, \tag{9}$$

[4]This concept was introduced by G. Dahlquist in 1963 with the article *A special stability problem for linear multistep methods.*

is given by the (implicit) formula

$$\mathbf{y}(x+h) = \mathbf{y}(x) + \int_x^{x+h} \mathbf{f}(\tau, \mathbf{y}(\tau))\,\mathrm{d}\tau = \mathbf{y}(x) + h\int_0^1 \mathbf{f}(x+h\tau, \mathbf{y}(x+h\tau))\,\mathrm{d}\tau\,.$$

Approximating the latter integral with a quadrature rule on $[0,1]$ with s nodes $c_1, \ldots, c_s$ and weights $b_1, \ldots, b_s$ returns

$$\mathbf{y}(x+h) \approx \mathbf{y}(x) + h\sum_{i=1}^{s} b_i \mathbf{f}(x+c_i h, \mathbf{y}(x+c_i h))\,. \tag{10}$$

Note that this approximation requires the values $\mathbf{y}(x+c_i h)$. To make the method explicit, we approximate the values $\mathbf{y}(x_0+c_i h)$ with explicit Runge–Kutta methods we already know. This way, we can construct s-stage explicit Runge–Kutta methods by induction.

Example 17. *If we choose the 1-point Gauss quadrature rule, that is,*

$$\mathbf{y}(x+h) \approx \mathbf{y}(x) + h\mathbf{f}(x+h/2, \mathbf{y}(x+h/2)) \tag{11}$$

and approximate $\mathbf{y}(x+h/2)$ with the explicit Euler method, the resulting scheme is (6).

Example 18. *If we use the trapezium rule, that is,*

$$\mathbf{y}(x+h) \approx \mathbf{y}(x) + \frac{h}{2}\mathbf{f}(x, \mathbf{y}(x)) + \frac{h}{2}\mathbf{f}(x+h, \mathbf{y}(x+h))\,,$$

and approximate $\mathbf{y}(x+h)$ with the explicit Euler method, the resulting scheme is (7).

A similar approach leads to the most famous explicit Runge–Kutta method *RK4*.

We have seen that s-stage explicit Runge–Kutta methods have at most order s. Next, we construct s-stage implicit Runge–Kutta methods whose order is at least s.

Definition 19. *Let $c_1, \ldots, c_s \in [0,1]$ be (pairwise distinct) collocation points. The corresponding collocation method is the one-step method defined by*

$$\mathbf{\Psi}(x, \mathbf{y}, h, \mathbf{f}) = \tilde{\mathbf{y}}(h)\,,$$

where $\tilde{\mathbf{y}}$ is the unique polynomial of degree s that satisfies

$$\tilde{\mathbf{y}}(0) = \mathbf{y} \quad \textit{and} \quad \tilde{\mathbf{y}}'(c_i h) = \mathbf{f}(x+c_i h, \tilde{\mathbf{y}}(c_i h))\,, \quad \textit{for } i = 1, \ldots, s\,. \tag{12}$$

Lemma 20. *Let Q be the highest-order quadrature rule on $[0,1]$ that can be constructed using the nodes $c_1, \ldots, c_s$, and let p_Q be its order (p_Q = 1 + the maximal degree of polynomials it integrates exactly). If $\mathbf{f}$ is sufficiently smooth and $h > 0$ is sufficiently small, the collocation method associated to $c_1, \ldots, c_s$ has order p_Q.*

Corollary 21. *If $\mathbf{f}$ is sufficiently smooth and $h > 0$ is sufficiently small, the order of the collocation method associated to $c_1, \ldots, c_s$ is at least s and at most $2s$ (Gauss quadrature).*

It is not obvious, but collocation methods are indeed Runge–Kutta methods.

Lemma 22. *Collocation methods are Runge–Kutta methods. Their coefficients are*

$$a_{ij} = \int_0^{c_i} L_j(\tau)\,\mathrm{d}\tau\,, \quad b_i = \int_0^1 L_i(\tau)\,\mathrm{d}\tau\,, \tag{13}$$

where $\{L_i\}_{i=1}^s$ *are the Lagrange polynomials associated to* $c_1, \dots, c_s$.

The Gauss collocation methods form a family of arbitrarily high-order A-stable methods whose stability region is exactly $\mathbb{C}^-$.

An example of a family of L-stable RK methods is the Gauss–Radau family. This is a family of collocation methods where the final quadrature point is fixed to $c_s = 1$ and the remaining points $c_1, \dots, c_{s-1}$ are chosen to obtain an associated quadrature rule of maximal order $2s-1$.

For more detailed discussions on RK methods, we refer to the books

- Süli and Mayer, “Introduction to Numerical Analysis”
- Hairer, Norsett, and Wanner, “Solving Ordinary Differential Equations”
- Butcher, “Numerical Methods for Ordinary Differential Equations”

Numerical Analysis Hilary Term 2026
Lecture 16: Multistep methods

Linear multi-step methods

Runge-Kutta methods deliver an approximate solution to

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \qquad \mathbf{y}(x_0) = \mathbf{y}_0, \tag{1}$$

but tacitly assume that it is possible to evaluate the right-hand side $\mathbf{f}(x, \mathbf{y})$ anywhere, and use a lot of such function evaluations. Instead, linear multi-step methods are more parsimonious, requiring values of $\mathbf{f}$ at grid points only. That is, these methods re-use quantities that have already been computed, and thus available without extra computation.

Definition 1. *Let $X > x_0$ be a final time, $N, k \in \mathbb{N}$, $N \geq k$, $h := (X - x_0)/N$, and $x_n := x_0 + hn$. A* linear k-step method *is an iterative method that computes the approximation $\mathbf{y}_{n+k}$ to $\mathbf{y}(x_{n+k})$ by solving*

$$\sum_{j=0}^{k} \alpha_j \mathbf{y}_{n+j} = h \sum_{j=0}^{k} \beta_j \mathbf{f}(x_{n+j}, \mathbf{y}_{n+j}), \tag{2}$$

where $\{\alpha_j\}_{j=0}^k$ and $\{\beta_j\}_{j=0}^k$ are real coefficients. To avoid degenerate cases, we assume that $\alpha_k \neq 0$ and that $\alpha_0^2 + \beta_0^2 \neq 0$.

Note that if $\beta_k = 0$, the method is explicit.

Non-examinable: It is also possible to construct multi-step methods on nonequidistant grids, and good timestepping software does so for you.

In the same way Runge-Kutta methods are summarized with Butcher tables, linear multi-step methods can be summarized with two polynomials.

Definition 2. *For the k-step method defined by (2),*

$$\rho(z) = \sum_{j=0}^{k} \alpha_j z^j \quad \text{and} \quad \sigma(z) = \sum_{j=0}^{k} \beta_j z^j \tag{3}$$

are called the first *and* second characteristic polynomials.

Example 3. *A simple linear 3-step method can be constructed using Simpson's quadrature rule. Indeed,*

$$\begin{aligned} \mathbf{y}(x_{n+1}) &= \mathbf{y}(x_{n-1}) + \textstyle\int_{x_{n-1}}^{x_{n+1}} \mathbf{f}(x, \mathbf{y}(x))\,\mathrm{d}x \\ &\approx \mathbf{y}(x_{n-1}) + \tfrac{2h}{6}\left(\mathbf{f}(x_{n-1}, \mathbf{y}(x_{n-1})) + 4\mathbf{f}(x_n, \mathbf{y}(x_n)) + \mathbf{f}(x_{n+1}, \mathbf{y}(x_{n+1}))\right). \end{aligned}$$

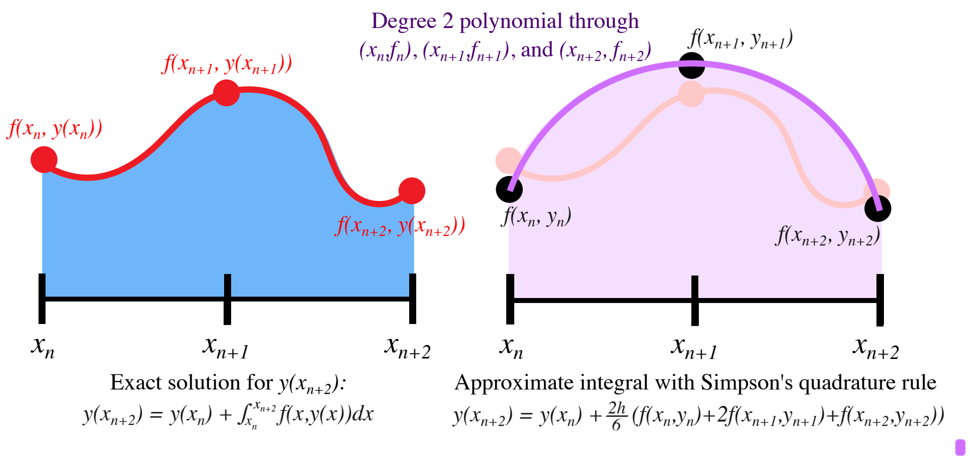
This motivates the following linear 2-step method

$$\mathbf{y}_{n+2} - \mathbf{y}_n = h\left(\frac{2}{6}\mathbf{f}(x_n, \mathbf{y}_n) + \frac{8}{6}\mathbf{f}(x_{n+1}, \mathbf{y}_{n+1}) + \frac{2}{6}\mathbf{f}(x_{n+2}, \mathbf{y}_{n+2})\right) \tag{4}$$

Its first and second characteristic polynomials are

$$\rho(z) = z^2 - 1 \quad \text{and} \quad \sigma(z) = \frac{2}{6}(z^2 + 4z + 1). \tag{5}$$

Here's an illustration.



$f(x_n, y(x_n))$
$f(x_{n+1}, y(x_{n+1}))$
$f(x_{n+2}, y(x_{n+2}))$
Degree 2 polynomial through
(x_n, f_n), (x_{n+1}, f_{n+1}), and (x_{n+2}, f_{n+2})
$f(x_{n+1}, y_{n+1})$
$f(x_n, y_n)$
$f(x_{n+2}, y_{n+2})$
x_n x_{n+1} x_{n+2}
x_n x_{n+1} x_{n+2}
Exact solution for $y(x_{n+2})$:
$y(x_{n+2}) = y(x_n) + \int_{x_n}^{x_{n+2}} f(x,y(x))dx$
Approximate integral with Simpson's quadrature rule
$y(x_{n+2}) = y(x_n) + \frac{2h}{6}(f(x_n,y_n)+2f(x_{n+1},y_{n+1})+f(x_{n+2},y_{n+2}))$

Example 4. *Two important families of multistep methods are* Adams-Moulton methods *and the* Adams-Bashforth methods. *The three-step Adams–Moulton method is (an implicit method)*

$$\mathbf{y}_{n+3} = \mathbf{y}_{n+2} + \frac{1}{24}h\left(9\mathbf{f}_{n+3} + 19\mathbf{f}_{n+2} - 5\mathbf{f}_{n+1} - 9\mathbf{f}_n\right), \tag{6}$$

and the four-step Adams-Bashforth method is (explicit)

$$\mathbf{y}_{n+4} = \mathbf{y}_{n+3} + \frac{1}{24}h\left(55\mathbf{f}_{n+3} - 59\mathbf{f}_{n+2} + 37\mathbf{f}_{n+1} - 9\mathbf{f}_n\right) \tag{7}$$

(If you're curious how they can be derived, see the non-examinable material at the end).

The methods listed above are all good ('convergent') methods, in that they compute solutions that converge to the exact ones as the step size $h \to 0$. Now let us look at

$$\mathbf{y}_{n+2} = -4\mathbf{y}_{n+1} + 5\mathbf{y}_n + h(4\mathbf{f}(x_{n+1}, \mathbf{y}_{n+1}) - 2\mathbf{f}(x_n, \mathbf{y}_n)) \tag{8}$$

and

$$\mathbf{y}_{n+3} = -2\mathbf{y}_{n+2} + \mathbf{y}_{n+1} + 2\mathbf{y}_n + h(2\mathbf{f}(x_{n+3}, \mathbf{y}_{n+3}) + \mathbf{f}(x_{n+2}, \mathbf{y}_{n+2}) + 3\mathbf{f}(x_n, \mathbf{y}_n)). \tag{9}$$

These methods are consistent— method (8) has consistency order 3, and (9) has 2 (see below for the precise definition). However, these methods are catastrophically bad—as $h \to 0$, the error does not decrease, it grows unboundedly! Our next goal is to understand why the methods in Examples 3 and 4 converge, while methods like (8), (9) do not.

Consistency+Zero-Stabiliy⇒Convergence To gain insight, let us examine what happens to $\mathbf{y}_n$ as $h \to 0$ in (9). For tiny h and $n = O(1)$, the recursion effectively yields $\mathbf{y}_{n+3} + 2\mathbf{y}_{n+2} - \mathbf{y}_{n+1} - 2\mathbf{y}_n = 0$. How does $\mathbf{y}_n$ behave? This is a difference equation, and one way to solve it is as follows: (as in "Note" in lecture 12) we rewrite (assuming for simplicity $\mathbf{y}_n$ are scalars)

$$\begin{bmatrix} \mathbf{y}_{n+3} \\ \mathbf{y}_{n+2} \\ \mathbf{y}_{n+1} \end{bmatrix} = \begin{bmatrix} -2 & 1 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{y}_{n+2} \\ \mathbf{y}_{n+1} \\ \mathbf{y}_n \end{bmatrix} =: A \begin{bmatrix} \mathbf{y}_{n+2} \\ \mathbf{y}_{n+1} \\ \mathbf{y}_n \end{bmatrix}.$$

Since this holds for all n, it follows immediately that (noting that the eigenvalues of A are $-2, 1, -1$)

$$\begin{bmatrix} \mathbf{y}_{n+2} \\ \mathbf{y}_{n+1} \\ \mathbf{y}_n \end{bmatrix} = A^n \begin{bmatrix} \mathbf{y}_2 \\ \mathbf{y}_1 \\ \mathbf{y}_0 \end{bmatrix} = X \begin{bmatrix} (-2)^n & & \\ & 1^n & \\ & & (-1)^n \end{bmatrix} X^{-1} \begin{bmatrix} \mathbf{y}_2 \\ \mathbf{y}_1 \\ \mathbf{y}_0 \end{bmatrix} \tag{10}$$

for an invertible matrix X of A's eigenvectors. Now since $(-2)^n$ blows up to $\pm\infty$, so does $\mathbf{y}_n$; regardless of f! Finally, notice that the matrix $A = \begin{bmatrix} -2 & 1 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ has the same structure as the companion matrix from lecture 8![1] Thus the eigenvalues of A are equal to the roots of the first characteristic polynomial $\rho(z) = z^3 + 2y^2 - y - 2 = 0$.

More generally, given any multistep method, any root of $\rho(z)$ outside the unit disk will lead to similar blowups. What about roots on the unit circle? The answer is that they are fine if they are simple roots; however multiple roots on the unit circle leads to divergence[2]. This motivates the following definition:

Definition 5. *A linear k-step method satisfies the* root condition *if all roots of its first characteristic polynomial $\rho(z)$ lie inside the closed unit disc, and every root that lies on the unit circle is simple.*

The root condition is thus a necessary condition for a multistep method to be convergent. The remarkable result by Dahlquist is that this property is also sufficient, as long as the method is consistent, which is a straightforward condition. That is, "Consistency + root condition ⇒ Convergence". Finally, the root condition can be shown to be identical to the so-called *zero-stability* (which we define and explain in the non-examinable appendix[3]). Thus Dahlquist's theorem is colloquially often known as

"Consistency + (Zero-)Stability ⇒ Convergence".

This result has sometimes been called the Fundamental Theorem of Numerical Analysis. It explains why the method (9) does not converge (the roots of $\rho(z)$ are -5 and 1, violating the root condition), whereas the other ones are convergent (by checking consistency, and that the root condition is satisfied).

To state Dahlquist's theorem precisely, let us define consistency properly for multistep methods.

[1] This is why we're solving the difference equation this way. You may well have seen other ways to solve it.

[2] This is because companion matrices cannot have eigenvalues of geometric multiplicity 2 or more, because $A - \lambda I$ has rank deficiency at most one. Put another way, if there is a multiple eigenvalue, A must have a Jordan block of the form $J_\lambda = \begin{bmatrix} \lambda & 1 \\ & \lambda \end{bmatrix}$. If $|\lambda| = 1$, then $\|J_\lambda^n\|_2 \to \infty$ as $n \to \infty$.

[3] Note that the word 'stability' in zero-stability is used in a rather different way than the stability we've been discussing, as in A-stability and L-stability, which relate to the question 'how small is small enough for h'. If confused, think of zero-stability as an exceptional use of the word.

Definition 6. *The* consistency error *of a linear k-step method with $\sigma(1) \neq 0$ is*

$$\boldsymbol{\tau}(h) = \frac{\sum_{j=0}^{k} \alpha_j \mathbf{y}(x_j) - h \sum_{j=0}^{k} \beta_j \mathbf{y}'(x_j)}{h \sum_{j=0}^{k} \beta_j}\,, \tag{11}$$

where $\mathbf{y}$ is a smooth function. A linear multi-step method has (consistency) order *p if $\boldsymbol{\tau}(h) = O(h^p)$.*

As in Runge-Kutta methods, The consistency error here is defined to model the local error of the method; for example it reduces to that for one-step methods when $k = 1$ and $\alpha_k = 1$.

By a simple Taylor expansion of $\mathbf{y}$, we can obtain the following theorem.

Theorem 7. *A linear multi-step method has consistency order p if and only if $\sigma(1) \neq 0$ and*

$$\sum_{j=0}^{k} \alpha_j = 0 \quad \textit{and} \quad \sum_{j=0}^{k} \alpha_j j^q = q \sum_{j=0}^{k} \beta_j j^{q-1} \quad \textit{for} \quad q = 1, \ldots, p\,. \tag{12}$$

A multi-step method is said to be *consistent* if these conditions are satisfied at least for $p = 1$.

From the above theorem we see that a linear multi-step method is consistent iff

$$\rho(1) = 0 \quad \text{and} \quad \rho'(1) = \sigma(1) \neq 0. \tag{13}$$

In general, these conditions can be reformulated more elegantly (nonexaminable): Equation (12) is equivalent to $\rho(e^h) - h\sigma(e^h) = O(h^{p+1})$.

To discuss convergence precisely for linear k-step methods, we need to specify some criteria about the choice of the starting conditions. Below, we say that a set of starting conditions $\mathbf{y}_i = \boldsymbol{\eta}_i(h)$, $i = 0, \ldots, k-1$ is consistent with the initial value $\mathbf{y}_0$ if $\boldsymbol{\eta}_s(h) \to \mathbf{y}_0$ as $h \to 0$ for every $s = 0, \ldots, k-1$.

We are now ready to state Dahlquist's Equivalence Theorem.

Theorem 8 (Dahlquist's Equivalence Theorem). *For a consistent linear k-step method with consistent starting values, the root condition (=zero-stability) is necessary and sufficient for convergence, that is, $\lim_{h\to 0} \mathbf{y}_N = \mathbf{y}(X)$ (with $N = (X - x_0)/h$).*

Moreover, if $\boldsymbol{\tau}(h) = O(h^p)$ and $\|\mathbf{y}(x_s) - \boldsymbol{\eta}_s(h)\| = O(h^p)$ for $s = 0, \ldots, k-1$, then $\max_{0 \le n \le N} \|\mathbf{y}(x_n) - \mathbf{y}_n\| = O(h^p)$.

The proof is long and non-examinable, but you should understand the statement.

Stability of linear multi-step methods Now that we understand convergence of multistep methods, we turn to stability[4]. Similar to one-step methods, stability is investigated by applying a linear multi-step method to the Dahlquist test equation $y' = zy$, $z \in \mathbb{C}$, $y(0) = 1$, and $h = 1$. Recall that the solution to this ODE is $y(x) = \exp(zx)$, that $|y(x)| \to 0$ as $t \to \infty$ whenever $\mathrm{Re}(z) < 0$, and that we call its numerical approximation $\{y_n\}_{n\in\mathbb{N}}$ (asymptotically) stable if $y_n \to 0$ as $n \to \infty$ when $\mathrm{Re}(z) < 0$.

[4]Not to be confused with zero-stability, which was crucial for convergence; here we mean the analogues of A-stability, L-stability, that is, related to the question of 'how small does h need to be?'

Our goal is to investigate when the sequence $\{y_n\}_{n\in\mathbb{N}}$ computed with a linear k-step method is stable. First of all, note that the n-th iterate y_n satisfies

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = \sum_{j=0}^{k} \beta_j z y_{n+j}\,, \quad \text{or equivalently,} \quad \sum_{j=0}^{k} (\alpha_j - z\beta_j) y_{n+j} = 0\,. \tag{14}$$

By an argument similar to that surrounding (10), we obtain

$$y_n = p_1(n) r_1^n + \ldots + p_\ell(n) r_\ell^n\,, \tag{15}$$

where the r_js are the roots of the polynomial $\pi(x) = \sum_{j=0}^{k}(\alpha_j - z\beta_j)x^j$, and the $p_j(n)$s are polynomials of degree $m_j - 1$, where m_j is the multiplicity of r_j.

With (15), we can fully analyze the asymptotic behavior of $\{y_n\}_{n\in\mathbb{N}}$. Indeed:

- if $\pi(x)$ has a zero r_j outside the unit disc, than y_n grows as $|r_j|^n$,
- if an r_j is on the unit circle and has multiplicity $m_j > 1$, then $y_n \sim n^{m_j - 1}$,
- otherwise, $y_n \to 0$ geometrically as $n \to \infty$.

This computation shows that the polynomial π plays a crucial role in this stability analysis. Therefore, similarly to one-step methods, we introduce the following definitions.

Definition 9. *The* stability polynomial *of a linear k-step method is*

$$\pi(x) = \pi(x; z) := \sum_{j=0}^{k} (\alpha_j - z\beta_j) x^j = \rho(x) - z\sigma(x)\,. \tag{16}$$

The stability domain *of a linear multistep method is*

$$S := \{z \in \mathbb{C} : \textit{if } \pi(x; z) = 0, \textit{ then } |x| \le 1\textit{; multiple zeros satisfy } |x| < 1\}\,. \tag{17}$$

Note that $0 \in S$ if the method is zero-stable (as $\pi(x; 0) = \rho(x)$).

Dahlquist's second barrier theorem places sharp limits on the stability domains of linear multi-step methods.

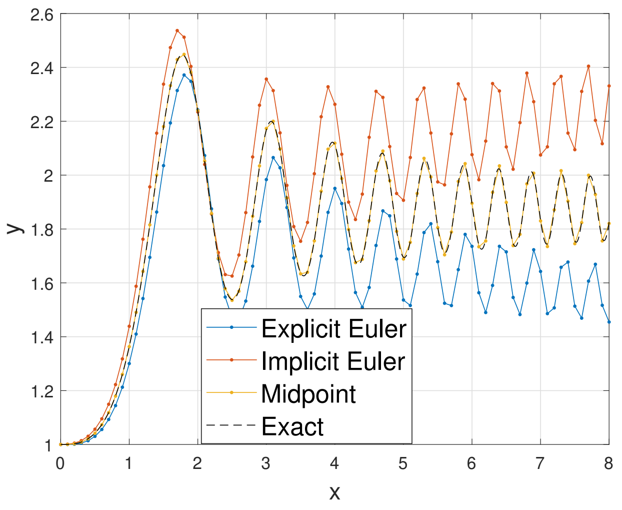
Theorem 10 (Dahlquist's second barrier). *An A-stable linear multi-step method must be implicit and of order $p \le 2$. The trapezium rule is the second-order A-stable linear multi-step method with the smallest error constant.*

The proof is long and omitted, and non-examinable. It is possible to break the Dahlquist barrier by hybridising between multi-stage (Runge-Kutta) and multi-step methods. Such methods are called *general linear methods*[5].

Example 11. *We conclude with an example illustrating some of the results. Consider the scalar IVP $y' = \sin(x^2)y$, $y(0) = 1$. We use explicit Euler, implicit Euler, implicit midpoint, explicit 4-stage Runge-Kutta, and 4th order Adam-Bashforth method to solve it.*

Here are the solutions.

[5]See *General linear methods*, J. C. Butcher, Acta Numerica (2006).



We now look at the error $y(x_n) - y_n$, shown in Figure 1. There we also examine the unstable multistep method (8), which is not zero-stable; we thus expect it to not converge. In fact the solution blows up and the error diverges to ∞—it hardly gets any worse than that!

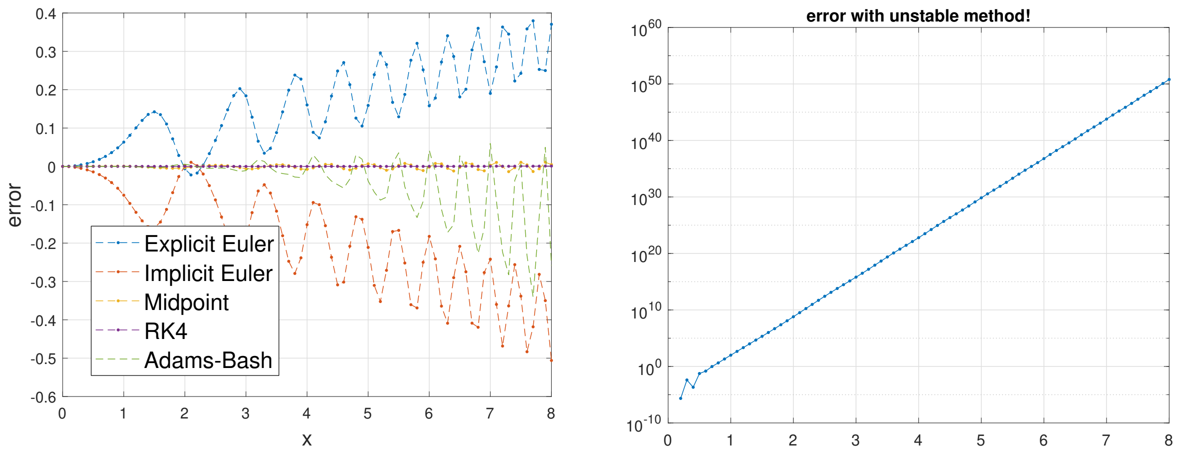
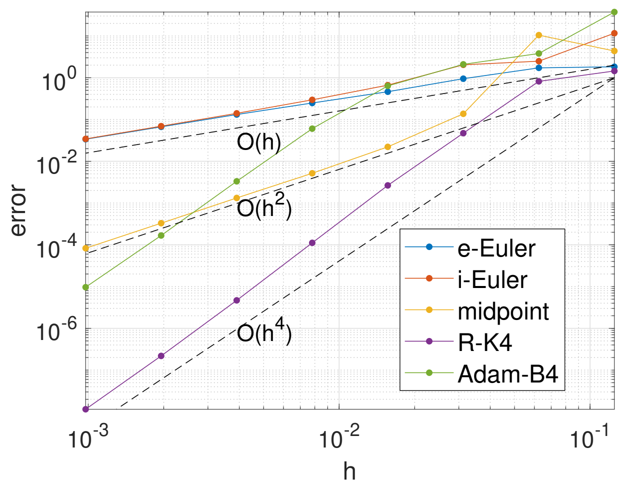


Figure 1: Errors with stable methods (left) and an unstable method (8)

Finally, we can vary the step size h and examine the convergence as $h \to 0$. Higher-order methods should have better accuracy especially for small h. We confirm this in the figure (note the loglog scale).



(MATLAB code is lec16_demo.m)

Summary of lectures 12–15 Here is a summary and key takeaways from the last part of the course on numerical solution of IVPs:

- Euler's method; explicit and implicit methods. Implicit methods are more expensive, but sometimes achieve higher order, and more importantly, more stable (can be A-stable, L-stable).
- Consistency error (local error as $h \to 0$), order of accuracy (global error) and convergence (does computed $\mathbf{y}$ tend to the exact solution as $h \to 0$?).
- Runge-Kutta methods, which achieve higher order of accuracy by evaluating $f(x, y)$ at 'intermediate points'. Convergence (relatively straightforward from the thereom in Lecture 12) and stability (A-stable, L-stable).
- Multistep methods, which achieve higher order of accuracy by using previously computed solutions. Convergence (requires zero-stability) and stability (A-stable).

This concludes this course—for further courses related to numerical analysis, check out e.g.

- Numerical Solution of Partial Differential Equations (Part B)
- Approximation of Functions (Part C, offered –2023)
- Numerical Linear Algebra (Part C)
- Finite Element Method for PDEs (Part C)
- Continuous Optimisation (Part C)

The remainder is nonexaminable.

Constructing multistep methods There is a formal calculus that can be used to construct families of multi-step methods.

Definition 12. *For a fixed small $h > 0$, we define:*

- *the shift operator $E : \mathbf{y}(x) \mapsto \mathbf{y}(x+h)$,*
- *its inverse $E^{-1} : \mathbf{y}(x) \mapsto \mathbf{y}(x-h)$,*
- *the difference operator $\Delta : \mathbf{y}(x) \mapsto \mathbf{y}(x) - \mathbf{y}(x-h)$,*
- *the identity operator $\mathbf{I} : \mathbf{y}(x) \mapsto \mathbf{y}(x)$,*
- *and the differential operator $D : \mathbf{y}(x) \mapsto \mathbf{y}'(x)$.*

Lemma 13. *Suppose that $\mathbf{y}(x)$ is analytic (hence infinitely differentiable) at x. Then formally, $hD = -\log(\mathbf{I} - \Delta)$.*

Proof. First, using Taylor expansion, we can show that

$$\begin{aligned} E\mathbf{y}(x) &= \mathbf{y}(x) + h\mathbf{y}'(x) + \tfrac{h^2}{2}\mathbf{y}''(x) + \dots \\ &= \mathbf{y}(x) + hD\mathbf{y}(x) + \tfrac{h^2}{2}D^2\mathbf{y}(x) + \dots = \exp(hD)\mathbf{y}(x)\,, \end{aligned}$$

and thus, $E = \exp(hD)$. This implies that $hD = \log(E)$.

Then, using the definition, we see that $E^{-1} = \mathbf{I} - \Delta$, and thus $E = (\mathbf{I} - \Delta)^{-1}$.

Therefore, $hD = \log(E) = \log((\mathbf{I} - \Delta)^{-1}) = -\log(\mathbf{I} - \Delta)$. □

Example 14. *We can construct a multi-step method using the previous lemma. Indeed, by Taylor expansion of the logarithm* $\log(1-x) = -\sum_{i=1}^{\infty} x^i/i$,

$$hD = -\log(\mathbf{I} - \Delta) = \left(\Delta + \frac{1}{2}\Delta^2 + \frac{1}{3}\Delta^3 + \dots\right)\,, \tag{18}$$

and thus

$$h\mathbf{f}(x_n, \mathbf{y}(x_n)) = \left(\Delta + \frac{1}{2}\Delta^2 + \frac{1}{3}\Delta^3 + \dots\right)\mathbf{y}(x_n)\,. \tag{19}$$

To construct a family of multi-step methods, we truncate the infinite series at different orders and replace $\mathbf{y}(x_n)$ with $\mathbf{y}_n$. These methods are called backward differentiation formulas, *and their simplest instances are*

$$\begin{aligned} \mathbf{y}_n - \mathbf{y}_{n-1} &= h\mathbf{f}(x_n, \mathbf{y}_n)\,, \quad \textit{(implicit Euler)} \\ \tfrac{3}{2}\mathbf{y}_n - 2\mathbf{y}_{n-1} + \tfrac{1}{2}\mathbf{y}_{n-2} &= h\mathbf{f}(x_n, \mathbf{y}_n)\,, \\ \tfrac{11}{6}\mathbf{y}_n - 3\mathbf{y}_{n-1} + \tfrac{3}{2}\mathbf{y}_{n-2} - \tfrac{1}{3}\mathbf{y}_{n-3} &= h\mathbf{f}(x_n, \mathbf{y}_n)\,. \end{aligned}$$

Example 15. *Explicit Euler's method arises from truncating the series*

$$hD = \left(\Delta - \frac{1}{2}\Delta^2 - \frac{1}{6}\Delta^3 + \dots\right)E\,, \tag{20}$$

which can be derived similarly.

Using the formal equalities

$$\begin{aligned} E\Delta &= h\left(\mathbf{I} - \tfrac{1}{2}\Delta - \tfrac{1}{12}\Delta^2 - \tfrac{1}{24}\Delta^3 - \tfrac{19}{720}\Delta^4 + \dots\right)D\,, \\ E\Delta &= h\left(\mathbf{I} + \tfrac{1}{2}\Delta + \tfrac{5}{12}\Delta^2 + \tfrac{3}{8}\Delta^3 + \tfrac{251}{720}\Delta^4 + \dots\right)D\,, \end{aligned}$$

we can derive the Adams-Moulton methods (6) and Adams-Bashforth methods (7).

More on zero-stability To compute $\mathbf{y}_k$ with a linear k-step method, we need the values $\mathbf{y}_0, \dots, \mathbf{y}_{k-1}$. These (except $\mathbf{y}_0$) must be approximated with either a one-step (e.g. Runge-Kutta) method or another multi-step method that uses fewer steps. At any rate, they will contain numerical errors. Clearly, a meaningful multistep method should be robust with respect to small perturbations of these initial values.

Definition 16. *A linear k-step method is said to be* zero-stable *if there is a constant $K > 0$ such that for every $N \in \mathbb{N}$ sufficiently large and for any two different sets of initial data $\mathbf{y}_0, \dots, \mathbf{y}_{k-1}$ and $\tilde{\mathbf{y}}_0, \dots, \tilde{\mathbf{y}}_{k-1}$, the two sequences $\{\mathbf{y}_n\}_{n=0}^N$ and $\{\tilde{\mathbf{y}}_n\}_{n=0}^N$ that stem from the linear k-step method with $h = (X - x_0)/N$ satisfy*

$$\max_{0 \le n \le N} \|\mathbf{y}_n - \tilde{\mathbf{y}}_n\| \le K \max_{j \le k-1} \|\mathbf{y}_j - \tilde{\mathbf{y}}_j\| . \tag{21}$$

Zero-stability of a k-step method can be verified algebraically with the following property, which is known as the *root condition.*

Definition 17. *A linear k-step method satisfies the* root condition *if all zeros of its first characteristic polynomial $\rho(z)$ lie inside the closed unit disc, and every zero that lies on the unit circle is simple.*

Theorem 18. *A linear multi-step method is zero-stable for any ODE $\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y})$ with Lipschitz right-hand side, if and only if the linear multi-step method satisfies the root condition.*

This theorem implies that zero-stability of a multi-step method can be determined by merely considering its behavior when applied to the trivial differential equation $y' = 0$; it is for this reason that it is called *zero*-stability.

The first Dahlquist barrier theorem For Runge–Kutta methods, we showed that one can construct s-stage methods of order $2s$. Unfortunately, it is not possible to construct linear k-step methods of order $2k$ without violating the zero-stability requirement (this result is non-examinable).

Theorem 19 (The first Dahlquist-barrier). *The order p of a zero-stable linear k-step method satisfies*

- *$p \le k + 2$ if k is even,*
- *$p \le k + 1$ if k is odd,*
- *$p \le k$ if $\beta_k/\alpha_k \le 0$ (in particular if the method is explicit).*