Interpolation of function value

outline

- definition of interpolation
- polynomial interpolation - Lagrange formula
- estimation of interpolation error
- cubic spline interpolation

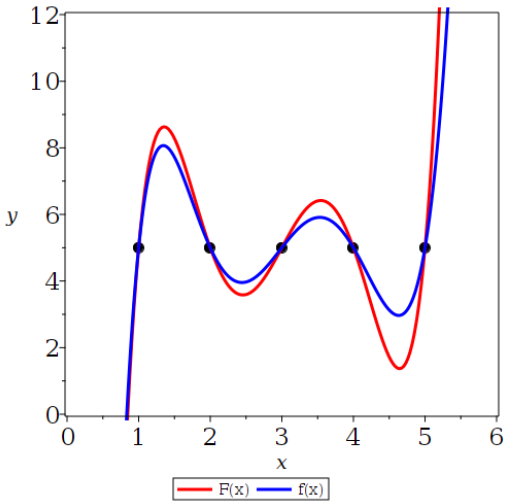
Definition of interpolation in 1D

For a given set of (interpolation) the nodes x_k and the values of unknown function $y_k=f(x_k)$

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$$

we need to find such interpolation function F(x) which at each node has the same value as the unknown function F(x). Moreover the difference between f(x) and F(x) (error) between neighbouring nodes shall be minimal.

This definition can be extended for more than one dimension but we limit our consideration to the basic 1D problems.



Remarks:

- interpolation is defined in an interval established by the left and right outermost nodes

$$F : \mathbb{R} \to \mathbb{R} \quad \Longleftrightarrow \quad x \in [x_0, x_n]$$

- we don't know what happens outside this region, such issue is classified as **extrapolation**

Aims of interpolation

- for a given tabulated sparse data we need to estimate the intermediate values
- polynomial interpolation is directly exploited in numerical intergration
- to model the shape of functions in 1D or surfaces in 2D/3D
- replacing complicated mathematical expressions with more simple ones

Polynomial Interpolation (Lagrange interpolation)

For any set of (n+1) interpolation data

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$$

we can determine only **one unique polynomial of n-degree** in interpolation region

$$W_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

In order to convince ourselves it's a truth let's calculate value of polynomial at each node

$$\begin{aligned} a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n &= y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n &= y_1 \\ \cdots\cdots\cdots\cdots\cdots &= \cdots \\ a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n &= y_n \end{aligned}$$

this operation provieds us with a square system of linear equations, it can be solved provided that the determinat of the matrix of this SLE does not vanish (matrix is nonsingular).

$$A\vec{a} = \vec{y}$$

$$Det(A) \neq 0?$$

Matrix of SLE has special form – it's **Vandermode matrix**

$$A = \begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^n \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_n & x_n^2 & x_n^3 & \cdots & x_n^n \end{bmatrix}$$

Its determinant can be written with a concise formula

$$Det(A) = D = \prod_{0 \le j < i \le n} (x_i - x_j) \neq 0$$

and we may conclude – the SLE defined for polynomial interpolation has a solution and it is given as

$$a_i = \frac{1}{D} \sum_{j=0}^{n} y_j D_{ij}$$

D_{ij} – minor obtained after removing i-th row and j-th column form matrix A

This expression proves the uniqe polynomial exists, but we must find a simpler way to determine the polynomial coeffiecients a_i.

Let's substitute sum of minors for coefficients a_i in the polynomial

$$a_i = \frac{1}{D}\sum_{j=0}^{n} y_j D_{ij} \quad \Rightarrow \quad W_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

and sort terms with y_i → notice here that every such terms contains contributions from all monomials

$$W_n(x) = y_0\Phi_0(x) + y_1\Phi_1(x) + \cdots + y_n\Phi_n(x)$$

the functions $\Phi_k(x)$ are different polynomials of at most n-degree, because each one stands at different y_k value they are **the nodal polynomials** and need to be found.

To determine the form of nodal polynomial we again exploit the interpolation condition.
After substituting position of nodes for an argument of polynomial we get

$$W_n(x_i) = y_0\Phi_0(x_i) + y_1\Phi_1(x_i) + \ldots + y_i\Phi_i(x_i) + \ldots + y_n\Phi_n(x_i) = y_i$$

and imediately deduce essential condition for the nodal polynomials

$$\Phi_j(x_i) = \begin{cases} 0 & \text{for } j \neq i \\ 1 & \text{for } j = i \end{cases}$$

In other words, we shall define such polynomial which has value 1 at the node it is ascribed to and vanishes at every other node.

$$\Phi_j(x_i) = \begin{cases} 0 & \text{for } j \neq i \\ 1 & \text{for } j = i \end{cases}$$

This is quite easy – just express polynomial in factor form

$$\Phi_j(x) = \lambda(x - x_0)(x - x_1) \dots\dots (x - x_{j-1})(x - x_j)(x - x_{j+1}) \dots (x - x_n)$$

to find the normalization constant λ use the interpolation condition

$$1 = \lambda(x_j - x_0)(x_j - x_1) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_n)$$

Finally, after dividing both expression we get the explicit form of nodal polynomial

$$\Phi_j(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_n)}{(x_j - x_0)(x_j - x_1) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_n)}$$

Now, interpolation polynomial can be constructed, to make the final formula more compact we use the full degree (n+1) factor polynomial containing all nodes as its roots

$$\omega_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n)$$

$$W_n(x) = \sum_{j=0}^{n} y_j \Phi_j(x) = \sum_{j=0}^{n} y_j \frac{\omega_n(x)}{(x - x_j)\left\{ \frac{\omega_n(x)}{x - x_j} \right\}\Big|_{x=x_j}} = \sum_{j=0}^{n} y_j \frac{\omega_n(x)}{(x - x_j)\omega_n'(x_j)}$$

Lagrange interpolation formula

$$W_n(x) = \sum_{j=0}^{n} y_j \frac{\omega_n(x)}{(x - x_j)\left\{ \frac{\omega_n(x)}{x - x_j} \right\}\Big|_{x=x_j}}$$

$$\omega_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n)$$

If one needs to calculate the value of Lagrange polynomial, there is no need to compute its coefficients explicilty

input: n, x, $\vec{x} = [x_0, x_1, \ldots, x_n]$, $\vec{y} = [y_0, y_1, \ldots, y_n]$

```
    W ← 0
for j=0 to n by 1 do
    for i=0 to n by 1 do
        α ← 1
        β ← 1
        if |i−j|>0 then
            α ← α · (x − x_i)
            β ← β · (x_j − x_i)
        end if
    end do
    W ← W + y_j α/β
end do
```

Interpolation error

In polynomial interpolation, the interpolation function exactly reconstruct the values of unknown function at nodes, however, no one should expect that finite degree polynomial would reconstruct all functions, especially these which have infinite Taylor expansion.

To assess the difference between an exact value of function F(x) and the interpolation polynomial $W_n(x)$ we use the following **interpolation error**

$$\varepsilon(x) = f(x) - W_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_n(x), \qquad \xi \in [x_0, x_n]$$

$f^{(n+1)}(x) -$ n+1 derivative of f

Remarks:

- for fixed outermost points of interpolation interval the supremum of (n+1) derivative of f is also fixed
- one may naively think that the factorial in denominator should dump the error for increasing number on node, we will see it's not the truth due to Runge effect
- the only effective way to improve the Lagrange polynomial interplation is to optimally select the positions of nodes → the optimal choice are the Chebyshev nodes

Optimal selection of nodes in Lagrange interpolation

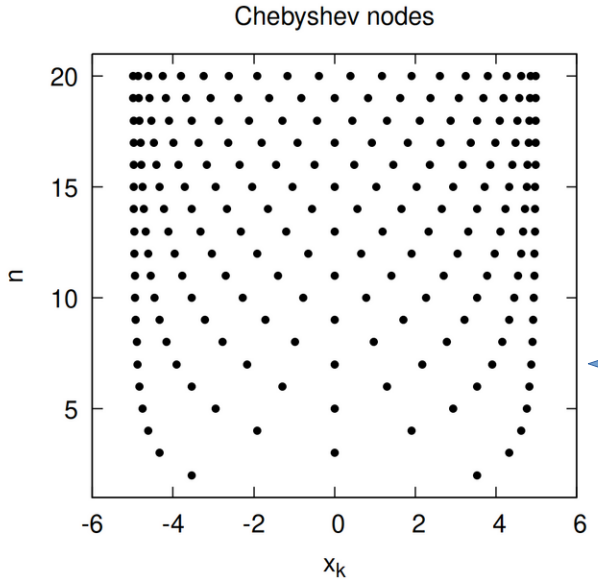
The optimal nodes are defined by the roots of Chebyshev polynomials

$$\mathrm{T}_n(x) = \cos\left[n \cdot arc\ \cos(x)\right], \qquad x \in [-1, 1]$$

$$x_m = \cos\left(\frac{2m+1}{2n+2}\pi\right), \ m = 0, 1, 2, \ldots, n$$

Chebyshev polynomial are defined for an interval [-1,1] if the roots/nodes are needed for different interval e.g. [a,b] their position must be then rescaled

$$z \in [-1, 1] \quad \rightarrow \quad x \in [a, b] \qquad \Longrightarrow \qquad x = \frac{1}{2}[(b-a)z + (b+a)], \ x \in [a, b]$$



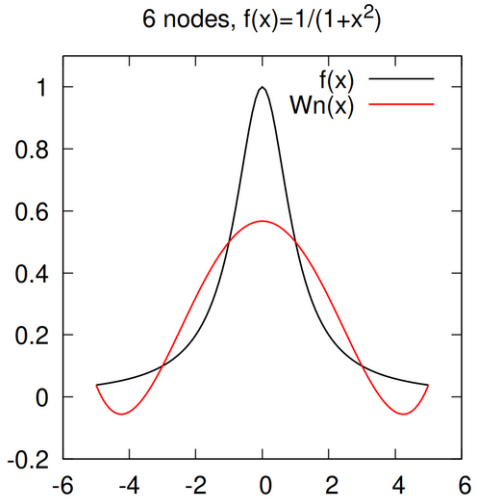
$$x_m = \frac{1}{2}\left[(b-a)\cos\frac{2m+1}{2n+2}\pi + (b+a)\right]$$

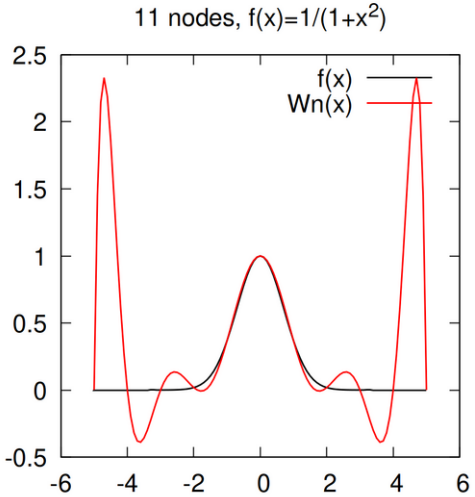
nodes condense at the end of interpolation interval

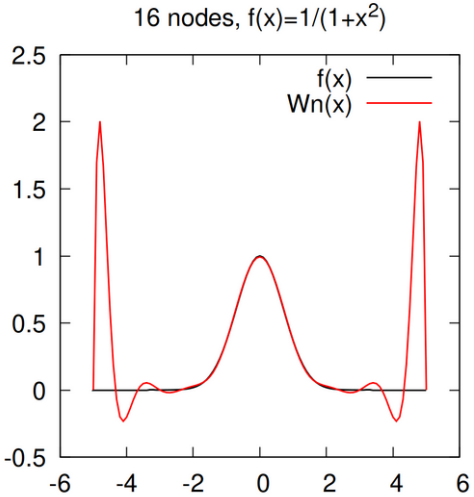
Example: interpolation of Lorentzian function with equidistant and Chebyshev nodes

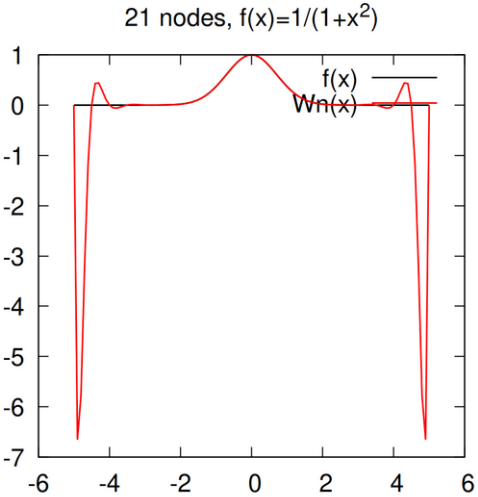
$$f(x) = \frac{1}{1+x^2}$$

equidistant nodes

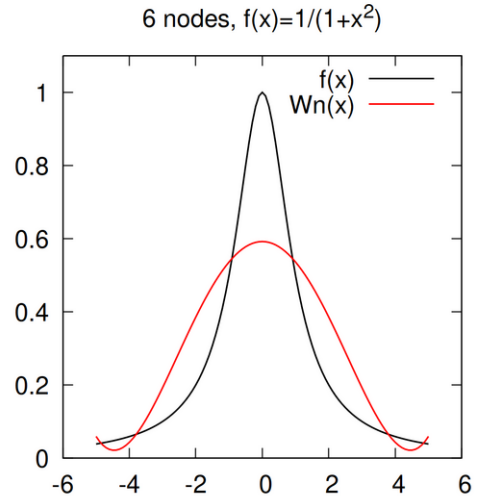


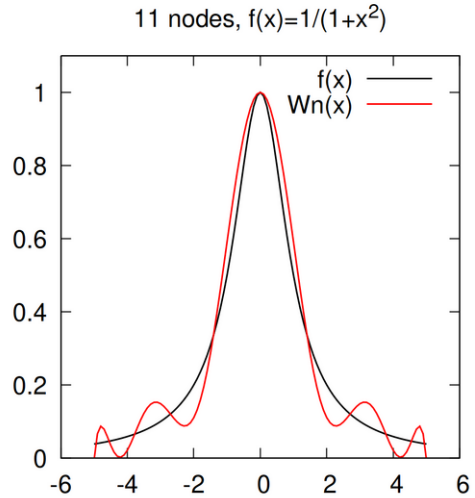


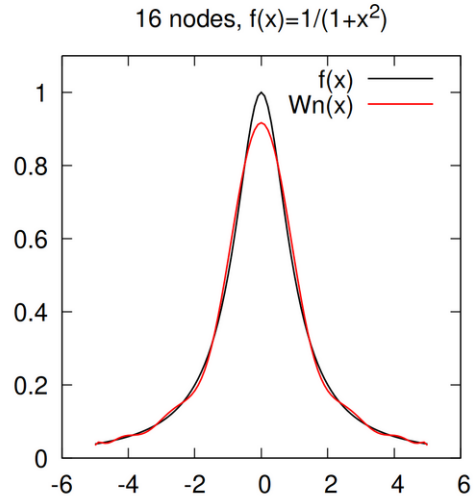


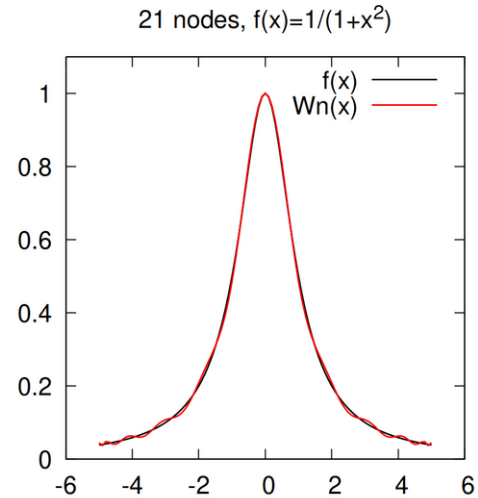


Chebyshev nodes









Remarks:

- high degree Lagrange interpolation is rarely used in practise due to **Runge effect**, it interpolates well the middle part of interpolation interval but strongly oscillates at its ends

- even shifting more nodes close to the ends largely dumps the oscilations, these are still significant making the data not reliable

- Lagrange interpolation formula is still used but for small number of nodes in numerical integration formulae (Newton-Cotes, Gauss quadrature)

- we need much better interpolation method like **cubic spline**

Cubic spline interpolation

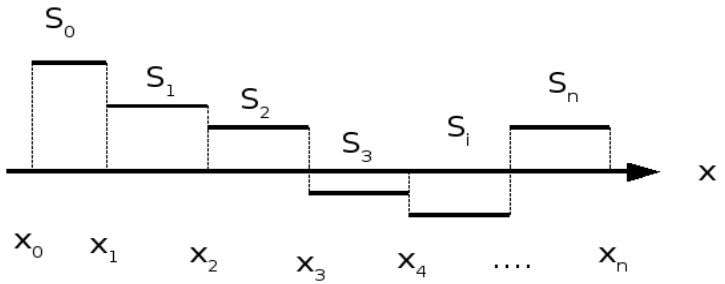
Lagrange polynomial interpolation suffers from enormously enlarged errors for increasing number of nodes, hence we have to go in the opposite direction.

- let's try to interpolate function with low m-degree polynomial, because it works for small number of nodes
- interpolation interval nodes shall be divided into small packets of nodes, interpolation is made for each segment with different polynomial, e.g. in i-th segment

$$s_i(x) = c_{im}x^m + c_{im-1}x^{m-1} + \cdots + c_{i1}x + c_{i0}, \qquad x \in (x_i; x_{i+1})$$

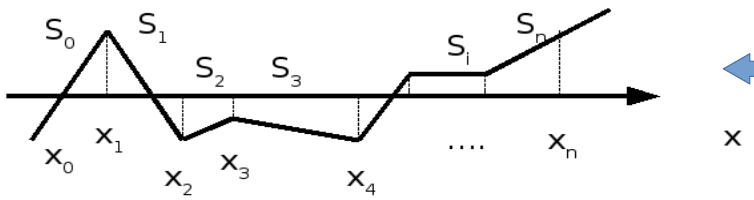
- at the border between two segments two polynomials must have the same: values of function and set of derivatives

m=0



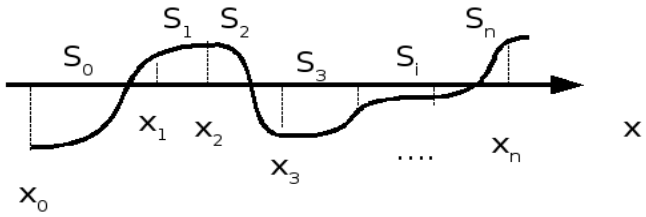
reject
interpolation function is not continuous, breaks interpolation condition

m=1



reject
interpolation is continuous but the 1-st order derivative is not

m=3



accept
cubic polynomial guarantees continuity up to 2-nd order derivative

From simple analysis of some possible choices of polynomial degree we deduce the minimal degree is m=3, it ensures continuity not only 1-st derivative (smoothness) but also 2-nd order one allowing for larger bending of polynomial within single segment. Now we may define the basics of cubic spline interpolation.

For given set of interpolation nodes (x_k) and function values (y_k)

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$$

we introduce set of elements with ends defined by a pair of consequtive nodes

$$[x_0, x_1],\ [x_1, x_2],\ \ldots [x_{n-1}, x_n]$$

and within each element we use polynomial of m=3 degree (cubic splines) to interpolate the tabulated function

$$s_i(x) = c_{im}x^m + c_{im-1}x^{m-1} + \cdots + c_{i1}x + c_{i0}, \qquad x \in (x_i; x_{i+1})$$

The total interpolation function is linear combination of cubic splines

$$s(x) = \sum_{i=0}^{n-1} c_i s_i(x), \qquad x \in [a, b]$$

Expressions for cubic splines can be determined in two ways

- we set c_i=1 and calculate the coefficients c_{im} of all cubic splines
- we chose c_{im} (cubic splines are predefined) and calulate c_i

Although, both method will give similar results we consider the former as it is more flexible.

Cubic splines defined by 2-nd order derivatives

For n cubic splines we need to determine

$$N = n(m+1)$$

coefficients $c_{i,m}$, by fitting 0-th, 1-st and 2-nd derivaties at inner nodes we get

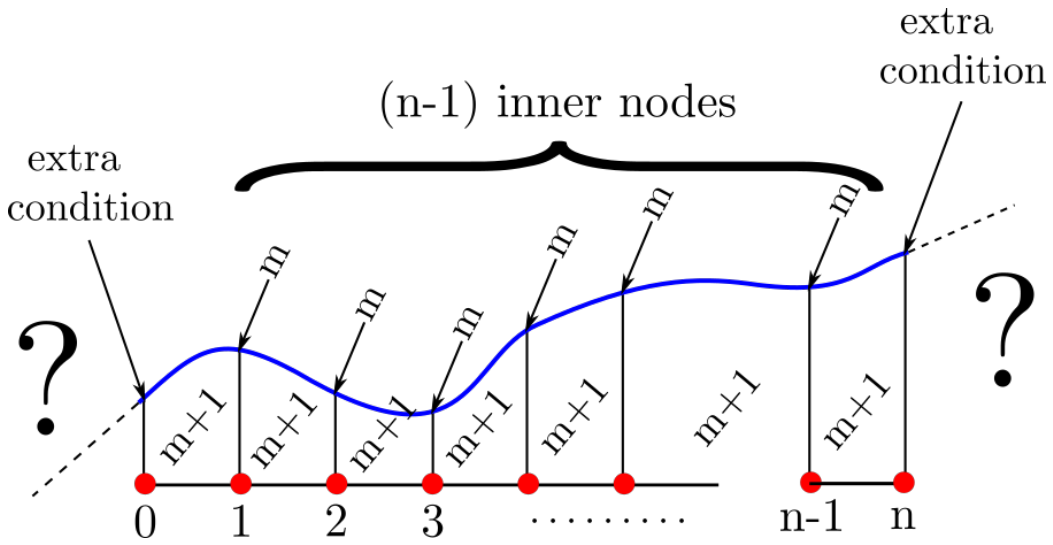
$$K = (n-1) \cdot m$$

conditions, and by utilizing the basic interpolation condition (values at nodes) we have additional conditions

$$L = n + 1$$

$$N - (K + L) = m - 1 = 2$$

We lack two conditions which are the free parameters of the method, these can be linked with function behaviour at two ends of interpolation interval



Boundary conditions in cubic spline interpolation

$$s^{(i)} = \frac{d^i s(x)}{dx^i}$$

- 1-st derivative

$$s^{(1)}(x_0) = \alpha_1$$
$$s^{(1)}(x_n) = \beta_1$$
$$\alpha_1, \beta_1 \in \mathbb{R}$$

- 2-nd derivative

$$s^{(2)}(x_0) = \alpha_2$$
$$s^{(2)}(x_n) = \beta_2$$
$$\alpha_2, \beta_2 \in \mathbb{R}$$

- periodic boundary condition

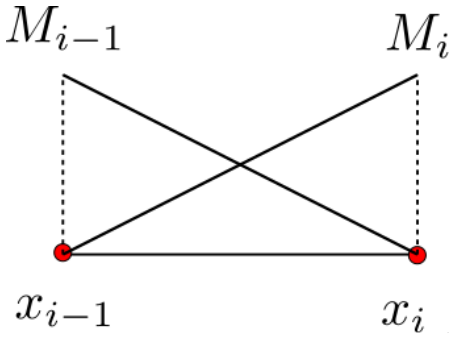
$$s^{(i)}(x_0) = s^{(i)}(x_n) = \alpha_3, \qquad i = 1, 2, \quad \alpha \in \mathbb{R}$$

In the following derivation we will use abbreviation for second order derivative

$$M_j = s^{(2)}(x_j), \qquad j = 0, 1, \ldots, n$$

We have previously assumed continuity of second order derivative at the joint of two elements.
This condition can be written as follows

$$\begin{aligned} s^{(2)}_{i-1}(x) &= M_{i-1}\frac{x_i - x}{h_i} + M_i\frac{x - x_{i-1}}{h_i} \\ x &\in [x_{i-1}, x_i] \\ h_i &= x_i - x_{i-1} \end{aligned}$$



After integrating this expression over x we get

$$s^{(1)}_{i-1}(x) = -M_{i-1}\frac{(x_i - x)^2}{2h_i} + M_i\frac{(x - x_{i-1})^2}{h_i} + A_i$$

and integrate again to get general formula describing the spline employing M_j

$$\begin{aligned} s_{i-1}(x) &= M_{i-1}\frac{(x_i - x)^3}{6h_i} + M_i\frac{(x - x_{i-1})^3}{6h_j} \\ &+ A_i(x - x_{i-1}) + B_i \end{aligned}$$

here we lack of:
A_i, B_i, M_{i-1}, M_i

- constants A_i and B_i we determine utilizing fundamental interpolation condition at both ends of element

$$s_{i-1}(x_{i-1}) = M_{i-1}\frac{h_i^2}{6} + B_i = y_{i-1}$$

$$s_{i-1}(x_i) = M_i\frac{h_i^2}{6} + A_i h_i + B_i = y_i$$

$$B_i = y_{i-1} - M_{i-1}\frac{h_i^2}{6}$$

$$A_i = \frac{y_i - y_{i-1}}{h_i} - \frac{h_i}{6}(M_i - M_{i-1})$$

Constants depend on second derivatives.

- at any node x_i the 1-st derivate is continuous

$$s_{i-1}^{(1)}(x_i) = s_i^{(1)}(x_i)$$

$$s_{i-1}^{(1)}(x_i - 0) = \frac{h_i}{6}M_{i-1} + \frac{h_i}{3}M_i + \frac{y_i - y_{i-1}}{h_i}$$

$$s_i^{(1)}(x_i + 0) = -\frac{h_{i+1}}{3}M_i - \frac{h_{i+1}}{6}M_{i+1} + \frac{y_{i+1} - y_i}{h_{i+1}}$$

by equaling both expression we get the linear equation, connecting three unknown 2-nd derivatives, valid for the inner nodes

$$\mu_i M_{i-1} + 2M_i + \lambda_i M_{i+1} = d_i, \qquad i = 1, 2, \ldots, n-1$$

$$\lambda_i = \frac{h_{i+1}}{h_i + h_{i+1}}, \qquad \mu_i = 1 - \lambda_i$$

We get the linear equation with unknown second derivatives

$$\mu_i M_{i-1} + 2M_i + \lambda_i M_{i+1} = d_i$$

$$\lambda_i = \frac{h_{i+1}}{h_i + h_{i+1}}, \qquad \mu_i = 1 - \lambda_i$$

$$d_i = \frac{6}{h_i + h_{i+1}} \left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i} \right)$$

In order to form complete system of linear equations which has unique solution we must adjoint the boundary conditions

- conditions for 1-st order derivatives

here we modify only rhs →

$$2M_0 + M_1 = d_0$$

$$M_{n-1} + 2M_n = d_n$$

$$d_0 = \frac{6}{h_1} \left(\frac{y_1 - y_0}{h_1} - \alpha_1 \right)$$

$$d_n = \frac{6}{h_1} \left(\beta_1 - \frac{y_n - y_{n-1}}{h_n} \right)$$

- condition for 2-nd order derivatives

here we must modify both, the lhs and the rhs →

$$M_0 = \alpha_2$$

$$M_n = \beta_2$$

Writing linear equation for all nodes + accounting for boundary condition we get SLE in matrix form which must be solved

- 1-st order derivative

$$\begin{bmatrix} 2 & 1 & 0 & \cdots & 0 \\ \mu_1 & 2 & \lambda_1 & \cdots & 0 \\ 0 & \mu_2 & 2 & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & \cdots & & 2 & \lambda_{n-1} \\ 0 & \cdots & & 1 & 2 \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ \vdots \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

- 2-nd order derivative

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \mu_1 & 2 & \lambda_1 & \cdots & 0 \\ 0 & \mu_2 & 2 & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & \cdots & & 2 & \lambda_{n-1} \\ 0 & \cdots & & 0 & 1 \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ \vdots \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} \alpha_2 \\ d_1 \\ \vdots \\ \vdots \\ d_{n-1} \\ \beta_2 \end{bmatrix}$$

The solution vector provides the values of M_i which allows to determine all splines

$$s_{i-1}(x) = M_{i-1}\frac{(x_i - x)^3}{6h_i} + M_i\frac{(x - x_{i-1})^3}{6h_i} + A_i(x - x_{i-1}) + B_i, \quad i = 1, 2, \ldots, n$$

Remarks:

- cubic spline interpolation is very effective, SLE is solved with LU designed for tridiagonal matrices, it is fast and stable

- the number of interpolation nodes is not limited, could be hundreds or more

- Runge effect do not occur due to low degree splines, an interpolation error could be made as small as needed but obviously this requires more and more nodes

- a small drawback is that we interpolate the function piecewise and we don’t have a concise expression for the unknown function f(x)

Example: interpolation of Lorentzian function with cubic splines

- number of nodes (n+1)
- boundary conditions: vanishing 2-nd derivative at both ends