Function Interpolation

Marcin Chrząszcz, Danny van Dyk
mchrzasz@cern.ch,
danny.van.dyk@gmail.com

University of Zurich UZH

Numerical Methods,
21 September, 2016

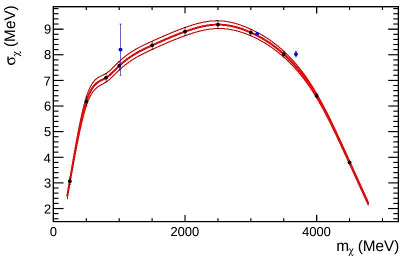
Interpolation - graphical interpretation

⇛ Let's assume we have a given function in a tabular form:

x_i	x_1	x_2	x_3	...	x_n
$f_i = f(x_i)$	f_1	f_2	f_3	...	f_n

⇛ The x_i are data points.

⇛ The problem is how to calculate the function values between the points (*interpolation*) or outside (*extrapolation* - see next lecture!) .



Interpolation - formal formulation

⇛ On the interval: $[a, b] \subset \mathbb{R}$ and set of points: $D = (x_i, y_i)$, $x_i \neq x_j \;\Leftrightarrow\; i \neq j$.
⇛ The interpolation is to find a function $F : [a, b] \to \mathbb{R}$ that satisfies:

$$\forall (x_i, y_i) \in D : \quad F(x_i) = y_i$$

⇛ The above equation is called the interpolation equation.
⇛ The interpolation error:

$$\epsilon(x) = f(x) - F(x),$$

where $f(x)$ is the function that is being interpolated.
⇛ Each interpolation method has different errors, each of it can be estimated.

Linear Interpolation

⇛ Let's say we have $n+1$ interpolation points: (y_i, x_i).
⇛ The linear interpolation is given with with the equation:

$$F(x) = \sum_{j=0}^{n} a_j \psi_j(x) = a_0\psi_0(x) + a_1\psi_1(x) + a_2\psi_2(x) + ... + a_n\psi_n(x),$$

where:

- a_j - parameter,
- $\psi_j(x)$ - base function.

⇛ Now the only thing one needs to do is to find the a_i coefficients. Now there are infinite number such solutions so we need to assume that the number interpolation points is equal to number of base functions.

Linear Interpolation

⇛ Now to find all the a_i one just needs to solve the linear equation system:

$$\begin{pmatrix} \psi_0(x_0) & \psi_1(x_0) & ... & \psi_n(x_0) \\ \psi_0(x_1) & \psi_1(x_1) & ... & \psi_n(x_1) \\ & ... & & \\ \psi_0(x_0) & \psi_1(x_0) & ... & \psi_n(x_2) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ ... \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ ... \\ y_n \end{pmatrix}$$

⇛ The only necessary condition to solve the equation is that the:

$$\det\left[\psi_i(x_j)\right] \neq 0$$

⇛ The polynomial of the order n has $n+1$ base functions, so system has only one solution.
⇛ We can choose the base functions on many different ways. This depends on the input points and our own judgement. Often use are trigonometric functions and polynomials.
⇛ The proper selection of the base function can hugely simplify the problem by making the matrix triangular or even diagonal.

Linear Interpolation, simplest base

Theorem:

If all of the interpolation points $x_0, x_1, ..., x_n$ are pair different there exists one and only one solution of the interpolation function of the order max. n.

The simplest base functions one can imagine to use is the natural base:

$$\psi_0 = 1, \quad \psi_1 = x, \quad \psi_2 = x^2, \quad ... \quad , \psi_n = x^n$$

Attention/Achtung:

This kind of interpolation is wrongly conditioned: small changes of input parameters will cause large output differences.

$\Rrightarrow$ Because of the above it is used almost not used in practice.

$\Rrightarrow$ For education purposes we will analyse it.

Linear Interpolation, simplest base

$\Rrightarrow$ When using natural base the interpolation polynomial will have the structure:

$$F(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 ... a_n x^n$$

$\Rrightarrow$ Now base matrix (so-called Vandermonde matrix) has the form:

$$V = \begin{pmatrix} 1 & x_0 & x_0^2 & ... & x_0^n \\ 1 & x_1 & x_1^2 & ... & x_1^n \\ & ... & & & \\ 1 & x_n & x_n^2 & ... & x_n^n \end{pmatrix}$$

$\Rrightarrow$ Now the only thing one needs to do is to solve:

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

$\Rrightarrow$ Because the natural base is wrongly conditioned the increase of the interpolation points significantly increases the condition parameter.

Lagrange interpolation

⇛ Now let's choose a better base function that are given by:

$$l_i(x) = \prod_{j=0,\ j\neq i}^{n} \frac{x - x_j}{x_i - x_j} = \frac{(x-x_0)(x-x_1)...(x-x_n)}{(x_i-x_0)(x_i-x_1)...(x_i-x_n)} \tag{1}$$

⇛ It is easy to notice:

$$l_i(x_j) = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

⇛ Now putting this into base matrix we get:

$$\begin{pmatrix} 1 & 0 & 0 & ... & 0 \\ 0 & 1 & 0 & ... & 0 \\ & ... & & & \\ 0 & 0 & 0 & ... & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ ... \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ ... \\ y_n \end{pmatrix}$$

⇛ Now this is something we like :) The interpolating function then takes the form:

$$F(x) = L_n(x) = \sum_{i=0}^{n} y_i l_i(x)$$

Lagrange interpolation - the algorithm

1. Get the input data $(x_i, y_i),\ i = 0, ..., n$.
2. The coefficients of the polynomial are y_i.
3. Calculate the base functions from Eq. 1.
4. Put everything together to get the L_n Lagrange polynomial.

Disadvantage:

The problem with this method is the fact that if you add another interpolation point you need to start over and recalculate everything.

Advantege

It is very popular and simple method. It is used in many different places: differential equations, numerical integrations, etc. Because of this the precision of the method depends on the precision of the interpolation is self:

$$\epsilon(x) = f(x) - F(x) = \frac{f^{(n+1}(\xi)}{(n+1)!}\omega_n(x)$$

Lagrange interpolation - the algorithm

1. Get the input data $(x_i, y_i),\ i = 0, ..., n$.
2. The coefficients of the polynomial are y_i.
3. Calculate the base functions from Eq. 1.
4. Put everything together to get the L_n Lagrange polynomial.

Disadvantage:

The problem with this method is the fact that if you add another interpolation point you need to start over and recalculate everything.

Advantege

where:
$\omega_n(x) = (x - x_0)(x - x_1)...(x - x_n)$
f function of class C^{n+1} on $[a, b]$
ξ average values.h

Newton interpolation

⇛ Sir Isaac Newton proposed a different base for the problem:

$$\begin{aligned}
p_0(x) &= 1\\
p_1(x) &= (x-x_0)\\
p_2(x) &= (x-x_0)(x-x_1)\\
&\ldots\\
p_n(x) &= (x-x_0)(x-x_1)\ldots(x-x_n)
\end{aligned}$$

⇛ Using the above functions the basis matrix takes a nice form:

$$V = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0\\ 1 & p_1(x_1) & 0 & \ldots & 0\\ & \ldots & & & \\ 1 & p_1(x_n) & p_2(x_n) & \ldots & p_n(x_n) \end{pmatrix}$$

Newton interpolation

⇛ Sir Isaac Newton proposed a different base for the problem:

$$
\begin{aligned}
p_0(x) &= 1\\
p_1(x) &= (x - x_0)\\
p_2(x) &= (x - x_0)(x - x_1)\\
&\dots\\
p_n(x) &= (x - x_0)(x - x_1)\dots(x - x_n)
\end{aligned}
$$

⇛ And the linear equation system:

$$
\begin{pmatrix}
1 & 0 & 0 & \dots & 0\\
1 & p_1(x_1) & 0 & \dots & 0\\
 & \dots & & & \\
1 & p_1(x_n) & p_2(x_n) & \dots & p_n(x_n)
\end{pmatrix}
\begin{pmatrix} a_0\\ a_1\\ \dots\\ a_n \end{pmatrix}
=
\begin{pmatrix} y_0\\ y_1\\ \dots\\ y_n \end{pmatrix}
$$

Newton interpolation

⇛ We can solve the aforementioned system by noticing that a_0 depends only on y_0, a_1 depends on y_0 and y_1, etc.
⇛ They can be calculated using difference quotient:

$$f\left[x_0, x_1\right] = \frac{y_1 - y_0}{x_1 - x_0}, \quad f\left[x_1, x_2\right] = \frac{y_2 - y_1}{x_2 - x_1}, \text{ etc.}$$

$$f\left[x_0, x_1, x_2\right] = \frac{f\left[x_1, x_2\right] - f\left[x_0, x_1\right]}{x_2 - x_1}$$

⇛ In general:

$$f\left[x_i, x_{i+1}, ..., x_{i+k}\right] = \frac{f\left[x_{i+1}, x_{i+2}, ..., x_{i+k}\right] - f\left[x_i, x_{i+1}, ..., x_{i+k-1}\right]}{x_{i+k} - x_i}$$

⇛ Using difference quotient we can calculate the coefficients:

$$a_i = f\left[x_0, x_1, ..., x_i\right]$$

Newton interpolation, the algorithm

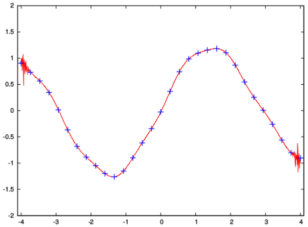
- We read the input interpolating points: (x_i, y_i), $i = 0, 1, ..., n$.
- Calculate the polynomial coefficients in an iterative way.
- Calculate the base functions.
- Put things together to get the Newton interpolation polynomial.

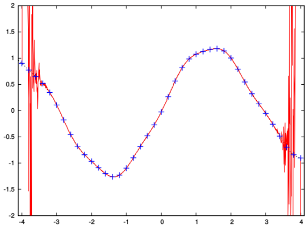
Why do we bother?

$\Longrightarrow$ Despite the procedure seems more complicated then the Lagrange interpolation it has a huge advantage: adding one more interpolation point does require to repeat the whole procedure, but we can reuse the previous steps.

Runge oscillation

$\Rrightarrow$ If one has many points one needs to use a high order polynomial.
$\Rrightarrow$ The problem with high order polynomials is that they start oscilationg at the edges $\mapsto$ so-called Runge oscillations.





$\Rrightarrow$ Danny will tell you more about this next lecture.

First degree spline function

⇛ Till now using couple numerical methods we were able to calculate coefficients of a big single polynomial equation.
⇛ Now we are smarter then Newton and Lagrange were over 300 years ago!
⇛ We can use several interpolating functions and then glue them (splice) them together!
⇛ In order to do so we divide the domain of the F function $[a, b]$ into:

$$a = x_0 < x_1 < x_2 < ... < x_n = b$$

⇛ The simplest is to use linear functions:

$$s(x) = \begin{cases} s_0(x) & x \in [x_0, x_1) \\ s_1(x) & x \in [x_1, x_2) \\ ... \\ s_{n-1}(x) & x \in [x_{n-1}, x_n) \end{cases},$$

where

$$s_k(x) = a_{k,0} + a_{k,1}(x - x_k) \quad x \in [x_k, x_{k+1})$$

First degree spline function

⇛ The above defined functions $s_k(x)$ need to obey the conditions:

$$s_k(x_k) = y_k$$
$$s_k(x_{k+1}) = s_{k+1}(x_{k+1})$$

⇛ From the above we can calculate the linear equation coefficients:

$$a_{k,0} = y_k$$
$$a_{k,1} = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

Advantage:

⇛ The biggest advantage of this method is it's simplicity. If you have a large number of interpolating points that are "densely " packed you can get very good and fast aproximation with this function.

Disadvantage:

⇛ The interpolating function is not smooth on the interpolation points.

Third degree spline function

⇛ As mentioned before the first degree spline function suffer from the continuity problem.
⇛ A solution would be to use the higher order polynomials to aproximate the function. For example the 3rd order:

$$s_k(x) = a_{k,0} + a_{k,1}(x - x_k) + a_{k,2}(x - x_k)^2 + a_{k,3}(x - x_k)^3, \quad x \in [x_k, x_{k+1})$$

⇛ Now this polynomial needs to obey conditions:

$$\begin{aligned} s_k(x_k) &= y_k \\ s_k(x_{k+1}) &= s_{k+1}(x_{k+1}) \\ s'_k(x_{k+1}) &= s'_{k+1}(x_{k+1}) \\ s''_k(x_{k+1}) &= s''_{k+1}(x_{k+1}) \end{aligned}$$

⇛ In total we will have $4n$ coefficients to find. But we will only have $4n - 2$ equations.
⇛ To solve the system we need to assume somethings about the endpoints (there are many options here).
⇛ The most popular ones are:

$$\begin{aligned} &s'(a) = f'(a) \quad s'(b) = f'(b) \quad \text{or} \\ &s''(a) = s''(b) = 0 \end{aligned}$$

Third degree spline function

⇛ Now to efficiently solve the system we will define temporary variables:

$$h_k = x_{k+1} - x_k$$
$$d_k = \frac{y_{k+1} - y_k}{h_k}$$
$$m_k = s''(x_k)$$

⇛ Now the second derivative can be interpolated using the Lagrange polynomial:

$$s''_k(x) = s''(x)\frac{x - x_{k+1}}{x_k - x_{k+1}} + s''(x)\frac{x - x_k}{x_{k+1} - x_k} = \frac{m_k}{h_k}(x_{k+1} - x) + \frac{m_{k+}}{h_k}(x - x_k)$$

⇛ Now if we integrate the above equation (two times):

$$s'_k(x) = -\frac{m_k}{2h_k}(x_{k+1} - x)^2 + \frac{m_{k+}}{2h_k}(x - x_k)^2 - p_k + q_k$$
$$s_k(x) = \frac{m_k}{6h_k}(x_{k+1} - x)^3 + \frac{m_{k+1}}{3h_k}(x - x_k)^3 + p_k(x_{k+1} - x) + q_k(x - x_k)$$

Third degree spline function

⇛ Now we need to calculate the $s_k(x)$ for x_k and x_{k+1} using relations: $s_k(x_k) = y_k$ and $s_k(x_k+1) = y_{k+1}$:

$$x_k(x_k) = y_k = \frac{m_k}{6h_k}(x_{k+1}-x_k)^3 + p_k(x_{k+1}-x_k)$$
$$x_k(x_{k+1}) = y_{k+1} = \frac{m_{k+1}}{3h_k}(x_{k+1}-x_k)^3 + q_k(x_{k+1}-x_k)$$

⇛ From which we get:

$$p_k = \frac{y_k}{h_k} - \frac{m_k h_k}{6}$$
$$p_{k+1} = \frac{y_{k+1}}{h_{k+1}} - \frac{m_{k+1} h_{k+1}}{6}$$

⇛ Putting all the things together:

$$S_k(x) = \frac{m_k}{6h_k}(x_k - x)^3 + \frac{m_{k+1}}{6h_k}(x-x_k)^3 + \left(\frac{y_k}{h_k} - \frac{m_k h_k}{6}\right)(x_{k+1}-x) \quad (2)$$
$$+ \left(\frac{y_{k+1}}{h_k} - \frac{m_{k+1} h_k}{6}\right)(x - x_k) \quad (3)$$

Third degree spline function

⇛ The only unknown in the above equation are the m_i variables. To get those we need to use the equation: $S'_{k-1}(x_k) = s'(x_k)$:

$$-\frac{1}{3}m_k h_k - \frac{1}{6}m_{k+1}h_k + d_k = \frac{1}{3}m_k h_{k-1} + \frac{1}{6}m_{k-1}h_{k-1} + d_{k-1}$$

⇛ So the complete solution is:

$$\begin{aligned} a_{k,0} &= y_k \\ a_{k,1} &= d_k - \frac{h_k}{6}(2m_k + m_{k+1}) \\ a_{k,2} &= \frac{m_k}{2} \\ a_{k,3} &= \frac{m_{k+1} - m_k}{6h_k} \end{aligned}$$

⇛ To get the m_0 and m_n one needs to use one of the aforementioned conditions.

Third degree spline function algorithm

- Get the input interpolation points.
- Calculate the temporary variables: d_i, h_i.
- Assume adequate conditions to get m_0, m_n.
- Solve linear equation system.
- Obtained values of the $a_{i,j}$ coefficients put in the interpolation equation.

Summary

$\Rrightarrow$ Interpolation playes essential role in almost all numerical methods!
$\Rrightarrow$ Even very old algorithms like Lagrange are still used(we used it in splines).
$\Rrightarrow$ There is a lot of algorithms on the market. From simple Lagrange and Newton interpolation algorithms up to modern splines.
$\Rrightarrow$ The most common used today are spline. They require a bit of work but they work very effectively.

Backup