Progressive Fourier (or trigonometric) interpolation

Andrej Liptaj[*]

April 12, 2017

Abstract

Method of progressive trigonometric interpolation is presented.

1 Introduction

I did not find a dedicated text about progressive polynomial interpolation, yet such interpolation method is know and mentioned in several texts. The task is simple: imagine one has a polynomial $P_N(x)$ which goes through N points $(x_i, y_i)_{i=1}^N$ and one wants to extend the polynomial description to an additional $(N+1)$-th point. The procedure is straightforward: First, one constructs a polynomial $\widetilde{p}_N(x)$ which vanishes for all $(x_i, y_i)_{i=1}^N$:

$$\widetilde{p}_N(x) = \alpha \prod_{i=1}^{N} (x - x_i),$$

where one is free to choose the value of α. Then, one writes down the new, extended interpolation polynomial

$$P_{N+1}(x) = P_N(x) + \widetilde{p}_N(x)$$

and tunes the value of α so as to describe the additional point (x_{N+1}, y_{N+1}). Such extension, of course (by construction), does not spoil the existing description of $(x_i, y_i)_{i=1}^N$ points. If one starts with one single point and the corresponding polynomial $P_1(x) = y_1$, one can apply the procedure repeatedly to get an interpolation polynomial describing all points of interest.

The method may have some computational advantages (mainly smaller computer memory consumption), several comments about it can be found in the (non-dedicated) text [1].

One may naturally wonder whether the "progressive approach" can be extended to other, non-polynomial interpolations. The Fourier series represent other important function form of data description and, in this text, I provide a complete algorithm for progressive Fourier interpolation together with the corresponding *SciLab* program.

2 Preparatory considerations

2.1 Nomenclature

To proceed, let me first define some terms I will be using in this text. By "degree" of a Fourier series I will refer to "n" of the highest frequency (non-vanishing) term present in the Fourier series

$$f(x) = \sum_{n=0}^{N} \left[a_n \cos(nx) + b_n \sin(nx) \right], \tag{1}$$

independently on whether it is sine or cosine term. So, for example, the degree of

$$2 + 3\cos(x) - \sin(x) + 7\sin(5x)$$

is 5 and the degree of

$$-1 - 2\cos(2x) - \sin(2x)$$

[*]Institute of Physics, Bratislava, Slovak Academy of Sciences, andrej.liptaj@savba.sk
I am willing to publish any of my ideas presented here in a journal, if someone (an editor) judges them interesting enough. Journals in the "*Current Contents*" database are strongly preferred.

is 2.

Note please, that I will be using a non-standard form of the Fourier series (1), where I include the a_0 coefficient into the cosine part, while keeping, in the sine part, the (arbitrary) coefficient b_0 for formal reasons that will become clear later.

I will further use the term "zero-crossing function" (ZCF) to point to the Fourier series which vanishes for certain set of points $(x_i, y_i)_{i=1}^{N}$ and should be regarded as an analogy of the $\widetilde{p}_N(x)$ polynomial. In equations I will write $zcf_N(x)$ and from what was said one has

$$zcf_N(x) = 0 \text{ for all } x \epsilon \{x_i\}_{i=1}^{N}.$$

The Fourier series describing the $(x_i, y_i)_{i=1}^{N}$ points will be noted Fs_N:

$$Fs_N(x_i) = y_i, \quad 1 \leq i \leq N.$$

I will also use "automatic" abbreviation system for numerical constants

$$\begin{aligned} s_i &\equiv \sin(x_i), \\ c_i &\equiv \cos(x_i), \\ t_i &\equiv \tan(x_i). \end{aligned}$$

2.2 What is tricky

The analogy with progressive polynomial interpolation is not really straightforward. For example, one might have the idea to construct the ZCF in the following way

$$zcf_N(x) = \alpha \prod_{i=1}^{N} \sin(x - x_i) \tag{2}$$

and use the formula

$$\sin(x - x_i) = s_i \cos(x) + c_i \sin(x).$$

Then, if the product $\prod_{n=1}^{N}$ is expanded, one ends up with terms of the type

$$z_j [\cos(x)]^{w_1} [\sin(x)]^{w_2},$$

where one can make use of the "trigonometric power formulas"

$$[\sin(x)]^{w_1} = \sum_k \left\{ q_k^{[s,s]} \sin(kx) + q_k^{[s,c]} \cos(kx) \right\},$$

$$[\cos(x)]^{w_2} = \sum_k \left\{ q_k^{[c,s]} \sin(kx) + q_k^{[c,c]} \cos(kx) \right\}.$$

The explicit form of the coefficients $q_k^{[a,b]}$ can be found, for example, of the web page [4]. The multiplication of the two series leads to three possible combinations of trigonometric functions and each of them can be treated in such way as to obtain a simple sum:

$$\begin{aligned} \sin(mx)\cos(nx) &= \frac{1}{2}\{\sin[(m-n)x] + \sin[(m+n)x]\}, \\ \sin(mx)\sin(nx) &= \frac{1}{2}\{\cos[(m-n)x] - \cos[(m+n)x]\}, \\ \cos(mx)\cos(nx) &= \frac{1}{2}\{\cos[(m-n)x] + \cos[(m+n)x]\}. \end{aligned}$$

One can so finally transform the $zcf_N(x)$ into the "standard Fourier" form and add it (term by term) to the existing description of previous points

$$Fs_{N+1}(x) = Fs_N(x) + zcf_N(x),$$

tuning, of course, α such as to describe the point (x_{N+1}, y_{N+1}). After the torturous transformations one is however not satisfied with the result: one obtains a valid Fourier series but the series in not of the minimal degree. Indeed, each multiplication by $\sin(x - x_i)$ increases the degree by one, which is too much. The "minimal" form is, of course, something what one naturally asks for.

So what should be the correct degree of the final Fourier series? As I pointed out in my previous texts [3, 2] (and is widely known anyway), the Fourier series naturally describes an odd number of points: the coefficients a_i and b_i come in pairs with the exception of $i = 0$ where the coefficient b_0 is arbitrary (but a_0 is not). So, if one has an odd number $2M + 1$ of points to describe then the appropriate degree is M. If the number of points is even, $2M$, then the right degree is (also) M with however *cutoff freedom*: one can reach the (highest) degree by a single cosine term, by a single sine term or by an appropriate combination of both. In other words, one has a common constraint on the two coefficients a_N and b_N: when changing one of them, one can satisfy the constraint by accordingly modifying the other.

2.3 Two zeros with cosine expression

Producing the ZCF using $\sin(x - x_i)$ terms needs to be modified. In this short paragraph my aim is to write down a trigonometric term (useful later), which becomes zero for *two* points, x_i and x_{i+1}. One possibility (which I will adopt from now on) is

$$\cos(x + \varphi) + K$$

with

$$\varphi = -\frac{x_i + x_{i+1}}{2}$$

and

$$K = -\cos(x_i + \varphi).$$

I will use the expanded form of this expression

$$\begin{aligned} \cos(x + \varphi) + K &= \cos(\varphi)\cos(x) - \sin(\varphi)\sin(x) + K \\ &= A\cos(x) + B\sin(x) + K \end{aligned}$$

with

$$\begin{aligned} A &= \cos(\varphi), \\ B &= -\sin(\varphi). \end{aligned}$$

For the rest of this text I will fix the meaning of the three capital letters A, B and K *always* to this context, i.e. A means the numerical factor in front of the cosine, B the one in front of the sine and K is the absolute (x independent) term. The same notation applied to sine terms gives

$$\begin{aligned} \sin(x - x_i) &= \sin(x + \varphi) \\ &= \cos(\varphi)\sin(x) + \sin(\varphi)\cos(x) \\ &= \sin(\varphi)\cos(x) + \cos(\varphi)\sin(x) \end{aligned}$$

thus

$$\begin{aligned} \varphi &= -x_i, \\ A &= \sin(\varphi), \\ B &= \cos(\varphi), \\ K &= 0. \end{aligned}$$

2.4 Multiplying Fourier series with $A\cos(x) + B\sin(x) + K$

Multiplication of a Fourier series by the $A\cos(x) + B\sin(x) + K$ expression is an important step in the progressive Fourier interpolation and deserves a dedicated paragraph. First of all, the Fourier series should be understood as two sequences of coefficients $a_0 \cdots a_N$ and $b_0 \cdots b_N$ and the procedure of the intended multiplication should be seen as an operation on these sequences.

Let me start by the most simple: multiplication by K is trivial and leads to

$$\begin{aligned} a_i &\to K a_i, \\ b_i &\to K b_i. \end{aligned}$$

Next, one can study the remaining possibilities:

$$\begin{aligned} a_n \cos(nx) \times A\cos(x) &= \frac{a_n A}{2}\{\cos[(n-1)x] + \cos[(n+1)x]\} \\ b_n \sin(nx) \times A\cos(x) &= \frac{b_n A}{2}\{\sin[(n+1)x] + \sin[(n-1)x]\} \\ a_n \cos(nx) \times B\sin(x) &= \frac{a_n B}{2}\{\sin[(n+1)x] - \sin[(n-1)x]\} \\ b_n \sin(nx) \times B\sin(x) &= \frac{b_n B}{2}\{\cos[(n-1)x] - \cos[(n+1)x]\} \end{aligned}$$

So what is the algorithm to generate the new "after-multiplication" coefficients? Clearly, each term has two "offsprings": one in the lower $(n-1)$ and one in the higher $(n+1)$ frequency. Reciprocally it means, that each new term has two "parents", one from the lower and one from the higher frequency terms (plus the one of the same frequency resulting from the multiplication by K). After little bit of thinking one can write:

$$a_i^{new} = \frac{A}{2}\left(a_{i-1} + a_{i+1}\right) + \frac{B}{2}\left(-b_{i-1} + b_{i+1}\right) + Ka_i \tag{3}$$

and

$$b_i^{new} = \frac{B}{2}\left(a_{i-1} - a_{i+1}\right) + \frac{A}{2}\left(b_{i-1} + b_{i+1}\right) + Kb_i. \tag{4}$$

These two equations should be seen as a recipe for a computer implementation of such a multiplication[1].

The two expressions deserve, however, additional attention. Clearly, the multiplication leads to a broadening of the series, it grows to higher frequencies, but also to lower and negative frequencies! The last behavior needs to be treated specifically and explains why it is important to consistently keep in memory the value of the "artificial" b_0 coefficient. The treatment is simple: once a negative frequency term is obtained, it can be turned into a positive frequency one:

$$\begin{aligned}\cos(-x) &= \cos(x)\\ \sin(-x) &= -\sin(x)\end{aligned}$$

which leads to a dedicated treatment of the $i = 1$ coefficient :

$$\begin{aligned}a_1^{new} &= \frac{A}{2}\left(a_0 + a_2\right) + \frac{B}{2}\left(-b_0 + b_2\right) + Ka_1 + \frac{A}{2}a_0 + \frac{B}{2}b_0,\\ b_1^{new} &= \frac{B}{2}\left(a_0 - a_2\right) + \frac{A}{2}\left(b_0 + b_2\right) + Kb_1 + \frac{B}{2}a_0 - \frac{A}{2}b_0,\end{aligned}$$

which one must not forget to implement.

3 Progressive Fourier interpolation

In this paragraph I provide an algorithm for the progressive Fourier interpolation of N points. It is, naturally, a recursive algorithm with some complexity arising from odd/even number of points and the cutoff freedom. I believe it is more clear to use separate sections to describe it.

3.1 Describing first point $i = 1$

The recursive approach is initiated by describing a single point (the "first" one, $i = 1$) using the absolute term

$$Fs_1(x) = a_0 = y_1.$$

The coefficient b_0 is arbitrary (I initiate it in my program to zero, see Appendix) and all other coefficients are considered to be zero. The ZCF can be also represented by its cosine c_i^{zcf} and sine s_i^{zcf} coefficients (because it is Fourier series) and is initiated to

$$\begin{aligned}c_0^{zcf} &= 1,\\ (s_0^{zcf} &= 0).\end{aligned}$$

3.2 Describing middle points $1 < i < N$

The steps to take when describing the i-th point depends on the parity of the numbers of points previously described.

3.2.1 Existing Fourier series describes an odd number of points (i is even)

In this case multiply the existing ZCF by $\sin(x - x_{i-i})$:

$$\widehat{zcf_{i-1}}(x) = zcf_{i-2}(x) \times \sin(x - x_{i-1})$$

[1] The "overflowing" or "underflowing" indices (non-existing a_{i+1}, b_{i+1} for $i = N$ and non-existing a_{i-1}, b_{i-1} for $i = 0$) should be, of course, considered as pointing to zero-valued coefficients.

and for that purpose use the "coefficient" representation of the ZCF, the "ABK" representation of the sine term and the approach described in the section 2.4. Once completed, determine the value of the α_i coefficient so as to fit the point (x_i, y_i):

$$\alpha_i = \frac{y_i - Fs_{i-1}(x_i)}{\widehat{zcf_{i-1}}(x_i)}.$$

With α_i known, compute the new Fourier series coefficients $\{a_n\}_{n=0}^{n=i/2}$, $\{b_n\}_{n=0}^{n=i/2}$ using a term-by-term addition of the coefficient-represented functions:

$$Fs_i(x) = Fs_{i-1}(x) + \alpha_i \widehat{zcf_{i-1}}(x).$$

Finally, forget the function $\widehat{zcf_{i-1}}(x)$.

3.2.2 Existing Fourier series describes an even number of points (i is odd)

In this case multiply the *last remembered* ZCF, i.e. $zcf_{i-3}(x)$ by $[\cos(x+\varphi)+K]$:

$$zcf_{i-1}(x) = zcf_{i-3}(x) \times [\cos(x+\varphi)+K],$$

where

$$\varphi = -\frac{x_{i-2}+x_{i-1}}{2}$$

and

$$K = -\cos(x_{i-1}+\varphi).$$

For that purpose use the "coefficient" representation of the ZCF, the "ABK" representation the cosine term and the approach described in the section 2.4. Once completed, determine the value of the α_i coefficient so as to fit the point (x_i, y_i):

$$\alpha_i = \frac{y_i - Fs_{i-1}(x_i)}{zcf_{i-1}(x_i)}.$$

With α_i known, compute the new Fourier series coefficients $\{a_n\}_{n=0}^{n=(i-1)/2}$, $\{b_n\}_{n=0}^{n=(i-1)/2}$ using a term-by-term addition of the coefficient-represented functions:

$$Fs_i(x) = Fs_{i-1}(x) + \alpha_i zcf_{i-1}(x).$$

Remember the function $zcf_{i-1}(x)$.

One can now see why the degree of the Fourier series remains minimal: when multiplying by $\sin(x-x_i)$ the degree rises (previous section), but the multiplication by the cosine term does not increase the degree. This is because the expression which is multiplied (i.e. the previously remembered ZCF) is *the same* as for the sine multiplication (we take the ZCF from two steps back), it is not of a higher degree. Its multiplication by the cosine term produces the same degree as its multiplication by the sine term. So, combining the "sine" with "cosine" multiplication we are able to describe *two* points by rising the degree only by *one* (which, of course, means *two* coefficients).

3.3 Describing the last point $i = N$

3.3.1 The total number of points is odd (i, N are odd)

In this case one should follow the prescription from the section 3.2.2.

3.3.2 The total number of points is even (i, N are even)

If one does not care about the high-frequency cutoff, one can use the procedure from the section 3.2.1. I want, however, present an approach where a cutoff strategy is implemented. In this case one should recall the formulas (3) and (4), and see how the highest-frequency sine and cosine terms are created. Clearly, they get no contributions from the previous same frequency terms, neither from the previous higher frequency terms. The later is obvious, the last version of the ZCF was of a not higher degree then the one we are going to get, so it does not contain higher frequency terms. Less obvious are the same frequency terms: they do not contribute because, when i is even, then the degree of the new ZCF is *always* higher then the degree of the previous ZCF. In other words: the last ZCF has no terms with the frequency corresponding to the highest frequency term in the new ZCF. So the formulas look like

$$a_i^{new} = \frac{1}{2}(Aa_{i-1} - Bb_{i-1}),$$
$$b_i^{new} = \frac{1}{2}(Ba_{i-1} + Ab_{i-1}).$$

To have enough parametric freedom to profit from these expressions one cannot use the standard multiplication by a single sine term. Rather, one has to use a multiplication by a cosine term

$$\cos(x+\varphi)+K=\cos(\varphi)\cos(x)-\sin(\varphi)\sin(x)+K$$

and, when creating the "new" ZCF, *tune* the values of φ and K so as to

- make vanish the (new) ZCF at x_{i-1} and
- reach the desired cutoff.

One has

$$\begin{aligned}A&=\cos(\varphi)\\B&=-\sin(\varphi)\end{aligned}$$

Cutting away the highest-frequency sine (cosine remains)
One requires

$$\begin{aligned}b_i^{new}&=0\\Ba_{i-1}+Ab_{i-1}&=0\\\cos(\varphi)\,b_{i-1}-\sin(\varphi)\,a_{i-1}&=0\\b_{i-1}-\tan(\varphi)\,a_{i-1}&=0\\\tan(\varphi)&=\frac{b_{i-1}}{a_{i-1}}\\\varphi&=\arctan\left(\frac{b_{i-1}}{a_{i-1}}\right)\end{aligned}$$

where one needs to use the highest-frequency coefficients (a_{i-1} and b_{i-1}) from the previous ZCF.

Cutting away the highest-frequency cosine (sine remains)
One requires

$$\begin{aligned}a_i^{new}&=0\\Aa_{i-1}-Bb_{i-1}&=0\\\cos(\varphi)\,a_{i-1}+\sin(\varphi)\,b_{i-1}&=0\\a_{i-1}+\tan(\varphi)\,b_{i-1}&=0\\\tan(\varphi)&=-\frac{a_{i-1}}{b_{i-1}}\\\varphi&=-\arctan\left(\frac{a_{i-1}}{b_{i-1}}\right)\end{aligned}$$

where one needs to use the highest-frequency coefficients (a_{i-1} and b_{i-1}) from the previous ZCF.

Symmetric cutoff
One requires

$$\begin{aligned}a_i^{new}&=b_i^{new}\\a_i^{new}-b_i^{new}&=0\\Aa_{i-1}-Bb_{i-1}-Ba_{i-1}-Ab_{i-1}&=0\\\cos(\varphi)\,a_{i-1}+\sin(\varphi)\,b_{i-1}+\sin(\varphi)\,a_{i-1}-\cos(\varphi)\,b_{i-1}&=0\\a_{i-1}+\tan(\varphi)\,b_{i-1}+\tan(\varphi)\,a_{i-1}-b_{i-1}&=0\\\tan(\varphi)\,(b_{i-1}+a_{i-1})&=b_{i-1}-a_{i-1}\\\tan(\varphi)&=\frac{b_{i-1}-a_{i-1}}{b_{i-1}+a_{i-1}}\\\varphi&=\arctan\left(\frac{b_{i-1}-a_{i-1}}{b_{i-1}+a_{i-1}}\right)\end{aligned}$$

where one needs to use the highest-frequency coefficients ($a_{i-1}$and b_{i-1}) from the previous ZCF.

Finalizing the computations

Once the φ is know, K can be directly computed:

$$K = -\cos\left(x_{N-1} + \varphi\right)$$

With A ($=-\sin\varphi$), B ($=\cos\varphi$) and K known (in any cutoff situation), one constructs the new ZCF (section 2.4) and finds the appropriate value of α, so that the last point is described

$$y_{i=N} = Fs_{N-1}\left(x_i\right) + \alpha \times zcf_{N-1}\left(x_i\right).$$

The final step consists in performing a term-by-term summation of the two series.

$$Fs_N\left(x\right) = Fs_{N-1}\left(x\right) + \alpha \times zcf_{N-1}\left(x\right).$$

4 Final remarks

My first remark concerns the creation of the ZCF. Clearly, the described procedure leads to a progressive creation of the ZCF and is thus is not really ready to extend an existing Fourier series to a new point (situation where no previous ZCF was constructed). If one asks for it, the ZCF is not difficult to construct and I plan to address this issue in some of my future texts. However, an example of creating an (inappropriate) ZCF from scratch (in a “non-progressive” way) is actually presented in this text in the “how not to do things” section 2.2. The term “non-progressive” is yet little bit misleading, because one can always perform a progressive multiplication of successive sines in the formula 2.

The second remark concerns cutoff procedure. One could actually implement a cutoff of a middle-frequency term, the complication would arise from additional contributions originating in the previous higher-frequency and same-frequency terms. Such a cutoff would be untypical: one usually wants the data to be described by low frequencies (as low as possible), not skipping one in the middle.

The question arises also about the performance of different existing methods for trigonometric interpolation (speed, memory consumption, numerical precision). I will try to address these questions in some of my future texts.

References

[1] J. M. Dias Pereira, O. Postolache, and P.M. Girão. Pdf-based progressive polynomial calibration method for smart sensors linearization. *IEEE Trans. on Instrumentation and Measurement*, 58(9):3245–3252, September 2009.

[2] A. Liptaj. Cosine and sine interpolation. interpolation for arbitrary series with multiplicative coefficients. *https://www.scribd.com/document/289673276/Cosine-and-sine-interpolation-Interpolation-for-arbitrary-series-with-multiplicative-coefficients, http://vixra.org/abs/1704.0066*, 2015.

[3] A. Liptaj. Short notice on (exact) trigonometric interpolation. *https://www.scribd.com/document/270904435/Short-notice-on-exact-trigonometric-interpolation, http://vixra.org/abs/1704.0048*, 2015.

[4] Wolfram MathWorld. Trigonometric power formulas. *http://mathworld.wolfram.com/TrigonometricPowerFormulas.html.*

A Appendix: *SciLab* program

The program serves two aims:

- Gives the reader an out-of-the-box implementation of the progressive Fourier interpolation.
- Contains the algorithm and so, in case something is unclear or missing in the text, the algorithm can be read-out from the program.

A difference one should be aware of is the indexing: in *SciLab* the array indices (unfortunately) start at one and so a shift in indexing had to be done on some places with respect to what is written in the text.

The program also contains the function “tsfData” which is meant to re-scale the data (in the x and y directions proportionally). Indeed, the whole algorithm is based on the 2π period. It could be, of course, scaled to any (finite) period length, but it seems to me easier to scale the data. The middle argument “f” of the function is a flag: when equal to 0 then the point with the maximal x coordinate will be scaled to exactly match the upper boundary chosen by the user. This usually happens in a situation when one wishes to scale the data to a smaller interval then $[0, 2\pi]$. If one desires to scale the data to the $[0, 2\pi]$ interval, then “f”=1 introduces a separation between the highest x point (after re-scaling) and 2π. Not implementing the separation, one would run (after re-scaling) into a pathology for $y_0 \neq y_N$ because the Fourier interpolation is 2π periodic.

One should also keep in mind that, for what concerns frequency analysis, an appropriate re-scaling is needed. If one tries to interpolate (without re-scaling) points for which $x_{max} - x_{min} \ll 2\pi$, then one usually runs into numerical problems, because coefficients become quickly very large. One is, of course, allowed to ask for such an interpolation, but with “wavelengths” larger then the data spread, one can hardly interpret the results as a “frequency analysis”.

The program also incorporates an alternative expression the interpolation can take. The described method of interpolation can be sketched as follows (for practical illustration I demonstrate it on 7 and 8 points):

$$\begin{aligned}
Fs_{7\{8\}}(x) = \alpha_0 & \\
&+ \alpha_1 \sin(x+\varphi_1) \\
&+ \alpha_2 [\cos(x+\varphi_2) + K_2] \\
&+ \alpha_3 [\cos(x+\varphi_2) + K_2] \sin(x+\varphi_3) \\
&+ \alpha_4 [\cos(x+\varphi_2) + K_2][\cos(x+\varphi_4) + K_4] \\
&+ \alpha_5 [\cos(x+\varphi_2) + K_2][\cos(x+\varphi_4) + K_4] \sin(x+\varphi_5) \\
&+ \alpha_6 [\cos(x+\varphi_2) + K_2][\cos(x+\varphi_4) + K_4][\cos(x+\varphi_6) + K_6] \\
&+ \left\{\alpha_7 [\cos(x+\varphi_2) + K_2][\cos(x+\varphi_4) + K_4][\cos(x+\varphi_6) + K_6]\left[\cos(x+\widetilde{\varphi}_7) + \widetilde{K}_7\right]\right\},
\end{aligned}$$

where the tilde in the last line represents the cutoff dependency of the coefficients. Using abbreviations $c_i \equiv \cos(x+\varphi_i)$, $s_i \equiv \sin(x+\varphi_i)$ the expression can be transformed into a nested factorized form

$$F_7 = \alpha_0 + \alpha_1 s_1 + (c_2 + K_2)\{\alpha_2 + \alpha_3 s_3 + (c_4 + K_4)[\alpha_4 + \alpha_5 s_5 + \alpha_6 (c_6 + K_6)]\}$$

for an odd number (=7) of points and in the form

$$F_8 = \alpha_0 + \alpha_1 s_1 + (c_2 + K_2)\Big(\alpha_2 + \alpha_3 s_3 + (c_4 + K_4)\Big\{\alpha_4 + \alpha_5 s_5 + (c_6 + K_6)\Big[\alpha_6 + \alpha_7\Big(c_{\widetilde{7}} + \widetilde{K}_7\Big)\Big]\Big\}\Big)$$

for an even number (=8) of points. The program remembers progressively the numbers

$$\alpha_0, \alpha_1, \varphi_1, \alpha_2, \varphi_2, K_2, \alpha_3, \varphi_3, \alpha_4, \varphi_4, K_4 \ldots$$

and is able to compute the interpolation value using the nested form. Unlike the standard form, the nested form contains the information about the data points and this makes the computational time grow. I checked this behavior placing the corresponding code at the and of the program. I decided to keep it, but only as an option: an “abort” command is used before. If interested in the speed comparison between the standard and the nested form, the user needs to comment this line off. The nested form may still be interesting, it allows to compute the interpolation with less transformations.

Finally, let me note that the program requires as data input a text file with two columns containing the x (first column) and y (second column) coordinates. I insert the program code using a very small font: it can be copy-pasted when needed (or zoomed on the screen), yet the document is not too long to print.

```
function [newX,newY]=tsfData(xmin,xmax,f,xDat,yDat)
    //f=0 - xmax occupied
    //f=1 - xmax empty

    L = length(xDat)
    oldMin = min(xDat)
    oldMax = max(xDat)

    if f==1 then
        xmax = xmax-(xmax-xmin)/L
    end

    deriv = (xmax-xmin)/(oldMax-oldMin)

    for i=1:L
        newX(i) = xmin+deriv*(xDat(i)-oldMin)
        newY(i) = deriv*yDat(i)
    end
endfunction

function [f] = fourier(c,s,x)
    f = 0
    L = length(c)

    for i=1:L
        n=i-1

        f = f +  c(i)*cos(n*x) + s(i)*sin(n*x)
    end
endfunction

function [c,s] = trigMultiply(cs,sn,K,A,B)
    L = length(cs)

    // multiply by K

    cs_0 = K*cs
    sn_0 = K*sn
    cs_0(L+1) = 0
    sn_0(L+1) = 0

    // multiply by cos and sin

    for i=1:L+1

        j = i-1
        k = i+1

        if j<1 then
            leftTerm_c = 0
            leftTerm_s = 0
        else
            leftTerm_c = cs(j)
            leftTerm_s = sn(j)
        end

        if k>L then
            rightTerm_c = 0
            rightTerm_s = 0
        else
            rightTerm_c = cs(k)
            rightTerm_s = sn(k)
        end

        cs_cc = A*(leftTerm_c+rightTerm_c)/2
        cs_ss = B*(-leftTerm_s+rightTerm_s)/2

        sn_sc = A*(leftTerm_s+rightTerm_s)/2
        sn_cs = B*(leftTerm_c-rightTerm_c)/2

        c(i) =  cs_0(i)+cs_cc+cs_ss
        s(i) =  sn_0(i)+sn_sc+sn_cs

        if i==2 then // left term treated as right (index 1 left from 2 but right from 0)
            c(i) = c(i) + A*(leftTerm_c)/2
            c(i) = c(i) + B*(leftTerm_s)/2

            s(i) = s(i) - A*(leftTerm_s)/2  // because sin(-x) = -sin(x)
            s(i) = s(i) - B*(-leftTerm_c)/2 // because sin(-x) = -sin(x)
        end
    end

endfunction

function [c,s,nestedFactors] = progTrigoInterpol(X,Y,f)
    N = length(X)
    zc_fun_c = []
    zc_fun_s = []

    for i=1:N // Loop over data points

        if i==1 then

            c(1)=Y(1)
            s(1)=0

            zc_fun_c(1)=1
            zc_fun_s(1)=0

            // Line related to the nested scheme
            nestedFactors(1) = Y(1)

            continue

        elseif i<N then

            if modulo(i,2)==1 then //  cos(x+phi) + K

                phi = -(X(i-2)+X(i-1))/2
                K = -cos(X(i-1)+phi)
                A = cos(phi)
                B = -sin(phi)

                // Two lines related to the nested scheme
                nestedFactors = cat(2,nestedFactors,phi)
                nestedFactors = cat(2,nestedFactors,K)
            else // sin(x+phi)
                phi = -X(i-1)
                K = 0
                A = sin(phi)
                B = cos(phi)

                // Line related to the nested scheme
                nestedFactors = cat(2,nestedFactors,phi)

            end

        else

            if modulo(i,2)==1 then // Odd number of points, "standard" situation

                phi = -(X(i-2)+X(i-1))/2
                K = -cos(X(i-1)+phi)
                A = cos(phi)
                B = -sin(phi)

            else // Even number of points, arbitrary cutoff needed

                L_zcf = length(c)
                s_left = zc_fun_s(L_zcf)
                c_left = zc_fun_c(L_zcf)

                if f==0 then // high sine cutoff
                    phi = atan(s_left/c_left)
                elseif f==1 then // high cosine cutoff
                    phi = atan(-c_left/s_left)
                else // symmetric cutoff
                    phi = atan( (s_left-c_left)/(s_left+c_left))
                end

                K = -cos(X(i-1)+phi)
                A = cos(phi)
                B = -sin(phi)


            end

            // Two lines related to the nested scheme
            nestedFactors = cat(2,nestedFactors,phi)
            nestedFactors = cat(2,nestedFactors,K)

        end

        // K, A, B are settled
        // Now construct zero-crossing function

        [coef_c,coef_s] = trigMultiply(zc_fun_c,zc_fun_s,K,A,B)

        if modulo(i,2)==1 then //remember last series
            zc_fun_c = coef_c
            zc_fun_s = coef_s
        end

        // Find appropriate multiplicative coefficient to match new data point

        upToNow = fourier(c,s,X(i))
        toBeAdded = fourier(coef_c,coef_s,X(i))
        coef = (Y(i)-upToNow)/toBeAdded

        // Line related to the nested scheme
        nestedFactors = cat(2,nestedFactors,coef)

        // Add fourier series

        L_four = length(c)
        L_coef = length(coef_c)

        for j=1:L_coef
            if j<L_coef then
                c(j)=c(j)+coef*coef_c(j)
                s(j)=s(j)+coef*coef_s(j)
            elseif L_coef==L_four
                c(j)=c(j)+coef*coef_c(j)
                s(j)=s(j)+coef*coef_s(j)
            else
                c(j)=coef*coef_c(j)
                s(j)=coef*coef_s(j)
            end
        end

    end

endfunction

function [f] = nestedForm(fcts,nData,x)
    L = length(fcts)

    if modulo(nData,2)==1 then // Odd number of data points
        f = fcts(L-3)*sin(x+fcts(L-4)) + fcts(L)*(cos(x+fcts(L-2))+fcts(L-1))
        pos = L-5
    else // Even number of data points
        f = fcts(L)*(cos(x+fcts(L-2))+fcts(L-1))
        pos = L-3
    end

    while 1==1
        f = fcts(pos-3)*sin(x+fcts(pos-4))+(cos(x+fcts(pos-2))+fcts(pos-1)).*(fcts(pos)+f)
        pos = pos-5

        if pos==1 then
            f = f + fcts(pos)
            break
        end
    end

endfunction

function [minX,maxX,minY,maxY]=winSize(datX,datY)
    minX = min(datX)
    maxX = max(datX)
    minY = min(datY)
    maxY = max(datY)
    deltaX = maxX-minX
    deltaY = maxY-minY
    minX = minX-0.2*deltaX
    maxX = maxX+0.2*deltaX
    minY = minY-0.2*deltaY
    maxY = maxY+0.2*deltaY
endfunction

// Start of the profram flow

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

datX = dataSet(:,1)
datY = dataSet(:,2)
nDat = length(datX)

// Transform data to a more appropriate interval - uncomment following
//datX = tsfData(0,2*%pi,1,datX,datY)
//datX = tsfData(-%pi/4,%pi/4,0,datX,datY)

[minX,maxX,minY,maxY] = winSize(datX,datY)

// Compute coefficients
[c,s,nf] = progTrigoInterpol(datX,datY,2)

nPts = 1000
xAx = linspace(minX,maxX,nPts)
yAx = fourier(c,s,xAx)

disp(" COSINE COEFS: ")
disp(c)
disp(" SINE COEFS: ")
disp(s)

scf(1)
plot2d(datX,datY,-2,rect=[minX,minY,maxX,maxY])
plot2d(xAx,yAx,rect=[minX,minY,maxX,maxY])

// Nested-factors scheme tested
y2_Ax =  nestedForm(nf,nDat,xAx)

scf(2)
plot2d(datX,datY,-2,rect=[minX,minY,maxX,maxY])
plot2d(xAx,y2_Ax,rect=[minX,minY,maxX,maxY])

// Time test comparing the standard and the nested form
abort() // Comment-out if interested in computation speed comparison

N = 1000
rand('uniform')

tic()
for i=1:N
    test_x = 2*%pi*rand()
    void = fourier(c,s,test_x)
end
t1 = toc()

tic()
for i=1:N
    test_x = 2*%pi*rand()
    void = nestedForm(nf,test_x)
end
t2 = toc()

mprintf("Standard = %f seconfd\n",t1)
mprintf("Nested = %f seconds\n",t2)
mprintf("Nested is %f time slower\n",t2/t1)
```