Short notice on (exact) trigonometric interpolation: 1) detailed calculation 2) symmetric cut-off for even number of points 3) general approach to a_N/b_N cut-off for even number of points 4) SciLab code

Andrej Liptaj*

Abstract

Method of trigonometric interpolation is presented in details and summarized. New ideas related to the high-frequency cutoff in the case of an even number of data points are presented.

Introductory note: This text was previously published on Scribd[1].

1 Foreword

Given a set of experimental points (x_i, y_i), one wants to find a cut-off Fourier series (i.e. its coefficients a_n and b_n)

$$y = \frac{a_0}{2} + \sum_{n=1}^{N} \left[b_n \sin(nx) + a_n \cos(nx) \right] \tag{1}$$

that exactly passes through *all* points. This problem has been (of course) solved and the solution can be found on many different places (on internet). This notice covers some points related to the trigonometric interpolation. First, it seems to me, that existing descriptions I have found are not detailed enough, or somewhat messy (my personal opinion). So I provide here a detailed calculation leading to the a_n and b_n coefficients. The second point concerns an even number of point pairs (x_i, y_i). The formula 1 naturally contains an odd number of parameters: they come in pairs in the sum, plus the stand-alone a_0 coefficient. The number of free parameters should match the number of points, in the case of an even number of points one has to find an *ad hoc* approach to cut-off the series. Here I present the symmetric cut-off with $a_N = b_N = w$. Then I also discuss a generalization of this approach. Finally, I provide a SciLab code to perform the trigonometric (and polynomial) interpolation.

2 Detailed calculation

The condition that coefficients need to fulfill is

$$y_i = \frac{a_0}{2} + \sum_{n=1}^{N} \left[b_n \sin(nx_i) + a_n \cos(nx_i) \right].$$

This condition can be transformed as follows:

$$\begin{aligned} y_i &= \frac{a_0}{2} + \sum_{n=1}^{N} \left[b_n \sin(nx_i) + a_n \cos(nx_i) \right] \\ &= \frac{a_0}{2} + \sum_{n=1}^{N} \left[b_n \frac{e^{inx_i} - e^{-inx_i}}{2i} + a_n \frac{e^{inx_i} + e^{-inx_i}}{2} \right] \end{aligned}$$

*Institute of Physics, Bratislava, Slovak Academy of Sciences, andrej.liptaj@savba.sk

I am willing to publish any of my ideas presented through free-publishing services in a journal, if someone (an editor) judges them interesting enough. Journals in the "*Current Contents*" database are strongly preferred.

[1] *https://www.scribd.com/document/270904435/Short-notice-on-exact-trigonometric-interpolation*

$$
\begin{aligned}
&= \frac{a_0}{2} + \sum_{n=1}^{N} \left[\frac{b_n}{2i} e^{inx_i} - \frac{b_n}{2i} e^{-inx_i} + \frac{a_n}{2} e^{inx_i} + \frac{a_n}{2} e^{-inx_i} \right] \\
&= \frac{a_0}{2} + \sum_{n=1}^{N} \left[\left(\frac{b_n}{2i} + \frac{a_n}{2} \right) e^{inx_i} + \left(\frac{a_n}{2} - \frac{b_n}{2i} \right) e^{-inx_i} \right] \\
&= \frac{a_0}{2} + \sum_{n=1}^{N} \left[\frac{1}{2} \left(a_n - ib_n \right) e^{inx_i} + \frac{1}{2} \left(a_n + ib_n \right) e^{-inx_i} \right] \\
&= \frac{a_0}{2} + \frac{1}{2} \sum_{n=1}^{N} \left(a_n - ib_n \right) e^{inx_i} + \frac{1}{2} \sum_{n=1}^{N} \left(a_n + ib_n \right) e^{-inx_i} \\
&= \frac{a_0}{2} + \frac{1}{2} \sum_{n=1}^{N} \left(a_n - ib_n \right) e^{inx_i} + \frac{1}{2} \sum_{k=1}^{N} \left(a_k + ib_k \right) e^{-ikx_i} \\
& \quad r = -k \\
& \quad k = -r \\
&= \frac{a_0}{2} + \frac{1}{2} \sum_{n=1}^{N} \left(a_n - ib_n \right) e^{inx_i} + \frac{1}{2} \sum_{-r=1}^{N} \left(a_{-r} + ib_{-r} \right) e^{irx_i} \\
&= \frac{a_0}{2} + \frac{1}{2} \sum_{n=1}^{N} \left(a_n - ib_n \right) e^{inx_i} + \frac{1}{2} \sum_{r=-1}^{-N} \left(a_{-r} + ib_{-r} \right) e^{irx_i} \\
&= \frac{a_0}{2} + \frac{1}{2} \sum_{n=1}^{N} \left(a_n - ib_n \right) e^{inx_i} + \frac{1}{2} \sum_{n=-1}^{-N} \left(a_{-n} + ib_{-n} \right) e^{inx_i} \\
&= \frac{a_0}{2} + \frac{1}{2} \left[\sum_{n=1}^{N} e^{inx_i} \left(a_n - ib_n \right) + \sum_{n=-1}^{-N} e^{inx_i} \left(a_{-n} + ib_{-n} \right) \right] \\
&= \frac{1}{2} \left\{ a_0 + \left[\sum_{n=1}^{N} e^{inx_i} \left(a_n - ib_n \right) + \sum_{n=-1}^{-N} e^{inx_i} \left(a_{-n} + ib_{-n} \right) \right] \right\} \\
& \quad c_{n>0} = \frac{1}{2} \left(a_n - ib_n \right) \\
& \quad c_{n<0} = \frac{1}{2} \left(a_{-n} + ib_{-n} \right) = \bar{c}_{-n} \\
& \quad c_0 = \frac{1}{2} a_0 \epsilon \Re \\
&= \sum_{n=-N}^{N} c_n e^{inx_i} \\
&= \sum_{n=-N}^{N} c_n \left(e^{ix_i} \right)^n \\
& \quad z_i \equiv e^{ix_i} \\
y_i &= \sum_{n=-N}^{N} c_n z_i^n
\end{aligned}
$$

We have a power series, however with negative powers. Multiplying by z_i^N, one can get a polynomial:

$$
\begin{aligned}
y_i &= \sum_{n=-N}^{N} c_n z_i^n \\
z_i^N y_i &= z_i^N \sum_{n=-N}^{N} c_n z_i^n \\
& \quad Y_i \equiv z_i^N y_i \\
Y_i &= \sum_{n=-N}^{N} c_n z_i^{N+n}
\end{aligned}
$$

$$
\begin{aligned}
& & k &= n+N \\
& & n &= k-N \\
Y_i &= & \sum_{k=0}^{2N} & c_{k-N} z_i^{N+k-N} \\
& & \alpha_k &\equiv c_{k-N} \\
Y_i &= & \sum_{k=0}^{2N} & \alpha_k z_i^k \\
Y_i &= & \sum_{n=0}^{2N} & \alpha_n z_i^n
\end{aligned}
$$

Next, one uses some standard interpolation procedure for the polynomial interpolation, interpolating the (complex) points (z_i, Y_i) by a (complex) polynomial.

Let me repeat, the procedure with $2N+1$ experimental points is:

- Take the x_i values and compute $z_i \equiv e^{ix_i}$.
- Take the y_i values and compute $Y_i \equiv z_i^N y_i$.
- Compute interpolation-polynomial coefficients α_n for the points (z_i, Y_i).
- Rename the coefficients $c_{n-N} \equiv \alpha_n$, $c_k \equiv \alpha_{k+N}$
- Compute the original coefficients $a_n = 2Re\,(c_{-n}) = 2Re\,(c_n)\,,\ b_n = 2Im\,(c_{-n}) = -2Im\,(c_n)$. (Is b_0 *automatically* zero? Yes, it is!)
- Construct the function $\frac{a_0}{2} + \sum_{n=1}^{N} \left[b_n \sin(nx_i) + a_n \cos(nx_i)\right]$ (do not forget to divide by 2 in the case of a_0).

3 Even number of data points: symmetric cut-off

In this case an *ad hoc* choice must be made to reconciliate the number of parameters with the number of data points:

- Cut the highest-frequency cosine?
- Cut the highest-frequency sine?
- Cut a_0?

I want to cut high-frequencies, so I do not want to cut the a_0, I want to be symmetric, so I introduce the term

$$w \sin(Nx_i) + w \cos(Nx_i)\,.$$

With this choice the things become little bit more complex. The number of data points is even, let me note it $2N$. In order to solve the coefficients, I introduce a new "data" point (x_{2N+1}, y_{2N+1}) and I will place it correctly so that $w = a_N = b_N$. With this point the situation looks similar to the "odd" case:

$$y_i = \frac{a_0}{2} + \sum_{n=1}^{N} \left[b_n \sin(nx_i) + a_n \cos(nx_i)\right]$$

and corresponds to $2N+1$ data points. The condition $w = a_N = b_N$ implies:

$$
\begin{aligned}
w \sin(Nx_i) + w \cos(Nx_i) &= \frac{1}{2}(w - iw)\, e^{iNx_i} + \frac{1}{2}(w + iw)\, e^{-iNx_i} \\
& \quad M = -N \\
& \quad N = -M \\
&= \frac{1}{2}(w - iw)\, e^{iNx_i} + \frac{1}{2}(w + iw)\, e^{iMx_i} \\
&= \frac{1}{2}(w + iw)\, e^{i(-|N|)x_i} + \frac{1}{2}(w - iw)\, e^{i|N|x_i}
\end{aligned}
$$

So that the polynomial looks like:

$$y_i = \sum_{n=-N}^{N} c_n z_i^n$$

where

$$\begin{aligned}
c_{N>n>0} &= \frac{1}{2}(a_n - ib_n) \\
c_{-N<n<0} &= \frac{1}{2}(a_{-n} + ib_{-n}) = \bar{c}_{-n} \\
c_{-N} &= \frac{1}{2}(w + iw) \\
c_N &= \frac{1}{2}(w - iw) = \bar{c}_{-N} \\
c_0 &= \frac{1}{2} a_0 \epsilon \Re
\end{aligned}$$

After renaming the indices (as in the "odd" case), we want to get to the polynomial interpolation

$$Y_i = \sum_{n=0}^{2N} \alpha_n z_i^n.$$

Now comes the reasoning: Imagine we have already the interpolation polynomial P for $2N$ points. We want to construct a new polynomial Π that keeps the interpolation property of P and, in addition, goes through the point (x_{2N+1}, y_{2N+1}). Π can be written as:

$$\Pi = P + K(x - x_1)(x - x_2)\cdots(x - x_{2N})$$

where the constant K is responsible for fitting the (x_{2N+1}, y_{2N+1}) point. How do the x^0 and x^{2N+1} terms of Π look like? They look like (p_0 comes from P):

$$\begin{aligned}
\Pi &= [p_0 \pm K x_1 x_2 \cdots x_{2N}] + \cdots + [K] x^{2N} \\
& q \equiv \pm x_1 x_2 \cdots x_{2N} \\
& [p_0 + qK] + \cdots + [K] x^{2N}
\end{aligned}$$

Coefficients of these two terms are just shifted coefficients c_{-N} and c_N, so one requires:

$$\begin{aligned}
K &= \frac{1}{2}(w - iw) \\
&\Rightarrow \\
p_0 + q\frac{1}{2}(w - iw) &= \frac{1}{2}(w + iw) \\
&\Leftrightarrow \\
2p_0 + q(w - iw) &= w + iw \\
&\Leftrightarrow \\
2p_0 + qw - iqw &= w + iw \\
&\Leftrightarrow \\
2p_0 &= w + iw - qw + iqw \\
&\Leftrightarrow \\
2p_0 &= (1 + i - q + iq) w \\
&\Leftrightarrow \\
w &= 2\frac{p_0}{1 + i - q + iq}
\end{aligned}$$

This needs to be a real number.

Let me repeat the procedure for case of $2N$ points:

- Take the x_i values and compute $z_i \equiv e^{ix_i}$.
- Take the y_i values and compute $Y_i \equiv z_i^N y_i$.
- Determine the interpolation polynomial P for the points (z_i, Y_i).

- Extract the absolute term p_0 of P. Compute $q = (-x_1) \times (-x_2) \times \ldots \times (-x_{2N})$. Then compute $w = \frac{2p_0}{1+i-q+iq}$ and $K = \frac{1}{2}(w - iw)$.
- Construct a new polynomial $\Pi = P + K(x - x_1)(x - x_2)\cdots(x - x_{2N})$.
- Treat the polynomial Π as the interpolation polynomial for $(2N + 1$ points) from the previous ("odd") case. From its coefficients (α_n) compute a_n and b_n. Automatically one gets $a_N = b_N = w$.

4 Even number of data points: general a_N/b_N cut-off

The approach from the previous paragraph can be straightforwardly generalized to any a_N/b_N cut-off. The algorithm is as follows:

- Formulate the constraints on the coefficients a_N and b_N.
- Find the corresponding constraints on c_N and c_{-N}.
- Build polynomial P. Find the appropriate equation which follows from the constraints, solve it, and build Π.
- Take the coefficients α_n of Π and treat them same as in the "odd" case, compute from them the coefficients a_n and b_n.

I give here the example of the highest-sine cut-off.

- The condition reads: $b_N = 0$.
- The coefficients $c_{\pm N}$ then look like $c_{\pm N} = \frac{a_N}{2}$.
- One has immediately:

$$\begin{aligned} K &= \frac{a_N}{2} \\ &\Rightarrow \\ p_0 + q\frac{a_N}{2} &= \frac{a_N}{2} \\ &\Leftrightarrow \\ p_0 &= \frac{a_N}{2} - q\frac{a_N}{2} \\ &\Leftrightarrow \\ p_0 &= \frac{a_N}{2}(1 - q) \\ &\Leftrightarrow \\ a_N &= \frac{2p_0}{(1 - q)} \\ &\Leftrightarrow \\ K &= \frac{p_0}{(1 - q)} \end{aligned}$$

- Polynomial $\Pi = P + K(x - x_1)(x - x_2)\cdots(x - x_{2N})$ can be built and its coefficients read-out.

5 Efficient approach for non-minimal series

An efficient alternative approach is achieved by interpolating the data with a complex exponential sum

$$y_i = \sum_{n=0}^{N} q_n e^{inx_i}.$$

One proceeds analogically

$$\begin{aligned} y_i &= \sum_{n=0}^{N} q_n \left(e^{ix_i}\right)^n \\ & z_i = e^{ix_i} \\ &= \sum_{n=0}^{N} q_n z_i^n. \end{aligned}$$

One constructs the interpolation polynomial for (y_i, z_i) and finds the coefficients q_n. However q_n can be complex. If y_i is real, then $\sum_{n=0}^{N} q_n z_i^n$ needs also to be real. Because a real part of a sum is sum of real parts, one can focus on an individual term

$$\begin{aligned} q_n e^{inx_i} &= (\alpha_n + i\beta_n)(\cos nx + i \sin nx) \\ &= \alpha_n \cos nx + i\alpha_n \sin nx + i\beta_n \cos nx - \beta_n \sin nx \\ &= [\mathrm{Re}\,(q_n) \cos nx - \mathrm{Im}\,(q_n) \sin nx] + i\,[\mathrm{Re}\,(q_n) \sin nx + \mathrm{Im}\,(q_n) \cos nx]\,. \end{aligned}$$

The second bracket is vanishing for purely real y_i, thus the trigonometric interpolation is fully given by the first one. One immediately gets the coefficients

$$\begin{aligned} a_n &= \mathrm{Re}\,(q_n)\,, \\ b_n &= -\mathrm{Im}\,(q_n)\,, \end{aligned}$$

where the coefficient a_0 is *not* divided by two

$$y_i = a_0 + \sum_{n=1}^{N} [b_n \sin(nx_i) + a_n \cos(nx_i)]$$

and the coefficient b_0 is irrelevant. In this approach, however, the series is (approximately) twice as long as in the "standard" approach ($2N-1$ terms in total for N experimental points). But still, it is a valid Fourier series.

6 SciLab code

Here is a SciLab code for the polynomial (works also for complex data points) and the trigonometric interpolation (works for real-valued data points only) passing (in both cases) through *all* data points ("standard" approach):

```
cutOffType = 0 //symmetric
//cutOffType = 1 //high-sine

// INTERPOLATING POLYNOMIAL
function [f] = interpolation_poly(x,y)
    nData = length(x)

    atom(1) = poly(1,"x","coeff")

    for i=2:nData
        atom(i) = atom(i-1)*poly(x(i-1),"x")
    end

    poly_interpol = poly(0,"x","coeff")
    for i=1:nData
        const = ( y(i) - horner(poly_interpol,x(i)) )/horner(atom(i),x(i))
        poly_interpol = poly_interpol + const*atom(i)
    end

    f = poly_interpol
endfunction

// INTERPOLATING FOURIER SERIES

// COMPUTING COEFFICIENTS
function [a,b,N] = trigo_cfs(x,y)
    nDat = length(x)

    if modulo(nDat,2)==1 then
        N = round((nDat-1)/2) //If nDat IS ODD
    else
        N = round((nDat)/2) //If nDat IS EVEN
```

```
    end

    for i=1:nDat
        z(i)= exp( %i*x(i))
        Y(i) = (z(i))^N*y(i)
    end

    z_polynomial = interpolation_poly(z,Y)

    // SPECIAL TRATMENT IN THE CASE OF EVEN NUMBER OF POINTS
    if modulo(nDat,2)==0 then

        p0 = coeff(z_polynomial,0)
        q = 1
        for i=1:nDat
            q = q*(-z(i))
        end

        if cutOffType==0 then //cutting symmetrically
            w = 2*p0/(1 + %i - q + %i*q)
            K = (1/2)*(w - %i*w)
        end

        if cutOffType==1 then // cutting high-frequency sine
            K = p0/(1-q)
        end

        polyToAdd = K*poly(z,"x")
        z_polynomial = z_polynomial + polyToAdd
    end

    cfs = coeff(z_polynomial)

    for i=-N:N
        if i<0 then
            continue
        end
        coef = cfs(i+N+1)
        a(i+1)=2*real(coef)
        b(i+1)=-2*imag(coef)
        //mprintf("a%i = %f, b%i = %f \n", i+1,a(i+1),i+1,b(i+1))
    end
endfunction

// COMPUTING TRIGONOMETRIC INTERPOLATION
function [f] = interpolation_trigo(x,a,b,N)
    f = a(1)/2
    for i=1:N
        f = f + b(i+1)*sin(i*x) + a(i+1)*cos(i*x)
    end
endfunction

// START OF THE PROGRAM FLOW

dataSet = read("trigoData.dat",-1,2)
nDat = length(dataSet)/2

for i=1:nDat
    x(i) = dataSet(i,1)
```

```
    y(i) = dataSet(i,2)
end

[a_cfs,b_cfs,N] = trigo_cfs(x,y)

delta = x(nDat)-x(1)
margin = delta/10
x_ax = [x(1)-margin:0.01:x(nDat)+margin]
plot(x_ax,interpolation_trigo(x_ax,a_cfs,b_cfs,N))
plot(x,y,'*')
```