I. Interpolation & Numerical Calculus

<u>Goals:</u> - How to "read between the lines" of a numerical table

- (Piecewise) polynomial interpolation
- Approximation of a function by poly. interp. (measure of errors)
- Compute derivatives / integrals approximately

<u>Task:</u> Given a table of some quantity q

i	0	1	2	...	n
x_i	0.00	0.51	1.05	...	x_n
q_i	0.00	0.22	0.25	...	q_n

compute approximations of $q(x)$, $q'(x)$, $\int_a^b q(x)\,dx$?

i.e. easy to evaluate, derive, integrate

$\rightsquigarrow$ Find a simple (& reasonable) function $q(x)$ that matches the data

$$q(x_i) = q_i \,, \quad i = 0, 1, \dots, n$$

I.1 Polynomial interpolation

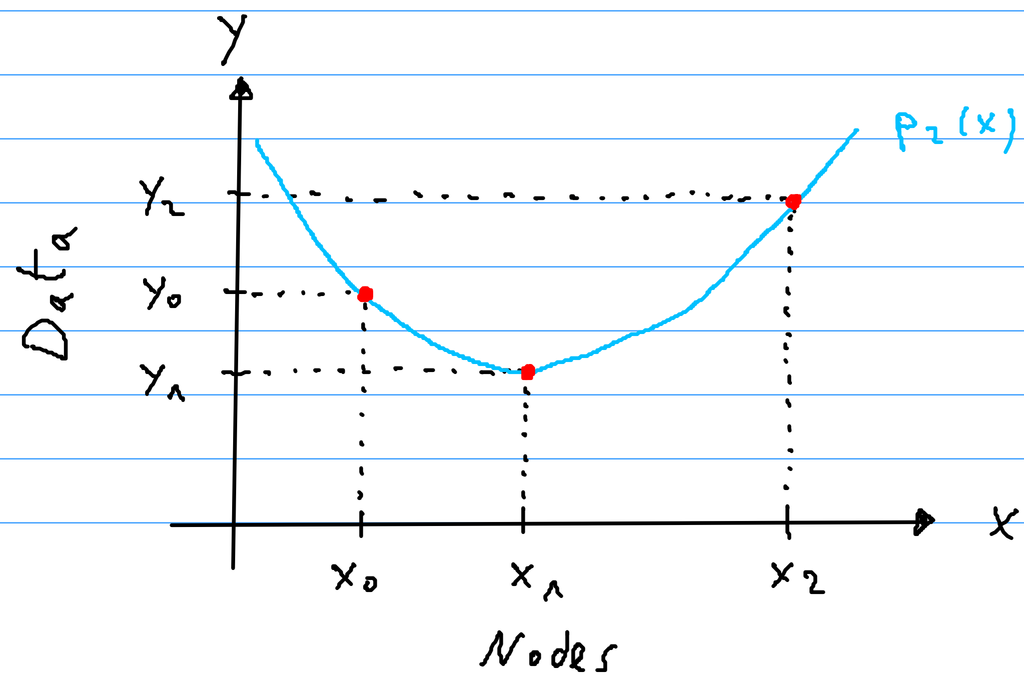
Given a set of $n+1$ distinct nodes,
$x_0 < x_1 < \ldots < x_n$, and corresponding data points
$y_0, y_1, \ldots, y_n$, find the n-th degree polynomial

$$p_n(x) = c_0 + c_1 \cdot x + c_2 \cdot x^2 + \ldots + c_n \cdot x^n$$

that satisfies the $n+1$ interpolation conditions (ICs)

$$p_n(x_j) = y_j \quad \text{for} \quad j = 0, 1, \ldots n.$$

The $n+1$ coefficients $c_0, c_1, \ldots, c_n$ of the so-called interpolating polynomial (IP) $p_n(x)$ result from the $n+1$ ICs ($\rightarrow$ linear system of equations (LSEs)).



Ex.: (1) find IP through

$$(x_0, y_0) = (1, 2)$$
$$(x_1, y_1) = (3, 5)$$
$$(x_2, y_2) = (4, 4)$$

So we have to find the coefficients c_0, c_1, c_2 of the IP $p_2(x) = c_0 + c_1 \cdot x + c_2 \cdot x^2$ fulfilling the ICs:

$$p_2(x_0) = p_2(1) = c_0 + c_1 + c_2 = 2$$
$$p_2(x_1) = p_2(3) = c_0 + 3c_1 + 9c_2 = 5$$
$$p_2(x_2) = p_2(4) = c_0 + 4c_1 + 16c_2 = 4$$

Or as

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \\ 4 \end{pmatrix}$$

Solving this LSE gives

$$c_0 = -2, \quad c_1 = \frac{29}{6}, \quad c_2 = -\frac{5}{6}$$

MATLAB: - p = polyfit(x, y, n)

nodes data degree

vector containing the coefficients

- convenient evaluation by polyval

Instead of solving a LSEs, the IP can also be found directly by the Lagrange Interpolation formula (LI)

$$p_n(x) = \sum_{j=0}^{n} y_j \cdot L_j^n(x)$$

where

$$L_j^n(x) = \prod_{\substack{i=0 \\ i \neq j}}^{n} \frac{x - x_i}{x_j - x_i} \quad \text{for } j = 0, 1, \dots, n$$

are the so-called Lagrange polynomials (LPs).

The LPs have the following properties:

(LP1) $L_j^n(x)$ is a polynom of degree n

(LP2) $L_j^n(x_k) = \delta_{jk} = \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases}$

(LP2) is the reason why the LI fulfills the ICs:

$$p_n(x_i) = \sum_{j=0}^{n} y_j \cdot L_j^n(x_i)$$

$$= 0 + \dots + 0 + y_i \cdot \underbrace{L_i^n(x_i)}_{1} + 0 + \dots + 0$$

$$= y_i \checkmark.$$

Ex.: (2) find IP through $(x_0, y_0) = (1, 2)$
$(x_1, y_1) = (3, 5)$
$(x_2, y_2) = (4, 4)$

same as Ex.(1)

with LI.

Compute the LPs:

$$L_0^2(x) = \frac{x - x_1}{x_0 - x_1} \cdot \frac{x - x_2}{x_0 - x_1} = \frac{x-3}{1-3} \cdot \frac{x-4}{1-4}$$

$$= \frac{1}{6}(x-3)(x-4)$$

$$L_1^2(x) = \frac{x - x_0}{x_1 - x_0} \cdot \frac{x - x_2}{x_1 - x_2} = \frac{x-1}{3-1} \cdot \frac{x-4}{3-4}$$

$$= -\frac{1}{2}(x-1)(x-4)$$

$$L_2^2(x) = \frac{x - x_0}{x_2 - x_0} \cdot \frac{x - x_1}{x_2 - x_1} = \frac{x-1}{4-1} \cdot \frac{x-3}{4-3}$$

$$= \frac{1}{3}(x-1)(x-3)$$

Now inserting into the LI

$$p_2(x) = 2 \cdot L_0^2(x) + 5 \cdot L_1^2(x) + 4 \cdot L_2^2(x)$$

$$= \ldots = 2 + \frac{29}{6} x - \frac{5}{6} x^2$$

(like Ex.(1), indeed!)

I.2 Interpolation error

e.g. from measurements

So far we have considered arbitrary data.
Now we assume that the data is generated by some function f and ask how well the IP approximates this function.

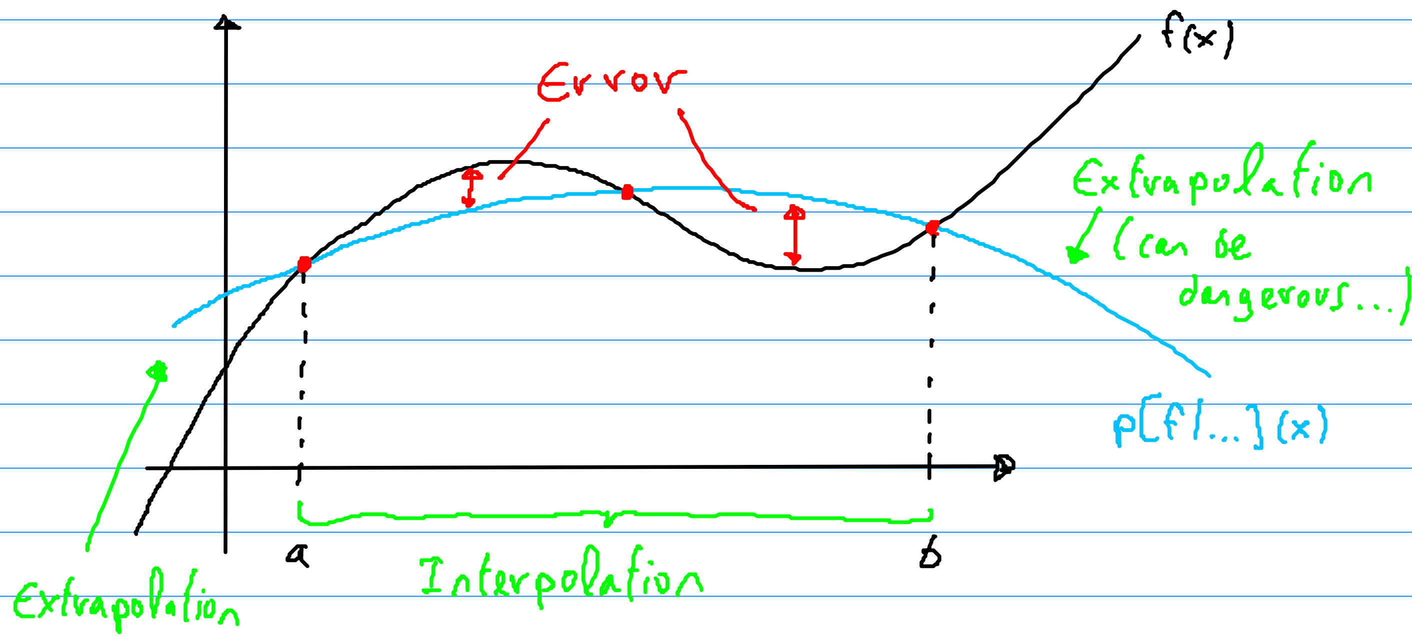
Let $f: I = [a,b] \to \mathbb{R}$ and we denote by

$$p[f|x_0,\dots,x_n](x) \in \mathbb{P}^n$$

Vector space of polynomials up to and including n

the IP fulfilling the ICs

$$p[f|x_0,\dots,x_n](x_j) = f(x_j) \quad \text{for} \quad j=0,1,\dots,n.$$



For f $(n+1)$-times continuously differentiable, one can show that for every $x \in I = [a,b]$ there is a $\xi(x) \in I$ such that

depends on x!

$$e(x) = f(x) - p[f|x_0,\dots,x_n](x) = \underbrace{\frac{\overset{(n+1)\text{-th derivative}}{f^{(n+1)}(\xi)}}{(n+1)!}}_{\text{depends on } f} \cdot \underbrace{\prod_{j=0}^{n}(x-x_j)}_{\text{nodes}}$$

Here $e(x)$ is a function over the whole interpolation interval I. Often, one is just interested in the biggest/maximum error over I:

$$\|e\|_\infty = \max_{x\in I} |e(x)| \qquad \text{(Maximum norm)}$$

$$= \max_{x\in I}\left|\frac{f^{(n+1)}(\xi(x))}{(n+1)!}\cdot\prod_{j=0}^{n}(x-x_j)\right|$$

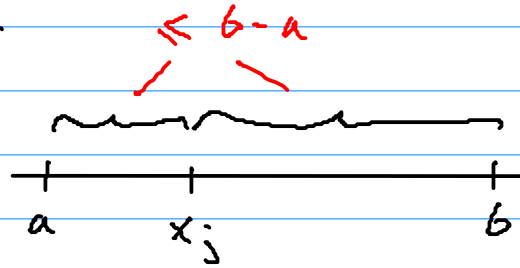
$$\leq \max_{x\in I}\left|\frac{f^{(n+1)}(\xi(x))}{(n+1)!}\right| \cdot \max_{x\in I}\left|\prod_{j=0}^{n}(x-x_j)\right|$$

$$= \frac{\|f^{(n+1)}\|_\infty}{(n+1)!}\cdot\Big\|\prod_{j=0}^{n}\underbrace{(x-x_j)}_{\leq\, b-a}\Big\|_\infty$$

Estimates

$$\leq \frac{\|f^{(n+1)}\|_\infty}{(n+1)!}(b-a)^{n+1}$$

The last estimate can best be understood graphically



The expression

"for f $(n+1)$-times continuously differentiable"

will pop up quite a few times in the course.

To shorten, one says:
- $f \in C^{n+1}[I]$
- f smooth enough
- f sufficiently many times cont. diff.

<u>Ex.:</u> (3) Runge's example (1901)

$\rightsquigarrow$ Slides

We note: (i) global interpolation with large n, i.e. many nodes and data points, is in general not recommendable

(ii) local, i.e. <u>piecewise</u>, works well (for f smooth enough)

Estimates of the form

$$\|e\|_\infty \leq \frac{h^{n+1}}{(n+1)!} \|f^{(n+1)}\|_\infty \qquad h = b-a$$

are very common and one introduces a special notation known as the Big-O or Big-oh notation.

One writes

$$\|e\| = \mathcal{O}(h^r) \qquad \text{some norm}$$

if there are positive constants C and r, independent of h, such that

$$\|e\| \leq C \cdot h^r$$

for h small enough. In the present context, r is called the order of accuracy.

Ex.: (4) $\|e\|_\infty \leq \frac{h^{n+1}}{(n+1)!} \|f^{(n+1)}\|_\infty = \mathcal{O}(h^{n+1})$

constants independent of h!

$\rightsquigarrow$ Slides

18.09.24

I.3 Numerical differentiation

We all know how to differentiate a function analytically ...

However, sometimes there are reasons to do this numerically:

- very complicated function (error prone)
  ... e.g. quasi-Newton methods ⇝ Chap. 2
- function not known analytically
  ... e.g. numerical solution of differential equations ⇝ Chap. 3 & 4

Idea: Find IP $p[f|x_0,\dots,x_n]$ approx. the function $f(x)$ and compute

$$f(x) \approx p[f|x_0,\dots,x_n](x)$$

$$f'(x) \approx p'[f|x_0,\dots,x_n](x)$$

$$f''(x) \approx p''[f|x_0,\dots,x_n](x)$$

$$\vdots$$

So suppose we want to approx. the derivatives of a (sufficiently) smooth function

$$f : I = [a,b] \to \mathbb{R}$$

Let $p[f|x_1,\dots,x_n]$ be the IP, then

k-th derivative

$$\frac{d^k f}{dx^k}(x) \underset{\text{approx.}}{\approx} \frac{d^k}{dx^k} p[f|x_0,\dots,x_n](x) = \frac{d^k}{dx^k} \sum_{j=0}^{n} L_j^n(x) \cdot f(x_j)$$

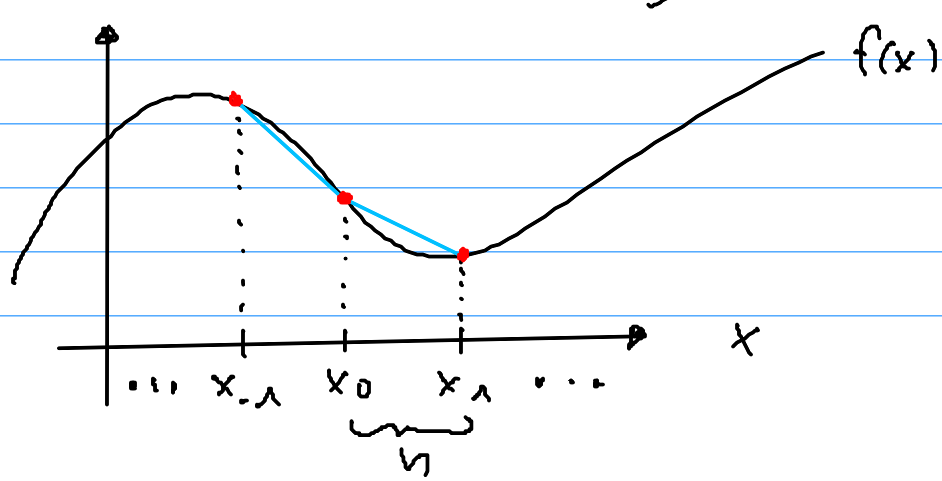
$$= \sum_{j=0}^{n} \frac{d^k L_j^n}{dx^k}(x) \cdot f(x_j)$$

This general procedure leads to so-called finite difference (FD) approximations of derivatives.

Usually one uses equidistantly spaced nodes

$$x_j = x_0 + j \cdot h \,, \quad j \in \mathbb{Z} \quad \text{integers}$$

where h is a constant spacing between nodes.



The resulting formulas are usually evaluated at x_0.

Using a linear IP:

$$f'(x_0) \approx p'[f \mid x_0, x_1](x_0) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$$= \frac{f(x_0+h) - f(x_0)}{h}$$

(so-called forward FD)

$$f'(x_0) \approx p'[f \mid x_{-1}, x_0](x_0) = \frac{f(x_0) - f(x_{-1})}{x_0 - x_{-1}}$$

$$= \frac{f(x_0) - f(x_0-h)}{h}$$

(so-called backward FD)

What about approx. to f''?

Using a quadratic IP:

$$f'(x_0) \approx p'[f|x_{-1},x_0,x_1] = \frac{f(x_1)-f(x_{-1})}{x_1-x_{-1}}$$

$$= \frac{f(x_0+h)-f(x_0-h)}{2h}$$

$$f''(x_0) \approx p''[f|x_{-1},x_0,x_1] = \frac{f(x_1)-2f(x_0)+f(x_{-1})}{h^2}$$

$$= \frac{f(x_0+h)-2f(x_0)+f(x_0-h)}{h^2}$$

(so-called centered FD)

Ex.: (5) Approx. first derivative of $f(x) = \sin(x)$ at $x = 1.2$ (exact $f'(x) = \cos(x)$ of course :-)

⇝ Slides

We observe

(i) The error $e = |p'[f|\ldots] - f'(x)|$ behaves as $\left\{ \begin{matrix} O(h) \\ O(h^2) \end{matrix} \right\}$ for $\left\{ \begin{matrix} \text{forward/backward} \\ \text{centered} \end{matrix} \right\}$ FD

(ii) The error grows if h is too small (round-off errors)

Why? Due to the finite precision of floating point numbers

The error estimates could be derived from the interpolation error... But there is another way with the help of Taylor expansions/series:

$$f(x+h) = f(x) + f'(x)\cdot h + \frac{f''(x)}{2} h^2 + \dots + \frac{f^{(k)}(x)}{k!} h^k + \frac{f^{(k+1)}(\xi)}{(k+1)!} h^{k+1}$$

for some $\xi \in [x, x+h]$

remainder term (sometimes error term)

Ex.: (6) forward FD approx of f':

$$\frac{f(x_0+h) - f(x_0)}{h} = \frac{\cancel{f(x_0)} + f'(x_0)\cdot h + \frac{f''(x_0)}{2} h^2 + \dots - \cancel{f(x_0)}}{h}$$

$$= f'(x_0) + \frac{h}{2} f''(x_0) + \dots$$

negligible?

$$= f'(x_0) + \mathcal{O}(h) \quad ✓.$$

$h > h^2 > h^3$ for $h \ll 1$ small

So forward FD has order of accuracy $r = 1$. They are first order accurate

I.4 Numerical integration (aka Quadrature)

Goal: Approx.

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

Idea: Use IP $p[f|\ldots]$ to approx. $f(x)$ and integrate

Why is this easier? → Integrating polynomials is easy.

Def.: a finite calculation rule to compute an approx. to $I[f]$

$$Q[f] = \sum_{j=0}^{n} w_j \cdot f(x_j)$$

is called quadrature rule (QR).
The $x_j \in I = [a,b]$ are called the quadrature nodes (QNs) and the w_j the quadrature weights (QWs).

QRs can now easily be derived...

Let $p[f|x_0,\dots,x_n]$ be the IP of $f(x)$:

$$\int_a^b f(x)\,dx \approx \int_a^b p[f|x_0,\dots,x_n]\,dx$$

$$= \int_a^b \sum_{j=0}^{n} L_j^n(x)\cdot f(x_j)\,dx$$

$$= \sum_{j=0}^{n} \underbrace{\int_a^b L_j^n(x)\,dx}\cdot f(x_j)$$

$$= \sum_{j=0}^{n} \underset{\text{QW}}{w_j}\cdot \underset{\text{QN}}{f(x_j)} = Q[f]$$

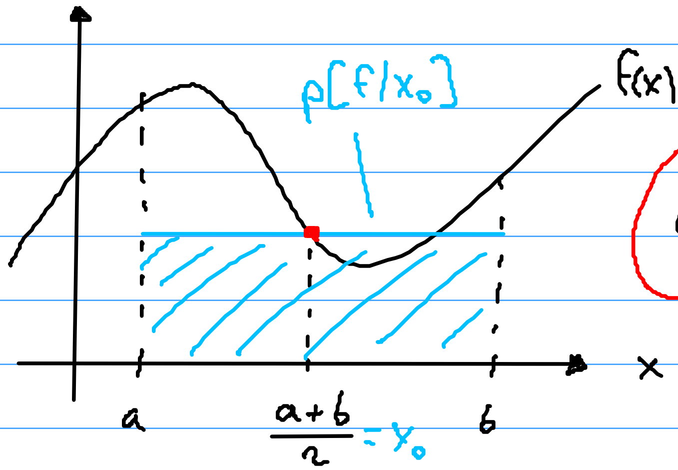
<u>Rem.:</u> The QWs do NOT depend on f!

For given QNs x_j compute them once and tabulate for posterity

⇝ for equally spaced QNs over $I=[a,b]$ this leads to so-called <u>Newton-Cotes QRs</u>

Popular examples

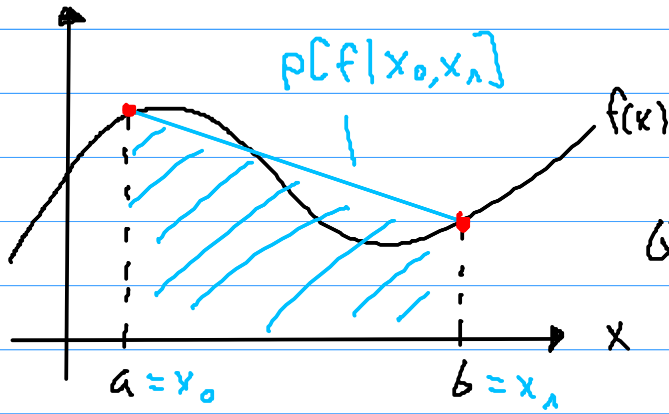
Ex.: (7) Midpoint rule (MR) $(n=0)$



$$Q_0[f] = (b-a)\cdot f\left(\frac{a+b}{2}\right)$$

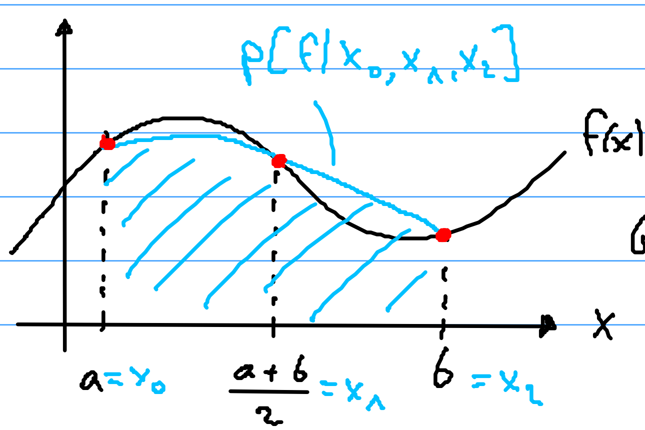
...for notation...

(8) Trapezoidal rule (TR) $(n=1)$



$$Q_1[f] = \frac{b-a}{2}\big(f(a)+f(b)\big)$$

(9) Simpson rule (SR) $(n=2)$



$$Q_2[f] = \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)$$

As a measure of quality one defines

Def.: the degree of exactness (DoE) q is defined as the maximum polynomial degree a QR can integrate exactly

We get directly:

$$\begin{aligned} &MR && q = \cancel{0} \rightsquigarrow 1 \\ &TR && q = 1 \\ &SR && q = \cancel{2} \rightsquigarrow 3 \end{aligned}$$

⌜Turns out that for even degree (and equidistantly spaced QNs) one wins a DoE for free :-) ⌟

Def.: We say that a QR is s-th order accurate if

for suff. smooth f!

$$\epsilon[f] = |Q[f] - I[f]| \overset{!}{=} \mathcal{O}((b-a)^s)$$

and call $\epsilon[f]$ the quadrature error (QE).

It holds: $s = q + 1$

As we have seen, high-degree (large n) interp. is in general not recommended

$\rightsquigarrow$ piecewise

Divide the integration interval $I = [a, b]$ in N subintervals

$$I_j = [x_{j-1}, x_j] \quad , \quad j = 1, 2, \ldots, N$$

$$x_j = a + \underbrace{\frac{b-a}{N}}_{h} \cdot j \quad , \quad j = 0, 1, \ldots, N$$

and apply a QR over each I_j.

This leads to so-called composite QRs (CQRs)

Ex.: (10) composite MR

$$Q_0^N[f] = h \cdot \sum_{k=1}^{N} f\left(\frac{x_{k-1} + x_k}{2}\right)$$

(11) composite TR

$$Q_1^N[f] = h \cdot \left(\frac{1}{2} f(a) + \sum_{k=1}^{N-1} f(x_k) + \frac{1}{2} f(b)\right)$$

(12) composite SR

$$Q_2^N[f] = \frac{h}{6}\left(f(a) + 2 \cdot \sum_{k=1}^{N-1} f(x_k) + 4 \cdot \sum_{k=1}^{N} f\left(\frac{x_{k-1} + x_k}{2}\right) + f(b)\right)$$

The QE of a CQR is (obviously) the sum of QEs over each subinterval.

One can show that

$$E^N[f] = | Q_n^N[f] - I[f] |$$

$$\leq \frac{\|f^{(q+1)}\|_\infty}{\underbrace{(q+1)!}_{\text{DoE}}}(b-a)h^{q+1} = \frac{\|f^{(s)}\|_\infty}{\underbrace{s!}_{\text{order of accuracy}}}(b-a)h^s$$

<u>Ex.:</u> (13) Compute approx. of $\int_0^1 \frac{1}{1+x}\,dx = \log(2)$

$\rightsquigarrow$ slides

25.09.24

Adaptive quadrature

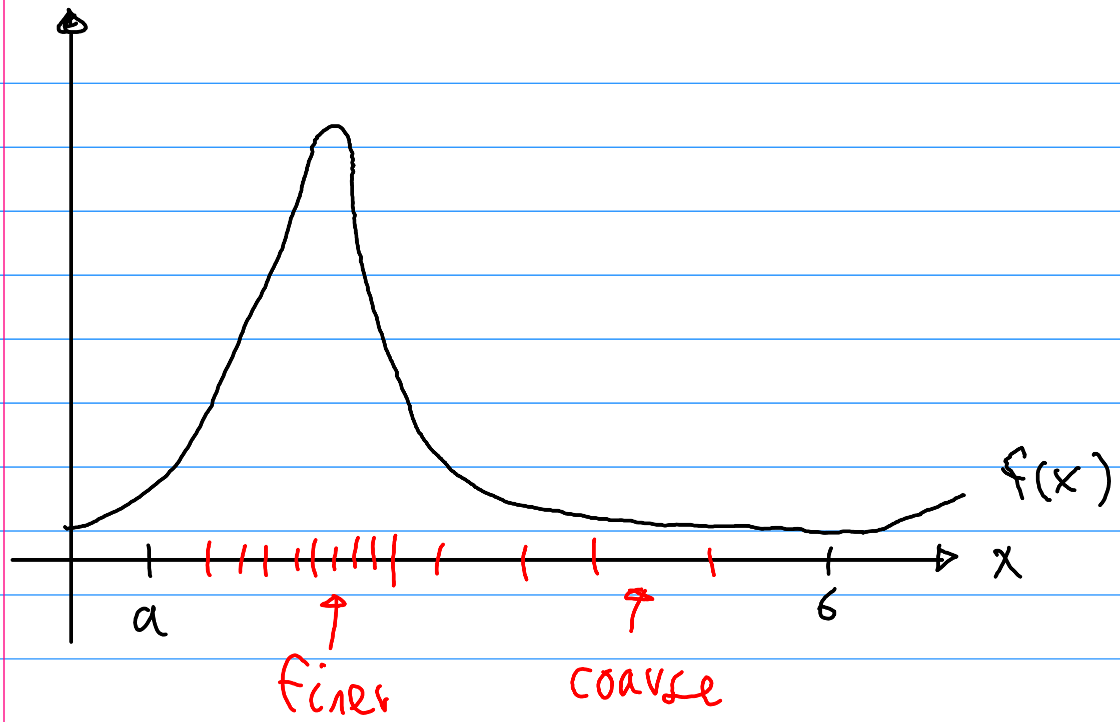
Goal: Approx.

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

up to a certain tolerance tol as efficiently as possible

i.e.: with the least number of function f evaluations as possible (because the can be computationally expensive)

Idea: Instead of dividing the interval $I = [a, b]$ into equally sized subintervals, put more subintervals where the function $f(x)$ varies "faster" / the quadrature error is greatest.



Need an estimate of the error and refine where the error is bigger than the tolerance.

Idea: Compute an approximation of the integral with two methods

$Q_1[f]$ method 1

$Q_2[f]$ method 2 (more precise than method 1)

and compare the results to get an estimate of the quadrature error.

Adaptive quadrature (~ Pseudo-MATLAB-Code)

```
function Q = adapt_quad(f, a, b, tol)
  Q1 = quad1(f, a, b)      method 1
  Q2 = quad2(f, a, b)      method 2
  E  = abs(Q1 - Q2)
  if E < tol
     Q = Q2     (keep the more precise
                        result)
  else
     Ql = adapt_quad(f, a, (a+b)/2, tol/2)
     Qr = adapt_quad(f, (a+b)/2, b, tol/2)
     Q = Qr + Ql
  end
end
```

Rem.: (i) Adaptive quadrature can be often very efficient, but not always.
There is no guarantee that the error is smaller than the chosen tolerance tol.

(ii) Choices for methods 1 / 2

Same method, but with half-subint.

$$Q_1[f] = Q_n^1[f]$$

$$Q_2[f] = Q_n^2[f]$$

Or method with higher order

$$Q_1[f] = Q_s^1[f]$$

$$Q_2[f] = Q_{s+1}^1[f]$$

E.g.: $s = 1$ trapezoidal

$s = 1+1=2$ Simpson