NTNU

Norwegian University of Science and Technology

TMA4125 Matematikk 4N

2 – Interpolation & Numerical Integration

Ronny Bergmann

Department of Mathematical Sciences, NTNU.

January 17, 2024

NTNU

Norwegian University of Science and Technology

Polynomial Interpolation

Interpolation: basic idea

Often we get some discrete measurement data

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

Goal. Find a function f that describes this data, i. e.

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

and that f is from a certain class (smoothness, polynomial,...) – let's say "nice" function

Alternatively. If we have a complicated function g:
take $(x_i, g(x_i))$ and find a "nice" function f that "explaines" the measurements well, since working with f is easier.
$\Rightarrow$ use approach from above.

Polynomial interpolation

Task. Given $n+1$ points $(x_i, y_i), i = 0, \ldots, n$, find a polynomial $p(x)$ of lowest possible degree satisfying the interpolation condition

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

The solution $p(x)$ is called interpolation polynomial.

The values x_i are called nodes, the points (x_i, y_i) are called interpolation points.

Example of an interpolation problem

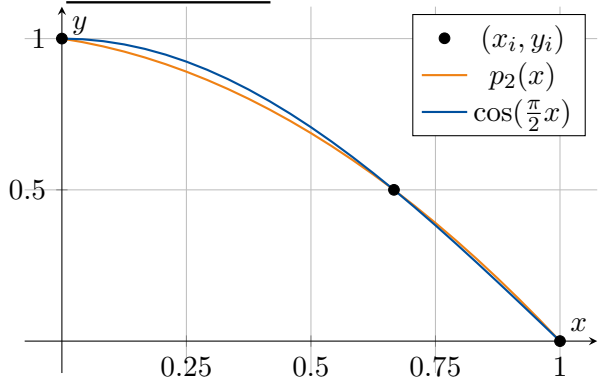
For given data

i	0	1	2
x_i	0	$\frac{2}{3}$	1
y_i	1	$\frac{1}{2}$	0

The corresponding interpolation polynomial is

$$p_2(x) = \frac{1}{4}\left(-3x^2 - x + 4\right).$$

The data are sample values of $\cos(\frac{\pi}{2}x)$ on $[0, 1]$.



- p_2 interpolates the data.
- Locally (on $[0, 1]$): p_2 explains $f(x) = \cos(\frac{\pi}{2}x)$ quite well.

Roadmap

We will discuss the following

- **Method.** How to compute the interpolation polynomial?
- Existence and uniqueness results
- Error analysis. If the polynomial is used to approximate a function, how good is the approximation?
- **Improvements.** If the nodes x_i can be chosen freely, how should we do it in order to reduce the error?
- **Splines.** What are they and how can we use them?

Polynomials: some useful facts

We already learned about

- $\mathbb{P}_n$ the set of polynomials of degree n or less
- $C^m[a, b]$ the set of all continuous functions that have continuous first m derivatives

and a polynomial of degree n we denote by $p_n \in \mathbb{P}_n$ written as

$$p_n(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0 = \sum_{i=0}^{n} c_i x^i,$$

where $c_i \in \mathbb{R}$, $i = 0, \ldots, n$, are some real coefficients.

Roots of a polynomial

The value r is a root or zero of a polynomial p if $p(r) = 0$.

A nonzero polynomial p_n of dergree n can never have more than n real roots (there are maybe less).

A polynomial p_n of degree n with n real roots $r_1, r_2, \ldots, r_n \in \mathbb{R}$ can be written as

$$p_n(x) = c(x - r_1)(x - r_2) \cdot \ldots \cdot (x - r_n) = c \prod_{i=1}^{n} (x - r_i).$$

Direct method

For a polynomial p_n of degree n we can write down the interpolation conditions that

$$p_n(x_j) = \sum_{i=0}^{n} c_i x_j^i = y_j, \qquad \text{for } j = 0, \ldots, n$$

has to hold.
These are $n+1$ equations and we have $n+1$ unknowns $c_0, c_1, \ldots, c_n$.

Example

Given the points

i	0	1	2	3	4
x_i	0	1	3	4	7
y_i	3	8	6	-1	2

find the lowest degree polynomial running through these points, i.e. a function $p(x)$ with

$$p(x_i) = y_i \qquad i = 0, \ldots, 4$$

of the form

$$p(x) = c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0.$$

Direct Method: Notes and Challenges.

For given data (x_i, y_i), $i = 0, \ldots, n$:

- the matrix we just constructed is called the Vandermonde matrix V

$$V = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^n \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}$$

- V is always of full rank
- But: V is ill-conditioned, that is
  1. shifting the nodes to $\tilde{x}_i = 100 + x_i$ we get $\tilde{x}_4 = 107$.
     $\Rightarrow$ bottom left entry: $107^4 = 131079601 \Rightarrow$ rounding errors!
  2. if two nodex x_i, x_j are close together
     $\Rightarrow$ small errors amplify!

A Remedy. "Rescale" to $[-1, 1]$

1. **Transfer** the nodes from $[a, b]$, $a = \min_i x_i$, $b = \max_i x_i$ to the "nicer" interval $[-1, 1]$:

$$x_i = \frac{a+b}{2} + t_i \frac{b-a}{2} \quad \text{and} \quad t_i = \frac{2}{b-a} x_i - \frac{b+a}{b-a}.$$

2. with (t_i, y_i) interpolate $q(t)$ ("lives" on $[-1, 1]$)
3. Use the formula from above to get $p(x)$ on $[a, b]$:

$$p(x) = q(t) = q\Big(\frac{2}{b-a} x - \frac{b+a}{b-a}\Big)$$

We avoid the too large numbers, since all $t_i \in [-1, 1]$

But the problem if two nodes are close persists.

Existence and uniqueness of interpolation polynomials

We have already proved the existence of such polynomials, simply by constructing them. But are they unique? The answer is yes!

Theorem. (Existence and uniqueness.)
Given $n+1$ points $(x_i, y_i)_{i=0}^n$ with distinct x values. Then there is one and only one polynomial $p_n(x) \in \mathbb{P}_n$ satisfying the interpolation condition

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

Proof. Suppose there exist two different interpolation polynomials p and q of degree n interpolating the same $n+1$ points. The polynomial $r(x) = p(x) - q(x)$ is of degree n with zeros in all the nodes x_i, that is a total of $n+1$ zeros. But then $r \equiv 0$, and the two polynomials p and q are identical.

Outlook: Lagrange interpolation

Given $n+1$ points (x_i, y_i), $i = 0, \ldots, n$, with distinct values of x_i.

The cardinal functions or Lagrange functions ℓ_i, $i = 0, \ldots, n$, are given by

$$\ell_i(x) := \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} = \frac{x - x_0}{x_i - x_0} \cdot \frac{x - x_1}{x_i - x_1} \cdot \ldots \cdot \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdot \ldots \cdot \frac{x - x_n}{x_i - x_n}$$

You will investigate their properties, e.g.

- $\ell_i \in \mathbb{P}_n$ for $i = 0, \ldots, n$
- $\ell_i(x_j) = \delta_{ij} = \begin{cases} 1 & \text{when } i = j, \\ 0 & \text{else} \end{cases}$
- they are constructed solely from the nodes x_i (no y_i involved)
- We obtain the interpolation polynomial directly by

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

Sketch: Error Analysis

1. Given some function $f \in C^{n+1}[a,b]$.
2. Choose $n+1$ distinct nodes x_i in $[a,b], i = 0, \ldots, n$
3. compute $p_n(x) \in \mathbb{P}_n$ to interpolate f

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

Error Analysis. What can be said about the error $e(x) = f(x) - p_n(x)$?

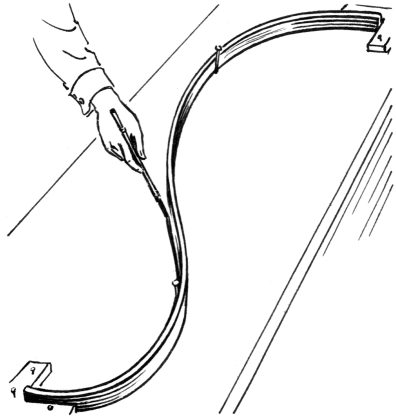
Theorem (without proof)

Given $f \in C^{(n+1)}[a,b]$, $p_n \in \mathbb{P}_n$ and $x_i \in [a,b]$, $i = 0, \ldots, n$ as above. Then, for each $x \in [a,b]$, there exists one $\xi(x) \in (a,b)$ such that

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

Goals. Find upper bound for $\max_{x \in [a,b]} f^{(n+1)}(n)$. and choose good x_i!

Splines: Motivation



a flat spline mainly used in shipbuilding. (source: wikipedia/flat spline)

Splines: Definition

Let $\Delta = [x_0, \dots, x_n]$ be an ordered set of points in $[a, b]$ with

$$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$$

and $k \in \mathbb{N}$. A spline of degree k relative to Δ is a function $s_k \colon [a, b] \to \mathbb{R}$ such that

1. $s_k\big|_{[x_j, x_{j+1}]} \in \mathbb{P}_k$ is a polynomial of degree $\leq k$ for $j = 0, \dots, n-1$
2. $s_k \in C^{(k-1)}([a, b])$

Note. These functions form a vector space $\mathcal{S}_{k,\Delta}$.

For $k = 1$ we speak of linear splines,
for $k = 2$ of quadratic splines, and
for $k = 3$ of cubic splines.

Example: Linear splines, degree $k = 1$

The "pieces" $s_1\big|_{[x_j, x_{j+1}]}$ are linear functions and s_1 to be in $C^0([a, b])$.

We can derive a basis of $S_{1,\Delta}$, the so-called "hat-functions"

$$\Lambda_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & \text{if } x_{i-1} \le x < x_i, \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & \text{if } x_i \le x < x_{i+1}, \\ 0 & \text{else,} \end{cases} \qquad i = 1, \ldots, n-1$$

and at the boundary we need Λ_0 and Λ_n to be "half-hats"

$$\Lambda_0(x) = \begin{cases} \frac{x_1 - x}{x_1 - x_0} & \text{if } x_0 \le x < x_1 \\ 0 & \text{else,} \end{cases} \quad \text{and} \quad \Lambda_n(x) = \begin{cases} \frac{x - x_{n-1}}{x_n - x_{n-1}} & \text{if } x_{n-1} \le x < x_n, \\ 0 & \text{else.} \end{cases}$$

$\Rightarrow$ to interpolate data (x_i, y_i), $i = 0, \ldots, n$, on Δ:

$$s_1(x) = \sum_{i=0}^{n} y_i \Lambda_i(x)$$

Cubic Splines & degrees of freedom

Splines of degree 3. Are the so called cubic splines $s_3(x)$. These are very widely used, e.g. in CAD.
On each interval we have a polynomial of degree 3:

$$s_3(x) = c_{i,0} + c_{i,1}x + c_{i,2}x^2 + c_{i,3}x^3, \quad x \in [x_i, x_{i+1})$$

But We can not choose them completely arbitrary,
since s_3 has to be twice continously differentiable, $s_3 \in C^2([a,b])$
$\Rightarrow$ at every inner node $x_1, \ldots, x_{n-1}$:
3 conditions $\Rightarrow 3(n-1)$ conditions in total.

We have $4n$ coefficients, that have to satifsy $3(n-1)$ conditions
$\Rightarrow S_{3,\Delta}$ has a dimension of $4n - 3(n-1) = n+3$

Cubic Splines: Boundary Conditions (BC)

Motivation.
$n+1$ interpolation conditions $s_3(x_i) = y_i$ and dimension $n+3$
$\Rightarrow$ 2 degrees of freedom / choices left.

Natural BC We require $s_3''(x_0) = 0$ and $s_3''(x_n) = 0$
$\Rightarrow$ Spline has no curvature at the boundary

Clamped BC We require $s_3'(x_0) = 0$ and $s_3'(x_n) = 0$
$\Rightarrow$ horizontal tangents at the ends
only useful if our function has this property, otherwise might look "odd".

not-a-knot BC remove x_1 and x_{n-1} from the grid, i.e.
Use $\tilde{\Delta} = [x_0, x_2, x_3, \ldots, x_{n-3}, x_{n-2}, x_n]$ for the interpolation ($n+1$ degrees of freedom left),
we still require $s_3(x_1) = y_1$ and $s_3(x_{n-1}) = y_{n-1}$ or in other words, we still consider $n+1$ conditions for interpolation.

NTNU

Norwegian University of Science and Technology

Numerical Integration

Introduction.

Imagine you want to compute the (finite) integral

$$I[f](a,b) \coloneqq \int_a^b f(x)\,\mathrm{d}x$$

The "usual" way is to find a primitive function F (also known as indefinite integral f) satisfying $F'(x) = f(x)$. Then we can compute

$$\int_a^b f(x)\,\mathrm{d}x = F(b) - F(a)$$

Challenge. Computing F analytically might be hard or F might not have a closed analytical form. For example

$f(x) = \mathrm{e}^{-x^2}$ (no elementary function F)

$f(x) = \dfrac{\log(2+\sin(\frac{1}{2}-\sqrt{x})^6)}{\log(\pi+\arctan(\sqrt{1-\exp(-2x-\sin(x))}))}$ (complicated)

Numerical quadrature.

A numerical quadrature or a quadrature rule is a formula for approximating $I[f](a,b)$. Quadratures are usually of the form

$$Q[f](a,b) = \sum_{i=0}^{n} w_i f(x_i),$$

where x_i, w_i, $i = 0, 1, \ldots, n$, are the nodes (points) and the weights of the quadrature rule $Q[f](a,b)$, respectively.

$Q[f](a,b)$ is defined by its nodes $\{x_i\}_{i=0}^n$ and weights $\{w_i\}_{i=0}^n$

- If f is given from the context, we write just short $I(a,b)$ and $Q(a,b)$.
- If $[a,b]$ is clear from context, we write $I[f]$ and $Q[f]$, respectively.
- quadrature rules are linear, i. e. for functions f, g and $\alpha, \beta \in \mathbb{R}$ it holds

$$Q[\alpha f + \beta g](a,b) = \alpha Q[f](a,b) + \beta Q[g](a,b)$$

Known examples.

You already know from Calculus 1:

Mid point rule. The mid point rule is the simplest possible rule

$$M[f](a,b) \coloneqq w_0 f(x_0) = (b-a) f\Big(\frac{a+b}{2}\Big)$$

The only node is the mid point $x_0 = \frac{a+b}{2}$ with weight $w_0 = b - a$.
Note. Instead of Q we use specific letters for these quadrature rules.

Known examples.

You already know from Calculus 1:

Trapezoidal rule. We use both boundaries to form a trapezoid.

$$T[f](a,b) \coloneqq w_0 f(x_0) + w_1 f(x_1) = (b-a) f\Big(\frac{f(a)+f(b)}{2}\Big)$$

So here we have $x_0 = a$, $x_1 = b$ and $w_0 = w_1 = \frac{b-a}{2}$.
Note. Instead of Q we use specific letters for these quadrature rules.

Known examples.

You already know from Calculus 1:

Simpson rule. We use all 3 nodes from before

$$S[f](a,b) := w_0 f(x_0) + w_1 f(x_1) + w_2 f(x_2) = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

with $x_0 = a$, $x_1 = \frac{a+b}{2}$, $x_2 = b$ and weights $w_0 = w_2 = \frac{b-a}{6}$ and $w_1 = \frac{2(b-a)}{3}$.
Note. Instead of Q we use specific letters for these quadrature rules.

Roadmap

1. construct the (known) quadratures from integration of interpolation polynomials
2. error analysis
3. composite quadrature rules – how to “divide and conquer”
4. adaptive quadrature rules – how to “divide cleverly”

Quadrature from integrating interpolation polynomials

Recap. Choose $n+1$ distinct nodes $x_0, \ldots, x_n$ in the interval $[a, b]$. Denote by p_n the interpolation polynomial satisfying the interpolation conditions

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

Idea. Integrating polynomials is easy!

$\Rightarrow$ Use $\int_a^b p_n(x)\,\mathrm{d}x$ as an approximation to $\int_a^b f(x)\,\mathrm{d}x$.

We consider the quadrature

$$I[f](a,b) \approx Q[f](a,b) \coloneqq \int_a^b p_n(x)\,\mathrm{d}x.$$

But what about the weights?

Weights for the quadrature based on p_n

To compute the weights we use the Lagrange form:

$$p_n(x) = \sum_{i=0}^{n} f(x_i)\ell_i(x), \qquad \text{where } \ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \quad i = 0, \ldots, n$$

Due to linearity of the integral we get for the weights w_i

$$\begin{aligned} Q[f](a,b) &= \int_a^b p_n(x)\,\mathrm{d}x = \int_a^b \sum_{i=0}^{n} f(x_i)\ell_i(x)\,\mathrm{d}x \\ &= \sum_{i=0}^{n} f(x_i) \int_a^b \ell_i(x)\,\mathrm{d}x = \sum_{i=0}^{n} f(x_i) w_i \end{aligned}$$

So the weights are simply computed as

$$w_i = \int_a^b \ell_i(x)\,\mathrm{d}x, \quad i = 0, \ldots, n,$$

and are independent of f.

Degree of precision

Definition. A numerical quadrature has degree of precision d if

$$Q[p](a,b) = I[p](a,b) \quad \text{for all} \quad p \in \mathbb{P}_d$$

and there is at least one $p \in \mathbb{P}_{d+1}$ such that $Q[p](a,b) \neq I[p](a,b)$.

Since both integrals and quadratures are linear, the degree of precision is d if

$$\begin{aligned} I[x^j](a,b) &= Q[x^j](a,b), \qquad j = 0, \ldots, d \\ I[x^{d+1}](a,b) &\neq Q[x^{d+1}](a,b). \end{aligned}$$

Let's revisit: The trapezoidal rule

If we want to approximate

$$I(0,1) = \int_0^1 \cos\left(\frac{\pi}{2}x\right) = \frac{2}{\pi} = 0.636619\ldots.$$

We can derive the Trapezoidal rule by the polynomial interpolation approach.

As a result we obtain

$$T(0,1) = \frac{1}{2}\big(f(0) + f(1)\big) = \frac{1}{2}$$

with an error of

$$I(0,1) - T(0,1) \approx 0.138\ldots.$$

and a degree of precision equal to 1.

Transfer the formula from $[-1, 1]$ to $[a, b]$

What if we have different intervals to tackle, say $[a, b]$ and $[c, d]$?

Construct your method on a reference interval $\hat{I} = [-1, 1]$, determine your quadrature points $\{t_i\}_{i=0}^n$ and weights $\{v_i\}_{i=0}^n$ (from Lagrange).

Use the transformation (cf. slide11)

$$x = \frac{b-a}{2}t + \frac{b+a}{2} \quad \text{so } \mathrm{d}x = \frac{b-a}{2}\,\mathrm{d}t$$

and thus we define the points $\{x_i\}_{i=0}^n$ and weights $\{w_i\}_{i=0}^n$ for $[a, b]$ as

$$x_i = \frac{b-a}{2}t_i + \frac{b+a}{2}, \quad w_i = \frac{b-a}{2}v_i \qquad \text{for } i = 0, \ldots, n.$$

Note. It is enough to store the weights v_i on $[-1, 1]$ and compute the transform for any given interval.

Example: Simpson's rule

Simpson's rule on $[-1, 1]$ uses the nodes $t_0 = -1$, $t_1 = 0$ and $t_2 = 1$.
With the cardinal functions

$$\ell_0(t) = \frac{1}{2}(t^2 - t), \qquad \ell_1(t) = 1 - t^2, \qquad \ell_2(t) = \frac{1}{2}(t^2 + t).$$

We get the weights

$$w_0 = \int_{-1}^{1} \ell_0(t)\,\mathrm{d}t = \frac{1}{3}, \qquad w_1 = \int_{-1}^{1} \ell_1(t)\,\mathrm{d}t = \frac{4}{3}, \qquad w_2 = \int_{-1}^{1} \ell_2(t)\,\mathrm{d}t = \frac{1}{3}$$

such that

$$\int_{-1}^{1} f(t)dt \approx \int_{-1}^{1} p_2(t)dt = \sum_{i=0}^{2} w_i f(t_i) = \frac{1}{3}\Big(f(-1) + 4f(0) + f(1)\Big).$$

The transformation yields the points $x_0 = a$, $x_1 = \frac{b+a}{2}$, $x_2 = b$ and we get

$$S(a, b) = \frac{b-a}{6}\Big(f(a) + 4f\big(\frac{b+a}{2}\big) + f(b)\Big).$$

Quadrature in Practice: Divide and Conquer

In the following, you will learn the steps on how to construct realistic algorithms for numerical integration, similar to those used in software like Matlab or SciPy/NumPy. The steps are:

1. Choose $n+1$ distinct nodes on a standard interval $[-1,1]$.
2. Let $p_n(x)$ be the polynomial interpolating some general function f in these nodes, and let the $Q[f](-1,1) = I[p_n](-1,1)$.
3. Transfer the formula Q from $[-1,1]$ to some interval $[a,b]$.
4. Find the composite formula, by dividing the interval $[a,b]$ into subintervals and applying the quadrature formula on each subinterval.
5. Find an expression for the error $E[f](a,b) = I[f](a,b) - Q[f](a,b)$.
6. Find an expression for an estimate of the error, and use this to create an adaptive algorithm.

Improving a quadrature rule by composition

To generate more accurate quadrature rules $Q[f](a,b)$ we have in principle two possibilities

- Increase the order of the interpolation polynomial used to construct $Q(a,b)$.
- Subdivide the interval $[a,b]$ into smaller subintervals and apply a quadrature rule on each of the subintervals, leading to Composite Quadrature Rules.

Composite quadrature rules

For a composite quadrature rule select $m \geq 2$
and divide $[a, b]$ into m equispaced subintervals

$$[x_{i-1}, x_i], \qquad i = 1, \ldots, m,$$

where $x_i = a + ih, \quad i = 0, \ldots, m$, and $h = \dfrac{b-a}{m}$.

Then for a given quadrature rule $Q[\cdot](x_{i-1}, x_i)$ and define the composite quadrature rule by

$$\int_a^b f(x)\,\mathrm{d}x \approx Q_m(f)([x_{i-1}, x_i]_{i=1}^m) := \sum_{i=1}^m Q[f](x_{i-1}, x_i)$$

Composite Simpson's rule

Idea. Split $[a, b]$ into m subintervals, do Simpson's rule on each.
$\Rightarrow$ we also need the mid points. So:

Divide $[a, b]$ into $2m$ equal intervals of length $h = (b - a)/(2m)$.
Let $x_j = a + jh$, $i = 0, \cdots, 2m$, and apply Simpson's rule on each subinterval $[x_{2j}, x_{2j+2}]$ (with nodes $x_{2j}, x_{2j+1}, x_{2j+2}$). We get

$$\int_a^b f(x)dx = \sum_{j=0}^{m-1} \int_{x_{2j}}^{x_{2j+2}} f(x)\,\mathrm{d}x \approx S_m(a, b) := \sum_{j=0}^{m-1} S(x_{2j}, x_{2j+2})$$

Plugging in all small Simpson's rules we get

$$\begin{aligned} S_m(a, b) &:= \sum_{j=0}^{m-1} \frac{h}{3}\Big(f(x_{2j}) + 4f(x_{2j+1}) + f(x_{2j+2})\Big) \\ &= \frac{h}{3}\Big(f(x_0) + 4\sum_{j=0}^{m-1} f(x_{2j+1}) + 2\sum_{j=1}^{m-1} f(x_{2j}) + f(x_{2m})\Big) \end{aligned}$$

Numerical Example for Composite Simpson's rule

We again consider $f(x) = \cos\left(\frac{\pi x}{2}\right)$.

Intuitively.
If we spend more points, the error should decrease.
But: How much does it decrease – or in other words – how fast?

From the experiment we observe that the error is reduced by a factor of approx. $0.0625 = \frac{1}{16}$ when doubling the number of subintervals m.

Two interpretations:
In number of points m. If we write $E_m(a,b) = |I(a,b) - S_m(a,b)|$, then

$$\frac{1}{16} E_m(a,b) \approx E_{2m}(a,b)$$

In step size $h = \frac{b-a}{m}$. We have $16 = 2^4$ so the error has to behave like a constant C times h^4, since

$$C\left(\frac{h}{2}\right)^4 = \frac{C}{2^4} h^4 = \frac{C}{16} h^4$$

Obtaining Error bounds for Composite Quadratures

Recap. We know the error bounds for the Quadrature of $f \in C^4[a,b]$:

- Trapezoidal rule $T(a,b) - I(a,b) = \dfrac{(b-a)^3}{12} f''(\xi)$ for some $\xi \in (a,b)$
- Simpson rule $S(a,b) - I(a,b) = -\dfrac{(b-a)^5}{2880} f^{(4)}(\xi)$ for some $\xi \in (a,b)$

For the composite rules we derived

1. $T_m(a,b) - I(a,b) = \dfrac{(b-a)h^2}{12} f''(\eta), \eta \in (a,b)$
2. analogously $S_m(a,b) - I(a,b) = -\dfrac{(b-a)h^4}{180} f^{(4)}(\eta), \eta \in (a,b)$

Adaptive Integration – Idea

Idea. Instead of equispaced points, use a basic function, for example `SimpsonBasic`, that returns a quadrature $Q(a,b)$ and an error estimate $\mathcal{E}(a,b)$ to partition the interval

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

such that (automatically) for any $k = 0, \ldots, m-1$ we have

$$|\mathcal{E}(x_k, x_{k+1})| \leq \frac{x_{k+1} - x_k}{b-a} \text{Tol}$$

where Tol is a given tolerance (by the user).

This way the accumulated error is

$$\mathcal{E}(a,b) \approx \sum_{j=0}^{m-1} \mathcal{E}(x_k, x_{k+1}) \leq \text{Tol}.$$

Algorithm. Adaptive quadrature

Given f, a, b and a user defined tolerance Tol.

1. Calculate $Q(a,b)$ and $\mathcal{E}(a,b)$.

2. If $|\mathcal{E}(a,b)| \leq$ Tol

- Accept the result, return $Q(a,b) + \mathcal{E}(a,b)$ as an approximation to $I(a,b)$.

else

- set $c = (a+b)/2$, and repeat the process on each of the subintervals $[a,c]$ and $[c,b]$, with tolerance Tol$/2$.

3. Sum up the accepted results from each subinterval.

How can we get a good estimate for the error?

If we consider (composite) Simpson $S(a,b) = S_1(a,b)$.
How to estimate its error E_1 in Step 1 on the last slide?

Idea. Use $S_2(a,b)$ as follows

1. Compute $S_1(a,b)$ (requires f at the mid point $c = \frac{a+b}{2}$)
2. Compute $S_2(a,b)$ (requires f additionally at $d = \frac{a+c}{2}$ and $e = \frac{c+b}{2}$)
3. Compute the approximate errors

$$\mathcal{E}_1(a,b) = \frac{16}{15}(S_2(a,b) - S_1(a,b)) \quad \text{and} \quad \mathcal{E}_2(a,b) = \frac{1}{15}(S_2(a,b) - S_1(a,b))$$

4. use $\mathcal{E}_2(a,b)$ to decide, whether on $[a,b]$ we are "good enough" ($\leq \mathrm{Tol}$)
   If we are not good enough, repeat Steps 1-5 for both $[a,c]$ and $[c,b]$.
5. if so, return the best approximate for $I(a,b)$, here: $S_2(a,b) + \mathcal{E}_2(a,b)$