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 Scribd¹.

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

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

This condition can be transformed as follows:

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

*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.

We have a power series, however with negative powers. Multiplying by $z^y = e^{ix}$, one can get a polynomial:

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

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

$$Y_i \equiv z_i^N y_i$$

$$Y_i = \sum_{n=-N}^{N} c_n z_i^{N+n}$$
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 \(c_n\) for the points \((z_i, Y_i)\).
- Rename the coefficients \(a_n \equiv c_n, c_k \equiv a_{k+N}\)
- Compute the original coefficients \(a_n = 2\text{Re}(c_n)\), \(b_n = 2\text{Im}(c_n)\). (Is \(b_0\) automatically zero? Yes, it is!)
- Construct the function \(\frac{a_0}{2} + \sum_{n=1}^{N} [b_n \sin (nx_i) + a_n \cos (nx_i)]\) (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 \textit{ad hoc} choice must be made to reconcile 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:

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

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

\[w \sin (Nx_i) + w \cos (Nx_i) = \frac{1}{2} (w - iw) e^{iNx_i} + \frac{1}{2} (w + iw) e^{-iNx_i}\]

\[M = -N\]

\[N = -M\]

\[= \frac{1}{2} (w - iw) e^{iNx_i} + \frac{1}{2} (w + iw) e^{iMx_i}\]

\[= \frac{1}{2} (w + iw) e^{i(-|N|)x_i} + \frac{1}{2} (w - iw) e^{i|N|x_i}\]
So that the polynomial looks like:

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

where

\[ c_{N>n>0} = \frac{1}{2} (a_n - ib_n) \]
\[ c_{-N<n<0} = \frac{1}{2} (a_n + ib_n) = \bar{c}_n \]
\[ c_{-N} = \frac{1}{2} (w + iw) \]
\[ c_N = \frac{1}{2} (w - iw) = \bar{c}_N \]
\[ c_0 = \frac{1}{2} a_0 \epsilon \mathbb{R} \]

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 \( \Pi \) that keeps the interpolation property of \( P \) and, in addition, goes through the point \( (x_{2N+1}, y_{2N+1}) \). \( \Pi \) 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 \( \Pi \) look like? They look like \( (p_0 \) comes from \( P \)):

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

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

\[ K = \frac{1}{2} (w - iw) \]
\[ p_0 + q \frac{1}{2} (w - iw) = \frac{1}{2} (w + iw) \]
\[ 2p_0 + q (w - iw) = w + iw \]
\[ 2p_0 + qw - iqw = w + iw \]
\[ 2p_0 = w + iw - qw + iqw \]
\[ 2p_0 = (1 + i - q + iq) w \]
\[ w = \frac{2}{1 + i - q + iq} \]

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 \( w = \frac{2p_0}{1+iq+q} \) 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 \( \Pi \) as the interpolation polynomial for \((2N + 1)\) points from the previous (“odd”) case. From its coefficients \((\alpha_n)\) 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 \(\Pi\).

• Take the coefficients \(\alpha_n\) of \(\Pi\) 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 \(c_{\pm N} = \frac{a_{\pm N}}{2} \).

• One has immediately:

\[
\begin{align*}
K &= \frac{a_N}{2} \\
p_0 + q \frac{a_N}{2} &= \frac{a_N}{2} \\
p_0 &= \frac{a_N}{2} - q \frac{a_N}{2} \\
p_0 &= \frac{a_N}{2} (1 - q) \\
a_N &= \frac{2p_0}{(1 - q)} \\
K &= \frac{p_0}{(1 - q)}
\end{align*}
\]

• 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

\[
y_i = \sum_{n=0}^{N} q_n e^{imx_i}.
\]

One proceeds analogically

\[
y_i = \sum_{n=0}^{N} q_n (e^{ix_i})^n = \sum_{n=0}^{N} q_n z_i^n.
\]
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

\[
q_n e^{inx_i} = (\alpha_n + i\beta_n) (\cos nx + i \sin nx) = \alpha_n \cos nx + i\alpha_n \sin nx + i\beta_n \cos nx - \beta_n \sin nx = [\text{Re}(q_n) \cos nx - \text{Im}(q_n) \sin nx] + i[\text{Re}(q_n) \sin nx + \text{Im}(q_n) \cos nx].
\]

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

\[
a_n = \text{Re}(q_n), \quad b_n = -\text{Im}(q_n),
\]

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):

```sciLab
//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
definterpolation_poly
endfunction

// INTERPOLATING FOURIER SERIES

// COMPUTING COEFFICIENTS
function [a,b,N] = trigcfs(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
```
\begin{verbatim}
end

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

z_polynomial = interpolation_poly(z,Y)

// SPECIAL TREATMENT 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)
end
\end{verbatim}
\[ y(i) = \text{dataSet}(i, 2) \]

end

\[ [a_{cfs}, b_{cfs}, N] = \text{trigo} \_cfs(x, y) \]

\( \delta = x(nDat) - x(1) \)

margin = \( \delta / 10 \)

\( x_{ax} = [x(1) - \text{margin}: 0.01:nDat + \text{margin}] \)

plot(x_{ax}, \text{interpolation} \_\text{trigo}(x_{ax}, a_{cfs}, b_{cfs}, N))

plot(x, y, 's')