Lecture 1: Introduction. (Monday, August 24)

1.1 Basics of error analysis.

1.1.1 Generic form of a problem.

Most problems can be cast into the form

$$F(x, d) = 0, \tag{1.1}$$

where

- F is a functional relation,
- x are the variables,
- d is the data.

Both x and d could be numbers, vectors, or even functions (where (1.1) could be an ODE or a PDE). The distinction between the variables and the data is usually one depending on context and physics (e.g. data could be coefficients such as mass, local speed of sound or light, etc; variables would usually be position, velocity, electric or magnetic fields...)
A **direct problem** is one where F and the data d are given, and the goal is to find the variables x.
Conversely, an **inverse problem** is one where F and x are given, and the goal is to recover the data d.

Examples.

1. Ohm's law:
$$U = RI \qquad \leftrightarrow U - RI = 0$$
Knowing the data U and R allows to predict the current intensity I.

2. Mass-spring system:
$$m\ddot{u} + ku - F(t) = 0$$
Knowing the data m, k, $F(t)$ allows to predict the motion $u(t)$.

3. Solve a polynomial equation:
$$x^5 - x^2 + a = 0$$
Note how the first two problems can be solved explicitely with a little work, however the last one cannot be solved explicitely - there is no formula for the solution $x(a)$.

Definition 1.1. *A problem of the form* (1.1) *is* ***well-posed****, or* ***stable****, if it admits a unique solution x depending continuously on the data d.*

A central question for mathematicians, especially in analysis, the determination of whether a problem is well-posed may sometimes seem fruitless - after all, it does not imply any actionable knowledge about the solution! However, in practice a problem which is ill-posed indicates a pathology in the model which numerical methods will not cure, and at worst leads to wrong predictions.

1.1.2 Absolute and relative errors.

Let x be some number, vector, or object in a vector space V equipped with a norm $\|\cdot\|$, and $\hat{x}$ some approximation of x in V.

Definition 1.2.
- *The error is defined as the quantity* $\delta x = \hat{x} - x$.
- *The absolute error is the positive number*

$$\|\delta x\| = \|\hat{x} - x\|. \tag{1.2}$$

- *If* $x \neq 0$*, the relative error is the strictly positive number*

$$\frac{\|\delta x\|}{\|x\|} = \frac{\|\hat{x} - x\|}{\|x\|}. \tag{1.3}$$

Example. Take the numbers $p = 0.2 \times 10^{-4}$, $\hat{p} = 0.15 \times 10^{-4}$. Then

$$\delta x = -5 \times 10^{-6}, \qquad \|\delta x\| = 5 \times 10^{-6}, \qquad \frac{\|\delta x\|}{\|x\|} = 0.25$$

Notice how the absolute error is quite small, yet the relative error is large! The converse can also happen.

1.2 Machine representation of Numbers: Floating-point systems.

1.2.1 Decimal system.

A standardized way to write a number, using the usual decimal notation:

$$x = (-1)^s \times (0.d_1 d_2 \dots d_t \dots) \times 10^e,$$

where

- $s \in \{0, 1\}$ determines the **sign** of x,
- The significant **digits** $d_1, \dots$ are such that the first $d_1 \in \{1, \dots, 9\}$ cannot be zero, and $d_t \in \{0, \dots, 9\}$ for $t \geqslant 2$.
- The **exponent** $e \in \mathbb{Z}$ is a signed integer.

Example. $\sqrt{2} = (-1)^0 \times (.14142135\dots) \times 10^1$.

1.2.2 General floating-point system:

The above notation can be generalized in a straightforward way to accomodate for a general basis $\beta \geqslant 2$: we write

$$x = (-1)^s \times (0.d_1 d_2 \dots d_t \dots) \times \beta^e \qquad = (-1)^s \times \sum_{i=1}^{\infty} d_i \beta^{e-i},$$

where

- $\beta \geqslant 2$ is an integer: the **base**.

1.2.3 Finite number systems

Computers must allow a finite amount of memory, and cannot store infinite numbers of digits (e.g. π). Only a fixed number of digits can be stored; usually the base used is 2 (except in that Russian ternary computer). The standard format is then

$$x = \underbrace{(-1)^s}_{\text{sign}} \times (0.\underbrace{d_1 d_2 \dots d_t}_{\text{mantissa}}) \times \underbrace{\beta}_{\text{base}}^{\overbrace{e}^{\text{exponent}}} = (-1)^s \times \sum_{i=1}^{t} d_i \beta^{e-i}, \tag{1.4}$$

where

- t is the **precision**, the number of digits stored;
- e, the exponent, must be in the interval $L \leqslant e \leqslant U$.

Each combination of sign, digits and exponent corresponds to a unique real number; all such combinations, plus zero, form a finite number system $\mathbb{F}$ of real numbers, determined by β, t, L, and U.

Remark 1.3. *Note that the number zero needs a special representation in this system, because the constraint $d_1 \neq 0$ prevents a number x of the form (1.4) to take the value 0.*

Cardinality. The set $\mathbb{F}(\beta, t, L, U) \subset \mathbb{R}$ is finite, and the number of its elements is

$$2 \times (\beta - 1) \times \beta^{t-1} \times (U - L + 1) + 1.$$

Largest positive number. The largest element in $\mathbb{F}(\beta, t, L, U)$ is

$$M = +\sum_{i}^{t} (\beta - 1) \times \beta^{U-i} = (\beta - 1)\beta^{U-1} \sum_{i=0}^{t-1} (1/\beta)^i = \frac{\beta - 1}{\beta} \frac{1 - (1/\beta)^t}{1 - 1/\beta} = (1 - \beta^{-t})\beta^U.$$

Smallest positive number. The smallest positive element in $\mathbb{F}(\beta, t, L, U)$ is

$$m = +(.\underbrace{100\dots0}_{\text{t}})_\beta \times \beta^L = \beta^{L-1}.$$

Range. The finite number system defined above is therefore contained in the three intervals:

$$\mathbb{F}(\beta, t, L, U) \subset \left[-(1 - \beta^{-t})\beta^U, -\beta^{L-1}\right] \cup \{0\} \cup \left[\beta^{L-1}, (1 - \beta^{-t})\beta^U\right].$$

Distribution. Fixing $s = 0$ and $e = 1$, and taking all combinations of digits yields all numbers in $\mathbb{F}(\beta, t, L, U)$ in the interval $[1, \beta)$, which read

$$1, 1 + \beta^{1-t}, \dots, \beta - \beta^{1-t}.$$

This is a total of $(\beta - 1)\beta^{(t-1)}$ numbers, evenly distributed in $[1, \beta)$ with gap size β^{1-t}.
Similarly, there are $(\beta - 1)\beta^{t-1}$ numbers distributed in each interval $[\beta^{e-1}, \beta^e)$ for $L < e \leqslant U$, with gap size β^{e-t} increasing as the exponent e increases.

Definition 1.4. *The* ***machine epsilon*** $\varepsilon_M = \beta^{1-t}$ *is the distance between* 1 *and the next floating-point number, or equivalently the gap size in the interval* $[1, \beta)$.
The ***unit roundoff*** *or* ***machine precision*** *is* $\mathrm{u} = \frac{1}{2}\varepsilon_M = \frac{1}{2}\beta^{1-t}$.

Remark 1.5. *When using a binary system* ($\beta = 2$)*, the formulae above simplify to:*

- *Cardinality:* $|\mathbb{F}(2, t, L, U)| = (U - L + 1)2^t + 1$,
- *Extreme numbers:* $M = (1 - 2^{-t})2^U$, $m = 2^{L-1}$,
- *Gap size in the interval* $[1, 2)$*:* 2^{1-t},
- *Unit roundoff:* $u = 2^{-t}$.

Lecture 2: IEEE Format: Rounding errors and floating-point arithmetic. (Wednesday, August 26)

2.1 IEEE standard.

Computers typically use a standardized **single-precision** or **double-precision** IEEE representation:

Single-precision format.

s (1 bit)	e (8 bits)	mantissa (23(+1) bits)

- Total of 32 bits.
- 1 bit is for the sign,
- 8 bits for the exponent ranging from (in base 2) 00000001 to 11111110,
  i.e. from 1 to $2^7 + 2^6 + \cdots + 2^1 + 0 = 254$. This number is shifted by a fixed value of -127 resulting in the effective range of values for the exponents:

$$L = -126 \leqslant e \leqslant U = 127,$$

- 23 bits for the mantissa, with the leading digit $d_1 = 1$ hidden, so an effective precision or number of binary digits $t = 24$.

Double-precision format.

s (1 bit)	e (11 bits)	mantissa (52(+1) bits)

- Total of 64 bits.
- 1 bit is for the sign,
- 11 bits for the exponent ranging from (in base 2) 00000000001 to 11111111110,
  i.e. from 1 to $2^10 + 2^9 + \cdots + 2^1 + 0 = 2046$. This number is shifted by a fixed value of -1022 resulting in the effective range of values for the exponents:

$$L = -1021 \leqslant e \leqslant U = 1024,$$

- 52 bits for the mantissa, with the leading digit $d_1 = 1$ hidden, so an effective precision or number of binary digits $t = 53$.

Special numbers:

- ± 0 with exponent $0 \ldots 0$ (all zeros) and mantissa $1 \ldots 1$ (all ones),
- $\pm \infty$ with exponent $1 \ldots 1$ (all ones) and mantissa $0 \ldots 0$ (all zeros),
- *NaN* with exponent $1 \ldots 1$ (all ones) and at least one nonzero mantissa digit.

Summary:

	β	t	L	U	u	m	M
Single	2	24	-126	127	$2^{-24} \approx 6 \cdot 10^{-8}$	$5.88 \cdot 10^{-39}$	$1.7 \cdot 10^{38}$
Double	2	53	-1021	1024	$2^{-53} \approx 1 \cdot 10^{-16}$	$2.2 \cdot 10^{-308}$	$1.8 \cdot 10^{308}$

Relative error and distribution of floating-point numbers. We can compute the absolute and relative distance between one floating point number and the next largest. This is easiest seen by writing

$$x = (-1)^s m(x) \beta^{e-t}, \tag{2.1}$$

where $m(x)$, the mantissa, is an integer taking values from β^{t-1} (corresponding to $(\underbrace{100000000}_{\text{t}})_\beta$ and $\beta^t - 1$.

- **Absolute:** for x in the form (2.1), it is clear that the next largest floating-point number is located at $\Delta x = (-1)^s \beta^{e-t}$. Hence
$$|\Delta x| = \beta^{e-t}$$
depends only on the exponent e.

- **Relative:** we find
$$\left|\frac{\Delta x}{x}\right| = \frac{\beta^{e-t}}{m(x)\beta^{e-t}} = \frac{1}{m(x)},$$
which depends only on the mantissa, decreasing in each interval from β^{1-t} to roughly β^{-t}.

2.2 Rounding of real numbers.

Consider a system of numbers $\mathbb{F}(\beta, t, L, U)$. Given a real number $x = (-1)^s (0.d_1 \dots d_t d_{t+1} \dots)_\beta \cdot \beta^e$, how to best represent this number in $\mathbb{F}$?
One reasonable option is to round to the nearest number in $\mathbb{F}$: assuming $L \leqslant e \leqslant U$, we define

$$fl(x) = \begin{cases} (-1)^s (0.d_1 \dots d_t)_\beta \cdot \beta^e, & \text{if } d_{t+1} < \beta/2, \\ (-1)^s (0.d_1 \dots d_t + \underbrace{0.0 \dots 01}_{t})_\beta \cdot \beta^e, & \text{if } d_{t+1} \geqslant \beta/2. \end{cases} \tag{2.2}$$

This rounding introduces an absolute error:

$$|x - fl(x)| \leqslant \frac{\beta}{2} \beta^{e-(t+1)} = \frac{1}{2} \beta^{e-t},$$

and a relative error:

$$\left|\frac{x - fl(x)}{x}\right| \leqslant \frac{\frac{\beta}{2}\beta^{e-(t+1)}}{\beta^{e-1}} = \frac{1}{2}\beta^{1-t} = u.$$

Property 2.1. *If $x \in \mathbb{R}$ is such that $m \leqslant |x| \leqslant M$, then*

$$fl(x) = x \cdot (1 + \delta) \qquad \textit{with} \qquad |\delta| \leqslant u, \tag{2.3}$$

where we recall $u = \frac{1}{2}\beta^{1-t} = \frac{1}{2}\varepsilon_M$ is the round-off unit.

Remark 2.2. *An alternative policy, at the cost of greater rounding error, would be to define rounding by simple chopping of the extra digits:*

$$fl_{chop}(x) = (-1)^s (0.d_1 \dots d_t)_\beta \cdot \beta^e.$$

Overflow: if $|x| > M$ or $e > U$, then the rounding operation $fl(x)$ is not yet defined. Typically, this is handled by an interruption, or by setting

$$fl(x) = \begin{cases} +\infty & x > M, \\ -\infty & x < -M. \end{cases}$$

Underflow: if $0 < |x| < m$, or $e < L$, then underflow happens (unless sub-normal numbers are included.) Usually, we set

$$fl(x) = \begin{cases} +0 & 0 < x < m, \\ -0 & -m < x < 0. \end{cases}$$

2.3 Machine arithmetic.

We next need operations that will approximate exact arithmetic within the finite number systems defined above. For example, suppose

$$x = (0.x_1 \dots x_t)_\beta \cdot \beta^{e_x}, \qquad y = (0.y_1 \dots y_t)_\beta \cdot \beta^{e_y}.$$

Assume that $e_x > e_y$. In order to perform the addition $x + y$ or substraction $x - y$, need to align exponents, by shifting y:

$$y = (0.\underbrace{0 \dots 0}_{e_x - e_y} y_1 \dots y_t)_\beta \cdot \beta^{e_x}.$$

Next, we carry out addition on the mantissa:

$$x + y = (0.x_1 \dots x_t + 0.\underbrace{0 \dots 0}_{e_x - e_y} y_1 \dots y_t)_\beta \cdot \beta^{e_x},$$

but since the sum usually has more than t nonzero digits, one will round the end result to the nearest floating-point number:

$$x \oplus y = fl(x + y).$$

Remark 2.3. *If $x, y > 0$ the $+$ operation can then be carrier out by just simply chopping y to its first t digits after shifting the exponent; otherwise, or for the $-$ operation, the machine implementation needs a so-called extra rounding digit to perform an accurate rounding.*

Definition 2.4. *The approximate floating point operations will be denoted respectively $\oplus$, $\ominus$, $\otimes$, $\oslash$. They are defined respectively using the model above: for $\cdot$ being any of the operations $+$, $-$, $\times$ or $\div$,*

$$\mathbb{F} \ni x \odot y = fl(x \cdot y), \qquad \text{for } x, y \in \mathbb{F}(\beta, t, L, U),$$

or more generally for any real numbers x, y by

$$\mathbb{F} \ni x \odot y = fl\left(fl(x) \cdot fl(y)\right), \qquad \text{for } x, y \in \mathbb{R}.$$

The considerations above show that the following property holds:

Proposition 2.5. *Error model. If $\cdot$ is one of the operations above, and no overflow occurs then*

$$x \odot y = (x \cdot y)(1 + \delta), \qquad \text{for } x, y \in \mathbb{F}(\beta, t, L, U), \quad |\delta| \leqslant u.$$

Remark 2.6. *For any $0 \leqslant \delta < u$, we have $1 \oplus \delta = 1$, whereas $1 \oplus \delta > 1$ for $\delta > u$.*
What happens for $\delta = u$ depends on the precise rounding policy: with the choice above (2.2) we get $1 \oplus u > 1$, whereas on most computer systems one would obtain $1 \oplus u = 1$.

Proposition 2.7.

- ***Commutativity:*** *we have $a \oplus b = b \oplus a$, $a \ominus b = b \ominus a$ and $a \otimes b = b \otimes a$.*
- ***Non-associativity:*** *in general, $a \oplus (b \oplus c) \neq (a \oplus b) \oplus c$ and $a \otimes (b \otimes c) \neq (a \otimes b) \otimes c$.*

Example. Take $x = 2^{-4}$, $y = 1$ and $z = -1$ in a binary floating-point system with $t = 3$, $L = -3$ and $U = 2$. The floating-point representation of x, y and z is

$$x = +(.100)_2 \times 2^{-3}, \qquad y = +(.100)_2 \times 2^{1}, \qquad z = -(.100)_2 \times 2^{1},$$

and one computes directly

$$\begin{aligned} x \oplus y &= fl(+(.0000100)_2 \times 2^1 + (.100)_2 \times 2^1) = fl\left(+(.10001)_2 \times 2^1\right) = (.100)_2 \times 2^1, \\ (x \oplus y) \oplus z &= fl\left(+(.100 - .100)_2 \times 2^1\right) = 0. \end{aligned}$$

On the other hand,

$$\begin{aligned} y \oplus z &= fl\left(+(.100 - .100)_2 \times 2^1\right) = 0, \\ x \oplus (y \oplus z) &= fl\left(+(.100)_2 \times 2^{-3} + 0\right) = +(.100)_2 \times 2^{-3}. \end{aligned}$$

Thus $(x \oplus y) \oplus z \neq x \oplus (y \oplus z)$.

Additional rules:

$$\begin{aligned} (\pm\infty) \oplus (\pm\infty) &= \pm\infty; & (\pm\infty) \ominus (\mp\infty) &= \pm\infty; \\ (\pm\infty) \oplus (\mp\infty) &= NaN; & (\pm\infty) \ominus (\pm\infty) &= NaN; \\ (\pm\infty) \otimes (\pm\infty) &= \pm\infty; & (\pm\infty) \otimes (\mp\infty) &= \mp\infty; \\ a \oslash (\pm 0) &= \pm\infty; & (a \oslash (\pm\infty) &= \pm 0 \qquad (a > 0); \\ 0 \oslash 0 &= NaN; & \infty \oslash \infty &= NaN; \\ NaN \odot a &= NaN. \end{aligned}$$

Lecture 3: Conditioning, Stability, Convergence. (Monday, August 31)

3.1 Condition number

We remember the general abstract form of an model or problem to solve:

$$F(x, d) = 0. \tag{3.1}$$

We will assume that this problem is well-posed: for any data d, there exists a unique solution x. This implies the existence of the *resolvent map*, an (a priori) unknown but continuous function $x = G(d)$ mapping the data to the corresponding solution. Given some relevant data, we seek to investigate the behavior of the problem in its vicinity, i.e. with hopefully small perturbations δd and δx of the data and variables, respectively:

$$F(x + \delta x, d + \delta d) = 0. \tag{3.2}$$

Assumption 3.1. *We will assume that the resolvent map is Lipschitz continuous in the neighborhood of some interesting data d, i.e. there is $\eta_0(d) > 0$ and $K_0(d) > 0$ such that*

$$\|\delta d\| \leqslant \varepsilon_0 \implies \|\delta x\| \leqslant K_0 \|\delta d\|.$$

Under this condition, we may define two *condition numbers*:

Definition 3.2. *The **absolute condition number** is the quantity*

$$K_{abs}(d) = \limsup_{\delta d \to 0} \frac{\|\delta x\|}{\|\delta d\|} = \limsup_{\hat{d} - d} \frac{\|G(\hat{d}) - G(d)\|}{\|\hat{d} - d\|}. \tag{3.3}$$

Definition 3.3. *The **relative condition number** is the quantity*

$$K(d) = \limsup_{\delta d \to 0} \frac{\|\delta x\|/\|x\|}{\|\delta d\|/\|d\|} = \limsup_{\hat{d} - d} \frac{\|G(\hat{d}) - G(d)\|/\|G(d)\|}{\|\hat{d} - d\|/\|d\|}. \tag{3.4}$$

The condition number measures how much a perturbation in the data d propagates to the variables x. A problem is said to be *ill-conditioned* if $K(d)$ is infinite or big for some relevant data d ("big" being a subjective, problem-dependent qualification). In the case where G is differentiable at d, the condition numbers can be expressed in terms of the derivative of G with respect to d, defined as

$$G(d + \delta d) - G(d) = G'(d) \cdot \delta d + o(\|\delta d|).$$

Then we have

$$K_{abs}(d) = \|G'(d)\| \qquad \text{and} \qquad K(d) = \frac{\|d\|}{\|G(d)\|} \|G'(d)\|. \tag{3.5}$$

Example. Let us study the conditioning of the root of a polynomial as a function of the coefficients. Let $p(x) = a_0 + a_1 x + \cdots + a_n x^n$, with a simple root λ, such that $p(x) = (x - \lambda)q(x)$ with some polynomial $q(x)$ such that $q(\lambda) \neq 0$. We see λ as a function of one of the coefficients,

$$\lambda = f(a_i),$$

and we want to compute the corresponding conditioning numbers $K_{abs}(a_i)$ and $K(a_i)$. *Solution.* The map $f(a_i)$ is our resolvent map, so we need to compute $f'(a_i)$. We know that $p(\lambda) = 0$, so

$$\sum_{j=0}^{n} a_j \left(f(a_i)\right)^j = 0.$$

Differentiating w.r.t $a - i$ we find

$$f'(a_i)\underbrace{\Big(\sum_{j=1}^{n} j a_j \left(f(a_i)\right)^{j-1}\Big)}_{=p'(\lambda)} + \underbrace{\left(f(a_i)\right)^i}_{=\lambda^i} = 0$$

so

$$f'(a_i) = \frac{-\lambda^i}{p'(\lambda)} \qquad \text{and} \qquad K_{abs}(a_i) = \frac{|\lambda|^i}{|p'(\lambda)|}.$$

The relative condition number is then

$$K(a_i) = |a_i|/|\lambda| K_{abs}(a_i) = \frac{|a_i||\lambda|^{i-1}}{|p'(\lambda)|}.$$

3.2 Errors in a practical situation.

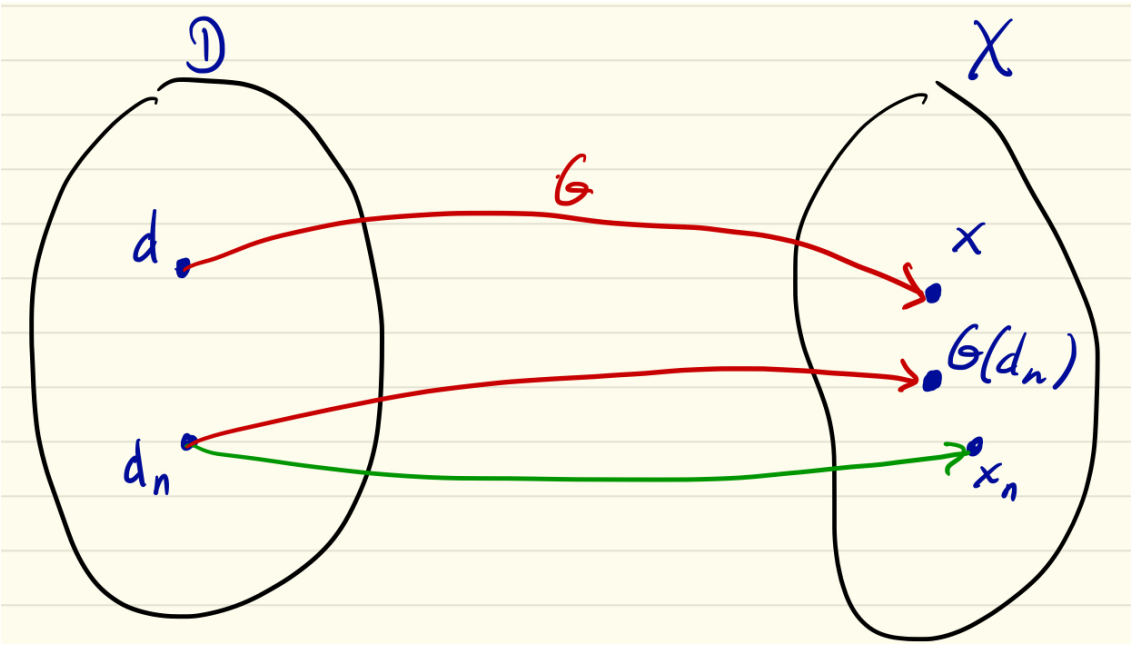
Given a difficult problem of the form (3.1), one usually sets up a sequence of more approachable problems,

$$F_n(d_n, x_n) = 0 \tag{3.6}$$

with the expectation that the x_n's can be used as approximations to the exact solution x because $x_n \to x$ as $d_n \to d$.
It is sometimes useful to categorize errors:

- The absolute backward error (data) $\|d_n - d\|$,
- the absolute forward error (solution) $\|x_n - x\|$, which can be further decomposed into

$$x_n - x \quad = \quad G_n(d_n) - G(d) \quad = \quad \underbrace{G_n(d_n) - G(d_n)}_{\text{computational error}} \quad + \quad \underbrace{G(d_n) - G(d)}_{\text{propagated data error}} .$$



Note that the propagated data error can be measured using the condition number of the problem, since

$$\|G(d_n) - G(d)\| \lessapprox K(d)\|d_n - d\|.$$

3.3 Concepts of consistency, stability, and convergence

In this section, we introduce in a very general setting the three most important concepts for the rigorous analysis of numerical methods.

Definition 3.4. *A scheme of the form* (3.6) *is said to be* ***consistent*** *if*

$$F_n(x,d) = F_n(x,d) - F(x,d) \to 0 \qquad as \qquad n \to \infty,$$

where x is the exact solution to the problem (3.1) *with exact data d.*

Example. Faced with the problem of computing the integral I of some function $f(t)$ over an interval (a,b), which can be cast into the form (3.1) with

$$F(I,f) = I - \int_a^b f(t)dt = 0,$$

one can decompose the interval into n uniformly smaller intervals $a = t_1 < t_2 < \cdots < t_n = b$ and use the midpoint rule to compute a numerical approximation I_n of the integral over each piece with a finite number of function evaluations:

$$F_N(I_n,f) = I_n - H\sum_{k=1}^{n-1} f\left(\frac{t_k + t_{k+1}}{2}\right),$$

where $H = \frac{b-a}{n}$ and $t_k = a + (k-1)H$.
Because it is known that the midpoint rule's accuracy is of order H^3 over each small interval of size H, one can show that $I_n \to I$ as $n \to \infty$. As a result, one easily checks that

$$F_N(I,f) - F(I,f) \to 0,$$

where I is the exact integral of the function over (a,b), as long as f is continuous, i.e. this numerical integration method is consistent for continuous functions.

Definition 3.5. *A scheme of the form* (3.6) *is said to be* ***stable*** *if it admits finite condition numbers:*

$$K_n(d_n) \quad = \quad \limsup_{\delta d_n \to 0} \frac{\|\delta x_n\|/\|x_n\|}{\|\delta d_n\|/\|d_n\|} \quad = \quad \limsup_{\delta d_n \to 0} \frac{\|G_n(d+\delta d_n) - G_n(d)\|}{\|\delta d_n\|}\frac{\|d_n\|}{\|G_n(d)\|},$$

and

$$K_{abs,n}(d_n) \quad = \quad \limsup_{\delta d_n \to 0} \frac{\|\delta x_n\|}{\|\delta d_n\|} \quad = \quad \limsup_{\delta d_n \to 0} \frac{\|G_n(d+\delta d_n) - G_n(d)\|}{\|\delta d_n\|},$$

which are bounded as $n \to \infty$, such that asymptotically we define

$$K^{num}(d) = \limsup_{n\to\infty} K_n(d) \qquad and \qquad K^{num}_{abs}(d) = \limsup_{n\to\infty} K_{abs,n}(d).$$

Remark 3.6. *A numerical scheme can unstable even if the underlying problem is itself stable (well-conditioned)!*

Definition 3.7. *A scheme of the form* (3.6) *is said to be* ***convergent*** *if the computed sequence of solutions converges to the exact solution:*

$$\lim_{n\to\infty}\lim_{\delta_n\to 0} G_n(d+\delta d_n) = G(d)$$

or more formally, for any $\varepsilon > 0$ *there exists* $n_0(\varepsilon)$ *and* $\delta_0(n_0,\varepsilon) > 0$ *s.t.*

$$\forall n > n_0(\varepsilon), \quad \forall \delta d_n \text{ s.t. } \|\delta d_n\| < \delta_0(n_0,\varepsilon), \qquad \|G(d) - G_n(d+\delta_n)\| < \varepsilon.$$

Relations between stability and convergence *Convergence* is clearly the main criterion for a numerical method to be useful, as it ensures that the result of an simulation is indeed an approximation of the exact solution, which can be improved with additional computational power. The usefulness of the *stability* concept is that it is usually much easier to investigate and prove, and stability is a necessary condition in order for a numerical method to be convergent. In addition, one of the most useful observations in numerical analysis is the statement

$$\text{STABILITY} \quad + \quad \text{CONSISTENCY} \quad \Longrightarrow \quad \text{CONVERGENCE.}$$

The following non-rigorous reasoning is the template for a "proof" of this fundamental statement, which can be made more precise for specific methods - in particular for the numerical solution of ODEs:

"Proof." Our goal is to control the absolute error $\|x(d) - x_n(d+\delta d_n)\|$. Using the triange inequality, we decompose it as:

$$\|x(d) - x_n(d+\delta d_n)\| \leqslant \|x(d) - x_n(d)\| + \|x_n(d) - x_n(d+\delta d_n)\| \tag{3.7}$$

and we observe that the first term on the right-hand side of (3.7) can be controlled thanks to the stability assumption:

$$\|x_n(d) - x_n(d+\delta d_n)\| \leqslant K_{abs,n}(d)\|\delta_n\|.$$

Thus it remains to control the second term on the right-hand side of (3.7). We will assume that $F(x,d)$ is locally differentiable:

$$F_n(x(d),d) - F_n(x_n(d),d) = \frac{\partial F_n}{\partial x}\Big|_{(\xi,d)}(x(d) - x_n(d))$$

where ξ is "between" x and x_n. Assuming that the derivative $\frac{\partial F_n}{\partial x}$ is an invertible linear map (which is reasonable, as the problem should be stable and hence well-conditioned) leads to the identity

$$x(d) - x_n(d) = \left(\frac{\partial F_n}{\partial x}\Big|_{(\xi,d)}\right)^{-1}(F_n(x(d),d) - F_n(x_n(d),d))$$

Passing to the norms, and using the fact that $F_n(x_n(d),d) = F(x(d),d) = 0$ leads to the identity

$$\|x(d) - x_n(d)\| = \left\|\left(\frac{\partial F_n}{\partial x}\Big|_{(\xi,d)}\right)^{-1}\right\| \|F_n(x(d),d) - F(x(d),d)\|.$$

Now *consistency* ensures that this last term goes to zero as $n \to \infty$, which completes the proof. Formally, given $\varepsilon > 0$, using consistency we may choose $n_0(\varepsilon)$ such that

$$\|F_n(x(d), d) - F(x(d), d)\| < \varepsilon/2 \left\| \left(\frac{\partial F_n}{\partial x}\Big|_{(\xi,d)} \right)^{-1} \right\|^{-1} \qquad \text{for all} \quad n \geqslant n_0(\varepsilon).$$

Then, given $K(n_0, d) = \sup_{n \geqslant n_0} K_{abs,n}(d)$ we select $\delta_0(n_0, \varepsilon)$ such that $K(n_0, d)\delta_0 < \varepsilon/2$. Then (3.7) yields

$$\|x(d) - x_n(d + \delta d_n)\| < \varepsilon. \qquad \square$$

Lecture 4: Rate and Order of Convergence. (Wednesday, September 2)

While *convergence* is an important notion, the usefulness of a numerical method really depends on how fast convergence happens. We introduce in this lecture a formalism to quantify the accuracy and efficiency of a numerical method.
First we recall:

Definition 4.1. *If α_n is a given numerical sequence, we define the* $\limsup$ *and* $\liminf$ *as*

$$\limsup_{n\to\infty} \alpha_n = \lim_{N\to\infty} (\sup\{\alpha_n; \quad n \geqslant N\}) \in \mathbb{R} \cup \{\pm\infty\}, \tag{4.1}$$

and

$$\liminf_{n\to\infty} \alpha_n = \lim_{N\to\infty} (\inf\{\alpha_n; \quad n \geqslant N\}) \in \mathbb{R} \cup \{\pm\infty\}. \tag{4.2}$$

4.1 Asymptotic rate of convergence

Definition 4.2. *Suppose we generate a sequence x_k indexed by $k \geqslant 0$, such that $\lim_{k\to\infty} x_k = \alpha$. Such a sequence is said to converge to α with* ***order*** *$p \geqslant 1$ if*

$$\frac{|x_{k+1} - \alpha|}{|x_k - \alpha|^p} \leqslant C, \qquad \forall k \geqslant k_0,$$

for some fixed real number $C > 0$ and k_0 large enough. In this case, we define

$$r = \limsup_{k\to\infty} \frac{|x_{k+1} - \alpha|}{|x_k - \alpha|^p},$$

and we classify the sequence as follows:

$p = \ldots$	$r = \ldots$	*the sequence (or method) convergence rate is...*
1	$r \geqslant 1$	*Sub-linear.*
1	$0 < r < 1$	*Linear.* ***r is called the convergence factor.***
$1 < p < 2$	$0 < r < \infty$	*Super-linear.*
2	$0 < r < \infty$	*Quadratic.*
$2 < p < 3$	$0 < r < \infty$	*Super-quadratic.*
3	$0 < r < \infty$	*Cubic.*
$2 < p < 3$	$0 < r < \infty$	*Super-cubic.*
integer $p > 1$	$0 < r < \infty$	*p^{th} order.*

Some examples.

- *Let $s > 0$ and consider the sequence $\{u_n\}_{n\geqslant 0} = \{n^{-s}\}$. We have $u_n \to 0$, and*

$$\frac{|u_{n+1} - 0|}{|u_n - 0|^p} = \frac{n^{ps}}{(n+1)^s} = \frac{n^{(p-1)s}}{(1+1/n)^s}.$$

For $p > 1$, this expression converges to $+\infty$, but for $p = 1$ we have

$$\lim \frac{|u_{n+1} - 0|}{|u_n - 0|} = \lim_{n\to\infty} \frac{1}{(1+1/n)^s} = 1.$$

Thus the sequence u_n converges sublinearly with convergence factor 1.

- *Consider the sequence* $\{v_n\}_{n\geqslant 0} = \{e^{-n}\}$. *Again* $v_n \to 0$, *and*

$$\frac{|v_{n+1} - 0|}{|v_n - 0|^p} = \frac{e^{-(n+1)}}{e^{-pn}} = e^{pn-(n+1)} = e^{(p-1)n-1}.$$

For $p > 1$, *the expression converges to* $+\infty$, *and for* $p = 1$ *it converges to* e^{-1}. *Hence the sequence* v_n *converges linearly (order 1) with convergence factor* $1/e$.

- *Let* $\{w_n\}_{n\geqslant 0}$ *be a sequence defined by the relations*

$$w_0 = \alpha > 0, \qquad w_{n+1} = \frac{w_n + w_n^{-1}}{2} \quad \text{for } n \geqslant 0.$$

It can be shown that $w_n \to 1$ *as* $n \to \infty$, *for any value of the initial term* $\alpha > 0$. *Furthermore, one has*

$$w_{n+1} - 1 = \frac{w_n^2 - 2w_n + 1}{2w_n} = \frac{(w_n - 1)^2}{2w_n}$$

so

$$\lim_{n\to\infty} \frac{|w_{n+1} - 1|}{|w_n - 1|^2} = \lim_{n\to\infty} \frac{1}{2w_n} = \frac{1}{2}.$$

The convergence rate of the sequence w_n *is thus quadratic (order 2).*

4.2 Order notation

Definition 4.3. *Let* r_n *be a given numerical sequence of strictly positive numbers converging to zero:* $r_n > 0$ *and* $\lim_{n\to\infty} r_n = 0$. *Given another sequence* $\{\alpha_n\}_{n\geqslant 0}$ *with* $\lim_{n\to\infty} \alpha_n = \alpha$ *in some normed space.*
If the quantity

$$\limsup_{n\to\infty} \frac{\|\alpha_n - \alpha\|}{r_n} = K \tag{4.3}$$

is finite ($0 \leqslant K < \infty$), *we say that*

- α_n *converges to* α *at the same rate as* r_n *(more precisely, if* $K > 0$*);*
- $\alpha_n - \alpha$ *is of* ***order*** r_n *or* $O(r_n)$ *[big O of* r_n*], denoted*

$$\alpha_n - \alpha = O(r_n) \quad \text{or} \quad \alpha_n = \alpha + O(r_n).$$

If $K = 0$ *in* (4.3) *then we use the same notation* $\alpha_n = \alpha + o(r_n)$ *[small O of* r_n*], meaning that the order of convergence of* α_n *to* α *is higher than the convergence order of* r_n.

Example. Let $\alpha_n = \dfrac{n+2}{n+e^{-n}}$. Then $\alpha \to 1$ and

$$|\alpha_n - 1| = \frac{2 - e^{-n}}{n + e^{-n}} \leqslant \frac{2}{n}.$$

So we have the following bound:

$$\frac{|\alpha_n - 1|}{1/n} \leqslant 2 \qquad \text{and} \qquad \limsup_{n\to\infty} \frac{|\alpha_n - 1|}{1/n} \leqslant 2.$$

Hence $\alpha_n = 1 + O(1/n)$.

Order and asymptotic rate. A sequence $\{\alpha_n\}_{n\geqslant 0}$ converging to α and such that

$$\|\alpha_n - \alpha\| \sim r^n \quad \text{meaning} \quad \|\alpha_n - \alpha\| = c_n r^n,$$

where $0 < r < 1$ and the sequence c_n is bounded uniformly from above and below:

$$0 < c \leqslant c_n \leqslant C \qquad \text{for two numbers } c, C \text{ independent of } n,$$

then clearly

$$\alpha_n = \alpha + O(r^n).$$

Furthermore, if $\lim_{n\to\infty} \frac{c_{n+1}}{c_n} = 1$ then the convergence rate is <u>linear</u>.
Similarly, if

$$\|\alpha_n - \alpha\| \sim r^{p^n} \quad \text{meaning} \quad \|\alpha_n - \alpha\| = c_n r^n,$$

with the same assumptions $0 < r < 1$ and the sequence c_n is bounded uniformly from above and below, then

$$\alpha_n = \alpha + O(r^{p^n}) \qquad \text{and the convergence order is } p.$$

4.3 Order notation for functions.

Definition 4.4. *Let F, G be two functions defined in a neighborhood of zero taking values in some normed space with $\lim_{h\to 0} F(h) - G(h) = 0$, and a rate function $r(h)$ defined in a neighborhood of zero with $r(h) > 0$ and $\lim_{h\to 0} r(h) = 0$. Then we say*

$$F(h) = G(h) + O\left(r(h)\right) \qquad \Leftrightarrow \qquad \limsup_{h\to 0} \frac{\|F(h) - G(h)\|}{r(h)} < \infty.$$

Note that typically, $r(h)$ is some power h^p with $p > 0$.

Example This notation is convenient for the remainder term with Taylor series. Let $F(h) = \cos(h)$ and $G(h) = 1 - h^2/2$, then applying the Taylor formula for $\cos(h)$ near zero yields:

$$\cos(h) = 1 - \frac{h^2}{2} + \frac{h^4}{24}\cos(\xi), \qquad \text{for some } \xi \in [0, h],$$

and thus

$$\frac{|F(h) - G(h)|}{h^4} \leqslant \frac{1}{24} \qquad \text{and} \qquad \cos(h) = 1 - \frac{h^2}{2} + O(h^4).$$

4.4 Root of a function

In this chapter, we want to find the roots of some continuously differentiable function $f : I \to \mathbb{R}$, where I is a given finite interval of $\mathbb{R}$.

Notation. $C^m(I)$ is the set of functions, defined on an interval $I \subset \mathbb{R}$, whose derivatives up to order m exist and are all continuous.
$C(I) := C^0(I)$ is then the set of continuous functions.

Theorem 4.5. *Given some function $f \in C^m(I)$ and some point $p \in I$ such that*

$$f(p) = f'(p) = \cdots = f^{(m-1)}(p), \qquad f^{(m)}(p) \neq 0,$$

then $f(x) = (x-p)^m h(x)$ for some continuous function $h(x)$ such that $h(p) = \frac{f^{(m)}(p)}{m!} \neq 0$.

Proof. Using the Taylor expansion of f at p with exact remainder, for $x \in I$, $x \neq p$, we find

$$f(x) = \underbrace{\sum_{j=0}^{m-1} \frac{f^j(p)}{j!}(x-p)^j}_{=0} + \frac{f^{(m)}(\xi)}{m!}(x-p)^m, \qquad \text{with } \xi \in [p,x],$$

and therefore we can define a function $h(x)$ for $x \neq p$, continuous on $I \backslash \{p\}$, satisfying the identity

$$h(x) = \frac{f(x)}{(x-p)^m} = \frac{f^{(m)}(\xi)}{m!} \qquad \text{for } \xi(x) \in [p,x].$$

Now since $f^{(m)}(p)$ is continuous at p, and by the squeeze theorem, $\lim_{x\to p} \xi(x) = p$, we find that

$$\lim_{x\to p} h(x) = \frac{f^{(m)}(p)}{m!}.$$

By defining $h(p) := \frac{f^{(m)}(p)}{m!} \neq 0$, we have built a continuous function h over the whole interval I such that $f(x) = (x-p)^m h(x)$, and the theorem is proved.

□

Definition 4.6. *For f satisfying the conditions of Theorem 4.5, i.e. $f \in C^m(I)$ and $p \in I$ such that the first $(m-1)$-th derivatives of f vanish at p and the m-th derivative does not, we say that p is* ***a root of f with multiplicity*** *m.*

Lecture 5: Rootfinding (continued). (Monday, September 7)

5.1 Root conditioning.

Problem. We want to study practical methods aiming to find (approximations to) solutions of nonlinear equations such as

$$f(x) = \phi(x) - d = 0.$$

This problem is well-posed in general if ϕ is an invertible map: in this case we may write $p = \phi^{-1}(d)$. Let us investigate the conditioning of this problem as a function of data d. Thanks to the chain rule,

$$\phi(\phi^{-1}(d)) \qquad \Longrightarrow \qquad (\phi^{-1})'(d) = \frac{1}{\phi'(p)},$$

so the condition number reads

$$K(d) = \frac{|d|}{|p||f'(p)|}, \qquad K_{abs}(d) = \frac{1}{|f'(p)|},$$

for a simple root (multiplicity 1) such that $f'(p) \neq 0$.
For higher multiplicities, we compute explicitely

$$f(p + \delta p) = d + \frac{(\delta p)^m}{m!} + o\left((\delta p)^m\right) = d + \delta d,$$

hence if $m > 1$, $K_{abs}(d) = \limsup_{\delta d \to 0} \frac{|\delta p|}{|\delta d|} = +\infty$.
In conclusion, rootfinding is generally ill-conditionned if the root is not simple, or if $|f'(p)|$ is small or zero. In such cases, care has to be taken as the usual methods will fail to converge or converge more slowly than usual.

5.2 Geometrical Rootfinding Methods

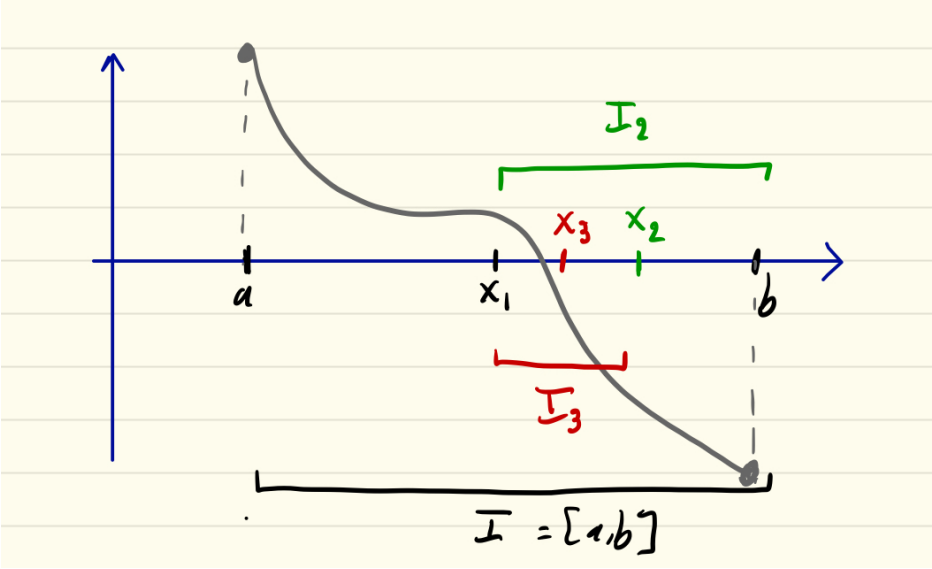
5.2.1 Bisection method

The very first result showing existence of a root is the following theorem:

Theorem 5.1 (Bolzano's theorem)**.** *Let $f \in C([a,b])$ and $f(a)f(b) < 0$, then there exists $p \in (a,b)$ such that $f(p) = 0$.*

Remark 5.2. *One also calls such a pair a, b such that $f(a), f(b)$ have opposite signs a* ***bracket*** *as this ensures the existence of a root in the segment $[a,b]$.*

Bisection method This simple result inspires the following rootfinding method. Given a, b as above, we can check one point inside the interval and use the same test to select a smaller interval, either to the left or right of this point, where a root will necessarily exist. The most natural candidate for this is the midpoint of the interval. In this way, the original interval is halved, and the process may continue until the remaining interval is small enough to satisfy some prescribed error tolerance.



Practically, this process writes as the following algorithm:
Initialization: Let $a_1 = a$, $b_1 = b$, the midpoint $x_1 = \frac{a+b}{2}$.

1. Compute $f(x_1)$.
2. If...
   - $f(x_1) = 0$: x_1 is a root, done.
   - $f(a_1)f(x_1) < 0$: we set $a_2 = a_1$, $b_2 = x_1$ and $x_2 = \frac{a_1+b_1}{2}$.
   - $f(x_1)f(b_1) < 0$: we set $a_2 = x_1$, $b_2 = b_1$ and $x_2 = \frac{a_1+b_1}{2}$.
3. Repeat steps 1-2, generating $a_3, b_3, x_3, \ldots, a_n, b_n, x_n, \ldots...$

This process generates a sequence of nested intervals

$$[a_1, b_1] \supset [a_2, b_2] \supset \cdots \supset [a_n, b_n] \supset \cdots$$

such that there is at least one zero $p_n \in [a_n, b_n]$ for any $n \geqslant 1$ per the bracketing property. Furthermore, the sequence $\{p_n\}$ is Cauchy and has a limit:

$$p = \lim_{n\to\infty} p_n \qquad \text{is a root of } f.$$

In fact, we have $p = \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n$, and the midpoint sequence also converges to p by the squeeze theorem, such that

$$p = \lim_{n\to\infty} x_n.$$

We use this sequence to approximate the root p.

Spped of convergence. The length of the search interval I_n is halved at each step. Hence

$$|I_n| = |I_1|/2^{n-1} = (b-a)/2^{n-1}.$$

Denoting by $e_n = x_n - p$ the absolute error at step n, it follows that

$$|e_n| < |I_n|/2 = (b-a)/2^n,$$

and we check again that $\lim_{n\to\infty} |e_k| = 0$.

Theorem 5.3. *The bisection method is globally convergent: it converges unconditionally of the starting interval, provided that a, b satisfy the bracketing property:*

$$f(a)f(b) < 0.$$

More precisely, the absolute error between the midpoint sequence and the root obeys

$$|p - x_n| < 2^{-k}(b-a) \qquad or \qquad x_n = p + O(1/2^n).$$

Remark 5.4. *Because we cannot guarantee a monotone reduction in the error, the method is technically not of order 1 as defined above.*

Property 5.5. *To achieve a desired accuracy ε, one needs at most*

$$m = \left\lceil \log_2 \left(\frac{b-a}{\varepsilon} \right) \right\rceil - 1 \qquad iterations.$$

Proof.

$$|e_n| < \frac{b-a}{2^n} < \varepsilon \qquad \Leftrightarrow \qquad n > \log_2 \left(\frac{b-a}{\varepsilon} \right).$$

□

Algorithm 1 Bisection method.

Input: Function f, a, b such that $f(a)f(b) < 0$, and a desired accuracy ε.

Output: Approximate value $p \in (a, b)$.

```
1: function BISECTION(f, a, b, ε)
2:     N = ⌈log_2((b−a)/ε)⌉ − 1;
3:     fa = f(a);
4:     for n = 1 ... N do
5:         x = (a + b)/2;
6:         fx = f(x);
7:         if fp == 0 then
8:             Break
9:         else if fa * fp < 0 then
10:            b = p;
11:        else
12:            a = p;
13:        end if
14:    end for
15:    return p
16: end function
```

Observations

- The bisection method is rather slow to converge: it needs around 2.3 steps to gain one decimal significant digit of accuracy.
- The computational bottleneck is usually the computation of the function $f(p)$, which is needed once per iteration.
- This is one of very few methods with guaranteed global convergence. That there is no equivalent for systems of equations is all the more sad.
- A few variations:
  1. One may prefer (for floating-point accuracy) to compute the quantity $e_n = \frac{b_n - a_n}{2}$, form the midpoint as $x_n = a_n + \frac{b_n - a_n}{2}$ and use $e_n < \varepsilon$ in the loop as a stopping criterion instead of computing N beforehand.
  2. A different stopping criterion altogether, similarly to that used in Newton or secant methods, reads $|fp| < tol$ where tol is a prescribed error tolerance. For the bisection method however, this is not necessary as we control directly the absolute error on the root (a unique feature), and in fact this modification would unnecessarily lose a key advantage of the bisection method: it is not sensitive to ill-conditioning of the root when $|f'(p)| \approx 0$.

5.2.2 Newton's method

A common factor to a different class of methods is to build a linear approximation of the problem at each step, which can be solved easily to get a (hopefully) better approximation based on the current one. Based on the Taylor expansion

$$f(p) = f(x) + f'(\xi)(p - x)$$

between the root p and some approximation x with $\xi \in [p, x]$, the idea is to use an approximation of the unknown slope $f'(\xi)$.

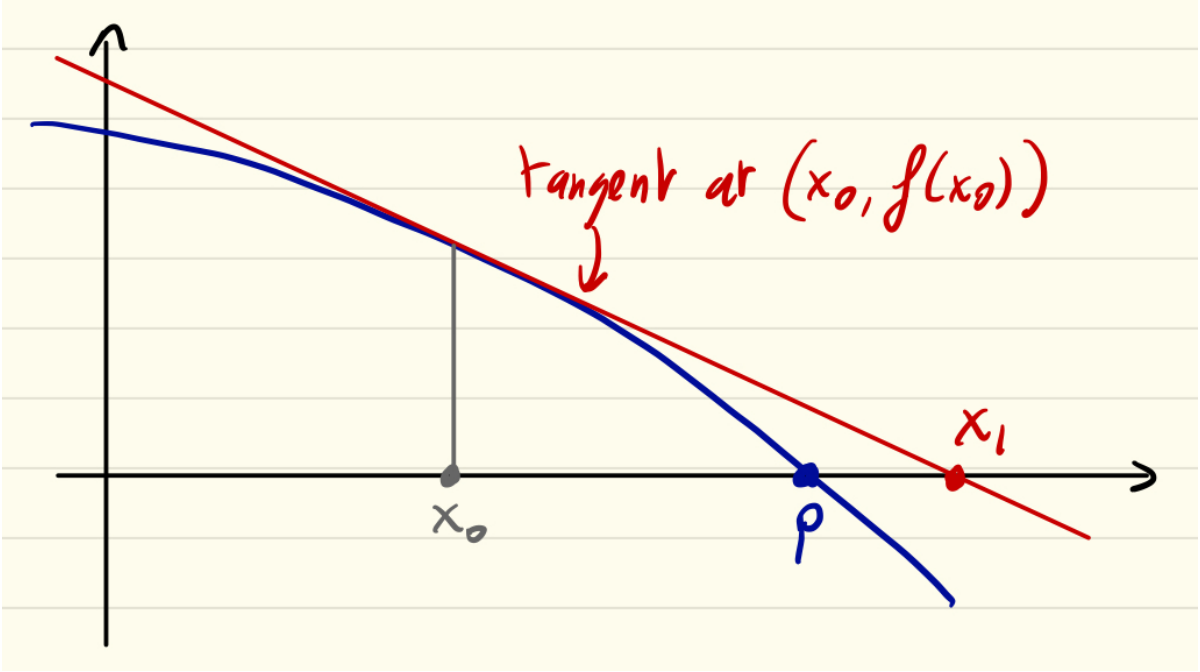
Iteration idea: given a current approximation x_k of the root p and q_k of the slope $f'(\xi)$, we can solve the linear equation

$$f(x_k) + q_k(x_{k+1} - x_k) = 0 \qquad \text{to obtain} \quad x_{k+1} = x_k - f(x_k)/q_k.$$

Newton's method A most prominent member of this class of methods is the so-called Newton-Raphson iteration, where we take the slope $q_k = f'(x_k)$. This leads to the Newton method:

$$x_{k+1} = x_k - f'(x_k)^{-1} f(x_k), \qquad k \geqslant 0.$$

The idea behind the method is to approximate the graph of the function around x_k by its best linear approximant: the tangent, and use it to construct the next approximation to the root.



Property 5.6. *If $f \in C^2(I)$, given x_1 close enough to a root p the Newton method constructs a sequence converging quadratically to p (order 2) if p is a simple root and linearly if it has multiplicity $m > 1$.*

The proof of this result is reserved for next week's lecture!

Cost analysis. Each iteration of Newton's method costs two function evaluations (f and f') and a few additional floating-point operations, which usually add a negligible cost.

Lecture 6: Rootfinding (continued, again). Fixed Points. (Wednesday, September 9)

6.1 Newton's method; methods of the Secant, chord and Regula Falsi.

We continue our exploration of rootfinding methods based on the common idea

$$f(x) \approx f(\underbrace{x_k}_{\text{approximation to the root}}) + \underbrace{q_k}_{\text{approximation to the slope}}(x - x_k)$$

where solving for the root on the right-hand side leads to the iteration idea

$$x_{k+1} = x_k - f(x_k)/q_k.$$

6.1.1 Newton's method

The idea is here to take $q_k = f'(x_k)$. This leads to the algorithm:

Algorithm 2 Newton method.

Input: Function f, derivative f', x_{old}, desired accuracy tol, maximum number of iterations $Nmax$.

Output: Approximate value p.

```
1: function NEWTON(f, f', x_old, tol, Nmax)
2:     for n = 1 ... Nmax do
3:         x_new = x_old − f(x_old)/f'(x_old);
4:         if |x_new − x_old| < tol then return x_new
5:         end if
6:         x_old = x_new
7:     end for
8:     return Error('Method failed after Nmax iterations.')
9: end function
```

6.1.2 Secant method.

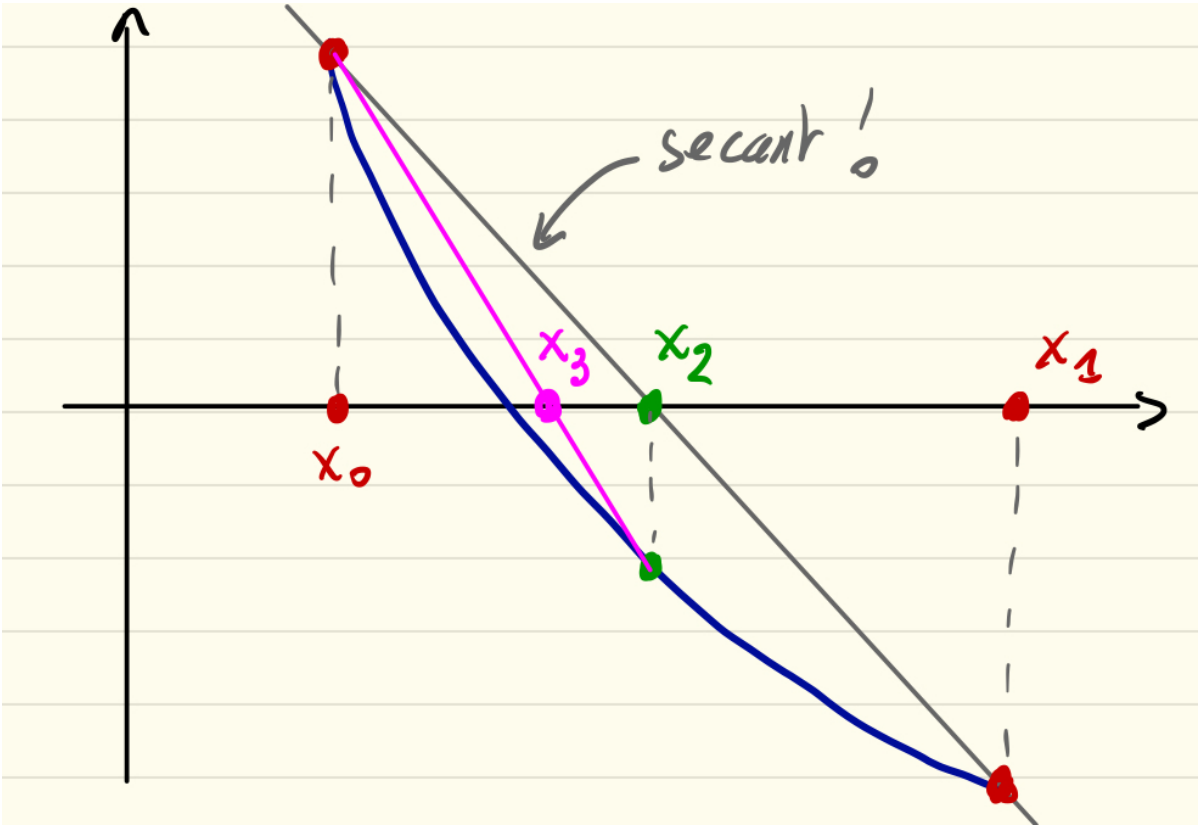
Similar to the bisection, we use here *two initial points* x_0 and x_1 to construct an approximation to the slope with the rule:

$$q_k = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}, \qquad \forall k \geqslant 1.$$

This leads to the two-point iteration:

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

Geometrically, this corresponds to the following scheme:



Property 6.1. *Given $f \in C^2(I)$ and two initial points x_0, x_1 close enough to a root $p \in I$, the secant method converges superlinearly with order $\Phi = \frac{1+\sqrt{5}}{2}$.*

Proof. Future lecture. □

Cost analysis. Each iteration of secant method costs one function evaluation of f and a few additional floating-point operations, which usually add a negligible cost. This means the secant method may well be faster (in wall-clock time) than the Newton method, since one can run almost 2 iterations of the secant method ($e_{n+2}^{secant} \sim (e_n^{secant})^{\Phi^2}$) in the same time one runs a single iteration of the Newton method ($e_{n+1}^{Newton} \sim (e_n^{Newton})^2$) - and since $\Phi^2 \approx 2.6 > 2$, the secant method may emerge as the winner of the race (provided round-off errors, etc. do not diminish the effective convergence rate).

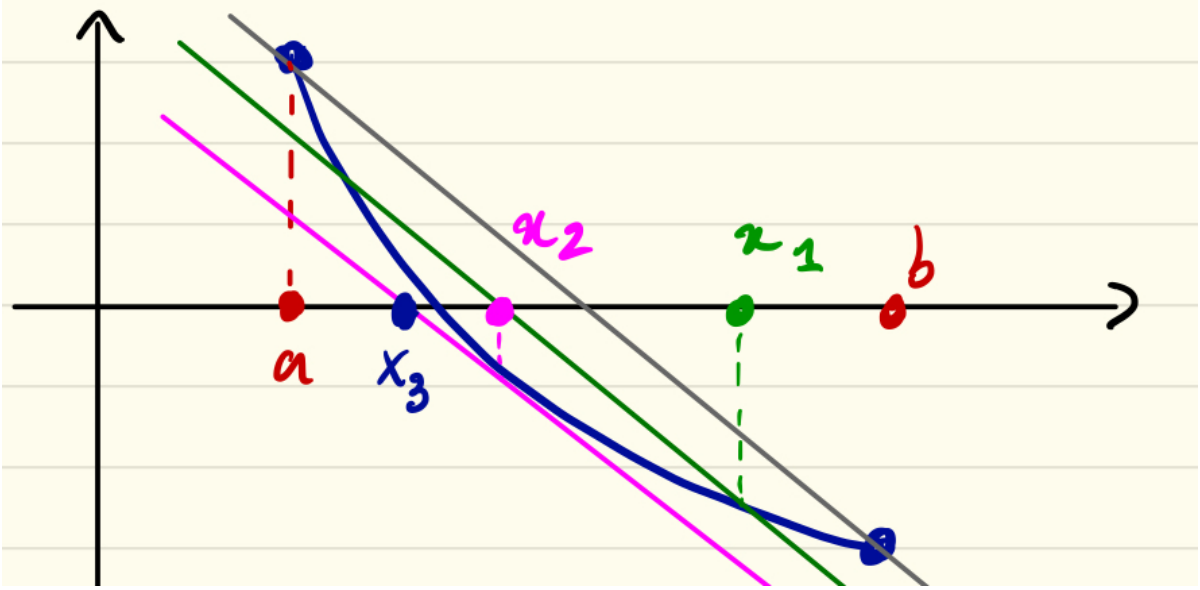
6.1.3 Chord method

Here, we keep a fixed value of the slope all along, using the slope from the chord between $(a, f(a))$ and $(b, f(b))$:

$$q_k = q = \frac{f(b) - f(a)}{b - a},$$

leading to the iteration

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



Property 6.2. *The chord method exhibits locally linear convergence.*

6.1.4 Regula Falsi.

This is a very old method, that we can see as a hybrid of the secant and bisection method. The idea is to keep the *bracketing* property of the bisection method by forming a sequence of pairs of points at which the function takes opposite signs. Formally, this writes as

$$q_k = \frac{f(x_k) - f(x_{k'})}{x_k - x_{k'}}, \qquad \forall k \geqslant 1.$$

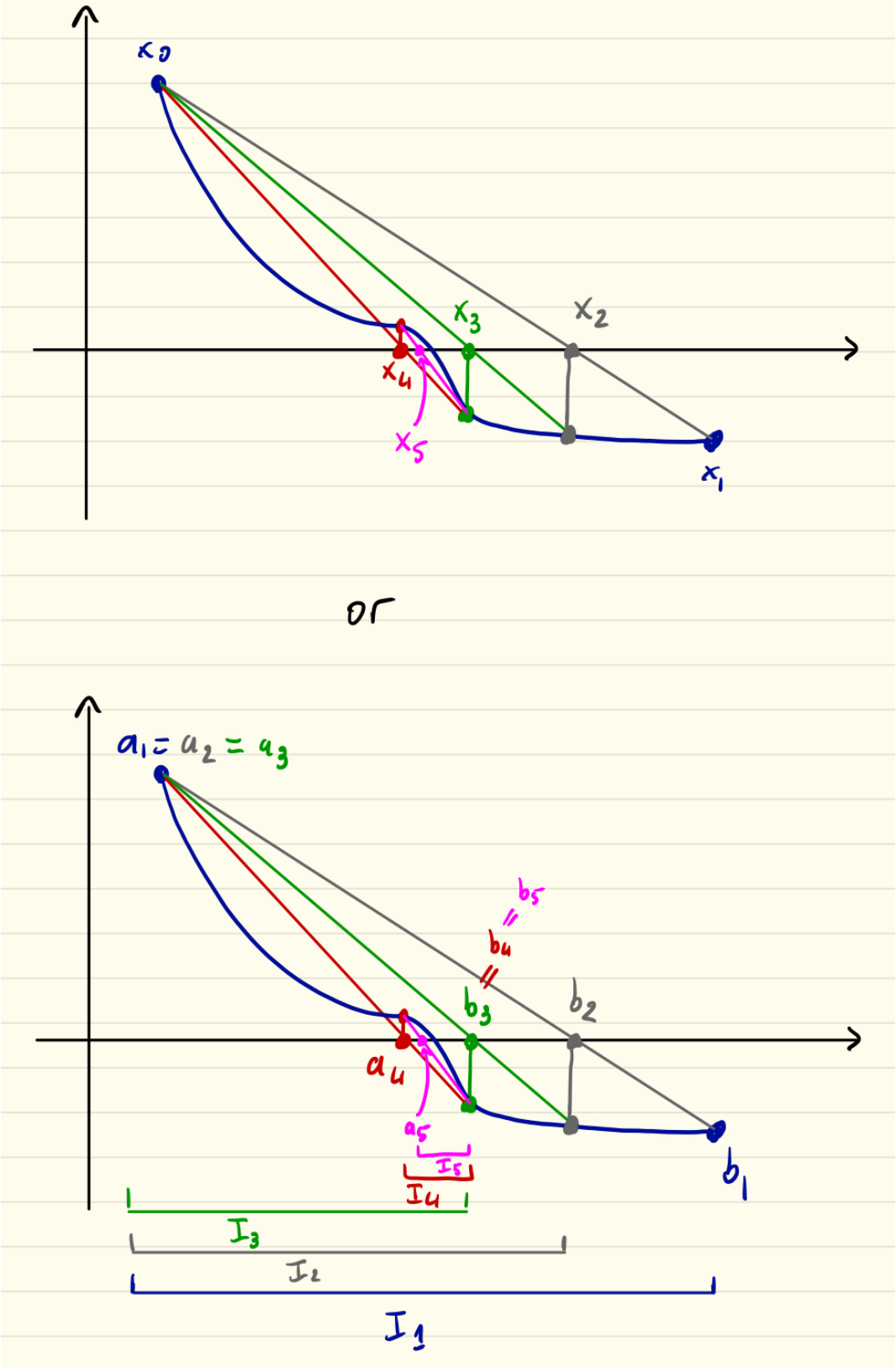
where k' is the largest index such that $f(x_k)f(x_{k'}) < 0$.
Another way to formulate this method is to see it as a bisection method where, instead of checking the midpoint $x_n = \frac{a_n+b_n}{2}$ we check the intersection of the secant between a_n and b_n with the horizontal axis:

Initialization: Let $a_1 = a$, $b_1 = b$ such that $f(a_1)f(b_1) < 0$.
Compute the root x_1 of the secant, $x \mapsto f(b_1) + \frac{f(b_1)-f(a_1)}{b_1-a_1}(x - b_1)$:

$$x_1 = b_1 - \frac{b_1 - a_1}{f(b_1) - f(a_1)} f(b_1) = \frac{a_1 f(b_1) - b_1 f(a_1)}{f(b_1) - f(a_1)}.$$

1. Compute $f(x_1)$.
2. If...
   - $f(x_1) = 0$: x_1 is a root, done.
   - $f(a_1)f(x_1) < 0$: we set $a_2 = a_1$ and $b_2 = x_1$.
   - $f(x_1)f(b_1) < 0$: we set $a_2 = x_1$ and $b_2 = b_1$.
3. Compute $x_2 = \dfrac{a_2 f(b_2) - b_2 f(a_2)}{f(b_2) - f(a_2)}$
4. Repeat steps 1-3, generating $a_3, b_3, \ldots, a_n, b_n, \ldots$...

This method has global convergence, and it is sometimes faster than the bisection method because it uses some information about the function to guess where the root might be. However, it can also be slower than the bisection method in some cases, limiting its usefulness in practice.



6.2 Analysis framework: the fixed-point iterations.

6.2.1 Fixed points.

Definition 6.3. *Given a function $g(x)$, a point p such that $g(p) = p$ is a fixed point of g.*

Example. Let $g(x) = x^2 - 2$. Fixed points of g are solutions of

$$g(x) = x \qquad \Leftrightarrow \qquad x^2 - x - 2 = 0 \quad \Leftrightarrow \quad (x-2)(x+1) = 0.$$

Hence g has two fixed points: 2 and -1.

A fundamental observation is that a fixed point problem can always be transformed into a root-finding problem and vice-versa. Indeed,

$$p \text{ fixed point of } g(x) \quad \leftrightarrow \quad p \text{ root of } f(x) = x - g(x).$$

Theorem 6.4 (Existence of fixed points.)**.** *Suppose $g(x) \in C([a,b])$ such that $a \leqslant g(x) \leqslant b$ for any $x \in [a,b]$. Then g has at least one fixed point on $[a,b]$.*

Proof. Let $f(x) = x - g(x)$, then $f(a) = a - g(a) < 0$ and $f(b) = b - g(b) > 0$ since $a \leqslant g(x) \leqslant b$. Hence, by the Intermediate Value Theorem, there exists a root p of $f(x)$ in (a,b), that ia a fixed point of g. □

Definition 6.5. *A function $g : [a,b] \to [a,b]$, Lipschitz continuous with constant $0 < K < 1$:*

$$|g(x) - g(y)| \leqslant K|x - y|,$$

is called a contraction mapping.

Proposition 6.6. *If $g : [a,b] \to [a,b]$ with $g \in C^1([a,b])$ and $|g'(x)| \leqslant K < 1$ for all $x \in [a,b]$, then g is a contraction mapping.*

Proof. Fix any $x, y \in [a,b]$. By Taylor expansion:

$$f(x) = f(y) + f'(\xi)(x - y) \qquad \text{for some } \xi \in [x,y].$$

Hence

$$|f(x) - f(y)| = |f'(\xi)||x - y| \leqslant K|x - y|.$$

□

Theorem 6.7. *Let $g : [a,b] \to [a,b]$ be a contraction mapping. Then g has a unique fixed point p.*

Proof.

- Existence: OK by the previous theorem.
- Uniqueness: by contradiction. Assume $p_1 \neq p_2$ are two different fixed points of g. Then

$$|p_1 - p_2| = |g(p_1) - g(p_2)| \leqslant K|p_1 - p_2| < |p_1 - p_2|.$$

This is not possible; hence $p_1 = p_2$ and g has a unique fixed point.

□

6.2.2 Fixed point iterations.

Contraction mappings further yield a practical, constructive method for a sequence approaching their fixed point. In general, given a map g and an initial value x_0, we generate a **fixed-point iteration**

$$x_{n+1} = g(x_n), \qquad \text{for } n \geqslant 0.$$

One property of such iterations is that if the sequence (x_n) converges, it must be to a fixed point p: Indeed, we check that since g is continuous,

$$\lim_{n\to\infty} p_n = p \qquad \text{implies} \quad \lim_{n\to\infty} p_{n+1} = p = \lim_{n\to\infty} g(p_n) = g(p).$$

The sequence is in fact guaranteed to converge if the mapping is a contraction:

Theorem 6.8. *Suppose $g(x)$ is a contraction mapping on $[a, b]$. For any $x_0 \in [a, b]$, the sequence generated by the fixed-point iteration*

$$x_{n+1} = g(x_n) \qquad \textit{for } n \geqslant 0$$

converges to the unique fixed point p of $g(x)$. Moreoever, we have the error bounds

$$|p - x_n| \leqslant \frac{K^n}{1-K}|x_1 - x_0|, \qquad \forall n \geqslant 1,$$

and

$$|p - x_n| \leqslant K^n \max(x_0 - a, b - x_0), \qquad \forall n \geqslant 1.$$

Proof. We know that for $n \geqslant 1$,

$$|x_{n+1} - p| = |g(x_n) - g(p)| \leqslant |x_n - p|,$$

so by induction, we obtain immediately

$$|x_n - p| \leqslant K^n |x_0 - p| \qquad \text{for } n \geqslant 0.$$

Since $0 < K < 1$, this shows already that $\lim_{n\to\infty} p_n = p$.
Furthermore,

$$|x_0 - p| \leqslant |x_0 - x_1| + |x_1 - p||x_0 - x_1| + \leqslant K|x_1 - x_0|$$

so that

$$|x_0 - p| \leqslant \frac{1}{1-K}|x_1 - x_0|.$$

This shows that the first error bound holds:

$$|x_n - p| \leqslant K^n \frac{1}{1-K}|x_1 - x_0| \qquad \text{for } n \geqslant 0.$$

Similarly, because $|p - x_0| \leqslant \max(p - a, b - p)$ we have

$$|x_n - p| \leqslant K^n \max(x_0 - a, b - x_0) \qquad \text{for } n \geqslant 0.$$

□

Lecture 7: Analysis of Rootfinding Methods. (Monday, September 14)

7.1 Convergence analysis of fixed-point iterations (cont.)

The theorem proved at the end of the previous lecture shows that *fixed-point iterations generated by contraction mappings $x \mapsto g(x)$ converge at least linearly to a unique fixed point p, with convergence factor $K \approx g'(p)$*. Let us now investigate and quantify the higher order of convergence expected when $g'(p) = 0$.

Proposition 7.1. *Let g be a contraction mapping on $I = [a, b]$ with fixed point p. Assume $g \in C^m(I)$, $m \geqslant 2$ with I a suitable neighborhood of p with*

$$g^{(i)}(p) = 0, \quad 1 \leqslant i < m \qquad \textit{and} \qquad g^{(m)}(p) \neq 0,$$

then the fixed-point iteration process converges to p with order m, and

$$\lim_{k\to\infty} \frac{x_{k+1} - p}{(x_k - p)^m} = \frac{g^{(m)}(p)}{m!}.$$

Proof. We know that the fixed-point iteration generates a sequence x_k converging to p. Furthermore, we write a Taylor expansion about p:

$$x_{k+1} - p = g(x_k) - g(p) = \sum_{i=1}^{m-1} \frac{g^{(i)}(p)}{i!}(x_k - p)^i + \frac{g^{(m)}(\xi_k)}{m!}(x_k - p)^m = \frac{g^{(m)}(\xi_k)}{m!}(x_k - p)^m$$

with $\xi_k \in (p, x_k)$. Thus

$$\lim_{k\to\infty} \frac{x_{k+1} - p}{(x_k - p)^m} = \lim_{k\to\infty} \frac{g^{(m)}(\xi_k)}{m!} = \frac{g^{(m)}(p)}{m!}$$

since $\xi_k \to p$ and $g^{(m)}$ is a continuous function. □

Recap. Suppose that p is the unique fixed-point of a function g (not necessarily a contraction) with g' continuous in the neighborhood of p. Let us summarize the properties of the fixed-point iteration sequence $x_{n+1} = g(x_n)$ with initial value x_0.

1. If $|g'(p)| > 1$ and $x_n \neq p \ \forall n$, then the sequence $\{x_n\}$ diverges in general since g is not a contraction around the fixed point.

2. If $|g'(p)| = 1$, the situation is underdetermined - the sequence may or may not converge to p.

3. If $|g'(p)| < 1$, then $x_n \to p$ if x_0 is close enough to p since g is locally a contraction around p. Furthermore, the sequence converges...

   - exactly linearly if $g'(p) \neq 0$,
   - with order m if $g'(p) = \cdots = g^{(m-1)}(p) = 0$ and $g^{(m)} \neq 0$ with $g \in C^m$ in a neighborhood of p.

Algorithm 3 Fixed-point iteration.

Input: Function $g(x)$, x_{old}, desired accuracy tol, maximum number of iterations $Nmax$.

Output: Approximate value of fixed-point p, or an error message.

```
1: function FixedPoint(g, x_old, tol, Nmax)
2:     for n = 1 ... Nmax do
3:         x_new = g(x_old);
4:         if |x_new − x_old| < tol then return x_new
5:         end if
6:         x_old = x_new
7:     end for
8:     return Error('Method failed after Nmax iterations.')
9: end function
```

Examples. Let us investigate the behavior of fixed-point iterations around the fixed point $p = 1$ for all three functions below.

1. $g_1(x) = 1/x$ on the interval $[1/2, 2]$. Here $g_1(1) = 1$, $g_1'(1) = 1$ and we do not know whether fixed-point iterations converge or not. In practice, they do not unless $x_0 = 1$ - the sequence repeats the cycle $x_0, 1/x_0, x_0, 1/x_0, \ldots$.

2. $g_2(x) = x^2 + x - 1$ on the interval $[1/2, 2]$. Here $g_2(1) = 1$, $g_2'(3) = 1$ and fixed-point iterations will diverge.

3. $g_3(x) = x/2 + 1/2x$ on the interval $(0, \infty)$. Here $g_3(1) = 1$, $g_3'(1) = 0$, $g_3''(1) = 1$ so the fixed-point iterations will converge quadratically if the starting point is close enough to 1.

   In fact, we can go further:

   $$g_3'(x) = 1/2 - 1/2x^2 < 1 \quad \text{and} \quad g_3'(x) > -1 \Leftrightarrow 1/2x^2 < 3/2 \Leftrightarrow x > 1/\sqrt{3}.$$

   so $|g_3'(x)| < 1$ for $1/\sqrt{3} < x < \infty$. Hence g is a contraction on any interval $[a, b]$ with $1/\sqrt{3} < a < b$ and the fixed-point iteration will converge for any $x_0 > 1/\sqrt{3}$. Actually, g is decreasing from 0 to 1 so for any $0 < x_0 \leqslant 1/\sqrt{3} < 1$ we can check that $x_1 = g(x_0) > g(1) = 1$ so the fixed-point iteration converges.

   In conclusion, for this function the fixed-point iteration generates a sequence converging quadratically to 1 for any $0 < x_0 < \infty$.

7.2 Application of fixed-point iteration analysis to 1-point rootfinding methods.

Many rootfinding methods can be cast into the model

$$x_{k+1} = g(x_k) \tag{7.1}$$

where the next iterate depends in some way on the previous one. Note that this is not the case for the secant method, which relies on the previous two iterates. We call methods following the model (7.1) 1-point rootfinding methods.

Chord method. The iteration for the chord method follows the model

$$x_{k+1} = x_k - \underbrace{\left(\frac{b-a}{f(b)-f(a)}\right)}_{q^{-1}} f(x_k),$$

with a, b two points chosen at the start. This can be seen as the fixed-point iteration with iteration function $g(x) = x - q^{-1}f(x)$, which satisfies $g(p) = p$ iff $f(p) = 0$. Then

$$g'(p) = 1 - q^{-1}f'(p),$$

so the convergence is guaranteed (locally) if

$$-1 < g'(p) < 1 \quad \Leftrightarrow \quad 0 < q^{-1}f'(p) < 2.$$

Hence the slope approximation q must have the same sign as $f'(p)$ and

$$|f'(p)| < 2\left|\frac{f(b)-f(a)}{b-a}\right|.$$

Newton method. The iteration for the Newton method follows the model

$$x_{k+1} = x_k - f(x_k)/f'(x_k),$$

which is a fixed-point iteration with function $g(x) = x - f(x)/f'(x)$. Assuming that p is a simple root of f, i.e. $f'(p) \neq 0$, we compute

$$g'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2} \qquad \Longrightarrow \qquad g'(p) = 0.$$

Furthermore

$$\begin{aligned} g''(x) &= \frac{(f'''(x)f(x) + f''(x)f'(x))(f'(x))^2 - f(x)f''(x)(2f''(x)f'(x))}{(f'(x))^4} \\ &= \frac{f(x)f'(x)f'''(x) + (f'(x))^2 f''(x) - 2f(x)(f''(x))^2}{(f'(x))^3} \qquad \Longrightarrow\ g''(p) = \frac{f''(p)}{f'(p)}. \end{aligned}$$

Assuming p is a simple root, that is $f'(p) \neq 0$, the Newton method will have quadratic order of convergence if $f''(p) \neq 0$, at higher order of convergence (at least 3) if $f''(p) = 0$.
On the other hand, if the root has multiplicity $m > 1$, then the convergence is only linear (Homework). However, we can recover quadratic convergence (at least) with the modified iteration

$$x_{k+1} = x_k - mf(x)/f'(x),$$

which assumes advance knowledge of the multiplicity m.

7.3 Analysis of two-point rootfinding methods

The proof of convergence for the Secant method is postponed to the next lecture.

7.4 Discussion: stopping criteria.

We often have to decide, based on computed or computable quantities, if the iteration has converged. Optimally, we would like to know whether a certain accuracy has been reached, that is given a prescribed tolerance $\varepsilon > 0$ we have reached

$$|x_k - p| = |e_k| < \varepsilon.$$

Option 1. Rarely, the method gives direct control over the absolute error $e_k = x_k - p$, a.k.a. yields a computable quantity ε_k which satisfies

$$|e_k| \leqslant \varepsilon_k.$$

The bisection method is such an example, since $|x_k - p| \leqslant \varepsilon_k := |b_k - a_k|/2$. In such as case, we can terminate the iteration at the first step where

$$\varepsilon_k < \varepsilon.$$

Option 2. Direct control of the residual. We may choose to terminate at the first step where

$$|f(x_k)| < \varepsilon.$$

$\oplus$ Simple test, straightforward to implement. Seems reasonable.

$\ominus$ This test may be too optimistic (if the problem is ill-posed) or pessimistic.

More precisely, assuming that p is a root of order m, we know that

$$f(x_k) - f(p) = \frac{f^{(m)}(p)}{m!}(x_k - p)^m + o((x_k - p)^{m+1}),$$

so since $f(p) = 0$ and $f^{(m)}(p) \neq 0$, we have

$$|e_k| \approx \left|\frac{m!}{f^{(m)}(p)}\right|^{1/m} |f(x_k)|^{1/m}.$$

- For multiple roots $m \geqslant 2$, the residual test is very misleading about the actual order of magnitude of the error.
- For a simple root, we have
$$|e_k| \approx \frac{1}{|f'(p)|}|f(x_k)|.$$
  - If $|f'(p)| \approx 1$, then $|e_k| \approx |f(x_k)|$. The residual test works great.
  - If $|f'(p)| \ll 1$, then $|e_k| \gg |f(x_k)|$. Hence the test is unreliable and too optimistic.
  - If $|f'(p)| \gg 1$, then $|e_k| \ll |f(x_k)|$. Hence the test is too pessimistic and will lead to more iterations than necessary.

Option 3. Control of the increment size. We may choose to terminate at the first step where

$$|x_{k+1} - x_k| < \varepsilon.$$

As it turns out, this test is better conditioned than the previous one, despite not involving the actual residual $f(x_k)$. Let us analyze this test in the context of the fixed-point iteration analysis with

$$x_{k+1} = g(x_k),$$

which can be applied to all 1-point rootfinding methods. Then

$$e_{k+1} = x_{k+1} - p = g(x_k) - g(p) = g'(\xi_k)(x_k - p) = g'(\xi_k)e_k.$$

Now $x_{k+1} - x_k = e_{k+1} - e_k = (g'(\xi_k) - 1)e_k$. Since $\xi_k \to p$ as $k \to \infty$ and g' is continuous, we obtain

$$|e_k| \approx \frac{|x_{k+1} - x_k|}{|g'(p) - 1|}.$$

Remember that convergence is expected iff $-1 < g'(p) < 1$. Hence,

- The test is unreliable (too optimistic) if $g'(p) \approx 1$, however in this case the convergence is very slow anyway.
- The test is optimal if $g'(p) = 0$ since in this case $|e_k| \approx |x_{k+1} - x_k|$. This is the case of quadratically converging methods such as the Newton method. This is also independent of the root conditioning.
- The test is satisfactory if $-1 < g'(p) < 1/2$, in which case $1/2|x_{k+1} - x_k| \lessapprox |e_k| \lessapprox 2|x_{k+1} - x_k|$.

Lecture 8: Analysis of the Secant method. (Wednesday, September 16)

The secant method is an example of Quasi-Newton method, i.e. it avoids to compute the derivative yet approximates it along the iteration, more and more accurately. Remember the scheme,

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k). \tag{8.1}$$

Alternatively, we have the equivalent formula

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

Such a method, which relies on the last two iterates to construct the next one, does not fit into the framework of fixed-point iterations we have relied on to understand Newton's method. It is more difficult to analyze!
In this lecture, we will prove the following result.

Theorem 8.1. *Let I be an open interval of $\mathbb{R}$, and assume that $f \in C^2(I)$ has a simple root $p \in I$. Then if x_0, x_1 are close enough to p, the sequence $\{x_k\}$ generated by (8.1) converges to p as $n \to \infty$, and setting $e_k = x_k - p$ we have*

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k e_{k-1}} = \frac{f''(p)}{2f'(p)}.$$

Furthermore, the convergence rate of the sequence x_k to p is superlinear with order $p = \frac{1+\sqrt{5}}{2}$.

We will divide the proof of this Theorem in four pieces. First, we prove a result showing that the slope of the secant of any two points close enough to the root is an approximation that is as good as desired to the slope of the tangent at any point also close enough to the root:

Lemma 8.2. *For any $0 < c < 1$, there exists $\delta > 0$ such that in the interval $I_\delta = \{x \in I \text{ s.t. } |x - p| < \delta\}$,*

$$\left| \left(\frac{x - y}{f(x) - f(y)} \right) f'(\xi) - 1 \right| < c$$

for all $x, y, \xi \in I_\delta$ with $x \neq y$.

Proof. As $\delta \to 0$, by the squeeze theorem we have $x, y, \xi \to p$ so

$$\lim_{\delta\to 0} \frac{f(x) - f(y)}{x - y} = \lim_{\delta\to 0} f'(\xi) = f'(p) \neq 0.$$

Hence for any $c > 0$, we can find δ small enough to ensure the desired result. □

Armed with this result, we next show that the sequence $\{x_k\}$ generated by the secant method converges.

Lemma 8.3. *Fix $0 < c < 1$, $\delta > 0$ and I_δ as per Lemma 1. Then if $x_0, x_1 \in I_\delta$, $x_k \in I_\delta$ for all $k \geqslant 0$, and*

$$\lim_{k\to\infty} x_k = p.$$

Notation. To facilitate some proofs, we use the shorthand $f_k = f(x_k)$, $f'_k = f'(x_k)$, ..., and $e_k = x_k - p$.

Proof. By Taylor's formula applied at x_k about p with exact remainder, there exists $\xi_k \in (p, x_k)$ such that (remember $f(p) = 0$):

$$f(x_k) = f(p) + f'(\xi_k)(x_k - p) \quad \Longrightarrow \quad f_k = f'(\xi_k)e_k,$$

hence

$$\left(\frac{x_k - x_{k-1}}{f_k - f_{k-1}}\right) f_k - (x_k - p) = \left[\left(\frac{x_k - x_{k-1}}{f_k - f_{k-1}}\right) f'(\xi_k) - 1\right] e_k.$$

Now we note that $x_{k+1} = x_k - \left(\frac{x_k - x_{k-1}}{f_k - f_{k-1}}\right) f_k$, so we have shown that

$$-e_{k+1} = \left[\left(\frac{x_k - x_{k-1}}{f_k - f_{k-1}}\right) f'(\xi_k) - 1\right] e_k.$$

By lemma 1, whenever $x_k, x_{k-1} \in I_\delta$, since $\xi_k \in (p, x_k) \subset I_\delta$, we have then

$$|x_{k+1} - p||e_{k+1}| < c|e_k| < c\delta.$$

Since $c < 1$ this shows that $x_{k+1} \in I_\delta$, and by recurrence, whenever $x_0, x_1 \in I_\delta$, then $x_k \in I_\delta$ for all $k \geqslant 0$. Furthermore, the error satisfies $|e_k| \leqslant c^k|e_0| < c^k\delta$ for all $k \geqslant 1$ and converges to zero. Hence we have shown that the secant method produces a linearly (at least) convergent sequence:

$$\lim_{k\to\infty} x_k = p.$$

□

Next, we show that a particular ratio of the errors converges to a simple quantity.

Lemma 8.4. *Fix $0 < c < 1$, $\delta > 0$ and I_δ as per Lemma 1. Then if $x_0, x_1 \in I_\delta$,*

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k e_{k-1}} = \frac{f''(p)}{2f'(p)}.$$

Proof. We start by recalling the alternative iteration formula

$$x_{k+1} = \frac{x_{k-1}f_k - x_k f_{k-1}}{f_k - f_{k-1}},$$

and similarly the sequence of errors satisfies

$$e_{k+1} = x_{k+1} - p = \frac{e_{k-1}f_k - e_k f_{k-1}}{f_k - f_{k-1}}.$$

Now, using a 2nd order Taylor expansion about p, we find

$$f_k = \underbrace{f(p)}_{=0} + f'(p)e_k + \frac{1}{2}f''(\xi_k)e_k^2, \qquad f_{k-1} = f(p) + f'(p)e_{k-1} + \frac{1}{2}f''(\xi_{k-1})e_{k-1}^2,$$

where $\xi_k \in (p, x_k)$, and we compute

$$\begin{aligned} e_{k-1}f_k - e_k f_{k-1} &= \underbrace{f'(p)(e_{k-1}e_k - e_k e_{k-1})}_{=0} + \frac{1}{2}\left(f''(\xi_k)e_{k-1}e_k^2 - f''(\xi_{k-1}e_{k-1}^2 e_k\right) \\ &= \frac{e_k e_{k-1}}{2}\left(f''(\xi_k)e_k - f''(\xi_{k-1})e_{k-1}\right) \end{aligned}$$

Continuing this computation and using the identity $x_k - x_{k-1} = e_k - e_{k-1}$:

$$\begin{aligned} \frac{e_{k-1}f_k - e_k f_{k-1}}{e_k e_{k-1}(x_k - x_{k-1})} &= \frac{f''(p)}{2} + \frac{1}{2}\frac{(f''(\xi_k) - f''(p))e_k - (f''(\xi_{k-1}) - f''(p))e_{k-1}}{e_k - e_{k-1}} \\ &= \frac{f''(p)}{2} + \frac{1}{2}\left[(f''(\xi_k) - f''(p))\frac{e_k}{e_k - e_{k-1}} - (f''(\xi_{k-1}) - f''(p))\frac{e_{k-1}}{e_k - e_{k-1}}\right]. \end{aligned}$$

Let us study the terms in the right-hand side of this equality. First, we note that

$$\lim_{k\to\infty} f''(\xi_k) - f''(p) = \lim_{k\to\infty} f''(\xi_{k-1}) - f''(p) = 0,$$

since f'' is continuous and $\lim_{k\to\infty} \xi_k = p$. Because $|e_k| < c|e_{k-1}|$ (see Lemma 1's proof),

$$\left|\frac{e_k}{e_k - e_k - 1}\right| = \left|\frac{e_k/e_{k-1}}{e_k/e_{k-1} - 1}\right| < \frac{c}{1-c} \quad \text{and} \quad \left|\frac{e_{k-1}}{e_k - e_{k-1}}\right| = \left|\frac{1}{e_k/e_{k-1} - 1}\right| < \frac{1}{1-c}.$$

As a consequence, we have the limit

$$\lim_{k\to\infty} \frac{e_{k-1}f_k - e_k f_{k-1}}{e_k e_{k-1}(x_k - x_{k-1})} = \frac{f''(p)}{2}.$$

To conclude, we now take a look at the desired quantity:

$$\frac{e_{k+1}}{e_k e_{k-1}} = \frac{e_{k-1}f_k - e_k f_{k-1}}{e_k e_{k-1}(x_k - x_{k-1})}\frac{x_k - x_{k-1}}{f_k - f_{k-1}} \to \frac{f''(p)}{2} \times \frac{1}{f'(p)}.$$

□

The final piece of the puzzle will now show the error goes down with order $\Phi = \frac{1+\sqrt{5}}{2}$, the golden ratio.

Lemma 8.5. *ix $0 < c < 1$, $\delta > 0$ and I_δ as per Lemma 1. Then $e_k = x_k - p \to 0$ with order $\Phi = \frac{1+\sqrt{5}}{2}$ and convergence factor $\left|\frac{f''(p)}{2f'(p)}\right|^{1/\Phi}$ as $k \to \infty$.*

Proof. Taking the log on the ratio $\left|\frac{e_{k+1}}{e_k e_{k-1}}\right|$ we obtain from the previous limit

$$a_{k+1} = a_k + a_{k-1} + C + o(1), \tag{8.2}$$

where $a_k = \log|e_k|$, $C = \log\left|\frac{f''(p)}{2f'(p)}\right|$, and $o(1)$ is a term going to zero as $k \to \infty$.
Observe that the (a_k) forms an approximate Fibonacci sequence, and also $a_k \to -\infty$ since $e_k \to 0$. Let us assume for a moment that

$$a_{k+1} = \Phi a_k + \beta + o(1), \tag{8.3}$$

with constants Φ, β to be determined. Plugging this formula into the earlier relation (8.2) we compute

$$\underbrace{\Phi\overbrace{(\Phi a_{k-1} + \beta + o(1))}^{a_k} + \beta + o(1)}_{a_{k+1}} = \overbrace{(\Phi a_{k-1} + \beta + o(1))}^{a_k} + a_{k-1} + o(1).$$

Rearranging the terms, we find

$$(\Phi^2 - \Phi - 1)a_{k-1} + \Phi\beta - C = o(1).$$

In order for the left-hand side to form a sequence converging to zero even as $a_{k-1} \to -\infty$, we must have the two equalities

$$\Phi^2 - \Phi - 1 = 0 \qquad \text{and} \qquad \Phi\beta - C = 0.$$

This leaves two possible values $\Phi = \dfrac{1 \pm \sqrt{5}}{2}$, but since Φ must be positive for (8.3) to hold true even as a_k and a_{k-1} converge to $-\infty$, we deduce

$$\Phi = \frac{1+\sqrt{5}}{2} \qquad \text{and} \qquad \beta = \frac{C}{\Phi}.$$

In conclusion, if we can prove (8.3) then we have concluded our proof: indeed taking the exponential yields

$$\log|e_{k+1}| = \Phi \log|e_k| + \beta + o(1) \qquad \Longrightarrow \qquad \lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^\Phi} = e^\beta = \left|\frac{f''(p)}{2f'(p)}\right|^{1/\Phi}.$$

We must now verify the assumption (8.3), which is the most technical part of the proof. Let us consider the residual

$$\varepsilon_k = a_k - (\Phi a_{k-1} + \beta),$$

we need to prove that $\varepsilon_k \to 0$, which is equivalent to (8.3). Using the known relation (8.2) we find

$$\begin{aligned}
\varepsilon_{k+1} &= a_{k+1} - (\Phi a_k + \beta) \\
&= (1-\Phi)a_k + a_{k-1} + (\Phi - 1)\beta + o(1) \\
&= \frac{-1}{\Phi}(a_k - \Phi a_{k-1} - \beta) + o(1),
\end{aligned}$$

where we have used the relations $\Phi - 1 = 1/\Phi$ and $C = \Phi\beta$. Hence we have

$$\varepsilon_{k+1} = \frac{-1}{\Phi}\varepsilon_k + o(1).$$

Since $\Phi > 1$, one expects a sequence satisfying such a relation to go to zero, unless the small $o(1)$ term on the right-hand side adds up to enough along the way to perturb convergence. We prove that convergence happens using the definition of the limit. Fix $\varepsilon > 0$. By definition of the notation $o(1)$, since $\varepsilon/2\Phi^2 > 0$ there exists $k_0 \geqslant 0$ large enough such that for $k \geqslant k_0$, we have

$$\left|\varepsilon_{k+1} + \frac{1}{\Phi}\varepsilon_k\right| < \frac{\varepsilon}{2\Phi^2}, \qquad \text{so} \qquad |\varepsilon_{k+1}| < \frac{|\varepsilon_k|}{\Phi} + \frac{\varepsilon}{2\Phi^2}.$$

Let us prove by induction that for $n \geqslant 0$, we have

$$|\varepsilon_{k_0+n}| < \frac{|\varepsilon_{k_0}|}{\Phi^n} + \frac{\varepsilon}{2}.$$

Indeed, this is obviously true for $n = 0$, and then if it is true for $n \geqslant 0$, the following computation shows it also holds true for $n + 1$:

$$|\varepsilon_{k_0+n+1}| < \frac{|\varepsilon_{k_0+n}|}{\Phi} + \frac{\varepsilon}{2\Phi^2} < \frac{1}{\Phi}\left(\frac{|\varepsilon_{k_0}|}{\Phi^n} + \frac{\varepsilon}{2}\right) + \frac{\varepsilon}{2\Phi^2} = \frac{|\varepsilon_{k_0}|}{\Phi^{n+1}} + \left(\frac{\Phi+1}{\Phi^2}\right)\frac{\varepsilon}{2} = \frac{|\varepsilon_{k_0}|}{\Phi^{n+1}} + \frac{\varepsilon}{2},$$

where we have used the relation $\Phi^2 = \Phi + 1$. By the induction principle, we have shown the recurrence hypothesis. Now for n_0 large enough such that $\dfrac{|\varepsilon_{k_0}|}{\Phi^{n_0}} < \dfrac{\varepsilon}{2}$, we find for all $k \geqslant k_0 + n_0$ (and $n = k - k_0 \geqslant n_0$),

$$|\varepsilon_k| = |\varepsilon_{k_0+n}| < \varepsilon.$$

This shows that $\varepsilon_k \to 0$ and ends the proof of Theorem 8.1

□

Lecture 9: Polynomial Interpolation. (Monday, September 21)

9.1 Motivation. Horner's method

An interpolation problem starts with some data pairs

$$(x_i, y_i), \qquad \text{for } i = 0 \dots m.$$

This data may come either

- some experimental data,
- a complex, closed-form function $f(x)$ which may be expensive to evaluate.

Definition 9.1. *A function $\phi(x)$ interpolates $\{y_i\}$ at the nodes $\{x_i\}$ if*

$$\phi(x_i) = y_i, \qquad \forall i = 0 \dots m.$$

We seek such a function, which should be simple to understand and cheap to evaluate. A natural candidate: polynomials!

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0, \qquad a_n \neq 0.$$

Why are polynomials so nice? Let us take a detour and think about how to evaluate polynomials in the cheapest manner (in terms of floating-point operations).

Evaluation of polynomials. Horner's method *Naive way:* we could simply use the natural order, computing in turn

$$P(x) = \overbrace{\underbrace{a_n x^n}_{\text{n multiplications}} + \underbrace{a_{n-1} x^{n-1}}_{\text{n-1 multiplications}} + \cdots + \underbrace{a_1 x}_{\text{1 multiplication}} + \underbrace{a_0}_{\text{0 multiplications}}}^{\text{n multiplications}}.$$

In total, this approach needs $0 + 1 + \cdots + n = \frac{n(n+1)}{2}$ multiplications and n additions.
Better way: the most expensive part of the computation above is the separate computations of the monomial terms x^k, which seems wasteful since $x^{k+1} = x \times x^k$ can be obtained without much more work from the value of x^k. We could thus compute the powers of x first,

$$1, \quad x^1 = x, \quad x^2 = x \times x^1, \quad x^3 = x \times x^2, \dots, x^n = x \times x^{n-1}.$$

The computation of these terms amounts to $n - 1$ total multiplications. Then we need another $n + 1$ multiplications to form the terms $a_k x^k$ and n additions to form $P(x)$. Hence we account in total for $2n$ multiplications and n additions.
Better-er way: Horner's method. Let us rewrite by factoring x recursively:

$$P(x) = a_0 + x\,(a_1 + x\,(a_2 + \cdots + x\,(a_{n-1} + x\underbrace{a_n}_{b_n})\cdots)).$$

$$\underbrace{\underbrace{\underbrace{\underbrace{a_0}_{b_0} + x\,(\underbrace{a_1}_{b_1} + x\,(\underbrace{a_2}_{b_2} + \cdots + x\,(\underbrace{a_{n-1}}_{b_{n-1}} + x\,\underbrace{a_n}_{b_n})\cdots))}_{b_0}}_{b_1}}_{b_2}$$

Grouping the terms inside parenthesis, we form the *nested multiplications*:

$$\begin{aligned} b_n &= a_n, \\ b_{n-1} &= a_{n-1} + x b_n, \\ &\vdots \\ b_0 &= a_0 + x b_1, \end{aligned}$$

and we observe that $b_0 = P(x)$.
The computation can be summed up as the *synthetic division algorithm* for evaluating P at x_0:

$$b_n = a_n, \qquad b_k = a_k + b_{k+1} x_0, \quad \text{for } k = n-1, n-2, \ldots, 0.$$

- This algorithm allows us to compute $P(x_0)$ efficiently, using only n multiplications and n additions.
- If we form the polynomial $Q(x; x_0) = b_1 + b_2 x + \cdots + b_n x^{n-1}$, then we observe

  $$P(x) = b_0 + (x - x_0) Q(x; x_0).$$

  Formally, Q is the result of dividing the polynomial P by the term $(x - x_0)$, which justifies the name of the algorithm.
- The algorithm provides also for fast evaluation of derivatives at x_0, for example $P'(x_0) = Q(x_0; x_0)$: this can be achieved by forming another backwards recursion,

  $$c_n = b_n, \qquad c_k = b_k + c_{k+1} x_0, \quad \text{for } k = n-1, n-2, \ldots, 1.$$

9.2 Polynomial interpolation.

Let's go back to the matter of interpolating data, in the form of $m+1$ pairs (x_i, y_i) indexed by $i = 0 \ldots m$.

Interpolation problem: find a polynomial of degree n interpolating $\{y_i\}$ at the nodes $\{x_i\}$.
A polynomial of degree at most n has $n+1$ free coefficients, $a_0, \ldots, a_n$, while the interpolation constraints $P(x_i) = y_i$ form $m+1$ equations.

- If $m \neq n$, the problem is over or under-determined, and there might be zero or an infinity of solutions.
- if $m = n$, then the problem is well-posed: there exists a unique interpolating polynomial of degree n, as we shall see.

Notation. We shall write $\mathbb{P}_n = \{$ polynomials of degree $\leqslant n\}$.

Theorem 9.2. *Given $n+1$ distinct nodes, say $x_0 < x_1 < \cdots < x_n$, and $n+1$ corresponding values $y_0, \ldots, y_n$, there exists a unique polynomial $P \in \mathbb{P}_n$ such that*

$$P(x_i) = y_i, \qquad \forall i = 0 \ldots n.$$

Proof 1. The monomials $x^0 = 1$, $x^1 = x$, x^2, $\dots$, x^n form a basis of $\mathbb{P}_n$. The above conditions form a linear system for the coefficients of the interpolating polynomial $a_0, \dots, a_n$:

$$\begin{cases} a_0 + x_0 \cdot a_1 + x_0^2 \cdot a_2 + \cdots + x_0^n \cdot a_n = y_0, \\ a_0 + x_1 \cdot a_1 + x_1^2 \cdot a_2 + \cdots + x_1^n \cdot a_n = y_1, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vdots \\ a_0 + x_n \cdot a_1 + x_n^2 \cdot a_2 + \cdots + x_n^n \cdot a_n = y_n, \end{cases}$$

or in matrix form,

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Now a Vandermonde matrix like $X = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix}$ is a very special object in Matrix theory, which has an explicit determinant:

$$\det(X) = \prod_{i>j} (x_i - x_j) \neq 0.$$

As a result, the linear system has a unique solution $a_0, \dots, a_n$ and there is a unique interpolating polynomial. □

Remark 9.3. *Vandermonde matrices are usually very ill-conditionned. The solution of the linear system is thus very sensitive to errors, noise in the data, etc. This indicates that this is not the right way to look at the problem.*

Proof 2. Let us try again by setting up a more appropriate basis for $\mathbb{P}_n$. Let us define

$$\ell_i(x) = \prod_{j=0 j \neq i}^{n} \frac{x - x_j}{x_i - x_j} \qquad \in \mathbb{P}_n.$$

We notice in particular $\ell_i(x_j) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$

Let $P(x) = \sum_{i=0}^n y_i \ell_i(x) \qquad \in \mathbb{P}_n$, then clearly $P(x_j) = \sum_{i=0}^n y_i \delta_{ij} = y_j$.
This already shows that there exists an interpolating polynomial for the data. Next, we want to show its **uniqueness**. This can be inferred from the standard result:

Proposition 9.4. *If $P \in \mathbb{P}_n$ vanishes at $n+1$ distinct points, then P vanishes uniformly.*

Proof. This result is proved by recurrence on n. It is trivial for $n = 0$, corresponding for constants $P(x) = C$ which vanish at some point, $P(x_0) = C = 0$. Then for $n > 0$, the polynomial $P(x)$ vanishing at $n+1$ points $x_0 < \cdots < x_n$ has a derivative $P'(x) \in \mathbb{P}_{n-1}$ which vanishes at n distinct points $t_i \in (x_i, x_{i+1})$. By recurrence, we find that $P'(x) \equiv 0$. Thus P is a constant, which vanishes at some point so is identically zero. □

Then, if P, Q are two polynomials interpolating the data, $R = P - Q$ vanishes at the $n+1$ distinct points $x_0, \dots, x_n$ so $R \equiv 0$ or P, Q are in fact the same polynomial. □

9.3 Lagrange representation.

From the previous proof, we extract a good way to represent the interpolating polynomial. Let us recall the useful basis:

Definition 9.5. *Given distinct nodes $x_0, \dots, x_n$, we define Lagrange interpolation basis functions:*

$$\ell_k(x) = \prod_{i=0 i\neq k}^{n} \frac{x - x_i}{x_k - x_i} = \frac{\omega_{n+1}(x)}{(x - x_k)\omega'_{n+1}(x_k)}$$

where $\omega_{n+1}(x) = (x - x_0) \cdots (x - x_n) = \prod_{i=0}^{n}(x - x_i)$.

Proof of this expression: exercise.

Proposition 9.6 (Lagrange Formula)**.** *The unique interpolating polynomial $P_n \in \mathbb{P}_n$ for the data (x_i, y_i), $i = 0 \dots n$ is given by:*

$$P_n(x) = \sum_{i=0}^{n} y_i \ell_i(x).$$

Lecture 10: Polynomial Interpolation II. (Wednesday, September 23)

We continue our exploration of the properties of interpolating polynomials. As a first step, we are interested in estimating the error between a given function and its interpolant at a set of nodes.

10.1 Error in the Lagrange formulation.

Given a function $f \in C^{n+1}(I)$, can we measure how close to $f(x)$ is an interpolant of the data

$$(x_i,\ f(x_i)), \qquad i = 0 \dots n,$$

for a given set of nodes $x_0, \dots, x_n$.

Definition 10.1 (Interpolation operator.)**.** *If the data is specified by a function, $y_i = f(x_i)$ for some function f, then we denote the corresponding interpolating polynomial $\Pi_n f \in \mathbb{P}_n$, given by*

$$\Pi_n f(x) = \sum_{i=0}^{n} f(x_i)\ell_i(x) \qquad \in \mathbb{P}_n.$$

Theorem 10.2 (Error formula, Lagrange.)**.** *Let $f \in C^{n+1}(I)$ with $I \subset \mathbb{R}$ some interval containing $n+1$ distinct nodes $x_0, \dots, x_n$ and $x \in I$. Then the interpolation error at x is given by*

$$E_n(x) = f(x) - \Pi_n f(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x),$$

where $x \in I$, $\omega_{n+1}(x) = (x - x_0)\cdots(x - x_n) = \prod_{i=0}^{n}(x - x_i)$, and ξ is some point in the smallest interval containing all the points $x_0, \dots, x_n$ and x.

Proof. First, the statement is clearly true if $x = x_i$, since $E_n(x_i) = f(x_i) - \Pi_n f(x_i) = 0$ by construction of the interpolating polynomial, and $\omega_{n+1}(x_i) = 0$ for $i = 0, \dots, n$.
Next, we assume that $x \neq x_i$ for any $i = 0, \dots, n$. Define the function $g(t)$ on the interval I such that

$$g(t) = E_n(t) - \lambda\omega_{n+1}(t),$$

where λ is a constant defined as $\lambda = E_n(x)/\omega_{n+1}(x)$.
Now, $g(x_i) = 0$ for $i = 0, \dots, n$ and $g(x) = 0$. Hence $g(t)$ has $n+2$ distinct zeros, forming $n+1$ pairs spanning as many disjoint open intervals, such that by Rolle's theorem g' has at least $n+1$ distinct zeros. Continuing the argument for g'', etc. we find that $g^{(n+1)}$ has at least one zero

$$\xi \quad \in \quad [\min(x_0, \dots, x_n, x), \max(x_0, \dots, x_n, x)] \qquad \text{such that } g^{(n+1)}(\xi) = 0.$$

Now we observe that $\Pi_n f$ is a polynomial of degree n, hence its $n+1$-th derivative vanishes. Hence

$$E_n^{(n+1)}(\xi) = f^{(n+1)}(\xi).$$

Moreover, we also have

$$\omega_{n+1}^{(n+1)}(\xi) = (n+1)!$$

so

$$g^{(n+1)}(\xi) = f^{(n+1)}(\xi) - \lambda(n+1)! = 0,$$

or using the definition of λ and reordering,

$$E_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x).$$

□

10.2 An example.

Given data pairs $(2,-1)$, $(-1,2)$ and $(1,-6)$, let us assemble the corresponding interpolating polynomial.

- First, we form the Lagrange basis functions:

$$\ell_0(x) = \frac{(x-(-1))(x-1)}{(-2-(-1))(-2-1)} = \frac{(x+1)(x-1)}{3},$$

$$\ell_1(x) = \frac{(x-(-2))(x-1)}{(-1-(-2))(-1-1)} = -\frac{(x+2)(x-1)}{2},$$

$$\ell_1(x) = \frac{(x-(-2))(x-(-1))}{(1-(-2))(1-(-1))} = \frac{(x+2)(x+1)}{6}.$$

- Next, we assemble the interpolating polynomial:

$$P_2(x) = 1 \times \frac{(x+1)(x-1)}{3} + 2 \times -\frac{(x+2)(x-1)}{2} + (-6) \times \frac{(x+2)(x+1)}{6}.$$

10.3 Newton representation

Motivation We already have two representations of the interpolating polynomial already.

- In the standard, monomial basis:

$$P_n(x) = a_n x^n + \cdots + a_0,$$

  where $a_0, \ldots, a_n$ are computed by solving a Vandermonde system. This form would be easy to evaluate using Horner's method, but it is not advisable due to ill-conditioning of the coefficients with respect to the data, which induces in particular huge round-off errors when computing the coefficients or evaluating the polynomial.

- In the Lagrange basis:

$$P_n(x) = y_0\ell_0(x) + \cdots + y_n\ell_n(x).$$

  This expression is mathematically simple, without the need to compute the coefficients through a linear system. However, it is expensive to evaluate, with $O(n^2)$ floating-point operations (although this can be amended by using the barycentric formula).

Because both representations have drawbacks, we introduce a third representation: Newton's formula. It is grounded in the new basis:

$$\{1, \quad (x-x_0), \quad (x-x_0)(x-x_1), \quad \ldots \quad (x-x_0)(x-x_1)\ldots(x-x_{n-1})\}$$

Each element of the basis is denoted, consistent with the notation introduced before,

$$\omega_k(x) = (x-x_0)\cdots(x-x_{k-1}) = \prod_{i=0}^{k-1}(x-x_i).$$

We can then expand the interpolating polynomial in this new basis:

$$P_n(x) = \sum_{k=0}^{n} c_k\omega_k(x).$$

Once the coefficients c_k of the expansion are obtained, this expression can be evaluated efficiently using the generalized Horner's method:

$$P_n(x) = c_0 + (x - x_0)\,(c_1 + (x - x_1)(c_2 + \cdots (c_{n-1} + (x - x_{n-1})\underbrace{\underbrace{\underbrace{b_n}_{b_n}}_{b_{n-1}}}_{b_1})\cdots))}_{b_0}$$

leading to the algorithm

$$b_n = c_n, \qquad b_k = c_k + (x - x_k)b_{k+1}, \quad \text{for } k = n-1, n-2, \ldots, 0.$$

The feasibility of this approach hinges on the availability of a stable method for the computation of the coefficient c_n of the expansion, which we present now. First, let us notice that if we write

$$P_n(x) = P_{n-1}(x) + c_n\omega_n(x), \tag{10.1}$$

where $P_{n-1}(x) = \sum_{k=0}^{n-1} c_k\omega_k(x) \in \mathbb{P}_{n-1}$, then

$$P_{n-1}(x_i) = P_n(x_i) = y_i \qquad \text{for } i = 0, \ldots, n-1,$$

since $\omega_n(x_i) = 0$ for $i < n$ by design. Hence, P_{n-1} is a polynomial of degree $n-1$ interpolating the values $y_0, \ldots, y_{n-1}$ at the n nodes $x_0, \ldots, x_{n-1}$: by uniqueness, it is *the* interpolating polynomial on these nodes. Hence $c_n\omega_n(x)$ is just the correction needed to transform the interpolating polynomial on the nodes $x_0, \ldots, x_{n-1}$ into the interpolating polynomial on the nodes $x_0, \ldots, x_n$, a process we can call *adding a node* to an existing set.
As a consequence, we notice that

- c_n is the leading coefficient (i.e., the coefficient in front of x^n in the monomial basis) of the polynomial P_n interpolating on the nodes $x_0, \ldots, x_n$,
- c_{n-1} is the leading coefficient of the polynomial P_{n-1} interpolating on the nodes $x_0, \ldots, x_{n-1}$,
- c_{n-2} is the leading coefficient of the polynomial P_{n-2} interpolating on the nodes $x_0, \ldots, x_{n-2}$, etc.

This leads us to the definition:

Definition 10.3. *The $k-th$ Newton divided difference, denoted*

$$c_k = f[x_0, \ldots, x_k],$$

is the leading coefficient of the interpolating polynomial P_k through the nodes $x_0, \ldots, x_k$.

Note that the expression "divided difference" for these coefficients will be explained below. In the meantime, a trivial recurrence on the expression above leads to:

Theorem 10.4. *Newton's formula for the polynomial interpolating a function f at distinct nodes $x_0, \ldots, x_n$ is*

$$\begin{aligned} P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ + \cdots + f[x_0, \ldots, x_n](x - x_0)\cdots(x - x_{n-1}) \end{aligned} \qquad = \sum_{k=0}^{n} f[x_0, \ldots, x_k]\omega_k(x). \tag{10.2}$$

10.4 Properties of the Newton divided differences.

Invariance with respect to index permutations: Since P_n does not depend on the order of the points $x_0, \ldots, x_n$, neither does its leading coefficient $c_n = f[x_0, \ldots, x_n]$. Hence given a permutation of the nodes, $\{i_0, \ldots, i_n\} = \{0, \ldots, n\}$, we have

$$f[x_0, \ldots, x_n] = f[x_{i_0}, \ldots, x_{i_n}].$$

Recursive definition: First, we note that for $n = 0$, we have $P(x) = f(x_0)$ the constant polynomial interpolating f at the node x_0. Its leading coefficient is $f(x_0)$, thus

$$f[x_0] = f(x_0).$$

This is a general formula for any 1-point divided difference. Next, let us compute a formula for the 2-point (first-order) divided difference. We find from the Lagrange formula,

$$P_1(x) \quad = \quad f(x_0)\frac{x - x_1}{x_0 - x_1} + f(x_1)\frac{x - x_0}{x_1 - x_0} \quad = \quad \frac{f(x_1) - f(x_0)}{x_1 - x_0}x + f(x_0).$$

We read its leading coefficient:

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

It turns out that this formula can be extended to all higher-order Newton finite differences, taking the form of the following recursion formula allowing to compute each $n+1$-point finite differences using two n-point ones:

$$f[x_i] = f(x_i), \qquad f[x_0, \ldots, x_n] = \frac{f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}]}{x_n - x_0}. \tag{10.3}$$

Thus the coefficients may be efficiently computed by forming a lower triangular table, with the first column being the values $f(x_0), \ldots, f(x_n)$ of the function at the nodes, and each column after that being computed from the values in the previous one using (10.3):

$$\begin{array}{c|ccccccccc}
x_0 & f[x_0] & & & & & & & & \\
 & & \searrow & & & & & & & \\
x_1 & f[x_1] & \to & f[x_0,x_1] & & & & & & \\
 & & \searrow & & \searrow & & & & & \\
x_2 & f[x_2] & \to & f[x_1,x_2] & \to & f[x_0,x_1,x_2] & & & & \\
 & & \searrow & & \searrow & & \searrow & & & \\
\vdots & \vdots & & \vdots & & \vdots & & \ddots & & \\
 & & \searrow & & \searrow & & & & \searrow & \\
x_n & f[x_n] & \to & f[x_{n-1},x_n] & \to & f[x_{n-2},x_{n-1},x_n] & \to & \cdots & \to & f[x_0,\ldots,x_n]
\end{array}$$

The coefficients of the Newton expansion (10.2) are then retrieved from the diagonal of this table: $f[x_0], f[x_0, x_1], \ldots, f[x_0, \ldots, x_n]$.

Proof. Let us now prove the recursive formula (10.3). Given the nodes $x_0, \ldots, x_n$, we define two polynomials of degree $n-1$:

- $P_{n-1}(x)$ interpolates f over the nodes $x_0, \ldots, x_{n-1}$. Hence its leading coefficient is
$$f[x_0, \ldots, x_{n-1}];$$
- $\widetilde{P}_{n-1}(x)$ interpolates f over the nodes $x_1, \ldots, x_n$. Hence its leading coefficient is
$$f[x_1, \ldots, x_n].$$

The polynomial $P_n = \Pi_n f$ interpolating f over the nodes $x_0, \ldots, x_n$ can be formed by adding respectively the node x_n to P_{n-1}, and x_0 to $\widetilde{P}_{n-1}$. By uniqueness and the formula (10.1), reordering the nodes if necessary, we have the identity

$$\begin{aligned} P_n(x) &= P_{n-1}(x) + f[x_0, \ldots, x_{n-1}, x_n](x - x_0) \cdots (x - x_{n-1}) \\ &= \widetilde{P}_{n-1}(x) + f[x_1, \ldots, x_n, x_0](x - x_1) \cdots (x - x_n). \end{aligned}$$

The coefficients in the monomial basis on both sides are equal. The leading term $a_n x^n$ is the same on both sides, that is
$$a_n = f[x_0, \ldots, x_n] = f[x_1, \ldots, x_n, x_0] :$$
this is the permutation invariance mentioned above. We are more interested in the next coefficient $a_{n-1}x^{n-1}$: since

$$\begin{aligned} (x - x_0) \cdots (x - x_{n-1}) &= x^n - (x_0 + \cdots + x_{n-1})x^{n-1} + \textit{lower order terms}, \\ (x - x_1) \cdots (x - x_n) &= x^n - (x_1 + \cdots + x_n)x^{n-1} + \textit{lower order terms}, \end{aligned}$$

the coefficient of P_n in front of x^{n-1} equals

$$\begin{aligned} a_{n-1} &= f[x_0, \ldots, x_{n-1}] - f[x_0, \ldots, x_n](x_0 + x_1 + \cdots + x_{n-1}) \\ &= f[x_1, \ldots, x_n] - f[x_0, \ldots, x_n](x_1 + \cdots + x_{n-1} + x_n). \end{aligned}$$

Reordering terms, we find

$$f[x_0, \ldots, x_n](x_1 + \cdots + x_{n-1} + x_n - x_0 - x_1 - \cdots - x_{n-1}) = f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}],$$

or

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

This is the formula you are looking for. □

Lecture 11: Interpolation Error. Piecewise interpolation. (Monday, September 28)

Let us now try to understand the interpolation error better. Does the Newton approach help in this regard?

11.1 Interpolation error and divided differences.

We have the two following results.

Theorem 11.1. *Assuming $f \in C^{n+1}(I)$ with I some interval, fix distinct nodes $x_0, \dots x_n$ and $x \in I$. Then*

$$E_n(x) = f[x_0, \dots, x_n, x]\omega_{n+1}(x).,$$

where $E_n(x) = f(x) - P_n(x)$ is the interpolation error and $\omega_{n+1}(x) = \prod_{i=0}^{n}(x - x_i)$.

Proof. Interpolating at the $n+2$ nodes $x_0, \dots, x_n, x$ using the Newton formula, we obtain using the notation from the previous lecture,

$$P_{n+1}(t) = P_n(t) + f[x_0, \dots, x_n, x]\omega_{n+1}(t),$$

where $P_n(t)$ is the polynomial interpolating f at the $n+1$ nodes $x_0, \dots, x_n$ and $\omega_{n+1}(t) = (t - x_0)(t - x_1)\cdots(t - x_n)$. Taking $t = x$, we know that $P_{n+1}(x) = f(x)$ since x is an interpolation node for P_{n+1}, hence

$$f(x) - P_n(x) = f[x_0, \dots, x_n, x]\omega_{n+1}(x).$$

□

Corollary 11.2. *Given $f \in C^{n+1}(I)$ with I some interval, fix distinct nodes $x_0, \dots x_n \in I$. Then*

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

for some $\xi \in [\min(x_0, \dots, x_n), \max(x_0, \dots, x_n)]$.

Proof. We apply the previous result with the $m + 1 = n$ nodes $z_0, \dots, z_m = x_0, \dots, x_{n-1}$ and $z = x_n$. Then using the Lagrange interpolation result, Theorem 10.2,

$$E_m(z) = f[z_0, \dots, z_m, z]\omega_{m+1}(z) = \frac{f^{(m+1)}(\xi)}{(m+1)!}\omega_{m+1}(z),$$

where $\xi \in [\min(z_0, \dots, z_{n-1}, z), \max(z_0, \dots, z_m, z)]$ and $\omega_{m+1}(z) = (z - z_0)\cdots(z - z_m)$. Since $x_n = z$ is distinct from all the other nodes, $\omega_{m+1}(z) \neq 0$, and we identify

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

as desired. □

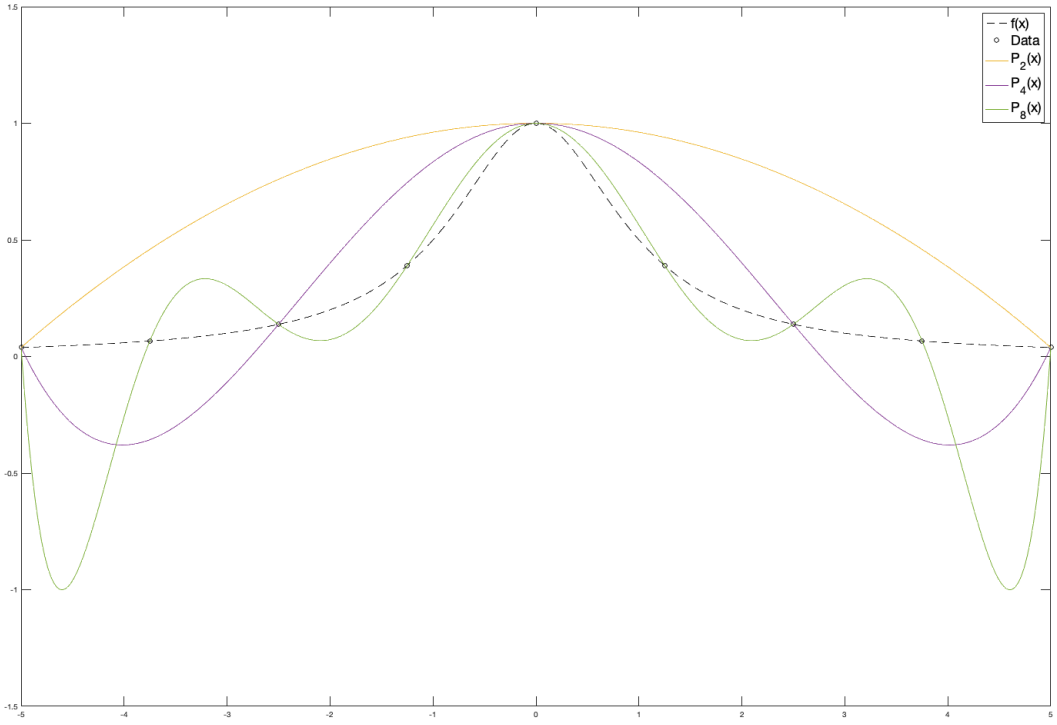
These two results clarify how divided differences are related to the derivatives of f. On the other hand, they do not shine much light on the behavior of the interpolation error $E_n(x)$, and we must use another approach to understand it.

11.2 Error investigation: Equidistant points.

The classical counter-example is Runge's function,

$$f(x) = \frac{1}{1+x^2} \qquad \text{with } x \in [-5, 5].$$

It is clear from the plots below that increasing the order of the interpolating polynomial, instead of increasing the accuracy of the approximation as expected, leads to large oscillations of the interpolating polynomial near the ends of the interval, $x \approx -5, 5$. This is not due to inaccuracies of the numerical implementation, but a fundamental difficulty in the interpolation process, which may not ensure $\Pi_n f(x) \to f(x)$ uniformly on $[-5, 5]$.



Property 11.3. *For a given set of nodes $X = \{x_0, \ldots, x_n\}$, the interpolation error can be compared to the best possible approximation by a polynomial with the following estimate:*

$$\|f - \Pi_n f\|_\infty \leqslant (1 + \Lambda_n(X)) \min_{P \in \mathbb{P}_n} \|f - P\|_\infty$$

where the best possible constant $\Lambda_n(X) = \left\| \sum_{i=0}^n |\ell_i^{(n)}| \right\|_\infty$ is called the Lebesgue constant of the set of interpolation nodes $x_0, \ldots, x_n$.

It can be shown that for any set of interpolation nodes,

$$\Lambda_n(X) \geqslant 2/\pi \log(n+1) - C,$$

where $C > 0$ is some constant. In particular, this implies that $\Lambda_n(X) \to \infty$. In fact, one can show that there exists always some continuous function such that, for any sequence of interpolating

nodes $X_1, X_2, \ldots$, the corresponding sequence of interpolating polynomials $\Pi_n f$ does not converge uniformly to f, meaning $\|f - \Pi_n f\|_\infty \not\to 0$.
For the particular case of equidistant points, the Lebesgue constant grows *much faster* than the lower bound above:

$$\Lambda_n(X) \approx \frac{2^{n+1}}{en\log(n)}.$$

Interpolation on equidistant nodes is thus very unstable for even moderate values of n, and unlikely to converge unless $f(x)$ can be approximated exponentially well by polynomials in the first place (e.g., if there is $P_n \in \mathbb{P}_n$ such that $\|f(x) - P_n(x)\|_\infty < Cr^n$ with $|r| < 1/2$).

11.3 Piecewise Lagrange interpolation.

Since high-order interpolation schemes may not converge as n increases, one must seek a different way to use interpolation in order to approximate functions accurately. The main idea in this paragraph is to divide the interval of interest in small pieces, and use a low-order interpolant on each piece.

Piecewise Interpolant: Ingredients.

- Interval of interest $[a, b]$,
- Partition in K intervals $I_k = [x_k, x_{k+1}]$ with

  $$a = x_0 < x_1 < \cdots < x_K = b$$

  with $x_{k+1} - x_k = |I_k| = h = \frac{|b-a|}{K}$.

  Remark 11.4. *The sub-intervals do not have to have the same length, but this makes the presentation and analysis somewhat easier to follow.*
- For $n \geqslant 1$, we define the piecewise polynomial space

  $$X_h^n = \{v \in C([a,b]); \quad v|_{I_k} \in \mathbb{P}_n(I_k), \quad k = 0, \ldots, K_1\}.$$

- For $f \in C([a,b])$, we construct the piecewise interpolation polynomial $\Pi_h^n f : \quad [a,b] \to \mathbb{R}$, such that for $x \in I_k$, i.e. $x_k \leqslant x \leqslant x_{k+1}$,

  $$\Pi_h^n f|_{I_k}(x) = \underbrace{\Pi_n f|_{I_k}}_{\text{interpolant with equidistant nodes } x_k^{(0)},\ldots,x_k^{(n)},\ x_k^{(i)} = x_k + ih/n}.$$

Because the ends of the sub-intervals are part of the interpolation nodes, we check that $\Pi_h^n f|_{I_k}(x_k) = f(x_k) = \Pi_h^n f|_{I_{k-1}}(x_k)$ for $1 \leqslant k \leqslant K-1$. Hence $\Pi_h^n f(x_k) = f(x_k)$ is well-defined and $\Pi_h^n f(x)$ is continuous across the interior points x_k separating the intervals I_k.
By construction, the piecewise interpolation polynomial $\Pi_n^h f$ thus belongs to the piecewise polynomial space X_h^n. Using the Lagrange error result on $[x_k, x_{k+1}]$ we obtain

$$|f(x) - \Pi_h^n f(x)| = \left| \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_k^{(0)}) \cdots (x - x_k^{(n)}) \right| \leqslant C(n)\|f^{(n+1)}(\xi)\|_\infty h^{n+1},$$

with $x \in I_k$ such that $|x - x_k^{(i)}| < h$ and $\xi \in I_k$, with $C(n)$ some constant depending only on n. Hence, on the whole interval we have the bound

$$|f(x) - \Pi_h^n f(x)| \leqslant C(n)\|f^{(n+1)}\|_\infty h^{n+1}.$$

For a fixed order n, we can thus ensure convergence $O(h^{n+1}) = O(K^{-(n+1)})$ by increasing the number of sub-intervals, for any smooth function $f(x)$.

11.4 Cubic Splines

Another widely employed scheme to ensure convergence of interpolating approximations, splines seek to enforce smoothness of the interpolant by enforcing continuity of some derivatives on the whole interval of interest. Note that this is not the case of the piecewise Lagrange interpolant above, whose derivative is in general not continuous at the points x_k.

Interpolatory Cubic Splines: Ingredients.

- Interval of interest $[a, b]$,
- Partition in K intervals $I_k = [x_k, x_{k+1}]$ with

$$a = x_0 < x_1 < \cdots < x_K = b$$

  with $x_{k+1} - x_k = |I_k| = h_k$.

- Cubic splines are functions $S(x) : [a, b] \to \mathbb{R}$ such that

$$S \in C^2([a, b]), \qquad S|_{I_k} \in \mathbb{P}_3.$$

  Since there are K intervals and $\dim \mathbb{P}_3 = 4$, each spline is uniquely determined by the knowledge of $4K$ coefficients at most.

- Interpolating splines are subject to the following set of contraints:

  (1) First, the spline must interpolate the data at the nodes x_k:

$$\begin{aligned} S(x_0) &= f(x_0), \\ S|_{I_{k-1}}(x_k) = S|_{I_k}(x_k) &= f(x_k), \quad \text{for } k = 1 \dots K-1, \\ S(x_K) &= f(x_K). \end{aligned}$$

  (2) Next, its first derivative must be continuous accross the nodes:

$$S|'_{I_{k-1}}(x_k) = S|'_{I_k}(x_k), \quad \text{for } k = 1 \dots K-1.$$

  (2) Finally, its second derivative must be continuous accross the nodes:

$$S|''_{I_{k-1}}(x_k) = S|''_{I_k}(x_k), \quad \text{for } k = 1 \dots K-1.$$

  The set (1) forms a total of $2K$ constraints, while (2) and (3) account together for $2K - 2$ constraints. In order to obtain a well-posed problem for the $4K$ coefficients of S, we need an additional 2 constraints. Two common ways to complete the system are:

  (4a) Natural spline:

$$S''(x_0) = S''(x_K) = 0.$$

  (4b) Clamped spline:

$$S'(x_0) = f'(x_0), \qquad S'(x_K) = f'(x_K).$$

Construction of natural splines. A natural, but non-efficient way of constructing the interpolatory spline is to form the system of $4K$ linear constraints above and solve for the $4K$ coefficients of the spline, e.g. using a monomial basis on each interval I_k.
A better approach is as follows. We form the unknowns

$$M_k = S''(x_k), \qquad k = 0 \dots K,$$

which are well-defined quantities because $S \in C^2([a,b])$, accounting for the set of constraints (3). Since S is a cubic polynomial on each I_k, S'' is linear and may be expressed as

$$S''|_{I_k} = M_k \frac{x_{k+1} - x}{h_k} + M_{k+1} \frac{x - x_k}{h_k}$$

Integrating twice leads to

$$S|_{I_k} = \frac{M_k (x_{k+1} - x)^3 + M_{k+1}(x - x_k)^3}{6h_k} + Cx + D,$$

where C, D are two integration constants to be determined. Rewriting

$$Cx + D = A \frac{x_{k+1} - x}{h_k} + B \frac{x - x_k}{h_k},$$

and because of the constraints (1) the new constants A, B must be such that

$$\begin{aligned} S|_{I_k}(x_k) = f(x_k) &= \frac{M_k (x_{k+1} - x_k)^3 + M_{k+1}(x_k - x_k)^3}{6h_k} + A \frac{x_{k+1} - x_k}{h_k} + B \frac{x_k - x_k}{h_k} \\ &= \frac{h_k^2 M_k}{6} + A \end{aligned}$$

so that $A = f(x_k) - \frac{h_k^2 M_k}{6}$ and

$$\begin{aligned} S|_{I_k}(x_{k+1}) = f(x_{k+1}) &= \frac{M_k (x_{k+1} - x_{k+1})^3 + M_{k+1}(x_{k+1} - x_k)^3}{6h_k} + A \frac{x_{k+1} - x_{k+1}}{h_k} + B \frac{x_{k+1} - x_k}{h_k} \\ &= \frac{h_k^2 M_{k+1}}{6} + B \end{aligned}$$

so that $B = f(x_{k+1}) - \frac{h_k^2 M_{k+1}}{6}$. Thus, constraints (1) and (3) have led us to the expression for the spline

$$\begin{aligned} S|_{I_k}(x) = &\frac{M_k (x_{k+1} - x)^3 + M_{k+1}(x - x_k)^3}{6h_k} \\ &+ \left(f(x_k) - \frac{h_k^2 M_k}{6} \right) \frac{(x_{k+1} - x)}{h_k} + \left(f(x_{k+1}) - \frac{h_k^2 M_{k+1}}{6} \right) \frac{(x - x_k)}{h_k}. \end{aligned}$$

It remains to compute the coefficients $M_0, \dots, M_K$ using the remaining constraints (2) and (4a) or (4b). Continuity of the first derivative:

$$S'|_{I_{k-1}}(x_k) = S'|_{I_k}(x_k), \quad \text{for } k = 1 \dots K-1,$$

is implemented by computing the derivative of the expression above:

$$S'|_{I_k}(x) = \frac{-M_k (x_{k+1} - x)^2 + M_{k+1}(x - x_k)^2}{2h_k} - \frac{1}{h_k} \left(f(x_k) - \frac{h_k^2 M_k}{6} \right) + \frac{1}{h_k} \left(f(x_{k+1}) - \frac{h_k^2 M_{k+1}}{6} \right),$$

so after some manipulations,

$$S|'_{I_k}(x_{k+1}) = \frac{f(x_{k+1}) - f(x_k)}{h_k} + \frac{h_k}{6}(M_k + 2M_{k+1}),$$

so that by substituting indices $k \to k-1$ we have

$$S|'_{I_{k-1}}(x_k) = \frac{f(x_k) - f(x_{k-1})}{h_{k-1}} + \frac{h_{k-1}}{6}(M_{k-1} + 2M_k).$$

and also

$$S|'_{I_k}(x_k) = \frac{f(x_{k+1}) - f(x_k)}{h_k} - \frac{h_k}{6}(2M_k + M_{k+1}).$$

Equality between these two quantities then means

$$6\frac{f(x_k) - f(x_{k-1})}{h_{k-1}} + h_{k-1}(M_{k-1} + 2M_k) = 6\frac{f(x_{k+1}) - f(x_k)}{h_k} - h_k(2M_k + M_{k+1}).$$

Defining $b_k = 6\left(\frac{f(x_{k+1})-f(x_k)}{h_k} - \frac{f(x_k)-f(x_{k-1})}{h_{k-1}}\right)$, we find that the coefficients $M_0, \ldots, M_K$ must satisfy:

$$h_{k-1}M_{k-1} + 2(h_{k-1} + h_k)M_k + h_kM_{k+1} = b_k, \qquad k = 1, \ldots, K-1.$$

Complementing these $K-2$ linear equations are the final conditions (4). For the natural spline, we have for example $M_0 = M_K = 0$, which leads to the tri-diagonal "M-continuity" system

$$\begin{bmatrix} 1 & 0 & & & & & \\ h_0 & 2(h_0+h_1) & h_1 & & & & \\ & h_1 & 2(h_1+h_2) & h_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & h_{K-2} & 2(h_{K-2}+h_{K-1}) & h_{K-1} & \\ & & & & 0 & 1 & \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ \vdots \\ M_{K-1} \\ M_K \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{K-1} \\ b_K \end{bmatrix}.$$

Solving this linear system can be done in $O(N)$ operations, using Gaussian elimination! See Math 782 for more details.

11.5 Properties of Cubic Spline Interpolants

Theorem 11.5 (Error bound). *Let $f \in C^4([a,b])$, $M = \|f^{(4)}\|_\infty$. Let $S(x)$ be an interpolatory cubic spline at $K+1$ equally space nodes $x_0, \ldots, x_K$ with $x_{k+1} - x_k = h = \frac{b-a}{K}$. Then:*

- *Natural spline case:*
$$\|f(x) - S(x)\|_\infty \leqslant Ch^2,$$
where C depends on $f''(a)$, $f''(b)$ and M;
- *Clamped spline case:*
$$\|f(x) - S(x)\|_\infty \leqslant \frac{5M}{384}h^4.$$

We skip the proof of this result.

Property 11.6 (curvature minimization). *Let $f \in C^2([a,b])$, and $S(x)$ a natural cubic spline interpolating f. Then*

$$\int_a^b |S''(x)|^2 \leqslant \int_a^b |f''(x)|^2.$$

Lecture 12: Hermite Interpolation. (Monday, October 5)

In this lecture, we seek to generalize interpolation to the case where values of the derivative of f are known at some or all of a set of distinct nodes:

$$\left(x_i, f^{(k)}(x_i)\right), \qquad i = 0 \dots n, \quad k = 0 \dots m_i,$$

where $m_0, \dots m_n \geqslant$ are some natural integers. These data pairs form a total of $N+1 = \sum_{i=0}^{n}(1+m_i)$ constraints.

Property 12.1. *There exists a unique polynomial $P_N \in \mathbb{P}_N$ with $N = n + \sum_{i=0}^{n} m_i$ such that*

$$P_N^{(k)}(x_i) = f^{(k)}(x_i), \qquad \textit{for all } i = 0 \dots n, \quad k = 0 \dots m_i.$$

In order to compute this interpolant in practice, we can employ either a Lagrange- or a Newton-type approach.

12.1 Lagrange-type formula.

In this approach, we find a basis such that

$$P_N(x) = \sum_{i=0}^{n} \sum_{k=0}^{m_i} y_i^{(k)} L_{ik}(x),$$

where $y_i^{(k)} = f^{(k)}(x_i)$. The Hermite characteristic polynomials $H_{ik}(x) \in \mathbb{P}_n$ need to satisfy

$$\frac{d^p H_{ik}}{dx^p}(x_j) = \begin{cases} 1, & \text{if } i = j \text{ and } k = p, \\ 0, & \text{else.} \end{cases}$$

Let us define

$$h_{ij}(x) = \overbrace{\underbrace{\frac{(x-x_i)^j}{j!}}_{h_{ij}^{(p)}(x_i) = 0,\ 0 \leqslant p < j, \text{ and } h_{ij}^{(j)}(x_i) = 1} \underbrace{\prod_{k \neq i} \left(\frac{x - x_k}{x_i - x_k}\right)^{m_k+1}}_{h_{ij}^{(p)}(x_i) = 0,\ 0 \leqslant p \leqslant m_k}}^{\text{polynomial of order} \leqslant N} \in \mathbb{P}_n.$$

By construction, these polynomials have the right derivatives at all points x_k for $k \neq i$, and also at x_i up to order j included. If $j < m_i$, then we must "correct" the remaining derivatives of order $j+1 \dots m_i$ in order to satisfy all the constraints above, which we achieve by the following recursive process.

- First, we set
$$H_{im_i}(x) = h_{im_i}(x), \qquad \text{for } i = 0 \dots n.$$
- Next, we set for $j = m_i - 1, \dots, 0$, assuming that we have computed polynomials $H_{ik}(x)$ with the right properties for all $j < k \leqslant m_i$,
$$H_{ij}(x) = h_{ij}(x) - \sum_{k=j+1}^{m_i} h_{ij}^{(k)}(x_i) H_{ik}(x).$$

One checks in particular that for $p = 0, \dots, m_i$, the resulting polynomial satisfies

$$H_{ij}^{(p)}(x_i) = h_{ij}^{(p)}(x_i) - \sum_{k=j+1}^{m_i} h_{ij}^{(k)}(x_i) H_{ik}^{(p)}(x_i) = h_{ij}^{(p)}(x_i) - \sum_{k=j+1}^{m_i} h_{ij}^{(k)}(x_i)\delta_{kp} = \begin{cases} 0 & \text{if } 0 \leqslant p < j, \\ 1 & \text{if } p = j, \\ 0 & \text{if } j < p < m_i. \end{cases}$$

By recurrence, the process thus leads to a set of polynomials functioning as advertised.

A particular case Let us work out this process in the case where $m_0 = m_1 = \cdots = m_n = 1$: we seek to match both values and first derivative of $f(x)$ at the set of nodes x_i. The degree of the resulting polynomial should be $N = 2n + 1$.

- First, we have $H_{i1}(x) = h_{i1}(x) = (x - x_i) \prod_{k \neq i} \left(\frac{x - x_k}{x_i - x_k} \right)^2$, so recalling the definition of the Lagrange characteristic polynomials $\ell_i(x) = \prod_{k \neq i} \frac{x - x_k}{x_i - x_k}$,

$$H_{i1}(x) = (x - x_i)\ell_i^2(x), \qquad i = 0, \dots, n.$$

- Next, we want to compute

$$h_{i0}(x) = \prod_{k \neq i} \left(\frac{x - x_k}{x_i - x_k} \right)^2 = \ell_i^2(x)$$

and

$$H_{i0}(x) = h_{i0}(x) - h'_{i0}(x_i) H_{i1}(x).$$

We have $h'_{i0}(x) = 2\ell'_i(x)\ell_i(x)$, so $h'_{i0}(x_i) = 2\ell'_i(x_i) = 2\sum_{i \neq k} \frac{1}{x_i - x_k}$. Hence,

$$H_{i0}(x) = \left[1 - 2\ell'_i(x_i)(x - x_i)\right] \ell_i^2(x).$$

Definition 12.2. *The following polynomials form a basis of Hermite characteristic polynomials for $\mathbb{P}_{2n+1}$:*

$$\begin{aligned} H_i(x) &= H_{i0}(x) = \left[1 - 2\ell'_i(x_i)(x - x_i)\right] \ell_i^2(x), \\ \widehat{H}_i(x) &= H_{i1}(x) = (x - x_i)\ell_i^2(x), \qquad i = 0, \dots, N, \end{aligned}$$

where $\ell_i(x) = \prod_{k \neq i} \frac{x - x_k}{x_i - x_k}$ are the characteristic Lagrange polynomials.

Property 12.3 (Lagrange-type formula for the Hermite interpolant)**.** *The unique Hermite interpolation polynomial for f at the nodes $x_0, \dots, x_n$ writes*

$$P_{2n+1}(x) = f(x_0)H_0(x) + \cdots + f(x_n)H_n(x) \quad + \quad f'(x_0)\widehat{H}_0(x) + \cdots + f'(x_n)\widehat{H}_n(x).$$

Furthermore, if $f \in C^{2n+2}(I)$ with $x_0, \dots, x_n, x \in I$ then

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

where $\xi \in [\min(x_0, \dots, x_n, x), \max(x_0, \dots, x_n, x)]$.

We skip the proof of this result. The general formula can be found in the textbook.

12.2 Newton-type formula.

Recall now the formula for divided differences:

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \qquad \text{for } x_0 \neq x_1.$$

It is clear than when $x_1 \to x_0$, the limit is well-defined allowing us to set

$$f[x_0, x_0] = f'(x_0).$$

This allows us to use the same Newton formula as before, but with a new set of nodes repeating the original ones as needed: for example, if $m_0 = \cdots = m_n = 1$, we form the new set of nodes

$$\{z_0, z_1, \ldots, z_{2n}, z_{2n+1}\} = \{x_0, x_0, \ldots, x_n, x_n\}.$$

The the Hermite interpolant is simply

$$P_{2n+1}(x) = f[z_0] + f[z_0, z_1](x - z_0) + f[z_0, z_1, z_2](x - z_0)(x - z_1) + \cdots + f[z_0, \ldots, z_{2n+1}](x - z_0) \cdots (x - z_{2n}),$$

or in terms of the original nodes,

$$\begin{aligned} P_{2n+1}(x) = f[x_0] &+ f[x_0, x_0](x - x_0) \\ &+ f[x_0, x_0, x_1](x - x_0)^2 \\ &+ \ldots \\ &+ f[x_0, x_0, \ldots, x_n, x_n](x - x_0)^2 \cdots (x - x_{n-1})^2 (x - x_n), \end{aligned}$$

where the divided differences are computed as before using the triangular table:

$$\begin{array}{c|ccccccccc} x_0 & f(x_0) & & & & & & & & \\ & & \searrow & & \searrow & & & & & \\ x_0 & f(x_0) & & f'(x_0) & & & & & & \\ & & \searrow & & \searrow & & & & & \\ x_1 & f(x_1) & \to & f[x_0, x_1] & \to & f[x_0, x_0, x_1] & & & & \\ & & & & \searrow & & \searrow & & & \\ x_1 & f(x_1) & & f'(x_1) & \to & f[x_0, x_1, x_2] & \to & f[x_0, x_0, x_1, x_1] & & \\ & & \searrow & & \searrow & & \searrow & & \searrow & \\ x_2 & f(x_2) & \to & f[x_1, x_2] & \to & f[x_0, x_1, x_2] & \to & f[x_0, x_1, x_1, x_2] & \to & f[x_0, x_0, x_1, x_1, x_2] \\ & & \searrow & & \searrow & & \searrow & & & \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & & \ddots \end{array}$$

A similar procedure can be employed to construct the general Hermite interpolant, using the convention

$$f[\underbrace{x, \ldots, x}_{\text{repeated } k+1 \text{ times}}] = \frac{f^{(k)}(x)}{k!}.$$

12.3 Example.

Let us find the Hermite interpolant of $f(x) = \frac{\sqrt{2}}{\sqrt{x^2+1}}$ at the nodes $-1, 0, 1$.

We compute the derivative $f'(x) = \frac{-\sqrt{2}x}{(x^2+1)^{3/2}}$ and thus the data:

x_k	$f(x_k)$	$f'(x_k)$
-1	1	$1/2$
0	$\sqrt{2}$	0
1	1	$-1/2$

Construction using the Lagrange-type formula: We know that

$$H_i(x) = [1 - 2(x - x_i)\ell_i'(x_i)]\ell_i^2(x), \qquad \widehat{H}_i(x) = (x - x_i)\ell_i^2(x).$$

Hence we start by computing the elementary Lagrange polynomials and their derivatives.

$$\begin{aligned}
&\ell_0(x) = \frac{x(x-1)}{2}, && \ell_1(x) = -(x-1)(x+1), && \ell_2(x) = \frac{x(x+1)}{2},\\
&\ell_0'(x) = x - \frac{1}{2}, && \ell_1'(x) = -2x, && \ell_2'(x) = x + \frac{1}{2},\\
&\ell_0'(x_0) = -\frac{3}{2}, && \ell_1'(x_1) = 0, && \ell_2'(x_2) = \frac{3}{2}.
\end{aligned}$$

This leads to the basis polynomials

$$\begin{aligned}
H_0(x) &= [1 - 2(x+1)(-3/2)]\left(\frac{x(x-1)}{2}\right)^2 &&= \frac{1}{4}(3x+4)x^2(x-1)^2,\\
H_1(x) &= [1 - 2x(0)]\left((x-1)(x+1)\right)^2 &&= (x-1)^2(x+1)^2,\\
H_2(x) &= [1 - 2(x-1)(3/2)]\left(\frac{x(x+1)}{2}\right)^2 &&= \frac{1}{4}(4-3x)x^2(x+1)^2,
\end{aligned}$$

and

$$\begin{aligned}
\widehat{H}_0(x) &= (x+1)x^2(x-1)^2,\\
\widehat{H}_1(x) &= x(x-1)^2(x+1)^2,\\
\widehat{H}_2(x) &= (x-1)x^2(x+1)^2.
\end{aligned}$$

Finally, we can assemble the Hermite interpolating polynomial:

$$\begin{aligned}
P_5(x) &= H_0(x) + \sqrt{2}H_1(x) + H_2(x) + \frac{1}{2}\widehat{H}_0(x) - \frac{1}{2}\widehat{H}_2(x)\\
&= \frac{1}{4}(3x+4)x^2(x-1)^2 + \sqrt{2}(x-1)^2(x+1)^2 + \frac{1}{4}(4-3x)x^2(x+1)^2 - x^2(x-1)(x+1).
\end{aligned}$$

Construction using the Newton formula: Here we first compute the table of divided differences:

$$\begin{array}{c|ccccccccccc}
-1 & 1 & & & & & & & & & & \\
 & & \searrow & & \searrow & & & & & & & \\
-1 & 1 & & 1/2 & & & & & & & & \\
 & & \searrow & & \searrow & & & & & & & \\
0 & \sqrt{2} & \to & \sqrt{2}-1 & \to & \sqrt{2}-3/2 & & & & & & \\
 & & & & \searrow & & \searrow & & & & & \\
0 & \sqrt{2} & & 0 & \to & 1-\sqrt{2} & \to & 5/2-2\sqrt{2} & & & & \\
 & & \searrow & & \searrow & & \searrow & & \searrow & & & \\
1 & 1 & \to & 1-\sqrt{2} & \to & 1-\sqrt{2} & \to & 0 & \to & \sqrt{2}-5/4 & & \\
 & & \searrow & & \searrow & & \searrow & & \searrow & & \searrow & \\
1 & 1 & \to & 1/2 & \to & 3/2-\sqrt{2} & \to & 2\sqrt{2}-5/2 & \to & \sqrt{2}-5/4 & \to & 0
\end{array}$$

Then we read directly on the diagonal the coefficients for the Newton expansion of the Hermite interpolation polynomial:

$$P_5(x) = 1 + \frac{1}{2}(x+1) + \left(\sqrt{2} - \frac{3}{2}\right)(x+1)^2 + \left(\frac{5}{2} - 2\sqrt{2}\right)(x+1)^2x + + \left(\sqrt{2} - \frac{5}{4}\right)(x+1)^2x^2.$$

Lecture 13: Numerical Integration. (Wednesday, October 7)

We want to develop methods to approximate definite integrals such as

$$I(f) = \int_a^b f(x)dx,$$

where $f(x)$ is a given function which typically does not have an explicit anti-derivative: for example,

$$f(x) = e^{-x^2}.$$

Main idea: given an approximation $f_n \approx f$ which as a closed-form antiderivative, we compute

$$I_n(f) = I(f_n).$$

Typically, $f_n(x) = \Pi_n f(x)$ is an interpolating polynomial at well-chosen nodes. The error commited by such an approach can be easily related to the error $f_n - f$ in the function approximation, since

$$E_n(f) = I(f) - I_n(f) = \int_a^b f(x) - f_n(x)dx,$$

so that the absolute error can be bounded:

$$|E_n(f)| = \left| \int_a^b f(x) - f_n(x)dx \right| \leqslant (b-a)\|f - f_n\|_\infty.$$

Interpolatory quadrature rules. Using the results from our previous sections about interpolation, we have a natural way of constructing approximations to the function f which are easily exactly integrable. Given distinct nodes $x_0, \ldots, x_n \in [a, b]$, we form the interpolation polynomial

$$f_n(x) = \Pi_n f(x) = f(x_0)\ell_0(x) + \cdots + f(x_n)\ell_n(x),$$

where $\ell_i(x) = \prod_{k \neq i} \frac{x - x_k}{x_i - x_k}$ for $i = 0, \ldots, n$ is the elementary Lagrange polynomial. Then,

$$I(f) \approx I_n(f) = \int_a^b \Pi_n f(x)dx = f(x_0)\int_a^b \ell_0(x)dx + \cdots + f(x_n)\int_a^b \ell_n(x)dx,$$

that is we have the Lagrange quadrature formula

$$I_n(f) = \sum_{i=0}^n \alpha_i f(x_i),$$

where the weights are given as $\alpha_i = \int_a^b \ell_i(x)$ and are independent of f.

Error analysis: since we have the formula

$$f(x) - \Pi_n f(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)\cdots(x - x_n),$$

the error for the integrals writes

$$E_n(f) = I(f) - I_n(f) = \frac{1}{(n+1)!}\int_a^b f^{(n+1)}(\xi(x))\Pi_{j=0}^n (x - x_j)dx.$$

In general, we cannot simplify this expression further.

Exactness: One remarkable property of interpolatory quadratures is that if $f \in \mathbb{P}_n$, then the quadrature is exact: $E_n(f) = 0$.
Let us formalize the main concepts so far.

Definition 13.1. *A **quadrature formula** is a weighted sum*

$$I_n(f) = \sum_{i=0}^{n} a_i f(x_i),$$

with specified nodes $x_0, \ldots, x_n$ and weights $a_0, \ldots, a_n$.

Note that the interpolatory rules presented above are fully specified by the nodes $x_0, \ldots, x_n$, since the weights are given by $\alpha_i = \int_a^b \ell_i(x)$.

Definition 13.2. *The **degree of exactness** of a quadrature rule is the largest integer $r \geqslant 0$ such that*

$$f \in \mathbb{P}_r \qquad \Longrightarrow \qquad I_n(f) = I(f).$$

Example: the interpolatory rule defined above using $n+1$ nodes has degree of exactness at least n.
Last comments:

- The reverse is true: is a rule with $n+1$ points has degree of exactness equal or larger than n, then the rule is interpolatory (i.e. the weights are given by the formula above).
- The degree of exactness of an interpolatory rule using $n+1$ points can be as high as $2n+1$ - we will explore this phenomenon, called Gaussian quadrature, later on.
- Integration as well as numerical quadrature are *linear maps* on the space of functions:

  $$I(\alpha f + \beta g) = \alpha I(f) + \beta I(g), \qquad I_n(\alpha f + \beta g) = \alpha I_n(f) + \beta I_n(g),$$

  for any pair of integrable functions f, g and real numbers α, β.

13.1 Closed Newton-Côtes quadrature rules.

A special class of rules is obtained by choosing $n \geqslant 1$ and $n+1$ equally spaced nodes $x_j = a + jh$, $j = 0, \ldots, n$, with step size $h = \frac{b-a}{n}$. Such rules are called *closed Newton-Côtes rules* - 'closed' because the end-points of the interval are part of the set of nodes.
We denote these rules

$$Q_n(f) = \sum_{i=0}^{n} A_i f(x_i), \qquad \text{where } A_i = \int_a^b \ell_i(x)dx.$$

Let us explore a few special cases.

Case $n = 1$: the trapezoidal rule.

- In this case, we have only 2 nodes: $x_0 = a$ and $x_1 = b$.
- The corresponding interpolating polynomial $\Pi_1 f(x) = f(a)\ell_0(x) + f(b)\ell_1(x)$, with the elementary Lagrange polynomials

  $$\ell_0(x) = \frac{b-x}{b-a} \qquad \text{and} \qquad \ell_1(x) = \frac{x-a}{b-a}.$$

- The interpolatory rule is thus obtained as

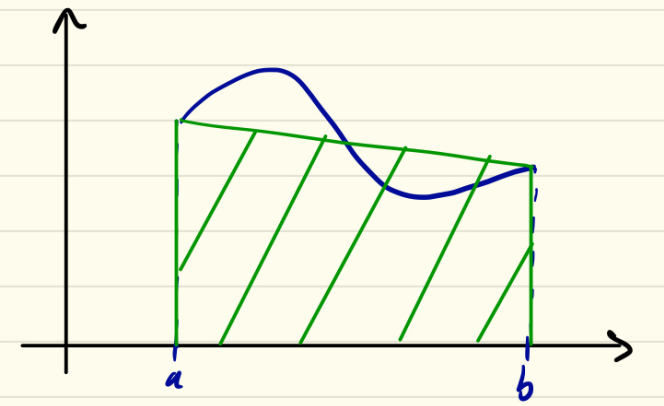
$$Q_1(f) = \int_a^b \Pi_1 f(x)dx = f(a)\int_a^b \ell_0(x)dx + f(b)\int_a^b \ell_1(x)dx,$$

whith the weights

$$\alpha_0 = \int_a^b \ell_0(x)dx = \int_a^b \frac{b-x}{b-a}dx = \left.\frac{-(b-x)^2}{2(b-a)^2}\right|_{x=a}^b = \frac{b-a}{2},$$
$$\alpha_1 = \int_a^b \ell_1(x)dx = \int_a^b \frac{x-a}{b-a}dx = \left.\frac{(x-a)^2}{2(b-a)^2}\right|_{x=a}^b = \frac{b-a}{2}.$$

We have thus derived the trapezoidal rule:

$$Q_1(f) = \frac{b-a}{2}\left(f(a)+f(b)\right) = \frac{h}{2}\left(f(a)+f(b)\right).$$



- Error analysis: we have from above

$$E_1(f) = \int_a^b \frac{f''(\xi(x))}{2}(x-a)(x-b)dx.$$

Now because $(x-a)(x-b) \leqslant 0$ does not change sign on the interval $[a,b]$, we can use the mean value theorem:

$$E_1(f) = \frac{f''(\xi}{2}\int_a^b (x-a)(x-b)dx, \qquad \text{for some } \xi \in [a,b].$$

We can compute explicitely this last expression:

$$\int_a^b (x-a)(x-b) = -\frac{(b-a)^3}{6},$$

so we obtain:

Property 13.3 (Error formula for the trapezoidal rule:)**.**

$$E_1(f) = -\frac{(b-a)^3}{12}f''(\xi), \qquad \textit{for some } \xi \in [a,b].$$

Case $n = 2$: Simpson's rule.

- In this case, we have 3 nodes: $x_0 = a$, $x_1 = m = \frac{a+b}{2}$, and $x_2 = b$, and step size $h = \frac{b-a}{2}$.
- The corresponding elementary Lagrange polynomials and weights are

$$\begin{aligned}
\ell_0(x) &= \frac{(x-m)(x-b)}{(a-m)(a-b)} \qquad \rightarrow \qquad A_0 = \int_a^b \ell_0(x)dx = \frac{h}{3},\\
\ell_1(x) &= \frac{(x-a)(x-b)}{(m-a)(m-b)} \qquad \rightarrow \qquad A_1 = \int_a^b \ell_1(x)dx = \frac{4h}{3},\\
\ell_2(x) &= \frac{(x-a)(x-m)}{(b-a)(b-m)} \qquad \rightarrow \qquad A_2 = \int_a^b \ell_2(x)dx = \frac{h}{3}.
\end{aligned}$$

We have thus derived Simpson's rule:

$$Q_2(f) = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) = \frac{h}{3}\left(f(a) + 4f(m) + f(b)\right).$$

Property 13.4 (Error formula for Simpson's rule:)**.**

$$E_2(f) = -\frac{h^5}{90}f^{(4)}(\xi), \qquad \textit{for some } \xi \in [a,b].$$

The proof will be presented later. Note that the degree of exactness of Simpson's rule is 3, which is one more than we expect!

General remark on Closed Newton-Côtes Rules: The weights of Newton-Côtes rules depend in general only on n and proportional to h, but not on a, b in particular. Indeed, using the change of variables $x = a + th$, $0 \leqslant t \leqslant n$, then we have

$$\ell_i(x) = \Pi_{k\neq i}\frac{x - x_k}{x_i - x_k} = \Pi_{k\neq i}\frac{(a+th) - (a+kh)}{(a+ih) - (a+kh)} = \Pi_{k\neq i}\frac{t-k}{i-k} = \phi_i(t).$$

Therefore the weights are given by

$$A_i = \int_a^b \ell_i(x)dx = \int_0^n \phi_i(t)hdt = h\left(\int_0^n \phi_i(t)\right).$$

The resulting quadrature rule is

$$Q_n(f) = h\sum_{i=0}^{n} w_i f(x_i), \qquad w_i = \int_0^n \phi_i(t)dt,$$

where the coefficients w_i depend only on n and have usually been computed and assembled into tables (see e.g. Table 9.2 in the textbook) for useful values of n.

Lecture 14: Open Newton-Côtes and Composite Rules. (Monday, October 12)

14.1 Open Newton-Côtes Quadrature Rules

We proceed similarly to the construction of closed Newton-Côtes rules. For a given $n \geqslant 0$, we choose $n+1$ equidistant nodes

$$x_j = a + (j+1)h, \qquad j = 0, \ldots, n, \qquad\qquad h = \frac{b-a}{n+2}.$$

This leads to the open Newton-Côtes rules:

$$Q_n(f) = \sum_{i=0}^{n} A_i f(x_i), \qquad \text{where } A_i = \int_a^b \ell_i(x)dx..$$

More precisely, using the change of variables $t = a + (t+1)h$, we find

$$\ell_i(x) = \prod_{k \neq i} \left(\frac{x - x_k}{x_i - x_k} \right) = \prod_{k \neq i} \left(\frac{(a + (t+1)h) - (a + (k+1)h)}{a + (i+1)h - (a + (k+1)h)} \right) = \prod_{k \neq i} \frac{t-k}{i-k} := \phi_i(t),$$

we find that the quadrature weights take the form

$$A_i = \int_{-1}^{n+1} \phi_i(t) h dt = h w_i, \qquad \text{with } w_i = \int_{i=-1}^{n+1} \phi_i(t) dt.$$

Note that the coefficients $w_0, \ldots, w_n$ only depend on n, and can generally be pre-computed and tabulated.

14.2 Example: the midpoint rule.

- Here, we choose $n = 0$ such that we have a single node $x_0 = \frac{a+b}{2}$,
- The corresponding weight is $A_0 = \int_a^b \ell_0(x)dx = \int_a^b 1 dx = b - a$.

This yields the *midpoint rule:*

$$Q_0(f) = (b-a) f\left(\frac{a+b}{2} \right).$$

The error formula reads

$$E_0(f) = I(f) - Q_0(f) = \frac{(b-a)^3}{24} f''(\xi) = \frac{h^3}{3} f''(\xi), \qquad\qquad a < \xi < b.$$

The degree of exactness of the midpoint rule is 1.

Proof. Using the Newton formula,

$$I(f) - Q_0(f) = \int_a^b f[x_0, x](x - x_0)dx.$$

Let $W(x) = \int_a^x (t-x_0)dt = \frac{1}{2}[(x-x_0)^2 - (a-x_0)^2] = \frac{1}{2}[(x-x_0)^2 - h^2] = \frac{1}{2}(x-a)(x-b)$, which we note is strictly negative in the interval (a,b). Integrating by parts we have

$$I(f) - Q_0(f) = f[x_0,x]W(x)|_{x=a}^b - \int_a^b \frac{d}{dx} f[x_0,x]W(x) = -\int_a^b \frac{1}{2} f''(\xi(x))W(x)dx.$$

We have used here the identity $\frac{d}{dx} f[x_0,x] = f[x_0,x,x] = \frac{1}{2}f''(\xi(x))$, and using the mean value theorem now we obtain

$$I(f) - Q_0(f) = -\frac{f''(\xi)}{2}\int_a^b W(x)dx \qquad \text{for some } \xi \in (a,b).$$

To conclude, we compute

$$\begin{aligned}\int_a^b W(x)dx &= \int_a^b \int_a^x (t-x_0)dtdx \qquad = \int_a^b \int_t^b (t-x_0)dxdt \qquad = \int_a^b (b-t)(t-x_0)dt \\ &= \underbrace{h^3 \int_0^2 s(1-s)ds}_{t=b-hs,\ dt=hds} = h^3\left[\frac{s^2}{2} - \frac{s^3}{3}\right]_{s=0}^2 = h^3\left(2 - \frac{8}{3}\right) = -\frac{2}{3}h^3.\end{aligned}$$

□

14.3 Error analysis.

Theorem 14.1. *Let $Q_n(f)$ be an open or closed Newton-Côtes rule with $n+1$ equidistant nodes in $[a,b]$ as defined above.*

(a) If n is odd, the degree of precision is n and provided $f \in C^{n+1}([a,b])$,

$$E_n(f) = I(f) - Q_n(f) = \frac{K_n}{(n+1)!}h^{n+2}f^{(n+1)}(\xi) \qquad \textit{for some } \xi \in (a,b),$$

where

$$K_n = \begin{cases} \int_0^n \pi_{n+1}(t)dt < 0, & \textit{(closed rule)}, \\ \int_{-1}^{n+1} \pi_{n+1}(t)dt > 0, & \textit{(open rule)}, \end{cases}$$

with $\pi_{n+1}(t) = \prod_{i=0}^n (t-i)$.

(b) If n is even, the degree of precision is $n+1$ and provided $f \in C^{n+2}([a,b])$,

$$E_n(f) = I(f) - Q_n(f) = \frac{M_n}{(n+2)!}h^{n+3}f^{(n+2)}(\xi) \qquad \textit{for some } \xi \in (a,b),$$

where

$$M_n = \begin{cases} \int_0^n t\pi_{n+1}(t)dt < 0, & \textit{(closed rule)}, \\ \int_{-1}^{n+1} t\pi_{n+1}(t)dt > 0, & \textit{(open rule)}. \end{cases}$$

Proof. We only sketch the main arguments in the case of even n and a closed rule (ex: Simpson's rule). Using the Newton formula, we write the error

$$I(f) - Q_n(f) = \int_a^b f[x_0,\ldots,x_n,x]\omega_{n+1}(x)dx.$$

Let $W(x) = \int_a^x \omega_{n+1}(t)dt$, we will accept the following (nontrivial) facts:

- $W(a) = W(b) = 0$, and
- $W(x) > 0$ for $a < x < b$.

Integrating by parts, we rewrite

$$I(f) - Q_n(f) = -\int_a^b \frac{d}{dx} f[x_0, \ldots, x_n, x] W(x) dx = -\int_a^b \frac{f^{(n+2)}(\xi(x))}{(n+2)!} W(x) dx,$$

and using the mean value theorem,

$$I(f) - Q_n(f) = -\frac{f^{(n+2)}(\xi)}{(n+2)!} \int_a^b W(x) dx \qquad \text{for some } a < \xi < b.$$

Finally, using a change of variables we compute

$$\int_a^b W(x) dx = -h^{n+3} \int_0^n t \pi_{n+1}(t) dt.$$

□

14.4 Composite Quadrature Rules.

In order to increase accuracy, we know that high-order interpolation should be avoided.
Instead, we proceed as for piecewise interpolation and divide the interval into many sub-intervals, before applying one of the above rules on each one and combining the results.

14.4.1 Composite Trapezoidal Rule.

In this case, we divide the intervals using the equidistant nodes

$$a < x_0 < x_1 < \cdots < x_n = b \qquad \text{with } x_i = a + ih, \quad h = \frac{b-a}{n}.$$

Then we write

$$\begin{aligned} I(f) &= \int_{x_0}^{x_1} f(x) dx + \int_{x_1}^{x_2} f(x) dx + \cdots + \int_{x_{n-1}}^{x_n} f(x) dx \\ &\approx \frac{h}{2} (f(x_0) + f(x_1)) + \frac{h}{2} (f(x_1) + f(x_2)) + \cdots + \frac{h}{2} (f(x_{n-1}) + f(x_n)) \\ &= \frac{h}{2} (f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)) \\ &:= Q_{1,n}(f). \end{aligned}$$

This is the **composite trapezoidal rule.**

Error analysis. If $f \in C^2([a, b])$, then we may write on each sub-interval

$$\int_{x_i}^{x_{i+1}} f(x) dx - Q_1(f) = -\frac{(x_{i+1} - x_i)^3}{12} f''(\xi_i), \qquad x_i < \xi_i < \xi_{i+1},$$

and thus by summing over i,

$$I(f) - Q_{1,n}(f) = -\frac{h^3}{12}\sum_{i=0}^{n-1} f''(\xi_i).$$

To proceed further, we need to use the following discrete Mean Value Theorem. Set ξ_{min} such that $f''(\xi_{min}) = \min_i f''(\xi_i)$ and ξ_{max} such that $f''(\xi_{max}) = \max_i f''(\xi_i)$. Then,

$$f''(\xi_{min}) \leqslant \frac{1}{n}\sum_{i=0}^{n-1} f''(\xi_i) \leqslant f''(\xi_{max})$$

and thus by the intermediate value theorem, there exists $\xi \in (a, b)$ such that

$$\frac{1}{n}\sum_{i=0}^{n-1} f''(\xi_i) = f''(\xi).$$

Now, we rewrite

$$E_{1,n}(f) = -\frac{h^2}{12}\frac{b-a}{n}\sum_{i=0}^{n-1} f''(\xi_i) = -\frac{b-a}{12}f''(\xi)h^2.$$

We have proved:

Theorem 14.2. *Suppose $f(x) \in C^2([a, b])$, set $n \geqslant 1$, $h = \frac{b-a}{n}$, $x_i = a + ih$, $i = 0, \ldots, n$. The n-th composite trapezoidal rule writes*

$$Q_{1,n}(f) = \frac{h}{2}\left(f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\right).$$

The error has the form

$$E_{1,n}(f) = I(f) - Q_{1,n}(f) = -\frac{b-a}{12}f''(\xi)h^2, \qquad \text{for some } a < \xi < b.$$

The degree of precision of $Q_{1,n}$ is 1.

14.4.2 Composite Simpson's Rule.

Similarly, we set $h = \frac{b-a}{2n}$ and $x_i = a + ih$, $i = 0, \ldots, 2n$. Applying Simpson's rule on each sub-interval $[x_0, x_2]$, $[x_2, x_4, \ldots$, we have the approximation

$$\begin{aligned}
I(f) &= \int_{x_0}^{x_2} f(x)dx + \int_{x_2}^{x_4} f(x)dx + \cdots + \int_{x_{n-2}}^{x_n} f(x)dx \\
&\approx \frac{h}{3}\left(f(x_0) + 4f(x_1) + f(x_2)\right) + \frac{h}{3}\left(f(x_2) + 4f(x_3) + f(x_4)\right) + \cdots + \frac{h}{3}\left(f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right) \\
&= \frac{h}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right) \\
&:= Q_{2,n}(f).
\end{aligned}$$

This is **the n-th composite Simpson's rule.**

Error analysis. Similarly to the above computation, we start from the elementary error formula

$$\int_{x_i}^{x_{i+1}} f(x)dx - Q_2(f) = -\frac{h^5}{90} f^{(4)}(\xi_i), \qquad x_{2i} < \xi_i < \xi_{2i+2},$$

such that

$$I(f) - Q_{2,n}(f) = -\frac{h^5}{90} \sum_{i=0}^{n-1} f^{(4)}(\xi_i) = -\frac{h^4}{90} \frac{b-a}{2n} \sum_{i=0}^{n-1} f^{(4)}(\xi_i) = -\frac{b-a}{180} h^4 f^{(4)}(\xi).$$

Theorem 14.3. *Suppose $f(x) \in C^4([a,b])$, set $n \geqslant 1$, $h = \frac{b-a}{2n}$, $x_i = a + ih$, $i = 0, \ldots, 2n$. The n-th composite Simpson's rule writes*

$$Q_{2,n}(f) = \frac{h}{3} \left(f(a) + 4 \sum_{i=0}^{n-1} f(x_{2i+1}) + 2 \sum_{i=1}^{n-1} f(x_{2i}) + f(b) \right).$$

The degree of precision of $Q_{2,n}$ is 3 and the error has the form

$$E_{2,n}(f) = I(f) - Q_{2,n}(f) = -\frac{b-a}{180} f^{(4)}(\xi) h^4, \qquad \text{for some } a < \xi < b.$$

Lecture 15: Composite Rules (cont.) Approximation theory, Orthogonal Polynomials. (Wednesday, October 14)

15.1 Last comments on composite rules

15.1.1 Composite Midpoint Rule.

As a final example, we take $n \geqslant 1$, $h = \frac{b-a}{n}$, $t_i = a + ih$, $i = 0, \ldots, n$ and $x_i = a + (i + 1/2)h$, $i = 0, \ldots, n-1$. Applying the midpoint rule on each sub-interval $[t_0, t_1]$, $[t_1, t_2], \ldots$, we have the approximation

$$\begin{aligned} I(f) &= \int_{t_0}^{t_1} f(x)dx + \cdots + \int_{t_{n-1}}^{t_n} f(x)dx \\ &\approx hf(x_0) + \cdots + hf(x_{n-1}) \\ &:= Q_{0,n}(f). \end{aligned}$$

This is **the n-th composite midpoint rule.**

Theorem 15.1. *Suppose $f(x) \in C^2([a,b])$, set $n \geqslant 1$, $h = \frac{b-a}{n}$, $x_i = a+(i+1/2)h$, $i = 0, \ldots, n-1$. The n-th composite Simpson's rule writes*

$$Q_{0,n}(f) = h \sum_{i=0}^{n-1} f(x_i).$$

The degree of precision of $Q_{0,n}$ is 1, and the error has the form

$$E_{0,n}(f) = I(f) - Q_{0,n}(f) = \frac{b-a}{24} f''(\xi)h^2, \qquad \text{for some } a < \xi < b.$$

15.1.2 General Composite Newton-Côtes Rule

Taking $n \geqslant 1$ sub-intervals and a rule of order m on each one, we form the composite quadrature rule

$$Q_{n,m}(f) = \sum_{i=0}^{n-1} \sum_{j=0}^{m} \alpha_j f(x_{i,j}),$$

where the nodes are defined by setting $y_i = a + ih$, $j = 0, \ldots, n$, $h = \frac{b-a}{n}$, and

$$x_{i,j} = \begin{cases} y_i + h(j/m), & j = 0, \ldots, m \text{ (closed rule)}, \\ y_i + h(j+1)/(m+2), & j = 0, \ldots, m \text{ (open rule)}, \end{cases}$$

and the weights $\alpha_j = hw_i$ are defined as before. The error formula for even n behaves like $O(h^{n+2})$ (degree of exactness $n+1$) and for odd n like $O(h^{n+1})$ (degree of exactness n.)

15.2 Approximation of Functions

15.2.1 The Weierstrass approximation theorem

We know that any continuous function $f \in C^(0)([a,b])$ may be approximated by polynomials: the Weierstrass approximation theorem states that there exists a sequence of polynomials p_n such that $\|f - p_n\|_\infty \to 0$ as $n \to \infty$, where we can impose that $p_n \in \mathbb{P}_n$. This result can be stated in the following form:

$$\lim_{n\to\infty} \min_{p\in\mathbb{P}_n} \max_{x\in[a,b]} |f(x) - P(x)| = 0.$$

The polynomial $P_n \in \mathbb{P}_n$ realizing this minimum for a given n is called the minimax polynomial, and is unique. It is in some sense the best approximation to the function for a given polynomial degree, however it is hard to achieve - and as we saw in previous studies, interpolation is unstable at high order and not a suitable method to approach the function. This is due in part to the nature of the norm $\|\cdot\|_\infty$, which is difficult to work with.

15.2.2 Scalar product. Generalized Fourier Series on $(-1,1)$

We introduce a more amenable structure by defining a scalar (or inner) product for integrable functions on $(-1,1)$: given a continuous weight $w(x) > 0$, defined for $-1 < x < 1$, such that $\int_{-1}^{1} w(x)dx < \infty$, we set

$$(f,g)_w = \int_{-1}^{1} f(x)g(x)w(x)dx.$$

This is a well-defined bilinear form for f, g continuous on $[a,b]$, which satisfies all the usual conditions to be a scalar product - in particular, $\|f\|_w := (f,f)_2^{1/2} = \left(\int_{-1}^{1} |f(x)|^2 w(x)dx\right)^{1/2}$ is a norm on functions on $(-1,1)$: for example, $\|f\|_w = 0$ implies $f(x) = 0$ for $-1 < x < 1$.
We further define the space of square-integrable functions

$$L^2_w = L^2_w(-1,1) = \{f : (-1,1) \to \mathbb{R}, \qquad \text{measurable, } \int_{-1}^{1} f^2(x)w(x)dx, \infty\}.$$

This space contains all continuous functions on $(-1,1)$: indeed,

$$\|f\|_w \leqslant \left(\int_{-1}^{1} \|f\|^2_\infty w(x)dx\right)^{1/2} \leqslant \left(\int_{-1}^{1} w(x)dx\right)^{1/2} \|f\|_\infty.$$

Since it is equipped with a scalar product, the Hilbert space L^2_w comes with a lot of useful notions (orthogonality, etc) for approximation. In particular, we are interested in forming a *orthogonal basis*, i.e. a set of functions $p_0, p_1, \ldots$ such that $(p_k, p_l)_w = 0$ for any $k \neq l$ and any function in L^2_w may be expanded as a series

$$f = \underbrace{\sum_{k=0}^{\infty} \widehat{f}_k p_k}_{\text{convergent series in } L^2_w \text{ norm}}, \qquad \widehat{f}_k \in \mathbb{R}.$$

Such an expansion is called a generalized Fourier series. The classical example is given by the functions

$$p_{2k} = \cos(k\pi x), \qquad p_{2k-1} = \sin(k\pi x),$$

which is an orthogonal basis for the weight $w(x) = 1$ for functions on the interval $(-1,1)$.

Fundamental properties

- **Fourier coefficients:** the expansion coefficients may be computed explicitely: since

$$(f, p_k)_w = \left(\sum_{l=0}^{\infty} \widehat{f}_l p_l, p_k \right)_w = \widehat{f}_k (p_k, p_k)_w = \widehat{f}_k \|p_k\|_w^2,$$

so

$$\widehat{f}_k = \frac{(f, p_k)_w}{\|p_k\|_w^2}.$$

- **Parseval identity:**

$$\|f\|_w^2 = \sum_{k=0}^{\infty} |f_k|^2 \|p_k\|_w^2.$$

- For any $n \geqslant 0$, the truncated Fourier series

$$f_n = \sum_{k=0}^{n} \widehat{f}_k p_k$$

is the best approximation of f within $P_n = \mathrm{Span}\{p_0, \dots, p_n\}$:

$$\|f - f_n\|_w = \min_{p \in P_n} \|f - p\|_w.$$

Indeed, we know that $f - f_n = \sum_{k=n+1}^{\infty} \widehat{f}_k p_k$ is orthogonal to any basis member $p_0, \dots, p_n$:

$$(f - f_n, p_k)_w = 0, \qquad k = 0, \dots, n,$$

such that by linearity, $(f - f_n, p) = 0$ for any $p \in P_n$. Then, we compute

$$\begin{aligned} \|f - f_n\|_w^2 &= (f - f_n, f - f_n)_w \\ &= (f - f_n, f - p)_w + (f - f_n, \overbrace{p - f_n}^{\in P_n})_w \\ &= \underbrace{(f - f_n, f - p)_w}_{\text{Cauchy-Schwartz inequality.}} \leqslant \|f - f_n\|_w \|f - p\|_w \end{aligned}$$

Hence $\|f - f_n\|_w \leqslant \|f - p\|_w$, which is the result we wanted to prove.

15.3 Families of orthogonal polynomials.

Here, we seek to form an orthogonal basis such that $p_n \in \mathbb{P}_n$ is a polynomial. In particular, $p_0, \dots, p_n$ must be linearly independent and therefore a basis of $\mathbb{P}_n$. These polynomials should satisfy the orthogonality relation

$$\int_{-1}^{1} p_k(x) p_l(x) w(x) dx = \begin{cases} > 0, & k = l, \\ 0, & k \neq l. \end{cases}$$

In general, the sequence of polynomials can be generated using the Gram-Schmidt orthogonalization process starting with the standard basis $\{1, x, \dots, x^n, \dots\}$: the algorithm proceeds recursively,

setting $p_0 = 1$ and then once $p_0, \ldots, p_n$ are formed the next polynomial in the basis is obtained by setting

$$p_{n+1}(x) = x^{n+1} - \sum_{k=0}^{n} \frac{(x^{n+1}, p_k)_w}{\|p_k\|_w^2} p_k(x).$$

This is an expensive process since at each step, all previously constructed polynomials are needed. As the following result shows, this is not actually the case for families of orthogonal polynomials, for which on the last two polynomials are needed to construct the next - a so-called 3-term recurrence.

Theorem 15.2. *A monic orthogonal family $\{p_n\}_{n\geqslant 0}$ satisfies the relations*

$$\begin{aligned} xp_0 &= p_1(x) + \alpha_0 p_0, \\ xp_n(x) &= p_{n+1}(x) + \alpha_n p_n(x) + \beta_n p_{n-1}(x) \qquad \text{for } n \geqslant 1 \end{aligned}$$

with constants α_n, β_n depending only on the weight $w(x)$.

Proof. We notice that the scalar product defined above satisfies the *shift property* $(xf, g)_w = (f, xg)_w$. Now, since p_n and p_{n+1} are assume to have leading coefficients equal to 1 (monic property) we know that $xp_n(x) - p_{n+1}(x)$ has degree at most n, so it may be expanded in the orthogonal basis $p_0, \ldots, p_n$ with the Fourier coefficients defined above:

$$xp_n(x) - p_{n+1}(x) + \sum_{k=0}^{n} \frac{(xp_n, p_k)_w}{\|p_k\|_w^2} p_k(x)$$

Furthermore, we note that $(xp_n, p_k)_w = (p_n, xp_k)_w$ per the property above; however, $xp_k(x)$ has degree equal to $k+1$, so it is orthogonal to p_n if $k+1 < n$, implying that $(xp_n, p_k)_w = 0$ for $k+1 < n$. Hence,

$$xp_n(x) - p_{n+1}(x) + \underbrace{\frac{(xp_n, p_n)_w}{\|p_n\|_w^2}}_{=\alpha_n} p_n(x) + \underbrace{\frac{(xp_n, p_{n-1})_w}{\|p_{n-1}\|_w^2}}_{=\beta_n} p_{n-1}(x)$$

□

This means that all orthogonal polynomial families are constructed by a 3-term recurrence of the form

$$\begin{aligned} p_0(x) &= A_0, \qquad p_1(x) = A_1 x - B_1, \\ p_{n+1}(x) &= (A_n x - B_n) p_n(x) - C_n p_{n-1}(x) \qquad \text{for all } n \geqslant 1, \end{aligned}$$

with some sequence of numbers A_n, B_n, C_n depending only on the weight $w(x)$ and normalization A_n.

Lecture 16: Chebyshev and Legendre polynomials; Gaussian Quadrature. (Monday, October 19)

16.1 Chebyshev polynomials.

Consider the weight function $w(x) = \frac{1}{\sqrt{1-x^2}}$, the Chebyshev polynomials of the first kind are defined by the formula

$$T_n(x) = \cos(n \arccos(x)).$$

This formula, which makes sense for $-1 < x < 1$, can be shown to define a sequence of polynomials by using trigonometric identities: in particular,

$$T_0(x) = 1, \qquad T_1(x) = x.$$

Properties.

- Using the change of variables $t = \arccos(x)$, one checks easily that

$$\int_{-1}^{1} T_k(x)T_l(x)w(x) = \int_{-\pi}^{\pi} \cos(kx)\cos(lx)dx = \begin{cases} \pi, & \text{if } k = l = 0, \\ \pi/2, & \text{if } k = l > 0, \\ 0, & \text{else.} \end{cases}$$

  Hence, the Chebyshev polynomials form an orthogonal basis for the weight $w(x) = \frac{1}{\sqrt{1-x^2}}$.

- The cosine formula implies many nice properties, in particular

$$|T_n(x)| \leqslant 1, \qquad \text{for } -1 \leqslant x \leqslant 1.$$

- $T_n(x)$ is an even (resp. odd) function of x if n is even (resp. odd).

- $T_n(x)$ has n distinct zeros in $(-1, 1)$, which are explicitely:

$$x_j = -\cos\left(\frac{(2j+1)\pi}{2n}\right), \qquad j = 0, \ldots, n-1.$$

- Three-term recurrence: we have by a trigonometric identity

$$\begin{aligned} T_{n+1}(x) + T_{n-1}(x) &= 2\cos\left(\frac{(n+1)+(n-1)}{2}\arccos(x)\right)\cos\left(\frac{(n+1)-(n-1)}{2}\arccos(x)\right) \\ &= 2xT_n(x), \end{aligned}$$

  which is the 3-term recurrence

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

  This is the cheapest and most stable way to compute the value of Chebyshev polynomials for a given x.

- The leading coefficient of $T_n(x)$ is 2^{n-1}.

- The Chebyshev polynomials have the *minimax* property:

$$\|\frac{1}{2^{n-1}}T_n\|_\infty = 2^{1-n} \leqslant \|P\|_\infty,$$

  for all $P \in \mathbb{P}_n$ with leading coefficient equal to 1, i.e. $p = x^n + \textit{lower degree terms}$.

Application: for any $f \in L^2_w(-1,1)$, the best approximation of f in $\mathbb{P}_n$ in the $\|\cdot\|_w$ norm is given by the truncated generalized Fourier series:

$$P_n(x) = \sum_{k=0}^{n} \widehat{f}_k T_k(x),$$

with the expansion coefficients given by the explicit formula

$$\widehat{f}_k = \frac{2-\delta_{k0}}{\pi} \int_{-1}^{1} f(x) T_k(x) \frac{dx}{\sqrt{1-x^2}}.$$

16.2 Legendre polynomials.

Now, let us take the weight $w(x) = 1$, leading to the classical L^2 scalar product

$$(f,g) = \int_{-1}^{1} f(x)g(x)dx, \qquad \|f\|_{L^2} = \left(\int_{-1}^{1} f(x)^2 dx \right)^{1/2}.$$

The orthogonal polynomials for this weight are the Legendre polynomials:

$$L_j(x) = \frac{1}{j! 2^j} \frac{d^j}{dx^j} (x^2-1)^j \qquad \text{(Rodrigues' formula)}$$

or recursively through the 3-term relation:

$$\begin{cases} L_0(x) = 1, & L_1(x) = x, \\ (j+1)L_{j+1}(x) = (2j+1)xL_j(x) - jL_{j-1}(x). \end{cases}$$

Properties:

- The polynomial $L_j(x)$ is an even (resp. odd) function of x if j is even (resp. odd).
- The scalar product and L_i and L_j is

$$(L_i, L_j) = \frac{2}{2j+1} \delta_{ij}.$$

Application: for any $f \in L^2(-1,1)$, the best approximation of f in $\mathbb{P}_n$ in the $\|\cdot\|$ norm is given by the truncated generalized Fourier series:

$$P_n(x) = \sum_{j=0}^{n} \widehat{f}_j L_j(x),$$

with the expansion coefficients given by the explicit formula

$$\widehat{f}_j = (j+1/2) \int_{-1}^{1} f(x) L_j(x) dx.$$

16.3 Gaussian quadrature

We are interested in quadrature rules with maxximum order of exactness to approximate weighted integrals:

$$I_w(f) = \int_{-1}^{1} f(x)w(x)dx.$$

Given nodes $x_0, \dots, x_n$, we know that the interpolatory quadrature rule

$$I_{n,w}(f) = \sum_{i=0}^{n} \alpha_i f(x_i), \qquad \alpha_i = \int_{-1}^{1} \ell_i(x)w(x)dx,$$

will have degree of exactness at least n.
In order to extract the maximum accuracy from the quadrature, it is of interest to choose the nodes $x_0, \dots, x_n$ to maximize the degree of exactness. To construct such rules, say with degree of exactness $n + m$ for $m > 0$, we rely on the following result:

Theorem 16.1. *For $m > 0$, an interpolatory rule $I_{n,w}$ has degree of exactness at least $n + m$ if and only if the polynomial*

$$\omega_{n+1}(x) = (x - x_0) \cdots (x - x_n)$$

is orthogonal to any polynomial in $\mathbb{P}_{n-1}$, meaning that

$$\int_{-1}^{1} \omega_{n+1}(x)p(x)w(x)dx = 0, \qquad \forall p \in \mathbb{P}_{m-1}.$$

Proof. If $f \in \mathbb{P}_{n+m}$, we use polynomial division by ω_{n+1} to write

$$f(x) = \omega_{n+1}(x)q_{m-1}(x) + r_n(x), \qquad q_{m-1} \in \mathbb{P}_{n-1},\ r_n \in \mathbb{P}_n.$$

Now we compute the quadrature. Since $\omega_{n+1}(x_i) = 0$ for $i = 0, \dots, n$ and $I_{n,w}$ is exact for all polynomials in $\mathbb{P}_n$,

$$\sum_{i=0}^{n} \alpha_i f(x_i) = \sum_{i=0}^{n} \alpha_i r(x_i) = \int_{-1}^{1} r_n(x)w(x)dx = \int_{-1}^{1} f(x)w(x)dx - \underbrace{\int_{-1}^{1} \omega_{n+1}(x)q_{m-1}(x)w(x)dx}_{=0 \text{ (by assumption).}}.$$

Since this must hold for all possible polynomials $q_{m-1}(x)$ in $\mathbb{P}_{m-1}$, the two propositions in the theorem are in fact equivalent. □

Corollary 16.2. *The maximum degree of exactness for an interpolatory quadrature formula is at most $2n + 1$.*

Proof. Pick $f(x) = \omega_{n+1}^2(x)$, then the exact integral is $I_w(f) = \int_{-1}^{1} \omega_{n+1}^2(x)w(x)dx > 0$, but the quadrature is $I_{n,w}(f) = \sum_{i=0}^{n} \alpha_i \omega_{n+1}^2(x_i) = 0$. Hence the rule cannot be exact for all polynomials of degree $2n + 2$. □

To achieve the maximum order of exactness $2n + 1$, we thus want $\omega_{n+1}(x)$ to be orthogonal to all polynomials of order at most n, i.e.

$$(\omega_{n+1}, p)_w = 0, \qquad \forall p \in \mathbb{P}_n.$$

This implies that ω_{n+1} is proportional to p_{n+1}, where $\{p_k(x)\}$ is an orthogonal basis of polynomials for $L_w^2(-1, 1)$.

Definition 16.3. *Given $n \geqslant 0$, let $p_{n+1}(x)$ be the $(n+1)$-th order orthogonal polynomial with respect to the weight $w(x)$.*

- *The roots $-1 < x_0 < \cdots < x_n < 1$ of p_{n+1} are the Gauss nodes associated with the weight function $w(x)$.*
- *The weights $\alpha_0, \ldots, \alpha_n$ defined by $\alpha_i = \int_{-1}^{1} \ell_i(x) w(x) dx$ are the Gauss weights.*
- *The Gauss quadrature formula*

$$G_{n,w}(f) = \sum_{i=0}^{n} \alpha_i f(x_i)$$

has degree of exactness $2n+1$, the highest possible ($2n+1$) using $n+1$ quadrature nodes.

Properties.

- The Gauss nodes are all internal to the interval: $-1 < x_0 < x_1 < \cdots < x_n < 1$.
- The Gauss weights are all positive: indeed $\alpha_i = G_{n,w}(\ell_i^2(x)) = I_w(\ell_i^2(x)) > 0$.

Gauss-Lobatto quadrature If including the end points is a desired property of the quadrature, one can reduce slightly the order of exactness of the quadrature in order to do so. The resulting quadrature rule is called Gauss-Lobatto. In order to achieve this result, we construct the polynomial

$$\overline{\omega}_{n+1}(x) = p_{n=1}(x) + A p_n(x) + B p_{n-1}(x)$$

with A, B chosen in such a way that $\overline{\omega}_{n+1}(-1) = \overline{\omega}_{n+1}(1) = 0$. Such a polynomial is orthogonal to all polynomials of order $n-2$, and includes as its roots the end points -1 and 1 by construction. As a consequence, the roots of $\overline{\omega}_{n+1}(x)$ form a set of quadrature nodes ensuring degree of exactness $2n-1$.

Definition 16.4. *Given $n \geqslant 0$, let $\overline{\omega}_{n+1}(x)$ be constructed as above.*

- *The roots $-1 = \overline{x}_0 < \overline{x}_1 < \cdots < x_n < \overline{x}_n = 1$ of $\overline{\omega}_{n+1}$ are the Gauss-Lobatto nodes associated with the weight function $w(x)$.*
- *The weights $\alpha_0, \ldots, \alpha_n$ defined by $\overline{\alpha}_i = \int_{-1}^{1} \overline{\ell_i}(x) w(x) dx$ are the Gauss-Lobatto weights.*
- *The Gauss-Lobatto quadrature formula*

$$GL_{n,w}(f) = \sum_{i=0}^{n} \overline{\alpha}_i f(\overline{x}_i)$$

has degree of exactness $2n-1$, the highest possible using $n+1$ quadrature nodes including the end points of the interval.

Convergence of Gauss quadrature rules.

Theorem 16.5. *For any continuous function $f : [-1, 1] \to \mathbb{R}$, the Gauss quadrature rules converge to $I(f)$ as $n \to \infty$:*

$$\lim_{n \to \infty} G_{n,w}(f) = \lim_{n \to \infty} GL_{n,w}(f) = I_w(f).$$

Proof. Skipped. □

Error formula. One recalls the error formula for the Hermite interpolation polynomial at $x_0, \dots, x_n$:

$$f(x) = P_{2n+1}(x) + \frac{f^{(2n+2)}(\xi(x))}{(2n+2)!}(x-x_0)^2 \cdots (x-x_n)^2.$$

Since P_{2n+1} is integrated exactly by the Gauss quadrature rule and the error term vanishes at the quadrature nodes, we compute

$$\begin{aligned} I_w(f) - G_{n,w}(f) &= \int_{-1}^{1} \frac{f^{(2n+2)}(\xi(x))}{(2n+2)!}(x-x_0)^2 \cdots (x-x_n)^2 w(x)dx \\ &= \frac{f^{(2n+2)}(\xi)}{(2n+2)!} \int_{-1}^{1} (x-x_0)^2 \cdots (x-x_n)^2 w(x)dx \end{aligned}$$

using the mean value theorem, with $\xi \in (-1,1)$. The integral on the right is a constant independent of f, which depends only on the weight and on the orthogonal polynomials: for example, in the case of Chebyshev weight ($w(x) = (1-x^2)^{-1/2}$) one has $T_{n+1}(x) = 2^n(x-x_0)\cdots(x-x_n)$, hence

$$\int_{-1}^{1} (x-x_0)^2 \cdots (x-x_n)^2 w(x)dx = \frac{1}{2^{2n}}(T_{n+1}, T_{n+1})_w = \frac{\pi}{2^{2n+1}}$$

yielding the error formula for Chebyshev-Gauss quadrature

$$I_w(f) - G_{n,w}(f) = \frac{\pi}{2^{2n+1}(2n+2)!} f^{(2n+2)}(\xi), \qquad \xi \in (-1,1).$$

Lecture 17: Numerical Quadrature: the Conclusion. (Wednesday, October 21)

17.1 Integration over arbitrary intervals.

Any rule over $[-1,1]$ may be apped onto an arbitrary interval of finite length $[a,b]$ using the change of variables

$$\phi(s) = \frac{a+b}{2} + \frac{b-a}{2}s,$$

since setting $x = \phi(s)$ such that $dx = \frac{b-a}{2}ds$ implies

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^1 f(\phi(s))ds \approx \frac{b-a}{2}\sum_{i=0}^n \alpha_i f(\phi(\xi_i)),$$

where ξ_i, α_i are respectively the nodes and weights of the quadrature rule over $[-1,1]$, for example a Gauss quadrature rule obtained in the previous lecture. Hence, on $[a,b]$ we have the new nodes and weights

$$x_i = \phi(\xi_i), \qquad A_i = \frac{b-a}{2}\alpha_i.$$

Remark 17.1. *When the quadrature over $[-1,1]$ corresponds to a weighted integral $I_w(f) = \int_{-1}^1 f(s)w(s)ds$, then the new quadrature over $[a,b]$ approximates $\int_a^b f(x)W(x)dx$, another weighted integral with the weight $W(x) = w(\phi^{-1}(x))$ where $\phi^{-1}(x) = \frac{2}{b-a}\left(x - \frac{a+b}{2}\right)$ is the inverse function of ϕ.*

Error formulae An error formula follows also by change of variables. For example, let us consider the Gauss-Legendre quadrature with error term

$$\int_{-1}^1 F(s)ds - G_n(F) = C_n F^{(2n+2)}(\xi), \qquad \xi \in (-1,1),$$

where $C_n = \frac{1}{(2n+2)!}\int_{-1}^1 (x-x_0)^2\cdots(x-x_n)^2 dx$ is a constant depending only on n. Then letting $F(s) = f(\phi(s))$, because $\phi'(s) = \frac{b-a}{2}$ a trivial recurrence using the chain rule implies

$$F^{(2n+2)}(\xi) = \left(\frac{b-a}{2}\right)^{2n+2} f^{(2n+2)}(\xi'), \qquad \xi' = \phi(\xi) \in (a,b).$$

Hence

$$\begin{aligned}\int_a^b f(x)dx - \sum_{i=0}^n A_i f(x_i) &= \frac{b-a}{2}\left(\int_{-1}^1 f(\phi(s))ds - \sum_{i=0}^n \alpha_i f(\phi(\xi_i))\right)\\ &= \frac{b-a}{2}\left(\int_{-1}^1 F(s)ds - G_n(F)\right)\\ &= C_n\left(\frac{b-a}{2}\right)^{2n+3} f^{(2n+2)}(\xi').\end{aligned}$$

Composite rules The mapping above by change of variables allows us for example to build composite rules based on Gaussian quadrature, which take advantage of their stability and high accuracy. Given

- a Legendre-Gauss or Legendre-Gauss-Lobatto quadrature rule with $n+1$ nodes $\xi_0, \ldots, \xi_n$ and weights $\alpha_0, \ldots, \alpha_n$ over $[-1, 1]$:

$$G_n(f) = \sum_{i=0}^{n} \alpha_i f(\xi_i) \approx \int_{-1}^{1} f(x)dx;$$

- the partition $a = y_0 < y_1 < \cdots < y_m = b$ of a given interval $[a, b]$, with equidistant nodes $y_k = a + kh$ for step $h = \frac{b-a}{m}$;

we can build a composite Legendre-Gauss quadrature rule over $[a, b]$:

$$G_{n,m}(f) = \frac{h}{2} \sum_{k=0}^{m-1} \sum_{i=0}^{n} \alpha_i f(x_i^{(k)}),$$

with the quadrature nodes $x_i^{(k)} = y_k + h\left(\frac{1+\xi_i}{2}\right)$.
The corresponding error term is of order $O(h^{2n+2})$ using the Gauss-Legendre, and $O(h^{2n})$ using the Lobatto-Gauss-Legendre rule and weights. Indeed, using the error formula for the Gauss quadrature from the previous lecture, we find that over each sub-interval, we have

$$\int_{y_k}^{y_{k+1}} f(x)dx - \frac{h}{2} \sum_{i=0}^{n} \alpha_i f(x_i^{(k)}) = C_n \left(\frac{h}{2}\right)^{2n+3} f^{(2n+2)}(\xi_k), \qquad \text{for } \xi_k \in (y_k, y_{k+1}).$$

Summing over all the intervals, we find that, using a discrete mean value theorem,

$$\begin{aligned} \int_a^b f(x)dx - G_{n,m}(f) &= C_n \left(\frac{h}{2}\right)^{2n+2} \frac{b-a}{2m} \sum_{k=0}^{m-1} f^{(2n+2)}(\xi_k) \\ &= C_n \frac{b-a}{2} \left(\frac{h}{2}\right)^{2n+2} f^{(2n+2)}(\xi), \qquad \text{for some } \xi \in (a, b). \end{aligned}$$

The composite rules uses a total $m(n+1)$ function evaluations in the Gauss-Legendre case and $mn+1)$ in the Lobatto-Gauss-Legendre case (since the end points of sub-intervals in the interior of $[a, b]$ appear twice in the sum).

Remark 17.2. *One can also construct composite rules based on other Gaussian rules (e.g., Chebyshev), but particular care should be taken to take the weight into account.*

17.2 Examples

Let us construct explicitely some low-order Gaussian quadrature rules.

17.2.1 Case n = 0.

We check here that the Legendre-Gauss quadrature with 1 node is the midpoint rule. We seek the node x_0 and weight w_0 such that the following quadrature rule has degree of precison $2 \cdot 0 + 1 = 1$:

$$G_0(f) = w_0 f(x_0) \approx I(f) = \int_{-1}^{1} f(x)dx.$$

Method of Undetermined Coefficients. Let us form directly two equations corresponding to the required exactness property:

$$G_0(1) = I(1) \qquad \rightarrow \qquad w_0 = \int_{-1}^{1} 1dx = 2,$$

$$G_0(x) = I(x) \qquad \rightarrow \qquad w_0 x_0 = \int_{-1}^{1} x dx = 0 \qquad \rightarrow \qquad x_0 = 0.$$

This yields the quadrature rule

$$G_0(f) = 2f(0),$$

which is just the midpoint rule applied to $I(f)$.

Using Legendre polynomials. Using the theory from the last lecture, we can also construct the node x_0 as the only root of the Legendre polynomial $L_1(x) = x$, that is $x_0 = 0$, and compute the corresponding weight as the integral of the corresponding elementary Lagrange interpolation polynomial, $\ell_0(x) = 1$. Hence

$$x_0 = 0, \qquad w_0 = \int_{-1}^{1} \ell_0(x)dx = \int_{-1}^{1} 1dx = 2.$$

This is indeed the same result.

17.2.2 Case n = 1.

Let us seek now the Legendre-Gauss quadrature with 2 nodes x_0, x_1 and weights w_0, w_1 that integrates exactly all polynomials of degree at most 3:

$$G_1(f) = w_0 f(x_0) + w_1 f(x_1) \approx I(f) = \int_{-1}^{1} f(x)dx.$$

Method of Undetermined Coefficients. Let us form directly four equations corresponding to the required exactness property:

$$G_1(1) = I(1) \qquad \rightarrow \qquad w_0 + w_1 = \int_{-1}^{1} 1dx \qquad \rightarrow \qquad w_0 + w_1 = 2,$$

$$G_1(x) = I(x) \qquad \rightarrow \qquad w_0 x_0 + w_1 x_1 = \int_{-1}^{1} x dx \qquad \rightarrow \qquad w_0 x_0 + w_1 x_1 = 0,$$

$$G_1(x^2) = I(x^2) \qquad \rightarrow \qquad w_0 x_0^2 + w_1 x_1^2 = \int_{-1}^{1} x^2 dx \qquad \rightarrow \qquad w_0 x_0^2 + w_1 x_1^2 = 2/3,$$

$$G_1(x^3) = I(x^3) \qquad \rightarrow \qquad w_0 x_0^3 + w_1 x_1^3 = \int_{-1}^{1} x^3 dx \qquad \rightarrow \qquad w_0 x_0^3 + w_1 x_1^3 = 0.$$

This is a nonlinear system of 4 equations for 4 unknowns. We can simplify its solution by guessing that, by symmetry, $x_0 = -x_1$ and $w_0 = w_1$, thus

$$2w_1 = 1 \qquad \rightarrow \qquad w_0 = w_1 = 1,$$

$$2w_1 x_1^2 = 2/3 \qquad \rightarrow \qquad x_1 = -x_0 = \frac{\sqrt{3}}{3}$$

This yields the quadrature rule

$$G_1(f) = f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right).$$

Using Legendre polynomials. We can also construct the nodes x_0, x_1 as the two roots of the Legendre polynomial $L_2(x)$. We know that $L_0(x) = 1$, $L_1(x) = x$, and the three-term recurrence for Legendre polynomials reads

$$(j+1)L_{j+1}(x) = (2j+1)xL_j(x) - jL_{j-1}(x),$$

so $2L_2(x) = 3x \cdot L_1(x) - L_0(x) = 3x^2 - 1$ or $L_2(x) = \frac{1}{2}(3x^2 - 1)$, which has roots $x_0, x_1 = \pm\frac{\sqrt{3}}{3}$. Next, one can compute the corresponding weight as the integral of the corresponding elementary Lagrange interpolation polynomials:

$$w_0 = \int_{-1}^{1} \frac{x - x_1}{x_0 - x_1} dx = \frac{-1}{2\sqrt{3}/3} \int_{-1}^{1} \left(x - \frac{\sqrt{3}}{3}\right) dx = \frac{-1}{2\sqrt{3}/3} \left(0 - 2\sqrt{3}/3\right) = 1,$$

$$w_1 = \int_{-1}^{1} \frac{x - x_0}{x_1 - x_0} dx = \frac{1}{2\sqrt{3}/3} \int_{-1}^{1} \left(x + \frac{\sqrt{3}}{3}\right) dx = \frac{1}{2\sqrt{3}/3} \left(0 + 2\sqrt{3}/3\right) = 1.$$

This computation yields the same quadrature rule:

$$G_1(f) = f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right).$$

Last remarks on Integration

- Gauss quadrature requires usually some extra work to set up (mainly the computation of the nodes and weights), but it is usually more stable and accurate than comparable Newton-Côtes rules.
- Adaptive quadrature rules, whether built out of simple Newton-Côtes or high-order Gaussian quadrature rules using an error estimator to guide the construction of a composite rule using a non-uniform partition of the interval, will usually be more effective as a general-purpose tool for accurate integration, especially when applied to a function which is not uniformly smooth throughout the interval of interest. A review of error estimators and adaptive algorithms can be found in the textbook, section 9.7.
- Acceleration methods such as Anderson acceleration and, in the case of quadrature, Romberg integration, are useful tools to accelerate the convergence of low-order methods such as the trapezoidal rule without requiring the additional work of setting up e.g. more complex Gaussian quadrature rules.
- There exist methods for both *improper integrals*, where the interval of interest is not bounded, as well as *singular integrals* where the function to be integrated blows up at a given point(s) in the interval but whose integral remains finite, but such cases require special care.

Lecture 18: Numerical Differentiation. (Monday, October 25)

Let us now turn to the different question of approximating derivatives $f'(x_i)$ at a set of nodes $x_0, \ldots, x_n \in [a, b]$ given values of $f(x)$ at the nodes.
In principle, one could use the interpolating polynomial to compute such approximations:

$$f'(x_i) \approx (\Pi_n f)'(x_i) = \sum_{k=0}^{n} f(x_k)\ell_k'(x_i).$$

However, this approach is both expensive (the computation of $\ell_k'(x_i)$ requires $O(n^2)$ work in general) and rather unstable because of the Runge phenomenon for high-order interpolation, so we focus on efficient low-order schemes called *finite differences*.

Remark 18.1. *One remarkable exception is interpolation using a set of Gauss quadrature nodes, usually using a scaled set of Chebyshev-Gauss* $x_j = -\cos[\pi(j+1/2)/(n+1)]$ *or Chebyshev-Gauss-Lobatto nodes* $\overline{x}_j = -\cos[j\pi/n]$, $j = 0, \ldots, n$. *This approach can be shown to yield an efficient, stable scheme whose accuracy is limited only by the smoothness of the function to be approximated, whether approximating the function, its derivatives or integrals, see Section 10.3 of the textbook and the Matlab package* `Chebfun` *developed by Prof. Nick Trefethen (Oxford) for a set of Matlab routines allowing to experiment with the power of Chebyshev interpolation.*

In the following, we assume the nodes $\{x_i\}$ are distributed uniformly on the interval $[a, b]$:

$$x_i = a + ih, \qquad i = 0, \ldots, n, \qquad\qquad h = \frac{b-a}{n}.$$

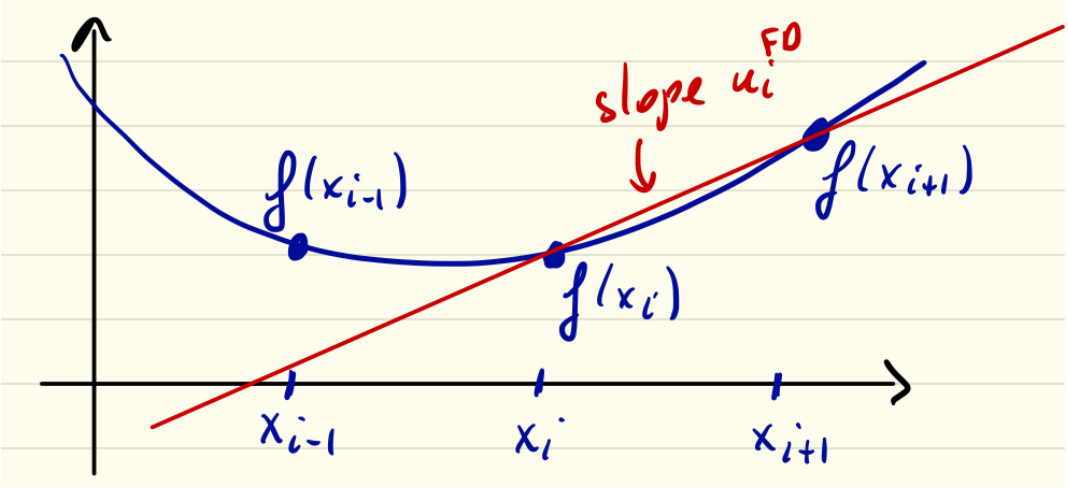
18.1 Classical Finite Difference Schemes

Forward Finite Difference formula. The very definition of the derivative gives us a first hint at an approximation:

$$f'(x_i) = \lim_{h \to 0^+} \frac{f(x_i + h) - f(x_i)}{h}.$$

Taking a *fixed* step size $h > 0$ in this limit, such that $x_i + h = x_{i+1}$, we obtain a first numerical approximation of the derivative, the **forward finite difference**:

$$f'(x_i) \approx u_i^{FD} = \frac{f(x_{i+1}) - f(x_i)}{h}, \qquad\qquad i = 0, \ldots, n-1.$$



Graphically, the formula computes the slope of the secant to the curve between x_i and x_{i+1}, which is expected to converge to the slope $f'(x_i)$ of the tangent to the curve as $h \to 0$.
To estimate the error attached to this formula, we expand $f(x)$ in a Taylor series of order 2 around x_i:

$$f(x_{i+1}) = f(x_i + h) = f(x_i) + hf_i'(x) + \frac{h^2}{2}f''(\xi_i), \qquad \xi_i \in (x_i, x_{i+1}),$$

and reorder the terms we find

$$f_i'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{h}{2}f''(\xi_i)$$

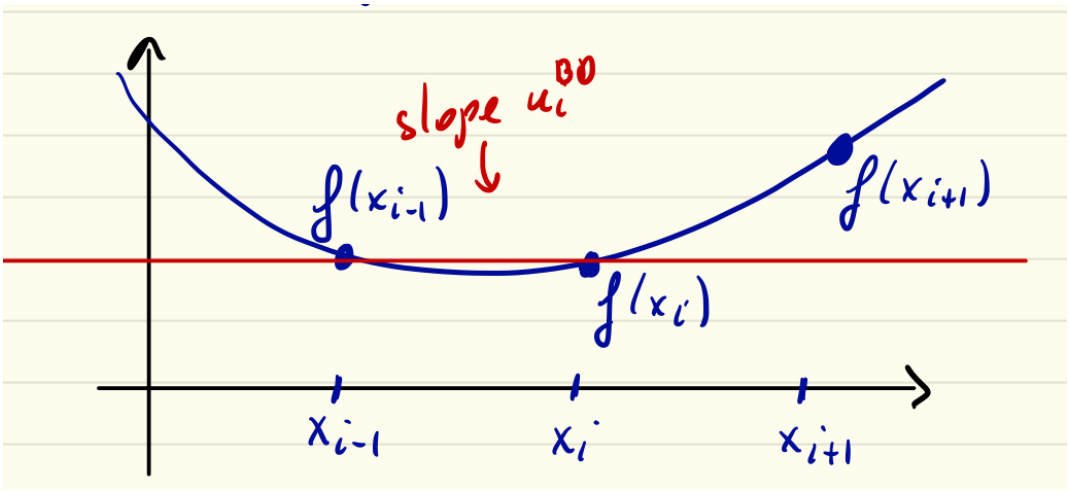
that is an error formula:

$$f_i'(x_i) - u_i^{FD} = \frac{-h}{2}f''(\xi_i),$$

where $\xi_i \in (x_i, x_{i+1})$. This shows in particular that the error is of order $O(h)$, which goes to zero as $h \to 0$.

Backwards finite difference formula. Next, we can also write a formula involving the previous node x_{i-1}:

$$f'(x_i) \approx u_i^{BD} = \frac{f(x_i) - f(x_{i-1})}{h}, \qquad i = 1, \ldots, n.$$



Graphically, the formula computes the slope of the secant to the curve between x_{i-1} and x_i, which is expected to converge to the slope $f'(x_i)$ of the tangent to the curve as $h \to 0$.
This formula behaves very similarly to the previous one, and using a similar Taylor expansion of order 2 around x_i, we derive an error formula:

$$f(x_{i-1}) = f(x_i - h) = f(x_i) + (-h)f_i'(x) + \frac{(-h)^2}{2}f''(\xi_i), \qquad \xi_i \in (x_{i-1}, x_i),$$

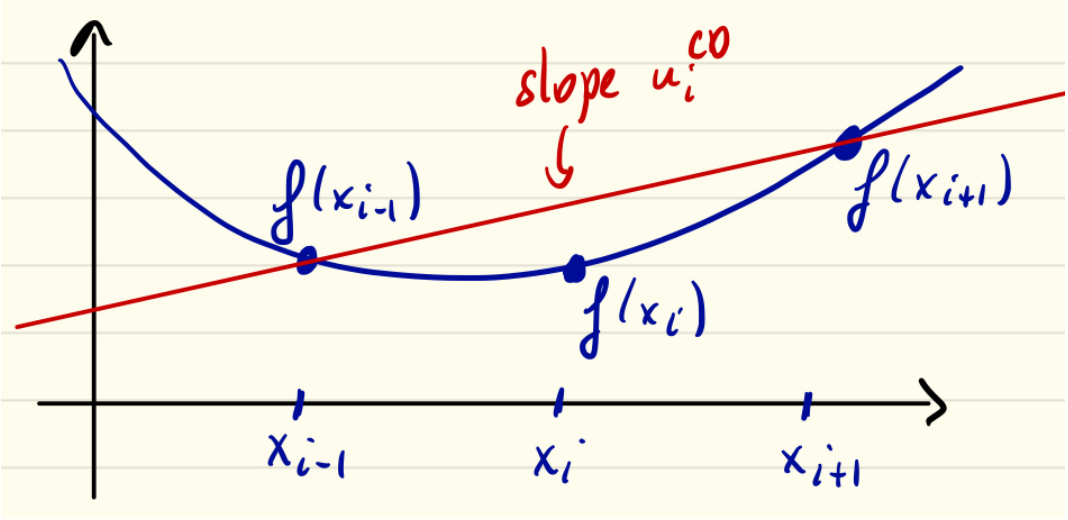
and thus

$$f_i'(x_i) - u_i^{BD} = \frac{h}{2}f''(\xi_i),$$

this time with $\xi_i \in (x_{i-1}, x_i)$. In particular, the error has the same order (power of h) as the forward finite difference formula.

A better approximation can be obtained by forming the average of the two previous formulas, which is the so-called **centered finite difference**:

$$f'(x_i) \approx u_i^{CD} = \frac{f(x_{i+1}) - f(x_{i-1})}{2h}, \qquad i = 1, \ldots, n-1.$$



Graphically, the formula computes the slope of the secant to the curve between x_{i-1} and x_{i+1}. The error term can be derived by using a Taylor series of order 3 for $f(x)$ around x_i: we find

$$f(x_{i+1}) = f(x_i + h) = f(x_i) + hf_i'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_i^+), \qquad \xi_i^+ \in (x_i, x_{i+1}),$$

$$f(x_{i-1}) = f(x_i - h) = f(x_i) + (-h)f_i'(x) + \frac{(-h)^2}{2}f''(x) + \frac{(-h)^3}{6}f'''(\xi_i^-), \qquad \xi_i^- \in (x_i, x_{i+1}),$$

such that

$$f(x_{i+1}) - f(x_{i-1}) = 2hf_i'(x) + \frac{h^3}{6}\left[f'''(\xi_i^+) + f'''(\xi_i^-)\right].$$

Dividing by $2h$ and reordering, we find that

$$f_i'(x) - u_i^{CD} = -\frac{h^2}{6}\frac{\left[f'''(\xi_i^+) + f'''(\xi_i^-)\right]}{2}.$$

We now observe that by the intermediate value theorem / discrete mean value theorem, there is $\xi_i \in (x_{i-1}, x_{i+1})$ such that $f(\xi_i) = \frac{\left[f'''(\xi_i^+) + f'''(\xi_i^-)\right]}{2}$, which simplifies the error formula down to

$$f_i'(x) - u_i^{CD} = -\frac{h^2}{6}f'''(\xi_i).$$

This shows that the formula has order of accuracy $O(h^2)$ - much better than the forward or backwards finite difference formulae.

End-point formulae To approximate the derivate at one of the end-points of the interval, only one of the three formulae above is usable: the forward FD (left end-point) or the backwards FD (right end-point). Neither of these formulae is as accurate as the centered finite difference, which is why we introduce the **left end-point formula** and **right end-point formula**:

$$u_0^{LEP} = \frac{-3f(x_0) + 4f(x_1) - f(x_2)}{2h}, \qquad u_n^{REP} = \frac{f(x_{n-2}) - 4f(x_{n-1}) + 3f(x_n)}{2h}.$$

Using Taylor expansions around x_0 and x_n respectively, one shows that both methods are $O(h^2)$ accurate (homework).

Definition 18.2.
- *A finite difference formula in which only 2 points appear, e.g. x_i and x_{i+1} (forward FD) or x_i and x_{i-1}, is called a 2-point finite difference formula.*
- *A finite difference formula in which 3 points appear, e.g. x_{i-1}, x_i and x_{i+1} (centered FD), or x_i, x_{i+1} and x_{i+2} (left end-point FD) is called a 3-point finite difference formula.*

18.2 Method of Undetermined Coefficients

Given a point x where one wishes to approximate the derivative and neighboring nodes $x \pm h$, $x \pm 2h$, ... we seek generically to compute approximations the derivative $f'(x)$ of the form of a linear combination of the values of f at these nodes,

$$u = \frac{1}{h} \sum_{j=-m}^{m} A_j f(x + jh),$$

with the right coefficients $A_{-m}, \ldots, A_m$.

Example. the centered difference formula takes the form: $m = 1$,

$$u^{CD} = \frac{1}{h} \left(A_{-1} f(x - h) + A_0 f(x) + A_1 f(x + h) \right)$$

To determine the right coefficients, one seeks to find A_j such that the error $f'(x) - u$ has the highest order of h possible. We achieve this by computing Taylor expansions of order at least $2m + 1$ (one term for each coefficient in the expansion, plus one for the error term) around x, here:

$$\begin{aligned} f(x - h) &= f(x) - h f'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(\xi_-), \\ f(x) &= f(x), \\ f(x + h) &= f(x) + h f'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(\xi_+). \end{aligned}$$

We form the linear combination u as above with coefficients A_{-1}/h, A_0/h, A_1/h: grouping terms of the same order in h,

$$u = \frac{A_{-1} + A_0 + A_1}{h} f(x) + (-A_{-1} + A_1) f'(x) + \frac{h}{2} (A_{-1} + A_1) f''(x) + \frac{h^2}{6} \left(-A_{-1} f'''(\xi_-) + A_1 f'''(\xi_+) \right).$$

We seek coefficients such that $u = f'(x) + O(h^3)$, leading to the three conditions

$$\begin{aligned} A_{-1} + A_0 + A_1 &= 0, \\ A_1 - A_{-1} &= 1, \\ A_1 + A_{-1} &= 0. \end{aligned}$$

We solve this linear system and find

$$A_{-1} = -1/2, \qquad A_0 = 0, \qquad A_1 = 1/2,$$

and we recovered the coefficients of the centered finite difference and its error formula:

$$u^{CD} = \frac{f(x + h) - f(x - h)}{2h} = f'(x) + \frac{h^2}{6} \frac{(f'''(\xi_-) + f'''(\xi_+))}{2} = f'(x) + \frac{h^2}{6} f'''(\xi).$$

18.3 Difference formulae for the second derivative.

Using the same ideas, we can also approximate higher derivatives, starting with $f''(x_i)$ at some of the nodes. Using the method of undetermined coefficients, we propose the generic form for a $2m + 1$-point formula:

$$f''(x) \approx \qquad v = \frac{1}{h^2} \sum_{j=-m}^{m} A_j f(x + jh),$$

with the right constants $A_{-m}, \ldots, A_m$ such that $f''(x) - v$ has the highest order of h possible.

Example: 3-point centered formula with nodes $x-h$, x and $x+h$ ($m=1$).
Using the Taylor expansions of order $2m+1=3$ around x from the previous paragraph, we form the linear combination

$$v = \frac{A_{-1}+A_0+A_1}{h^2}f(x) + \frac{-A_{-1}+A_1}{h}f'(x) + \frac{1}{2}(A_{-1}+A_1)f''(x) + \frac{h}{6}\left(-A_{-1}f'''(\xi_-) + A_1 f'''(\xi_+)\right).$$

We seek coefficients such that $u = f''(x) + O(h)$, leading to the three conditions:

$$\begin{aligned} A_{-1}+A_0+A_1 &= 0,\\ A_1 - A_{-1} &= 0,\\ A_1 + A_{-1} &= 2. \end{aligned}$$

We solve this linear system and find

$$A_{-1}=1, \qquad A_0=-2, \qquad A_1=1,$$

and we find **the centered finite difference for the second derivative**:

$$v^{CD} = \frac{f(x-h)-2f(x)+f(x+h)}{h^2} = f''(x) + \frac{h}{6}\left(-f'''(\xi_-) + f'''(\xi_+)\right).$$

Now, we notice that, if f is of class C^4 then $f'''(\xi_+) - f'''(\xi_-)$ is itself of order $O(h)$, meaning the formula is more accurate than indicated by the previous error formula. To show this, we write Taylor expansions of order 4 for $f(x \pm h)$:

$$\begin{aligned} f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_-),\\ f(x) &= f(x),\\ f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_+). \end{aligned}$$

Then we recompute the error formula:

$$v^{CD} = \frac{f(x-h)-2f(x)+f(x+h)}{h^2} = f''(x) + \frac{h^2}{24}\left(f^{(4)}(\xi_-) + f^{(4)}(\xi_+)\right).$$

Using the intermediate value theorem, we find $\xi \in (x_{i-1}, x_{i+1})$ such that $f(\xi) = \frac{f^{(4)}(\xi_-)+f^{(4)}(\xi_+}{2}$ leading to

$$v^{CD} = f''(x) + \frac{h^2}{12}f^{(4)}(\xi).$$

This shows that the 3-point finite difference formula for the second derivative is also of order $O(h^2)$.

18.4 Application:

Finite Differences method for boundary value problems.
Assume we seek to find an approximate solution to a boundary value problem such as the Poisson equation,

$$\begin{cases} -u''(x) = f(x) \quad \text{for } x \in (a,b),\\ u(a) = u(b) = 0, \end{cases}$$

for a given continuous function $f(x)$ on $[a,b]$. To achieve this, we set up a grid of equidistant points $x_j = a + jh$ for $j = 0, \ldots, n$, $h = \frac{b-a}{n}$, and we seek values $u_0 \approx u(x_0), \ldots, u_n \approx u(x_n)$ such that

- $u''(x_j) \approx \frac{-u_{j-1}+2u_j-u_{j+1}}{h^2} = f(x_j)$, for $j = 1, \ldots, n-1$, and
- $u_0 = u_n = 0$ (the boundary conditions).

This yields a linear system of $n+1$ equations for $n+1$ unknowns:

$$\frac{1}{h^2}\begin{bmatrix} 1 & 0 & & \ldots & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & \ldots & 0 & 1 \end{bmatrix}\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \begin{bmatrix} 0 \\ f_1 \\ \vdots \\ f_{n-1} \\ 0 \end{bmatrix}.$$

This system can be solved easily using e.g. Matlab's backlash operator for the values $u_0, \ldots, u_n$. This is the simplest example of the **method of finite differences** for solving ODEs and PDEs in boundary value problems.

18.5 Pseudo-spectral differentiation (Chebfun).

If the function $f(x)$ is very smooth, a high-order, high-accuracy alternative to finite differences is the pseudo-spectral derivative

$$D_n f = \left(\Pi_n^{CGL} f\right)' \qquad \in \P_{n-1},$$

where $\Pi_n^{CGL} f$ is the interpolation operator using a well-chosen set of quadrature nodes, which is here the Chebyshev-Gauss-Lobatto quadrature nodes on $[a, b]$, that is

$$x_j = \frac{a+b}{2} + \frac{a-b}{2}\cos\left(\frac{\pi j}{n}\right), \qquad\qquad j = 0, \ldots, n.$$

This set of points ensures stability of interpolation such that, on the interval $[-1, 1]$ to keep estimates simple, for any $s \geqslant 1$ such that f has at least s derivatives in the space L^2_w,

1. In the L^2_w weighted norm $\|g\|_w = \left(\int_{-1}^1 |g(x)|^2 \frac{dx}{\sqrt{1-x^2}}\right)^{1/2}$, we have the error estimate:

$$\|f - \Pi_n^{CGL} f\|_w \leqslant C n^{-s} \left(\sum_{k=0}^{n} \|f^{(k)}\|_w^2\right)^{1/2},$$

   where the constant $C > 0$ depends in general on s. This means that the interpolant converges to f as $n \to \infty$ with order $1/n^s$.

2. In the ∞-norm $\|g\|_\infty = \sup_{-1\leqslant x\leqslant 1} |g(x)|$, we have the interpolation error estimate:

$$\|f - \Pi_n^{CGL} f\|_\infty \leqslant C n^{1/2-s} \left(\sum_{k=0}^{n} \|f^{(k)}\|_w^2\right)^{1/2},$$

   where the (possibly different) constant $C > 0$ also depends in general on s. This means that the interpolant also converges uniformly to f as $n \to \infty$ with (slightly lower) order $1/n^(s - 1/2)$.

3. Finally, the derivative of $\Pi_n^{CGL} f$ also converges to f' with order $1/n^{s-1}$, with the error estimate

$$\|f' - D_n f\|_\infty \leqslant C' n^{1-s} \left(\sum_{k=0}^{n} \|f^{(k)}\|_w^2 \right)^{1/2},$$

where the constant $C' > 0$ also depends in general on s.

Notice how the order of convergence is only limited by the smoothness of f, in particular if $f(x)$ can be differentiated an infinity of times, then the interpolant and its derivatives converge to f and its derivatives faster than any power of n. This kind of convergence is termed "spectral convergence".

Implementation. We have the Lagrange formula for the interpolant,

$$\Pi_n^{CGL} f(x) = \sum_{k=0}^{n} f(x_k) \ell_k(x),$$

so the derivative takes the form

$$D_n f(x_i) = \left(\Pi_n^{CGL} f \right)'(x_i) = \sum_{k=0}^{n} \ell_k'(x_i) f(x_k).$$

The right-hand side can be formulated as a matrix-vector product, using the *differentiation matrix* D with coefficients $D_{ik} = \ell_k'(x_i)$. Note that these coefficients have *explicit formulae* for the chosen set of points (see the textbook), which makes this a very practical expression:

$$\underbrace{\vec{f'}}_{\text{Vector of derivative values } [f'(x_0), \ldots, f'(x_n)]} = \underbrace{D}_{\text{Differentiation matrix}} \underbrace{\vec{f}}_{\text{Vector of function values } [f(x_0), \ldots, f(x_n)]}.$$

Lecture 19: Numerical Solution of ODEs. (Wednesday, October 27)

19.1 The Cauchy problem.

Given an interval $I \subset \mathbb{R}$, an initial point $t_0 \in I$ and value y_0, and a function $f : I \times \mathbb{R} \to \mathbb{R}$, we consider the initial value problem:
Find $y \in C^1(I)$ such that:

$$\begin{cases} y'(t) = f(t, y(t)), & \text{for all } t \in I, \\ y(t_0) = y_0. \end{cases} \tag{19.1}$$

Equivalent integral formulation Integrating directly these equations with respect to time yields an equivalent problem: find $y \in C^1(I)$ such that

$$y(t) = y_0 + \int_{t_0}^{t} f(s, y(s))ds.$$

Existence and Uniqueness of solutions. Before desigining numerical methods to approximate the solution of this problem, we need to understand under which condition the Cauchy problem is **well-posed** and **stable**. The following theorem provides a condition for the existence and uniqueness of solutions.

Theorem 19.1. *If $f(t, y)$ is continuous as a function of the two variables t, y and satisfies a Lipschitz condition in the variable y, i.e.*

$$|f(t, y_1) - f(t, y_2)| \leqslant L|y_1 - y_2|, \tag{19.2}$$

for all $t \in I$ and y_1, y_2 in $\mathbb{R}$, that is in the region $D = \{(t, y) \text{ s.t. } t \in I, y \in \mathbb{R}\}$,
then the Cauchy problem (19.3) *has a unique global solution $y(t)$ on I.*

The Lipschitz condition is obviously satisfied if the partial derivative f_y exists and is bounded, leading to the following result, which is slightly less general but easier to apply:

Corollary 19.2. *The Cauchy problem* (19.3) *has a unique global solution $y(t)$ on I if $f(t, y)$ is continuous, differentiable with respect to y and*

$$|f_y(t, y)| \leqslant L, \qquad \forall (t, y) \in D.$$

We skip both proofs.

Example. Let us show that the following Cauchy problem is well-posed:

$$y' = 1 + t\cos(ty(t)), \qquad y(0) = 0, \qquad t \in [0, 2].$$

Solution. Clearly, the function $f(t, y) = 1 + t\cos(ty)$ is continuous. Furthermore, it is differentiable with respect to y and

$$|f_y(t, y)| = |-t^2 \sin(ty)| \leqslant |t^2| \leqslant 4,$$

for all $t \in [0, 2]$ and $y \in \mathbb{R}$. By Corollary 2, this IVP has a unique solution.

19.2 Stability.

Now that we have some understanding of when the Cauchy problem has a unique solution, we turn our attention to the conditionning of the problem, that is how much the solution varies if one perturbs the data, that is y_0, f. We investigate the perturbed problem:
Find $z \in C^1(I)$ such that:

$$\begin{cases} z'(t) = f(t, z(t)) + \delta(t, z(t)), & \text{for all } t \in I, \\ z(t_0) = y_0 + \delta_0, \end{cases} \tag{19.3}$$

where the perturbations $\delta(t, y)$ is continuous and satisfies a Lipschitz condition in y of the type (19.2) to ensure this problem has a unique solution as in Theorem 19.1.

Definition 19.3. *Let I be a bounded interval of $\mathbb{R}$. The Cauchy problem* (19.3) *is called* ***stable*** *in the sense of Lyapunov, or* ***well-posed****, if for any perturbations δ_0, δ such that*

$$|\delta_0|, |\delta(t, y)| < \varepsilon, \qquad \forall (t, y) \in D,$$

then there exists $C > 0$ such that $|y(t) - z(t)| < C\varepsilon$, for all $t \in I$.

Letting $u(t) - z(t) - y(t)$, we see that u satisfies

$$\begin{cases} u'(t) = f(t, z(t)) + \delta(t, z(t)) - f(t, y(t)), & \text{for all } t \in I, \\ u(t_0) = \delta_0. \end{cases}$$

so that, using the Lipschitz condition for $f(t, y)$: $|u'(t)| \leqslant L|u(t)| + \varepsilon$. Now we integrate from t_0 to t and obtain

$$\begin{aligned} u(t) &= \delta_0 + \int_{t_0}^{t} u'(s)ds \\ |u(t)| &\leqslant |\delta_0| + \left| \int_0^t |u'(s)|ds \right| \\ &\leqslant \varepsilon + \left| \int_{t_0}^t L|u(s)| + \varepsilon ds \right| \\ &\leqslant (1 + |t - t_0|)\varepsilon + L \left| \int_{t_0}^t |u(s)|ds \right|. \end{aligned}$$

In order to conclude, we need the following result:

Lemma 19.4. *(simplified Gromwall's Lemma.) Let $v(t) \geqslant 0$ be a positive, continuous function on I that satisfies the integral inequality*

$$v(t) \leqslant A + B \int_{t_0}^t v(s)ds, \qquad A, B > 0.$$

Then $v(t) \leqslant Ae^{B(t-t_0)}$ for any $t \geqslant t_0$.

Proof. Define the function $V(t) = Be^{-B(t-t_0)} \int_{t_0}^t v(s)ds$, then

$$\begin{aligned} V'(t) &= B \left[v(s) - B \int_{t_0}^t v(s)ds \right] e^{-B(t-t_0)} \\ &\leqslant ABe^{-B(t-t_0)}, \end{aligned}$$

where we have used the integral inequality. Furthermore, $V(t_0) = 0$ so that, integrating V' from t_0 to t,

$$V(t) = \int_{t_0}^{t} V'(s)ds \leqslant A \int_{t_0}^{t} Be^{-B(s-t_0)}dt = A\left[1 - e^{-B(t-t_0)}\right].$$

Recalling the definition of $V(t)$, this means

$$B \int_{t_0}^{t} v(s)ds \leqslant Ae^{B(t-t_0)} - A,$$

and recalling the integral inequality $v(t) \leqslant A + B\int_{t_0}^{t} v(s)ds \leqslant Ae^{B(t-t_0)}$, we have finished. □

Let us resume our analysis of the perturbed solution. Let $T = \max_{t\in I} |t-t_0|$. If $t \geqslant t_0$, we conclude from Gromwall's lemma applied to $v(t) = |u(t)|$ with $A = (1+T)\varepsilon$ and $B = L$ that

$$|u(t)| \leqslant (1+T)\varepsilon e^{L|t-t_0|} \leqslant (1+T)e^{LT}\varepsilon,$$

and if $t < t_0$, we set $t' = t_0 - t \geqslant t'_0 = 0$, and $v(t') = |u(t_0 - t')|$ such that

$$v(t') = |u(t_0 - t')| \leqslant (1+T)\varepsilon + L\left|\int_{t_0}^{t_0-t'} |u(s)|ds\right| = (1+T)\varepsilon + L\int_{0}^{t'} v(s)ds.$$

Applying Gromwall's lemma to $v(t')$ with $t'_0 = 0$, $A = (1+T)\varepsilon$, $B = L$ yields the desired inequality

$$|u(t)| = v(t') \leqslant (1+T)e^{LT}.$$

To conclude, we have found that for $T = \max_{t\in I} |t - t_0|$ and $C = (1+T)e^{LT}$, for any $t \in I$,

$$|u(t)| \leqslant C\varepsilon.$$

We have proved:

Theorem 19.5. *If f is continuous and satisfies the Lipschitz condition* (19.2)*, then the Cauchy problem is stable / well posed.*

Remark 19.6. *We have shown that the condition number of the problem is at most $C = (1+T)e^{LT}$, which grows exponentially fast in the interval width T. This means the problem may be quite ill-posed on long intervals, depending on the properties of the problem, a result which should be improved for particular problems to be really usable.*

19.3 One-step numerical methods.

Fix $T > 0$ and the integration interval $I = [t_0, t_0 + T]$. We seek to approximate the solution of the Cauchy problem on I,

$$\begin{cases} y'(t) = f(t, y(t)), & \text{for all } t \in I, \\ y(t_0) = y_0. \end{cases}$$

We form a set of regularly spaced nodes $t_n = t_0 + nh$, $n = 0, \ldots, N$ with step size $h = T/N$, and we seek approximate solution values $u_n \approx y(t_n)$.
For convenience, we will denote

$$y_n = y(t_n), \qquad f_n = f(t_n, u_n).$$

19.3.1 Forward Euler method.

There are many ways to derive the following approach.

Using a forward finite difference formula. We seek that

$$\frac{u_{n+1} - u_n}{h} \approx y'(t_n) = f(t_n, y_n) \approx f(t_n, u_n),$$

Since we know $u_0 = y_0$, this leads to the recurrence (**Euler's method**):

$$u_0 = y_0, \qquad u_{n+1} = u_n + hf_n, \qquad f_n = f(t_n, u_n), \qquad \text{for } 0 \leqslant n \leqslant N-1. \tag{19.4}$$

Since we know the initial value exactly, we can understand the error committed during the first step exactly!

$$u_1 = y_0 + hf(t_0, y_0) = y(t_0) + hy_0'(t_0) \approx y(t_0 + h) = y(t_1).$$

This corresponds to making a step of length h along the tangent of the curve $y(t)$ at $t = t_0$.

Using a Taylor expansion:

$$y(t_1) = y(t_0 + h) = \underbrace{y(t_0) + hy'(t_0)}_{=u_1} + \frac{h^2}{2} y''(\xi)$$

leading to the error formula for the first step of Euler's method:

$$y_1 - u_1 = \frac{h^2}{2} y''(\xi).$$

Integration and quadrature: Using the alternative integral formulation to the Cauchy problem:

$$\begin{aligned} y(t_1) &= y_0 + \int_{t_0}^{t_1} f(t, y(t))dt \\ &\approx y_0 + \underbrace{(t_1 - t_0) f(t_0, y(t_0))}_{left rectangle rule} \\ &= y_0 + hf_0 \\ &= u_1. \end{aligned}$$

These are three different interpretations of Euler's method: using the forward finite difference approximation to the derivatives of y at the nodes $t_0, \ldots, t_N$; and for the first step at least, a Taylor expansion of order 1 of $y(t)$ around t_0, or as an approximation of the integral formulation using the left rectangle rule.

Algorithm 4 Forward Euler method.

Input: Function $f(t, y)$, initial data t_0, y_0, T, number of steps N.

Output: Approximate values $u_0, \ldots, u_N$ of the solution.

function ForwardEuler(f, t_0, y_0, N)
    $h = T/N$;
    $u = \text{zeros}(N + 1,1)$;
    $u_0 = y_0$;
    for $n = 0 \ldots N - 1$ **do**
        $u_{n+1} = u_n + f(t_n, u_n)$;
    end for
    return u
end function

Lecture 20: Analysis of one-step methods, I. (Monday, November 2)

20.1 Some one-step methods

Remember that we are looking for approximations

$$u_n \approx y(t_n),$$

where $y(t)$ is the solution to an initial value problem

$$\begin{cases} y'(t) = f(t, y(t)), & \text{for all } t \in [t_0, t_0 + T], \\ y(t_0) = y_0. \end{cases} \tag{20.1}$$

Definition 20.1. *A **one-step numerical scheme** for the approximation of* (20.1) *is one where* u_{n+1} *depends only on* u_n.
If u_{n+1} *depends on* u_n, u_{n-1}, ... *the scheme is **multistep.***

20.1.1 Forward Euler method

See the previous lecture.

20.1.2 Backwards Euler method

Instead of using a forward finite difference formula to approximate the derivatives $y'(t_n)$, which leads to the forward Euler method, one can use a backwards finite difference formula for $y'(t_{n+1})$, and this leads to the backwards Euler method:

$$\frac{u_{n+1} - u_n}{h} \approx y'(t_{n+1}) \approx f(t_{n+1}, u_{n+1}).$$

This can be reformulated as the scheme:

$$u_{n+1} = u_n + hf(t_{n+1}, u_{n+1}).$$

Note that unlike the previous case, this equation does not define explicitely u_{n+1},but rather defines it implicitely as the solution of an equation (nonlinear, in general). In practice, a root-finding scheme studied earlier in the semester, such as Newton's method, may be used to find u_{n+1}.

Algorithm 5 Backwards Euler method.

Input: Function $f(t, y)$, initial data t_0, y_0, T, number of steps N.

Output: Approximate values $u_0, \ldots, u_N$ of the solution.

```
1: function BackwardsEuler(f, t_0, y_0, N)
2:     h = T/N;
3:     u = zeros(N + 1,1);
4:     u_0 = y_0;
5:     for n = 0 ... N − 1 do
6:         SOLVE u_{n+1} = u_n + h f(t_{n+1}, u_{n+1});
7:     end for
8:     return u
9: end function
```

20.1.3 Trapezoidal or Crank-Nicholson method

The forward and backwards Euler method can also be seen as left- and right-rectangle quadrature rules applied to the integral formulation of the initial value problem:

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t))dt.$$

A more accurate approximation can be achieved by using a trapezoidal rule:

$$y_{n+1} \approx y_n + \frac{t_{n+1} - t_n}{2} \left[f(t_n, y(t_n)) + f(t_{n+1}, y(t_{n+1})) \right],$$

which leads to the Crank-Nicholson scheme:

$$u_{n+1} = u_n + \frac{h}{2} \left(f(t_n, u_n) + f(t_{n+1}, u_{n+1}) \right).$$

Just like the backwards Euler method, this formula defines u_{n+1} only implicitely, as a solution to a nonlinear equation.

20.1.4 Heun's method

In order to avoid having to solve a nonlinear equation for u_{n+1}, we can replace its value on the right-hand side of the Crank-Nicholson scheme by an approximation, obtained with one step of the forward Euler method: $u_{n+1} \approx u_n + hf(t_n, u_n)$. This approximation yields Heun's method:

$$u_{n+1} = u_n + \frac{h}{2} \left[f(t_n, u_n) + f\left(t_{n+1}, u_n + hf(t_n, u_n)\right) \right].$$

20.1.5 Explicit vs Implicit schemes

We have introduced four schemes so far. In two of them (the forward Euler and Heun's method), u_{n+1} is given as an explicit formula in terms of u_n, while for the other two (backwards Euler and Crank-Nicholson), u_{n+1} is given as the solution of a nonlinear equation. This distinction can be generalized to any scheme, including ones that are multistep.

Definition 20.2. *A numerical method (or scheme) for the solution of the initial value problem* (20.1) *is called* ***explicit*** *if u_{n+1} can be computed directly, as an explicit formula in terms of the previous values u_k, $k \leqslant n$.*
A scheme is called ***implicit*** *if u_{n+1} is given as the solution of an implicit, nonlinear equation.*

20.2 Analysis of 1-step methods

Let us recall now the concepts introduced in the first chapter, when we investigated the general idea of numerical schemes to solve equations or systems of equations for given data d of the form

$$F(x, d) = 0 \qquad \text{with} \qquad F_n(x_n, d) = 0,$$

where F_n is an equation than can be solved in practice for an approximate solution x_n to the exact solution x. In particular, we recall the general result that

$$\text{CONSISTENCY} \quad + \quad \text{STABILITY} \quad \Longrightarrow \quad \text{CONVERGENCE.}$$

20.2.1 Consistency.

We recall our original definition in Section 3.3:

Definition 20.3. *A scheme is said to be* ***consistent*** *if*

$$F_n(x, d) = F_n(x, d) - F(x, d) \to 0 \qquad \textit{as} \qquad n \to \infty,$$

where x is the exact solution with exact data d.

We put this definition into action for one-step numerical schemes, which broadly take the form $u_{n+1} = F_h(t_n, u_n)$. We can rewrite these as the set of equations:

$$\begin{aligned}
&u_1 - F_h(t_0, u_0) = 0,\\
&u_2 - F_h(t_1, u_1) = 0,\\
&\vdots\\
&u_{n+1} - F_h(t_n, u_n) = 0,\\
&\vdots\\
&u_N - F_h(t_{N-1}, u_{N-1}) = 0.
\end{aligned}$$

To implement the idea of consistency, we plug in the exact solution $y_n = y(t_n)$ into these equations, resulting in error terms on the right-hand side since the exact solution does *not* satisfy the approximate scheme:

$$\begin{aligned}
&y_1 - F_h(t_0, y_0) = \varepsilon_1,\\
&y_2 - F_h(t_1, y_1) = \varepsilon_2,\\
&\vdots\\
&y_{n+1} - F_h(t_n, y_n) = \varepsilon_{n+1},\\
&\vdots\\
&y_N - F_h(t_{N-1}, y_{N-1}) = \varepsilon_N.
\end{aligned}$$

To sum up, consistency is concerned with the difference arising at each point t_{n+1} between $y_{n+1} = y(t_{n+1})$, the exact solution at t_{n+1}, and $\widetilde{y}_{n+1} = F_h(t_n, y_n)$, the result of applying our numerical scheme for one step only using the exact initial data $y_n = y(t_n)$:

$$\varepsilon_{n+1} = \underbrace{y(t_{n+1})}_{\text{exact solution at } t_{n+1}} - \underbrace{\widetilde{y}_{n+1}}_{\text{result of 1 step of the numerical scheme with data } y_n = y(t_n)}.$$

More precisely, by inspection of the schemes above we notice that one-step schemes broadly take the form

$$u_{n+1} = F_h(t_n, u_n) \qquad = u_n + h\Phi(t_n, u_n, u_{n+1}; h),$$

for $n = 0, \dots, N_1$, where the function $\Phi(t_n, u_n, u_{n+1}; h)$ is called the *increment function* of the time-stepping scheme. Note that Φ depends in practice on u_{n+1} only for implicit schemes. In this case, we can rearrange the equations above and divide by h on both sides:

$$\frac{u_{n+1} - u_n}{h} - \Phi(t_n, u_n, u_{n+1}; h) = 0 \quad \rightarrow \quad \frac{y_{n+1} - y_n}{h} - \Phi(t_n, y_n, y_{n+1}; h) = \frac{\varepsilon_{n+1}}{h} = \tau_{n+1}(h).$$

The new quantity $\tau_{n+1}(h)$ measures the difference between the actual increment $(y_{n+1} - y_n)/h$ and the approximate increment of the scheme $\Phi(t_n, y_n, y_{n+1}; h)$, both computed using values of the exact solution $y(t)$. We shall see that this definition of the consistency error is the relevant one in the next sections.

Definition 20.4. *The quantity* $\tau_{n+1}(h) = \dfrac{\varepsilon_{n+1}}{h} = \dfrac{y_{n+1} - \widetilde{y}_{n+1}}{h}$ *is called the* ***local truncation error*** *or LTE.*
The ***global truncation error*** *is* $\tau(h) = \max_{0 \leqslant n \leqslant N-1} |\tau_{n+1}(h)|$.

Definition 20.5.

- *A one-step numerical scheme is called* ***consistent*** *if*

$$\lim_{h \to 0} \tau(h) = 0,$$

  i.e. the local truncation error converges to zero as $h \to 0$*, uniformly in* n*.*

- *A scheme has order* p *for* $p \geqslant 1$ *if it is consistent and*

$$|\tau_{n+1}(h)| \leqslant Ch^p,$$

  where $C > 0$ *is a constant independent of* n*, but which may depend on* f*,* y_0 *and the scheme.*

Applications: the forward Euler scheme. Here, we have $u_{n+1} = u_n + hf(t_n, u_n)$, so $\widetilde{y}_{n+1} = y_n + f(t_n, y_n)$, so the difference reads

$$\varepsilon_{n+1} = y_{n+1} - [y_n + hf(t_n, y_n)].$$

Using a Talor expansion, we compute:

$$y_{n+1} = y(t_n + h) = \underbrace{y(t_n) + hy'(t_n)}_{= \widetilde{y}_{n+1}} + \frac{h^2}{2} y''(\xi_n),$$

such that we have the error formula

$$\varepsilon_{n+1} = \frac{h^2}{2} y''(\xi_n)$$

for some $\xi_n \in (t_n, t_{n+1})$.
The local and global truncation errors of the forward Euler scheme thus read

$$\tau_{n+1}(h) = \frac{h}{2} y''(\xi_n), \qquad \text{and} \qquad \tau(h) \leqslant \frac{h}{2} \max_{\xi \in [t_0, t_0+T]} |y''(\xi)|.$$

The forward Euler scheme is consistent with order 1.

Backwards Euler scheme. The forward Euler scheme is consistent with order 1 **(homework)**.

Crank-Nicholson scheme. Here, we have $u_{n+1} = u_n + \frac{h}{2}\left[f(t_n, u_n) + f(t_{n+1}, u_{n+1})\right]$, so the increment function for this implicit scheme reads

$$\Phi_{CN}(t_n, u_n, u_{n+1}; h) = \frac{1}{2}\left[f(t_n, u_n) + f(t_{n+1}, u_{n+1})\right].$$

Let us compute the difference between the exact $(y_{n+1} - y_n)/h$ and approximate increment Φ. Using the integral formulation of the Cauchy problem and the error formula for the trapezoidal rule, we have

$$\begin{aligned}
\frac{y_{n+1} - y_n}{h} &= \frac{1}{h} \int_{t_n}^{t_{n+1}} y'(t) dt \\
&= \frac{1}{h} \left(\frac{t_{n+1} - t_n}{2} \left(y'(t_n) + y'(t_{n+1})\right) - \frac{h^3}{12} y'''(\xi_n) \right) \\
&= \frac{1}{h} \left(\frac{h}{2} \left(f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right) - \frac{h^3}{12} y'''(\xi_n) \right) \\
&= \Phi_{CN}(t_n, y_n, y_{n+1}; h) - \frac{h^2}{12} y'''(\xi_n).
\end{aligned}$$

This computation gives us the local truncation error:

$$\tau_{n+1}(h) = \frac{y_{n+1} - y_n}{h} - \Phi_{CN}(t_n, y_n, y_{n+1}; h) = -\frac{h^2}{12} y'''(\xi_n), \qquad \xi \in (t_n, t_{n+1}),$$

such that $\tau(h) \leqslant \frac{h^2}{12} \|y'''\|_\infty$:
The Crank-Nicholson scheme is consistent with order 2.

Example: Heun's method. Here, we have $u_{n+1} = u_n + \frac{h}{2}\left[f(t_n, u_n) + f(t_{n+1}, u_n + hf(t_n, u_n))\right]$, so the increment function for this explicit scheme reads

$$\Phi_H(t_n, u_n; h) = \frac{1}{2}\left[f(t_n, u_n) + f(t_{n+1}, u_n + hf(t_n, u_n))\right].$$

Heun's method is consistent with order 2 (homework).

20.2.2 Zero-stability

The next concept on the road to convergence is the stability of the numerical scheme, that is its resilience to perturbations of the data y_0 and Φ. In parallel to the stability of the continuous problem, we introduce the concept of zero-stability:

Definition 20.6. *A one-step numerical scheme for the Cauchy problem* (20.1) *is called* ***zero-stable*** *if, for $h < h_0$ small enough, there exists $C > 0$ such that, for $\varepsilon > 0$ small enough, given perturbations $|\delta_n| \leqslant \varepsilon$ for $0 \leqslant n \leqslant N_h = T/h$, then*

$$|u_n - z_n| \leqslant C\varepsilon, \qquad n = 0, \dots, N_h,$$

where u_n is the solution to the exact scheme

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = u_n + h\Phi(t_n, u_n, u_{n+1}; h), \end{cases}, \qquad n = 0, \dots, N_h - 1, \tag{20.2}$$

and z_n is the solution to the perturbed iteration

$$\begin{cases} z_0 = y_0 + \delta_0, \\ z_{n+1} = z_n + h\left[\Phi(t_n, z_n, z_{n+1}; h) + \delta_{n+1}\right], \end{cases} \qquad n = 0, \dots, N_h - 1. \tag{20.3}$$

A zero-stable scheme is one that will keep under control the cumulative effect of errors occuring at each step of the computation, such as rounding errors, errors due to solving approximately the equations for implicit schemes, etc.
While the definition is quite complicated, it turns out that proving zero-stability is similar to proving stability for the continuous Cauchy problem.

Theorem 20.7. *Consider a one-step scheme with increment function $\Phi(t_n, u_n, u_{n+1}; h)$ which is Lipschitz-continuous w.r.t. u_n for $h < h_0$ small enough:*

$$|\Phi(t_n, u_n, u_{n+1}; h) - \Phi(t_n, z_n, z_{n+1}; h)| \leqslant \Lambda\left(|u_n - u_{n+1}| + |u_{n+1} - z_{n+1}|\right),$$

where Λ is a constant independent of h, n and t_n. Then the scheme is zero-stable.

Proof. Set $w_n = z_n - u_n$, then by taking the difference between (20.3) and (20.2):

$$\begin{aligned} w_{n+1} &= w_n + h\left[\Phi(t_n, z_n, z_{n+1}; h) + \delta_{n+1} - \Phi(t_n, u_n, u_{n+1}; h)\right] \\ &= w_n + h\left[\Phi(t_n, z_n, z_{n+1}; h) - \Phi(t_n, u_n, u_{n+1}; h)\right] + h\delta_{n+1}. \end{aligned}$$

Using the Lipschitz condition from the theorem, we find that

$$|w_{n+1}| \leqslant |w_n| + h\Lambda(|w_n| + |w_{n+1}|) + h|\delta_{n+1}|,$$

hence if $h < h_0 = 1/2\Lambda$, $\delta_{n+1} \leqslant \varepsilon$, then

$$|w_{n+1}| \leqslant \left(\frac{1 + h\Lambda}{1 - h\Lambda}\right)|w_n| + \frac{h\varepsilon}{1 - h\Lambda}.$$

Let us set $e = \frac{h\varepsilon}{1-h\Lambda}$ and $K = \frac{2h\Lambda}{1-h\Lambda}$ such that $1 + K = \frac{1+h\Lambda}{1-h\Lambda}$, then we find by recursion,

$$\begin{aligned} |w_0| &= |\delta_0|, \\ |w_1| &\leqslant (1+K)|w_0| + e &&= (1+K)|\delta_0| + e, \\ |w_2| &\leqslant (1+K)|w_1| + e &&\leqslant (1+K)^2|\delta_0| + \left[(1+K) + 1\right]e, \\ |w_3| &\leqslant (1+K)|w_2| + e &&\leqslant (1+K)^3|\delta_0| + \left[(1+K)^2 + (1+K) + 1\right]e, \\ &&&\vdots \\ |w_{n+1}| &\leqslant (1+K)|w_n| + e &&\leqslant (1+K)^{n+1}|\delta_0| + \left[(1+K)^n + \cdots + (1+K) + 1\right]e, \end{aligned}$$

and using the geometric series formula, we find:

$$|w_{n+1}| \leqslant (1+K)^{n+1}|\delta_0| + \frac{(1+K)^{n+1}-1}{1+K-1}e.$$

Since $1 + K \leqslant e^K$, we have the bound

$$|w_{n+1}| \leqslant e^{K(n+1)}|\delta_0| + \frac{e}{K}\left(e^{K(n+1)} - 1\right).$$

Re-inserting the values of the constants K, e, since $t_{n+1} - t_0 = (n+1)h$ we find

$$|w_{n+1}| \leqslant |\delta_0| e^{\frac{2\Lambda}{1-h\Lambda}(t_{n+1}-t_0)} + \frac{\varepsilon}{2\Lambda}\left(e^{\frac{2\Lambda}{1-h\Lambda}(t_{n+1}-t_0)} - 1\right).$$

Finally, assuming $\delta_0 \leqslant \varepsilon$, $h < h_0 = 1/2\Lambda$ and $t_{n+1} - t_0 < T$ we obtain the desired zero-stability bound:

$$|w_n| < \left[e^{4\Lambda T} + \frac{e^{4\Lambda T}-1}{2\Lambda}\right]\varepsilon, \qquad n = 0, \ldots, N.$$

□

Note that, while this proves the desired result, the bound which is obtained is quite unsatisfying - the constant grows exponentially fast with the length of the integration interval T, apparently limiting the usefulness of the numerical methods to at most a few $1/\Lambda$ units of time before unacceptable deviations from the exact solution.

Example. The forward Euler method is zero-stable provided $f(t, y)$ satisfies the usual Lipschitz condition (19.2) in y. Indeed, in this case

$$\Phi_{FE}(t_n, u_n, u_{n+1}; h) = f(t_n, u_n),$$

hence

$$|\Phi(t_n, u_n, u_{n+1}; h) - \Phi(t_n, z_n, z_{n+1}; h)| = |f(t_n, u_n) - f(t_n, z_n)| \leqslant L|u_n - z_n|,$$

and the theorem applies with $\Lambda_{FE} = L$. The Crank-Nicholson method, indeed is also zero-stable

$$\Phi_{CN}(t_n, u_n, u_{n+1}; h) = \frac{1}{2}\left(f(t_n, u_n) + f(t_{n+1}, u_{n+1})\right),$$

hence

$$\begin{aligned}|\Phi(t_n, u_n, u_{n+1}; h) - \Phi(t_n, z_n, z_{n+1}; h)| &\leqslant \frac{1}{2}|f(t_n, u_n) - f(t_n, z_n)| + \frac{1}{2}|f(t_{n+1}, u_{n+1}) - f(t_{n+1}, z_{n+1})| \\ &\leqslant \frac{L}{2}\left(|u_n - z_n| + |u_{n+1} - z_{n+1}|\right),\end{aligned}$$

so the theorem applies with $\Lambda_{CN} = L/2$. As an exercise, you can check that the backwards Euler and Heun's schemes are both zero-stable, and in general most numerical schemes are zero-stable for reasonable functions $f(t, y)$.

Lecture 21: Analysis of One-Step Methods, II. (Wednesday, November 4)

After defining and investigating in the previous section the consistency and zero-stability of one-step numerical schemes for the Cauchy problem, we now turn to the *convergence* which results from these two properties.

21.1 Convergence analysis

We use in this paragraph the notations from the previous lecture. Let us define the **global error**

$$e_n = |y_n - u_n|.$$

Definition 21.1.

- *A scheme is called **convergent** if*

$$\lim_{h\to 0}\left[\max_{0\leqslant n\leqslant N_h}|u_n - y_n|\right] = 0.$$

- *It **converges with order** p if*

$$|u_n - y_n| \leqslant Ch^p, \qquad n = 0, \dots, N_h,$$

with a constant $C > 0$ independent of h and n, but which may depend on the data f, y_0, T and the scheme itself.

Theorem 21.2 (Lax-Richtmeyer equivalence theorem.)**.** *A numerical scheme which is both consistent and zero-stable is convergent.*
Moreover, if $|y_0 - u_0| = O(h^p)$ and the method has order p, then it converges with order p.

The global error thus has the same order $O(h^p)$ as the local truncation error.

Proof. We make the observation that, by definition of the *local truncation error*, the values of the exact solution $y_n = y(t_n)$ are in fact obtained by the following perturbation of our numerical scheme:

$$\begin{cases} y_0 = u_0 + (y_0 - u_0), \\ y_{n+1} = y_n + h\Phi(t_n, y_n, y_{n+1}; h) + \tau_{n+1}(h), \qquad n = 0, \dots, N_h - 1. \end{cases}$$

Now, if we define perturbations $\delta_{n+1} = \tau_{n+1}(h)$ and $\delta_0 = y_0 - u_0$, this system has exactly the form (20.3). Hence, if the scheme is zero stable, then for $h > 0$ small enough such that $h < h_0$ and $|\delta_{n+1}| = |\tau_{n+1}(h)| \leqslant \varepsilon = \max(\tau(h), |y_0 - u_0|)$ is small enough (which is possible since $\lim_{h\to 0}\tau(h) = 0$) then

$$|u_n - z_n| < C\max(\tau(h), |y_0 - u_0|).$$

Hence, if the scheme is consistent and $u_0 \to y_0$, the scheme is also convergent. Furthermore, if the scheme has order p and $|y_0 - u_0| = O(h^p)$ we obtain

$$|u_n - z_n| \leqslant C\left(C'h^p + O(h^p)\right) \leqslant C''h^p$$

for h small enough, so the scheme converges with order p and the second assertion is proved. □

21.2 Analysis of the forward Euler scheme.

To see once more the steps of the convergence proof, let us show the convergence of the forward Euler method without using the theorem above. We define

$$u_0 = y_0, \qquad u_{n+1} = u_n + hf(t_n, u_n), \qquad y_n = y(t_n),$$

and we set $\widetilde{y}_{n+1} = y_n + hf(t_n, y_n)$ obtained with one step of Euler's method with initial data y_n. The global error may be expanded as

$$\begin{aligned} e_{n+1} &= |y_{n+1} - u_{n+1}| && \text{(error at } t_{n+1}) \\ &\leqslant \underbrace{|y_{n+1} - \widetilde{y}_{n+1}|}_{\text{Local truncation error}} + \underbrace{|\widetilde{y}_{n+1} - u_{n+1}|}_{\text{Propagated error}}. \end{aligned}$$

Now we consider each part of the error separately. First, the local truncation error

$$y_{n+1} - \widetilde{y}_{n+1} = h\tau_{n+1}(h),$$

which we computed earlier as $\tau_{n+1}(h) = \frac{h}{2}y''(\xi_n)$, and second the propagated error

$$\begin{aligned} \widetilde{y}_{n+1} - u_{n+1} &= y_n + hf(t_n, y_n) - u_n - hf(t_n, u_n) \\ &= (y_n - u_n) + h(f(t_n, y_n) - f(t_n, u_n)), \end{aligned}$$

which we bound using the Lipschitz condition on f,

$$|\widetilde{y}_{n+1} - u_{n+1}| \leqslant e_n + hLe_n,$$

such that finally

$$e_{n+1} \leqslant h\tau(h) + (1 + hL)e_n.$$

Now, we show by recursion on n,

$$\begin{aligned} e_0 &= 0, \\ e_1 &\leqslant h\tau(h) + (1 + hL)e_0 && = h\tau(h), \\ e_2 &\leqslant h\tau(h) + (1 + hL)e_1 && = (1 + (1 + hL))h\tau(h), \\ e_3 &\leqslant h\tau(h) + (1 + hL)e_2 && = (1 + (1 + hL) + (1 + hL)^2)h\tau(h), \\ e_{n+1} &\leqslant h\tau(h) + (1 + hL)e_n && = (1 + (1 + hL) + \cdots + (1 + hL)^n)h\tau(h) \\ &\leqslant \frac{(1 + hL)^{n+1} - 1}{1 + hL - 1}h\tau(h) \\ &\leqslant \frac{e^{hL(n+1)} - 1}{L}\tau(h). \end{aligned}$$

Since $h(n + 1) \leqslant T$ and

$$\tau(h) \leqslant \frac{Mh}{2} \qquad \text{where} \quad M = \max_{t_0 \leqslant \xi \leqslant t_0 + T} |y''(\xi)|,$$

we conclude with the bound

$$|y_n - u_n| \leqslant \frac{e^{LT} - 1}{L}\frac{M}{2}h, \qquad \forall n \geqslant 0,$$

which shows that the Euler method converges (but the constant grows exponentially with L and T).

Remark. If we further account for the possibility of rounding or approximation errors at each step, leading to a perturbed solution:

$$\bar{u}_0 = y_0 + \varepsilon_0, \qquad \bar{u}_{n+1} = \bar{u}_n + hf(t_n, \bar{u}_n) + \varepsilon_{n+1},$$

where $\varepsilon_0, \dots, \varepsilon_N$ are the errors, then zero-stability allows us to bound the deviation from the numerical solution u_n in exact arithmetic, provided $\delta_{n+1} = \varepsilon_{n+1}/h$ is small enough. In particular, we get a bound of the form

$$|\bar{u}_{n+1} - u_{n+1}| \leqslant |\varepsilon_0| e^{L(t_{n+1}-t_0)} + \frac{\varepsilon}{hL}\left(e^{L(t_{n+1}-t_0)} - 1\right),$$

where $\varepsilon = \max_{j \geqslant 1} |\varepsilon_j|$. In conjunction to the error formula for $|y_{n+1} - u_{n+1}|$, the triangle inequality yields

$$|\bar{u}_{n+1} - y_{n+1}| \leqslant |\varepsilon_0| e^{L(t_{n+1}-t_0)} + \left(\frac{Mh}{2} + \frac{\varepsilon}{h}\right) \frac{e^{L(t_{n+1}-t_0)} - 1}{L}.$$

This shows that for h too small, the error will actually start to increase due to the accumulation of small rounding errors at each step.

21.3 Absolute Stability

The notion of zero-stability introduced in the previous sections is a useful one in theory, since it ensures robustness with regard to perturbations and convergence of the scheme, however it comes with a caveat: the exponential dependence of the condition number on the length of the integration interval, T, which persists even as $h \to 0$. In practice, one will be using a fixed time-step $h > 0$, and may want to compute solutions over a long time interval. We investigate in this paragraph the behavior of numerical schemes in this regime, starting with the following.

Definition 21.3. *The following linear Cauchy problem is called the test problem:*

$$\begin{cases} y'(t) = \lambda y(t), & \lambda \in \mathbb{C}, \\ y(0) = 1, \end{cases} \tag{21.1}$$

with the exact solution $y(t) = e^{\lambda t}$.

Now, if the real part of λ is strictly negative, then $\lim_{t\to\infty} y(t) = 0$. This is not necessarily true for the numerical approximation!

Definition 21.4. *A numerical method is* ***absolutely stable*** *if* $|u_n| \to 0$ *as* $t_n \to \infty$ *when applied to problem* (21.1).

A method will be absolutely stable for certain values of h, λ and not for others.

Definition 21.5. *The region of absolute stability is the subset of the complex plane*

$$\mathcal{A} = \{z = h\lambda \in \mathbb{C} \mid \lim_{n\to\infty} |u_n| = 0\}.$$

Examples

21.3.1 Forward Euler scheme.

Applied to problem (21.1), the forward Euler scheme yields

$$\begin{cases} u_0 = 1, \\ u_{n=1} = u_n + hf(t_n, u_n) = u_n + h\lambda u_n = (1 + h\lambda)u_n. \end{cases}$$

By recurrence, the numerical solution is given by the formula

$$u_n = (1 + h\lambda)^n, \qquad \forall n \geqslant 0,$$

and thus $u_n \to 0$ if and only if $|1 + h\lambda| < 1$, i.e. $h\lambda$ lies within the open disk of center $(-1, 0)$ and radius 1.

Application. For a more general problem of the form $y' = f(t, y)$ such that $\lambda \leqslant f_y < 0$, the forward Euler scheme will be unstable, i.e. develop oscillations of exponentially large amplitude, unless we pick a timestep $h < 2/|\lambda|$.

21.3.2 Backwards Euler scheme.

Applied to problem (21.1), the backwards Euler scheme yields

$$\begin{cases} u_0 = 1, \\ u_{n=1} = u_n + hf(t_{n+1}, u_{n+1}) = u_n + h\lambda u_{n+1} \qquad \text{or } u_{n+1} = (1 - h\lambda)^{-1} u_n. \end{cases}$$

By recurrence, the numerical solution is given by the formula

$$u_n = (1 - h\lambda)^{-n}, \qquad \forall n \geqslant 0,$$

and thus $u_n \to 0$ if and only if $|1 - h\lambda| > 1$, i.e. $h\lambda$ does *not* lie within the closed disk of center $(1, 0)$ and radius 1.

Lecture 22: Absolute Stability. Multistep Methods. (Monday, November 9)

22.1 Absolute stability: some more examples.

Recall that we investigate the behavior of our numerical methods applied to the test Cauchy problem

$$\begin{cases} y'(t) = \lambda y(t), \\ \qquad y(0) = 1, \end{cases}$$

which has exact solution $y(t) = e^{\lambda t}$. In particular, the region of absolute stability is the set $\mathcal{A} = \{h\lambda \in \mathbb{C} \text{ such that } |u_n| \to 0\}$.

22.1.1 Trapezoidal or Crank-Nicholson scheme.

We find here the recurrence relation

$$u_0 = 1, \qquad u_{n+1} = u_n + \frac{h}{2}\left(\lambda u_n + \lambda u_{n+1}\right),$$

leading to $\left(1 - \frac{h\lambda}{2}\right) u_{n+1} = \left(1 + \frac{h\lambda}{2}\right) u_n$, and by immediate induction, for $h\lambda \neq 2$,

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

Now, we observe that for any complex number $z = x + iy \neq 2$,

$$\left|\frac{1+z}{1-z}\right| < 1 \quad \Leftrightarrow \quad |1 + z^2|^2 < |1 - z^2|^2 \quad \Leftrightarrow \quad x < 0,$$

so the region of absolute stability is the entire left half-plane $\mathrm{Re}(h\lambda) < 0$. Note that this matches exactly the set of parameters λ for which the exact solution also converges to zero.

22.1.2 Heun's method

For this last example, we have the recurrence

$$u_0 = 1, \qquad u_{n+1} = u_n + \frac{h}{2}\left(\lambda u_n + \lambda(u_n + h\lambda u_n)\right),$$

leading to $u_{n+1} = \left(1 + h\lambda + \frac{(h\lambda)^2}{2}\right) u_n$, and by immediate induction,

$$u_n = \left(1 + h\lambda + \frac{(h\lambda)^2}{2}\right)^n.$$

Hence the region of absolute stability is the set

$$\mathcal{A}_{Heun} \left\{ h\lambda \in \mathbb{C} \text{ such that } \left|\frac{1 + (1 + h\lambda)^2}{2}\right| < 1 \right\}.$$

The shape of this set is a somewhat elliptic set containing the disk centered at -1 with radius 1 and contained in the rectangle with $-2 \leqslant \mathrm{Re}(h\lambda) \leqslant 0$ and $-1.75 \leqslant \mathrm{Im}(h\lambda) \leqslant 1.75$.

22.1.3 A-stability

Definition 22.1. *A method is called A-stable if its region of absolute stability contains the entire left half-plane* $\mathbb{C}^- = \mathrm{Re}(z) < 0$*, i.e. the method is absolutely stable whenever* $\mathrm{Re}(\lambda) < 0$*.*
If a method is not absolutely stable, it is called conditionally stable.

22.1.4 Summary

	Order	Type	A-stable?	Stability region
Forward Euler	1	Explicit	No	$\lvert 1 + h\lambda\rvert < 1$
Backwards Euler	1	Implicit	Yes	$\lvert 1 - h\lambda\rvert < 1$
Crank-Nicholson	2	Implicit	Yes	$\mathrm{Re}(h\lambda) < 1$
Heun's method	2	Explicit	No	$\lvert 1 + h\lambda + \frac{(h\lambda)^2}{2}\rvert < 1$

Remark 22.2.
- *There are no A-stable (or unconditionnally stable) explicit schemes.*
- *Not all implicit methods are A-stable. There are also consistent yet unstable or conditionally stable implicit schemes.*

22.2 Multistep methods.

Previous methods are limited in the order of convergence because we only used values u_n, u_{n+1} and $f_n = f(t_n, u_n)$, $f_{n+1} = f(t_{n+1}, u_{n+1})$. In order to gain accuracy, we may use the idea behind interpolation: to gain accuracy, we can use more nodes. In particular, we can use some of the previous values generated by the scheme: u_n, but also u_{n-1}, u_{n-2}, etc.

Notation In this entire section, we will use the notation

$$f_n := f(t_n, u_n).$$

Definition 22.3. *A numerical scheme called* ***q-step*** *is a method where* u_{n+1} *depends only on the values* $u_n, \ldots, u_{n+1-q}$*.*

Examples.

- The midpoint method, based on the centered finite difference:

$$y'(t_n) \approx \frac{u_{n+1} - u_{n-1}}{2h} \qquad \rightarrow \qquad u_{n+1} = u_{n-1} + 2hf(t_n, u_n), \qquad \text{for all } n \geqslant 2.$$

  This is an explicit 2-step scheme, since u_{n+1} depends only on u_n and u_{n-1}.

- The Simpson method, based on the Simpson quadrature rule:

$$y(t_{n+1}) = y_{t_{n-1}} + \int_{t_{n-1}}^{t_{n+1}} y'(t)dt \approx y_{t_{n-1}} + \frac{2h}{6}\left[f(t_{n-1}, y_{n-1}) + 4f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right]$$

  leading to the scheme

$$u_{n+1} = u_{n-1} + \frac{h}{3}\left[f_{n-1} + 4f_n + f_{n+1}\right], \qquad \text{for all } n \geqslant 2.$$

  This is an implicit 2-step scheme, since u_{n+1} depends only on u_n and u_{n-1}.

Remark 22.4. *Any q-step method needs q initial values to take off:*

$$u_0, \dots, u_{q-1}.$$

Since the initial value problem provides only one starting value $u_0 = y_0$, one way to compute these starting values is to resort to an explicit one-step method of the same order. For high-order multi-step methods like the Adams method introduced below, this cannot be achieved with the schemes of order 1 or 2 we have seen so far, but higher-order Runge-Kutta schemes can be used for this purpose.

22.3 Explicit Adams-Bashforth schemes

Idea. To build a family of $q = p+1$-step schemes, we use the nodes $t_{n-p}, \dots, t_n$ to construct and interpolatory quadrature approximating the integral

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} y'(t)dt = \int_{t_n}^{t_{n+1}} f(t, y(t))dt.$$

To achieve this, we build a polynomial interpolating the values $f(t_{n-p}, y_{n-p}), \dots, f(t_n, y_n)$ at the nodes $t_{n-p}, \dots, t_n$:

$$P_p(t) = \sum_{j=0}^{p} f(t_{n-j}, y_{n-j})\ell_j(t),$$

where $\ell_j(t)$ is the elementary Lagrange polynomial:

$$\ell_j(t) = \prod_{k=0,\ k\neq j}^{p} \left(\frac{t - t_{n-k}}{t_{n-j} - t_{n-k}}\right).$$

This allows us to build an interpolatory quadrature rule:

$$\int_{t_n}^{t_{n+1}} f(t, y(t))dt \approx \int_{t_n}^{t_{n+1}} P_p(t)dt = \sum_{j=0}^{p} \alpha_j f(t_{n-j}, y_{n-j}), \qquad \text{where } \alpha_j = \int_{t_n}^{t_{n+1}} \ell_j(t)dt.$$

Let us compute the coefficients α_j more precisely. Introduce the change of variables $t = t_n + hs$, $dt = hds$, and recall $t_{n-j} = t_n - jh$, then

$$\ell_j(t) = \prod_{k\neq j} \frac{(t_n + hs) - (t_n - kh)}{(t_n - jh) - (t_n - kh)} = \prod_{k\neq j} \frac{k+s}{k-j},$$

such that

$$\alpha_j = \int_{t_n}^{t_{n+1}} \ell_j(t)dt = h \underbrace{\int_0^1 \prod_{k=0,\ k\neq j}^{p} \frac{k+s}{k-j} ds}_{=w_j}.$$

Note that the coefficients w_j only depend on j and p.
Replacing above the exact values y_{n-j} by their approximation u_{n-j}, we obtain:

Definition 22.5. *For any $p \geqslant 0$, we define the $(p+1)$-step Adams-Bashforth scheme:*

$$u_{n+1} = u_n + h \sum_{j=0}^{p} w_j \underbrace{f(t_{n-j}, u_{n-j})}_{:=f_{n-j}}.$$

This relation defines an explicit scheme.

Examples.

- Case $p = 0$: here, we interpolate at the single node t_n, with $P_0(t) = f(t_n, y_n)$. The result is the forward Euler scheme:
$$u_{n+1} = u_n + hf(t_n, u_n).$$

- Case $p = 1$: we build a linear interpolant at t_{n-1}, t_n: using the Newton formula,
$$P_1(t) = f(t_n, y_n) + \frac{f(t_{n-1}, y_{n-1}) - f(t_n, y_n)}{t_{n-1} - t_n}(t - t_n),$$
and in particular
$$P_1(t_{n+1}) = f(t_n, y_n) - (f(t_{n-1}, y_{n-1}) - f(t_n, y_n)) = 2f(t_n, y_n) - f(t_{n-1}, y_{n-1}).$$
Because the trapezoidal rule integrates exactly polynomials of order 1, we have
$$\int_{t_n}^{t_{n+1}} P_1(t)dt = \frac{h}{2}\left(P_1(t_n) + P_1(t_{n+1})\right) = \frac{h}{2}\left[3f(t_n, y_n) - f(t_{n-1}, y_{n-1})\right].$$
This leads to the two-step Adams-Bashforth scheme
$$u_{n+1} = u_n + \frac{h}{2}(3f_n - f_{n-1}).$$

- Formulae for the 3-step ($p = 2$) and 4-step ($p = 3$) Adams-Bashforth schemes can be found in the textbook.

Consistency analysis. Because these schemes are based on an interpolatory quadrature for $y'(t)$, we may use the Lagrange interpolation error formula:
$$y'(t) = P_p(t) + \frac{y^{(p+2)}(\xi(t))}{(p+1)!}\underbrace{(t - t_{n-p})\cdots(t - t_n)}_{=\omega_{p+1}(t)}dt.$$
Integrating this formula from t_n to t_{n+1} leads to the identity
$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} y'(t)dt = y_n + \int_{t_n}^{t_{n+1}} P_p(t)dt + \int_{t_n}^{t_{n+1}} \frac{y^{(p+2)}(\xi(t))}{(p+1)!}\omega_{p+1}(t)dt.$$
For the first term, we note that by the construction above,
$$y_n + \int_{t_n}^{t_{n+1}} P_p(t)dt = y_n + h\sum_{j=0}^{p} w_j f(t_{n-j}, y_{n-j}) \qquad = \widetilde{y}_{n+1},$$
where $\widetilde{y}_{n+1}$ is the result of one step of the Adams-Bashforth scheme with exact initial values $y_{n-p}, \ldots, y_n$. Now for the second term, let $t = t_n + hs$, we compute
$$\begin{aligned}\omega_{p+1}(t) &= (t - t_{n-p})\cdots(t - t_{n-1})(t - t_n)\\ &= [t_n + hs - (t_n - hp)]\cdots[t_n + hs - (t_n - h)[\,[t_n + hs - t_n]\\ &= h^{p=1}(s + p)\cdots(s + 1)s.\end{aligned}$$

In particular $\omega_{p+1}(t)$ is positive on the interval $[t_n, t_{n+1}]$, so we can use the mean value theorem:

$$\begin{aligned}\int_{t_n}^{t_{n+1}} \frac{y^{(p+2)}(\xi(t))}{(p+1)!}\omega_{p+1}(t)dt &= \frac{y^{(p+2)}(\xi_n)}{(p+1)!}\int_{t_n}^{t_{n+1}} \omega_{p+1}(t)dt \\ &= \frac{y^{(p+2)}(\xi_n)}{(p+1)!}h^{p+2}\int_0^1 (s+p)\cdots(s+1)sds.\end{aligned}$$

for some $\xi_n \in (t_n, t_{n+1})$. Define the constant

$$C_{p+1} := \frac{1}{(p+1)!}\int_0^1 s(s+1)\cdots(s+p)ds \geqslant 0,$$

then we have shown that the local truncation error satisfies

$$y_{n+1} - \widetilde{y}_{n+1} = C_{p+1}y^{(p+2)}(\xi_n)h^{p+2} \quad \text{or} \quad \tau_{n+1}(h) = \frac{y_{n+1}-\widetilde{y}_{n+1}}{h} = C_{p+1}y^{(p+2)}(\xi_n)h^{p+1}.$$

This shows that the $q = p+1$-step Adams-Bashforth schemes are consistent for any $p \geqslant 0$, and furthermore have order q.

Lecture 23: Multi-step methods: Adams-Moulton schemes and analysis. (Monday, November 16)

23.1 Implicit Adams-Moulton schemes.

Using a similar idea as for the Adams-Bashforth schemes in the last lecture, we can use the nodes $t_{n-p}, \ldots, t_n, t_{n+1}$ to construct an interpolatory quadrature rule approximating

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t))dt.$$

By writing an interpolant $Q_{p+1}(t) = \sum_{j=-1}^{p} f(t_{n-j}, y_{n-j})$, we find weights w_j such that

$$y_{n+1} \approx y_n + h \sum_{j=-1}^{p} \bar{w}_j f(t_{n-j}, y_{n-j}),$$

which are given by the formula

$$\bar{w}_j = \int_0^1 \prod_{k=-1,\ k\neq j}^{p} \left(\frac{k+s}{k-j}\right) ds.$$

Using the approximate values $u_n \approx y_n$ yields a numerical scheme:

Definition 23.1. *For any $p \geqslant -1$, we define the $(p+1)$-step (for $p \geqslant 0$) or 1-step (for $p = -1$) Adams-Moulton scheme:*

$$u_{n+1} = u_n + h \sum_{j=0}^{p} \bar{w}_j f_{n-j} \ + h\bar{w}_{-1} f_{n+1}.$$

This relation defines an implicit scheme.

In general, a q-step Adams-Moulton method corresponding to $p \geqslant 0$ has order $q+1$. An exception to this rule is made for $p = -1$, which corresponds to the backwards Euler scheme, an implicit 1-step scheme with order $p + 2 = 1$.

Example

- Case $p = 0$: we find the Crank-Nicholson scheme, which has order 2.
- Case $p = 1$:

$$u_{n+1} = u_n + \frac{h}{12}\left(5f_{n+1} + 8f_n - f_{n-1}\right).$$

23.2 Stability and Convergence Analysis

The general formula for a linear $p+1$-step method takes the form

$$u_{n+1} = \underbrace{\sum_{j=0}^{p} a_j u_{n-j}}_{\text{linear combination of past } u \text{ values}} + \underbrace{h\sum_{j=0}^{p} b_j f_{n-j}}_{\text{linear combination of past } f \text{ values}} + \underbrace{h b_{-1} f_{n+1}}_{\text{if implicit}}. \tag{23.1}$$

We define in general the local truncation error

$$\tau_{n+1}(h) = \frac{y_{n+1} - \widetilde{y}_{n+1}}{h}, \qquad \widetilde{y}_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j f(t_{n-j}, y_{n-j}).$$

Example: the Adams scheme share the coefficients $a_0 = 1$, $a_1 = \cdots = a_p = 0$.

- We have shown in the previous lecture that the explicit Adams-Bashforth scheme with $q = p + 1$ step, with coefficients

$$b_{-1} = 0, \qquad b_j = w_j = \int_0^1 \prod_{k=0,\ k\neq j}^{p} \left(\frac{k+s}{k-j}\right) ds, \quad j = 0, \ldots, p,$$

  are consistent with order q.

- We can show that the implicit Adams-Moulton scheme with $q = p + 1$ step, with coefficients

$$b_j = \bar{w}_j = \int_0^1 \prod_{k=-1,\ k\neq j}^{p} \left(\frac{k+s}{k-j}\right) ds, \quad j = -1, \ldots, p,$$

  are consistent with order $q + 1$.

23.2.1 What about stability?

Definition 23.2. *A linear q-step method like* (23.1) *is called zero-stable if there exists $K > 0$ such that, given two sequences $\{u_n\}_{n\geqslant 0}$, $\{z_n\}_{n\geqslant 0}$ generated by the scheme with different starting values $u_0, \ldots, u_{p-1}$ and $z_0, \ldots, z_{p-1}$, then*

$$|u_n - z_n| \leqslant K \max\left\{|u_0 + z_0| + \cdots + |u_{p-1} - z_{p-1}|\right\},$$

for all n such that $t_0 \leqslant t_n \leqslant t_0 + T$, with K independent of h as $h \to 0$.

Remark 23.3. *This is a simplified version of the zero-stability notion introduced for one-step methods; we could also include perturbations δ_n representing errors commited at each step of the scheme* (23.1) *for $n \geqslant p$.*

This is a difficult property to check *a priori*. However, we may use results from the theory of sequences generated by multi-step recurrence relations like (23.1). Let us introduce the **characteristic polynomials**:

$$\begin{cases} \rho(z) = z^{p+1} - \displaystyle\sum_{j=0}^{p} a_j z^{p-j}, \\ \sigma(z) = b_{-1} z^{p+1} + \displaystyle\sum_{j=0}^{p} b_j z^{p-j}. \end{cases} \tag{23.2}$$

We study the special case where $f(t, y) = 0$ in the Cauchy problem, which as we will see is the key to understanding (and justifies the name of) *zero-stability*. The values of the sequence $\{u_n\}$ are then generated from the recurrence relation

$$u_{n+1} = \sum_{j=0}^{p} a_j u_j = a_0 u_n + \cdots + a_p u_{n-p} \qquad \text{for } n \geqslant p,$$

given a set of initial values $u_0, \ldots, u_p$. In particular, we are interested in the behavior of this sequence as $n \to \infty$.

Lemma 23.4. *Consider the $q = p + 1$-th order homogeneous linear recurrence relation given by coefficients $a_0, \dots, a_p$:*

$$u_{n+1} = \sum_{j=0}^{p} a_j u_j,$$

where we assume $a_p \neq 0$, and its characteristic polynomial

$$\rho(z) = z^{p+1} - \sum_{j=0}^{p} a_j z^{p-j}.$$

Let $z_1, \dots, z_l$ be the distinct (complex) roots of ρ with multiplicity $m_1, \dots, m_l$ such that $m_1 + \cdots + m_l = q$. Then, for any set of initial values $u_0, \dots, u_p$ there exists polynomials $p_r(n)$ of degree at most $m_r - 1$ for $r = 1, \dots, l$ such that

$$u_n = \sum_{r=1}^{l} p_r(n) z_r^n, \qquad \text{for all } n \geqslant 0.$$

In the particular case where $m_r = 1$ (the root z_r is simple), then p_r is a constant.

Proof. We only sketch the proof. Assume that the polynomial $\rho(r)$ has q distinct, simple roots $z_1, \dots, z_q$ different from zero (since $a_p \neq 0$ and thus $\rho(r) \neq 0$).
Clearly, any sequence of the form $u_n^{(r)} = (z_r)^n$ satisfies

$$z_r^{n+1} - \sum_{j=0}^{p} a_j z_r^{n-j} = z_r^{n-p} \left(z_r^{p+1} - \sum_{j=0}^{p} a_j z_r^{p-j} \right) = z_r^{n-p} \rho(z_r) = 0.$$

Hence it satisfies the recurence relation. Furthermore, by linearity any linear combination of such sequences $\sum_{r=1}^{q} C_r z_r^n$ also satisfies the recurrence relation. To show that any sequence satisfying the recurrence relation and generated from initial values $u_0, \dots, u_p$ has this form, we may solve the linear system

$$\begin{cases} C_1 + \cdots + C_q = u_0, \\ C_1 z_1 + \cdots + C_q z_q = u_1, \\ C_1 z_1^2 + \cdots + C_q z_q^2 = u_2, \\ \qquad\qquad\qquad \vdots \\ C_1 z_1^p + \cdots + C_q z_q^p = u_p. \end{cases}$$

The coefficient matrix has the specific *Vandermonde* form, and its determinant is the Vandermonde determinant

$$\begin{vmatrix} 1 & \cdots & 1 \\ z_1 & \cdots & z_q \\ \vdots & & \vdots \\ z_1^{q-1} & \cdots & z_q^{q-1} \end{vmatrix} = \prod_{r<s} (z_s - z_r) \neq 0.$$

Hence the system has a unique solution $C_1, \dots, C_q$, showing that any sequence generated by initial values $u_0, \dots, u_p$ and the recurrence relation writes as

$$u_n^{(r)} = \sum_{r=1}^{q} C_r z_r^n.$$

If the polynomial has one (or several) roots with multiplicity $m_r > 1$, then the construction above does not work since there are only $l < q$ distinct sequences of the form $\{(z_r)^n\}_{n\geqslant 0}$ where z_r is a root of $\rho(z)$. To fix this, we include in the basis for each $r = 1, \dots, l$ the m_r sequences:

$$u_n^{(r,0)} = z_r^n, \qquad u_n^{(r,1)} = nz_r^n, \qquad \dots, \qquad u_n^{(r,m_r-1)} = n(n-1)\cdots(n-m_r+2)z_r^n.$$

In total, this is a set of $\sum_{r=1}^{l} m_r = q$ sequences. Some more complex computations show that each such sequence satisfies the recurrence relation, and that the linear system allowing to find the coefficients $C_{r,j}$ for $r = 1, \dots, l$ and $j = 0, \dots, m_r$ from the initial values $u_0, \dots, u_p$ is well-posed. Finally, we note that each linear combination

$$\sum_{j=0}^{m_r-1} C_{r,j} u_n^{(r,j)} = \left[C_{r,0} + C_{r,1}n + \cdots + C_{r,m_r-1}n(n-1)\cdots(n-m_r+2)\right] z_r^n$$

is of the form $p_r(n)z_r^n$ with $p_r(n)$ a polynomial in n of degree at most $m_r - 1$. □

Now that we understand the behavior of sequences generated by recurrence relations of the form above, it follows that the roots of the characteristic polynomial $\rho(z)$ are critical for the zero-stability property.

Theorem 23.5. ***(Root condition)***
A linear multi-step method applied to a Cauchy problem where f satisfies a Lipschitz condition is ***zero-stable*** *if and only if all roots of the first characteristic polynomial are inside the closed unit disk of $\mathbb{C}$, with any lying on the unit circle being simple, i.e.*

$$\textit{Zero-stability} \quad \Leftrightarrow \quad \textit{(Root condition): if } \rho(z) = 0, \textit{ then } \begin{cases} |z| \leqslant 1 \\ \textit{and} \\ \rho'(z) \neq 0 \quad \textit{if} \quad |z| = 1. \end{cases}$$

Proof. Again, we sketch the proof, and in particular the necessary component of the equivalence above. Consider the homogeneous case $y' = 0$ and $a_p \neq 0$. By Lemma 23.4, any numerical solution of the scheme takes the form

$$u_n = \sum_{r=1}^{l} p_r(n) z_r^n,$$

with $p_r(n)$ a polynomial of degree $m_r - 1$ at most. Then $|z_r| > 1$, then for some choice of the starting values, $p_r(n) \neq 0$ and the sequence will grow to infinity like $|z_r|^n$. If $|z_r| = 1$ and $m_r > 1$ (the root is not simple), then the sequence will grow to infinity like n^{m_r-1}. Thus, as $h \to 0$ such solutions will grow to infinity as $t_n = t_0 + nh$ is fixed and the scheme is not zero-stable.
It is easy to extend this analysis to the case where $a_p = 0$ and $z = 0$ is a (possibly multiple) root of the characteristic polynomial. The other direction (sufficient) is quite technical, and we skip the rest of the proof. □

Examples

- The Euler methods are example of 1-step linear multistep methods: $u_{n+1} = u_n + hf_n$ or $u_{n+1} = u_n + hf_{n+1}$. In this case, $p = 0$ and the characteristic polynomial is $\rho(z) = z - 1$. It has a simple root $z = 1$, and hence satisfies the root condition: the Euler methods are zero-stable.

- The Adams methods are $p+1$-step linear multistep methods of the form

$$u_{n+1} = u_n + h \sum_{j=0/-1}^{p} w_j f_{n-j},$$

which have the characteristic polynomials

$$\rho(z) = z^{p+1} - z^p = z^p(z-1),$$

which have a simple root at $z = 1$ and a root at $z = 0$ with multiplicity $m = p$. The methods hence satisfy the root condition, and they are zero-stable when applied to a Cauchy problem with f satisfying a Lipschitz condition.

- The midpoint method: $u_{n+1} = u_{n-1} + 2hf_n$ and the Simpson method: $u_{n+1} = u_{n-1} + \frac{h}{3}(f_{n-1} + 4f_n + f_{n+1})$ are both 2-step linear multistep methods with the characteristic polynomial

$$\rho(z) = z^2 - 1 = (z-1)(z+1).$$

Both roots $z = -1$ and $z = 1$ are simple, hence the methods satisfy the root condition and they are zero-stable.

- Finally, consider the three-step method

$$u_{n+1} = u_{n-2} + u_{n-1} - u_n + 2h(f_{n-1} + f_n),$$

which has the characteristic polynomial $\rho(z) = z^3 + z^2 - z - 1 = (z+1)^2(z-1)$. There is a simple root at $z = 1$ and a double root at $z = -1$, so the method is not zero-stable (and hence practically useless).

Lecture 24: Multistep methods (the conclusion). Runge-Kutta methods. (Wednesday, November 18)

24.1 Consistency for multistep methods.

Recall

Definition 24.1. *We define the local truncation error*

$$\tau_{n+1}(h) = \frac{y_{n+1} - \left[\sum_{j=0}^{p} a_j y_j + h \sum_{j=-1}^{p} b_j f(t_{n-j}, y_{n-j}\right]}{h} \qquad where y_k = y(t_k),$$

and the global truncation error

$$\tau(h) = \max_{0 \leqslant t_{n+1} - t_0 \leqslant T} \tau_{n+1}(h).$$

Theorem 24.2. *The $p+1$-step linear multistep method (23.1) is **consistent**, i.e. $\lim_{h\to 0} \tau(h) = 0$ if the exact solution $y(t)$ is twice continuously differentiable on $[t_0, t_0+T]$ and the coefficients satisfy the algebraic conditions*

$$\sum_{j=0}^{p} a_j = 1, \qquad b_{-1} + \sum_{j=0}^{p} (b_j - j a_j) = 1,$$

or in terms of the characteristic polynomials $\rho(z) = z^{p+1} - \sum_{j=0}^{p} a_j z^{p-j}$ and $\sigma(z) = \sum_{j=-1}^{p} b_j z^{p-j}$,

$$\rho(1) = 0, \qquad \sigma(1) = \rho'(1).$$

Remark 24.3. *Note that 1 must be a root of the first characteristic polynomial to achieve consistency. To satisfy the root condition (and the method to be zero-stable), this root must be simple, meaning that*

$$\sigma(1) = \rho'(1) \neq 0.$$

Proof. Let us expand $y(t)$ and $y'(t)$ in Taylor expansions of order 1 and 0 respectively around t_n:

$$y_{n-j} = y(t_n - jh) = y_n - jhy'_n + O(h^2),$$
$$y'_{n-j} = y'_n + O(h) + f(t_n, y_n) + O(h).$$

Hence, since $f(t_{n-j}, y_{n-j}) = y'(t_{n-j}) = y'_{n-j}$,

$$\widetilde{y}_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j y'_{n-j} = \sum_{j=0}^{p} a_j (y_n - jhy'_n) + h \sum_{j=-1}^{p} b_j y'_n + O(h^2),$$
$$y_{n+1} = y_n + hy'_n + O(h^2),$$

such that, grouping terms with like powers of h,

$$\tau_{n+1}(h) = \frac{y_{n+1} - \widetilde{y}_{n+1}}{h} = \frac{1}{h}\left(1 - \sum_{j=0}^{p} a_j\right) y_n + \left(1 - \left[\sum_{j=-1}^{p} b_j - \sum_{j=0}^{p} j a_j\right]\right) y'_n + O(h).$$

To obtain $\tau_{n+1}(h) \to 0$, we thus need $\sum_{j=0}^{p} a_j = 1$ and $\sum_{j=-1}^{p} b_j - \sum_{j=0}^{p} j a_j = 1$, which are the two algebraic conditions of the theorem. □

Remark 24.4. *By taking Taylor expansions of higher order, one finds further algebraic conditions which are necessary to increase the order of the method.*

24.2 Dahlquist's Theorems

To conclude this section on the analysis of multistep methods, the following two theorems give necessary and sufficient conditions about convergence and order of accuracy of the methods.

Theorem 24.5 (Dahlquist Equivalence Theorem)**.** *For a consistent method of the form* (23.1)*, zero-stability is equivalent to convergence.*
Furthermore, if the solution has $p+1$ continuous derivatives, the truncation error satisfies $\tau(h) = O(h^p)$ and the initial conditions satisfy $|u_j - y_j| = O(h^p)$ for $j = 0, \dots, p-1$ then the global error satisfies $e_n = |u_n - y_n| = O(h^p)$, i.e. the method is convergent with order p.

Theorem 24.6 (Dahlquist Barrier Theorem)**.** *The order of accuracy of a zero-stable q-step method cannot exceed $q+1$ if q is odd, and $q+2$ if q is even.*

Examples.

- The Crank-Nicholson method achieves the highest order of accuracy (2) for a 1-step method.
- The Simpson method achives the highest order of accuracy (4) for a 2-step method.

Remark 24.7. *To be useful in practice, each method need also fulfill absolute stability for small enough time-steps. We do not study this topic in detail, but refer to the textbook for more details.*

24.3 Higher-order single-step methods.

24.3.1 The Taylor methods.

The first idea to make a one-step method more accurate is to exploit the Taylor expansion around t_n:

$$\begin{aligned} y(t_{n+1}) &= y(t_n + h) \\ &= y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \cdots + \frac{h^p}{p!}y^{(p)}(t_n) + \dots \end{aligned}$$

We know how to approximate y_n with u_n, y'_n with $f(t_n, u_n)$ but what about the higher order derivatives? Using the multi-variate chain rule, we have the sequence

$$\begin{aligned} &y(t) \\ &y'(t) = f(t, y(t)) \\ &y''(t) = \frac{d}{dt}f(t, y(t)) = \frac{\partial f}{\partial t}(t, y(t)) + y'(t)\frac{\partial f}{\partial y}(t, y(t)) \qquad = \overbrace{(f_t + ff_y)}^{:=D^{(1)}f}(t, y(t)) \\ &y'''(t) = \frac{d}{dt}\underbrace{\left[(f_t + ff_y)(t, y(t))\right] = \left[f_{tt} + f_t f_y f f_{ty} + f(f_{ty} + f_y^2 + ff_{yy})\right]}_{:=D^{(2)}f}(t, y(t)) \qquad \vdots \end{aligned}$$

We can thus define expressions $D^{(1)}f$, $D^{(2)}f$, $D^{(3)}f$, …, $D^{(p)}f$ of any order, in terms of f and its partial derivatives only, such that $y^{(p+1)}(t) = D^{(p)}f(t, y(t))$. This allows to build a method:

$$u_{n+1} = u_n + hf(t_n, u_n) + \frac{h^2}{2}D^{(1)}f(t_n, u_n) + \cdots + \frac{h^p}{p!}D^{(p-1)}f(t_n, u_n)$$

which has by construction the local truncation error

$$y_{n+1} = \widetilde{y}_{n+1} + \frac{h^{p+1}}{(p+1)!} y^{p+1}(\xi_n), \qquad t_n < \xi_n < t_{n+1},$$

and thus satisfies $\tau_{n+1}(h) = O(h^p)$.

Issues: While this construction allows to build methods that have any order, they suffer from some disadvantages:

- Requires to compute high-order derivatives of f;
- Formulae are very complex;
- Hard to generalize to systems of equations.

24.3.2 Runge-Kutta methods.

Goal: we seek to achieve high-order approximation without the use of derivatives of f.

Main idea: build a quadrature using s nodes $t_n + c_1 h, \ldots, t_n + c_s h$ with $0 \leqslant c_1 \leqslant c_2 \leqslant \ldots \leqslant c_s \leqslant 1$ to approximate

$$\int_{t_n}^{t_{n+1}} y'(t)dt \approx h b_1 y'(t_n + c_1 h) + \cdots + h b_s y'(t_n + c_s h),$$

with quadrature weights $a_1, \ldots, a_s$. Since the values $y'(t_n + c_k h)$ are not known or approximated by the values u_j, we will construct appropriate approximations

$$K_j \approx y'(t_n + c_j h) = f(t_n + c_j h, y(t_n + c_j h)) \qquad \text{for } j = 1, \ldots, s.$$

Remark 24.8. *In most cases, we have $c_1 = 0$ such that $K_1 = f(t_n, u_n)$.*

General Runge-Kutta scheme. From the construction above, we expect to write the scheme as

$$u_{n+1} = u_n + hF(t_n, u_n, h; f),$$

where F is an increment function built as

$$\begin{cases} F(t_n, u_n, h; f) = \displaystyle\sum_{j=1}^{s} b_j K_j, \\ K_i = f\left(t_n + c_i h, u_n + h \displaystyle\sum_{j=1}^{s} a_{ij} K_j\right) \qquad \text{for } i = 1, \ldots, s. \end{cases}$$

s is called the **number of stages** of the Runge-Kutta method.

Butcher tableau / array. The Runge-Kutta method above is fully specified by the knowledge of the coefficients $c_1, \dots, c_s$ (the nodes of the quadrature), $b_1, \dots, b_s$ (the weights of the quadrature), and a_{ij} for $i, j = 1, \dots, s$, the coefficients of the linear combinations such that $u_n + h \sum_{j=1}^{s} a_{ij} K_j \approx y(t_n + c_i h)$. We organize these $s(s+2)$ coefficients in the shape of the following *Butcher tableau*:

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} \qquad \begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

Note that by consistency, we expect the following conditions on the rows of the tableau to hold:

- $c_i = \sum_{j=1}^{s} a_{ij}$ for all $i = 1, \dots, s$,
- $\sum_{j=1}^{s} b_j = 1$

Such methods are in general implicit, in the sense that the coefficients K_i have to be computed by solving (coupled) systems of nonlinear equations. However, if $a_{ij} = 0$ for any $j \geqslant i$, then each K_i may be computed in terms of $K_1, \dots, K_{i-1}$ only, and the method is explicit. This corresponds to a *strictly lower triangular* Butcher tableau.

Lecture 25: Runge-Kutta methods: conclusion. (Monday, November 23)

Recall the general form of a Runge-Kutta scheme associated to a Butcher tableau

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} \qquad \begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

- First, we compute the values K_i satisfying,

$$K_i = f\left(t_n + c_i h, u_n + \sum_{j=1}^{s} a_{ij} K_j\right), \qquad j = 1, \ldots, s,$$

- Next, we compute the next iterate,

$$u_{n+1} = u_n + h \sum_{j=1}^{s} b_j K_j.$$

25.1 Second-order, two-stage explicit Runge-Kutta schemes

We base these methods on a quadrature rule

$$\int_{t_n}^{t_{n+1}} y'(t)dt \approx h\left[b_1 y'(t_n) + b_2 y'(t_n + c_2 h)\right].$$

(Note that we fixed the first node, $c_1 = 0$.)
The method of undetermined coefficients can be applied to find appropriate values of b_1, b_2, c_2 such that the quadrature above has degree of exactness 1. This means

$$\int_{t_n}^{t_{n+1}} 1dt = h = h\left[b_1 \cdot 1 + b_2 \cdot 1\right] = h(b_1 + b_2), \qquad \text{or } b_1 + b_2 = 1;$$

and

$$\int_{t_n}^{t_{n+1}} tdt = \frac{t_{n+1}^2 - t_n^2}{2} = ht_n + \frac{h^2}{2} = h\left(b_1 t_n + b_2(t_n + c_2 h)\right) = h(b_1 + b_2)t_n + b_2 c_2 h^2,$$

so $b_1 = 1 - b_2$ and $c_2 = 1/(2b_2)$. Let us define $\beta = b_2 = \frac{1}{2c_2}$, such that $\beta \geqslant \frac{1}{2}$. We have found a family of quadrature rules:

$$\int_{t_n}^{t_{n+1}} y'(t)dt = h\left[(1-\beta)y'(t_n) + \beta y'\left(t_n + \frac{h}{2\beta}\right)\right].$$

Next, we have

$$\begin{cases} y'(t_n) = f(t_n, y_n), \\ y'\left(t_n + \dfrac{h}{2\beta}\right) = f\left(t_n + \dfrac{h}{2\beta}, y\left(t_n + \dfrac{h}{2\beta}\right)\right). \end{cases}$$

such that these values may be approximated, using Euler's method for the second one:

$$\begin{cases} y'(t_n) \approx f(t_n, u_n), \\ y'\left(t_n + \dfrac{h}{2\beta}\right) \approx f\left(t_n + \dfrac{h}{2\beta}, u_n + \dfrac{h}{2\beta} f(t_n, u_n)\right). \end{cases}$$

This construction amounts to the following Runge-Kutta scheme:

$$\begin{cases} K_1 = f(t_n, u_n), \\ K_2 = f\left(t_n + \dfrac{h}{2\beta}, u_n + \dfrac{h}{2\beta} K_1\right), \end{cases}$$

and then

$$u_{n+1} = u_n + h\left[(1-\beta)K_1 + \beta K_2\right].$$

This defines a family of two-stage, order 2 Runge-Kutta methods for any choice of $\beta \geqslant 1/2$. The Butcher tableau for these methods writes:

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2\beta} & \frac{1}{2\beta} & \\ \hline & 1-\beta & \beta \end{array}$$

(Note that an empty cell in the tableau indicates a zero value.)
Some popular choices of β are:

Examples.

- Case $\beta = 1$, or $c_2 = 1/2$: here

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}$$

  This creates the scheme

$$\begin{cases} K_1 = f(t_n, u_n), \\ K_2 = f\left(t_n + \dfrac{h}{2}, u_n + \dfrac{h}{2} K_1\right), u_{n+1} = u_n + hK_2, \end{cases}$$

  or in a single line,

$$u_{n+1} = u_n + hf\left(t_n + \frac{h}{2}, u_n + \frac{h}{2} f(t_n, u_n)\right).$$

  This is another 'midpoint method'.

- Case $\beta = 3/4$ or $c_2 = 2/3$:

$$\begin{array}{c|cc} 0 & & \\ \frac{2}{3} & \frac{2}{3} & \\ \hline & \frac{1}{4} & \frac{3}{4} \end{array}$$

such that

$$\begin{cases} K_1 = f(t_n, u_n), \\ K_2 = f\left(t_n + \dfrac{2h}{3}, u_n + \dfrac{2h}{3}K_1\right), u_{n+1} = u_n + \dfrac{h}{4}(K_1 + 3K_2), \end{cases}$$

or in a single line,

$$u_{n+1} = u_n + \frac{h}{4}\left(f(t_n, u_n) + 3f\left(t_n + \frac{2h}{3}, u_n + \frac{2h}{3}f(t_n, u_n)\right)\right).$$

- Case $\beta = 1/2$ or $c_2 = 1$:

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

such that

$$\begin{cases} K_1 = f(t_n, u_n), \\ K_2 = f\left(t_n + h, u_n + hK_1\right), u_{n+1} = u_n + \dfrac{h}{2}(K_1 + K_2), \end{cases}$$

or in a single line,

$$u_{n+1} = u_n + \frac{h}{2}\left(f(t_n, u_n) + f\left(t_n + h, u_n + hf(t_n, u_n)\right)\right).$$

This is just Heun's method!

25.2 Analysis of Runge-Kutta methods

We can use the one-step formalism here to find most answers.

Zero-stability: By Theorem 20.7, this is equivalent to the increment function $F(t_n, u_n, h; f)$ satisfying a Lipschitz condition. Because any finite combination of Lipschitz functions (sum, product, composition) also satisfies a Lipschitz condition, this is usually OK, hence Runge-Kutta methods will be zero-stable.

Consistency: This is usually studied case by case, using multivariate Taylor series. As an example, let us investigate the midpoint method

$$\begin{cases} K_1 = f(t_n, u_n), \\ K_2 = f\left(t_n + \dfrac{h}{2}, u_n + \dfrac{h}{2}K_1\right), u_{n+1} = u_n + hK_2, \end{cases}$$

Let us expand to the 3rd order $y(t)$ around t_n, using the formulae derived above in the Taylor methods section:

$$\begin{aligned} y_{n+1} = y(t_n + h) &= y_n + hy'_n + \frac{h^2}{2}y''_n + \frac{h^3}{6}y'''_n + O(h^4) \\ &= y_n + hf + \frac{h^2}{2}(f_t + ff_y) + \frac{h^3}{6}(f_{tt} + f_t f_y + 2ff_{ty} + ff_y^2 + f^2 f_{yy}) + O(h^4), \end{aligned}$$

where all functions are implicitely evaluated at the point (t_n, y_n). On the other hand, plugging into the scheme the exact value y_n, we find

$$\begin{aligned}
K_1 &= f, \\
K_2 &= f + \frac{h}{2} f_t + \frac{hK_1}{2} f_y + \frac{1}{2}\left(\frac{h}{2}\right)^2 f_{tt} + \frac{h}{2}\frac{hK_1}{2} f_{ty} + \frac{1}{2}\left(\frac{hK_1}{2}\right)^2 f_{yy} + O(h^3) \\
&= f + \frac{h}{2}(f_t + f f_y) + \frac{h^2}{8}\left(f_{tt} + 2 f f_t y + f^2 f_{yy}\right) + O(h^3).
\end{aligned}$$

Hence,

$$\widetilde{y}_{n+1} = y_n + hf + \frac{h^2}{2}(f_t + f f_y) + \frac{h^3}{8}\left(f_{tt} + 2 f f_t y + f^2 f_{yy}\right) + O(h^4).$$

Taking the difference between this expansion and that of y_{n+1}, we find

$$\begin{aligned}
y_{n+1} - \widetilde{y}_{n+1} =& y_n + hf + \frac{h^2}{2}(f_t + f f_y) + \frac{h^3}{6}\left(f_{tt} + f_t f_y + 2 f f_{ty} + f f_y^2 + f^2 f_{yy}\right) \\
&- y_n - hf - \frac{h^2}{2}(f_t + f f_y) - \frac{h^3}{8}\left(f_{tt} + 2 f f_t y + f^2 f_{yy}\right) + O(h^4) \\
=& \frac{h^3}{24}\left(f_{tt} + 2 f f_t y + f^2 f_{yy}\right) + \frac{h^3}{6}\left(f_t f_y + f f_y^2\right) + O(h^4).
\end{aligned}$$

This yields the leading term in the local truncation error:

$$\tau_{n+1}(h) = \frac{y_{n+1} - \widetilde{y}_{n+1}}{h} = \frac{h^2}{24}\left(f_{tt} + 2 f f_t y + f^2 f_{yy}\right) + \frac{h^2}{6}\left(f_t f_y + f f_y^2\right) + O(h^3).$$

In particular, the method is consistent of order 2.

Convergence: Runge-Kutta methods are convergent as a by product of their zero-stability and consistency, per the Lax-Richtmeyer Theorem 21.2.

25.2.1 Classical Fourth-Order Runge-Kutta Method

One of the most useful Runge-Kutta methods is given by the Butcher tableau:

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

This corresponds to the 4-stage explicit Runge-Kutta scheme:

$$\begin{cases}
K_1 = f(t_n, u_n), \\
K_2 = f\left(t_n + \dfrac{h}{2}, u_n + \dfrac{h}{2} K_1\right), \\
K_3 = f\left(t_n + \dfrac{h}{2}, u_n + \dfrac{h}{2} K_2\right), \\
K_4 = f\left(t_n + h, u_n + hK_3\right), \\
u_{n+1} = u_n + \dfrac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4).
\end{cases}$$