Fourier (or trigonometric) interpolation based on summation of Fourier series F_i^δ with data-related delta-function property $F_i^\delta(x_j) = y_j \delta_{ij}$ (analogy to Lagrange form of interpolation polynomial) and use of F_i^δ functions to extend existing trigonometric interpolation.

Andrej Liptaj*

April 19, 2017

Abstract

Full analogy to the Lagrange form of the interpolation polynomial is constructed for Fourier series. As a straightforward consequence one gets the ability to extend an existing trigonometric interpolation to additional data point(s).

1 Introduction

A function expressed in a form of a series with multiplicative coefficients

$$f(x) = \sum_i c_i f_i(x)$$

can be easily used to interpolate data points

$$(x_i, y_i)$$

if "one-point describing, other points vanishing" functions f_i^δ can be found

$$f_i^\delta(x) = \sum_j c_{ij}^\delta f_j(x),$$
$$f_i^\delta(x_j) = y_j \delta_{ij}.$$

I call such functions as having "data-related delta-function property". The interpolation over all data points can be then written as

$$f(x) = \sum_i f_i^\delta(x).$$

The well know example is the Lagrange form of the interpolation polynomial [1], where

$$f_i^\delta(x) = y_i \prod_{j, j \neq i} \frac{x - x_j}{x_i - x_j}.$$

This text aims to generalize this method also to the trigonometric interpolation.

*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.

For that purpose let me use the terminology introduced in [2]: I will use the term "degree" to refer to the highest non-zero n (frequency) in a Fourier series expression

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

The degree is directly related to the number of data points to be interpolated. Because b_0 is arbitrary, a Fourier series of degree M is supposed to describe $2M+1$ or $2M$ data points. In the latter case one has a "cutoff freedom": different combinations of the highest-frequency coefficients a_N and b_N can be used to provide an interpolation (i.e. different valid interpolations exist).

2 Fourier series with data-related delta-function property

The method has two steps. If the number of data points is N then, in the first step, one needs to construct a Fourier series $\widetilde{F}_i^{\delta}$ which vanishes for $N-1$ data points $x_{j,j\neq i}$. In the second step one normalizes the (non-zero) value of the function at x_i

$$F_i^{\delta}(x) = y_i \frac{\widetilde{F}_i^{\delta}(x)}{\widetilde{F}_i^{\delta}(x_i)}.$$

The first step can be understood as an interpolation task for points with zero y coordinates $(x_{j,j\neq i}, 0)$. Usually, the number of parameters (to be tuned) is the same as the number of points to be described. However our situation is a non-standard one: a straightforward application of an interpolation method to describe $(x_{j,j\neq i}, 0)$ would lead to all coefficients vanishing $a_n = b_n = 0$, thus preventing us to perform the second, normalization step. Therefore, to construct the Fourier series $\widetilde{F}_i^{\delta}$, one needs one more parameter in the series than what corresponds to the number of data points.

The first of the following subsections describes a useful technique of Fourier series multiplication by an "atomic" term. Next follows the description of the method which is split into two subsections corresponding to a different data points parity. This separation is convenient because of a possible cutoff freedom which occurs only for an even number of data points.

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

This is a preparatory section for further calculations, where terms of the structure $\cos(x+\varphi) + K$ will play an important role. Any of these terms can be expanded

$$\cos(x+\varphi) + K = A\cos(x) + B\sin(x) + K$$

with

$$A = \cos(\varphi)$$

and

$$B = -\sin(\varphi).$$

The multiplication of a Fourier series by $\cos(x+\varphi) + K = A\cos(x) + B\sin(x) + K$ can be written as

$$\left\{ \sum_{n=0}^{N} \left[a_n \cos(nx) + b_n \sin(nx) \right] \right\} \times \left[A\cos(x) + B\sin(x) + K \right] = \sum_{n=0}^{N+1} \left[a_n^{new} \cos(nx) + b_n^{new} \sin(nx) \right],$$

where

$$\begin{aligned} a_n^{new} &= \frac{A}{2}(a_{n-1} + a_{n+1}) + \frac{B}{2}(-b_{n-1} + b_{n+1}) + K a_n, \\ b_n^{new} &= \frac{B}{2}(a_{n-1} - a_{n+1}) + \frac{A}{2}(b_{n-1} + b_{n+1}) + K b_n \\ &\quad \text{for } 1 < n \end{aligned} \tag{1}$$

and (for $n = 1$)

$$a_1^{new} = \frac{A}{2}(a_0 + a_2) + \frac{B}{2}(-b_0 + b_2) + Ka_1 + \frac{A}{2}a_0 + \frac{B}{2}b_0,$$
$$b_1^{new} = \frac{B}{2}(a_0 - a_2) + \frac{A}{2}(b_0 + b_2) + Kb_1 + \frac{B}{2}a_0 - \frac{A}{2}b_0. \tag{2}$$

The justification can be found in Sec. 2.4 of [2].

2.2 Odd number of data points

If the total number of data points is odd, $2N + 1$, then the set $(x_{j,j\neq i}, 0)$ contains $2N$ points. The function $\widetilde{F}_i^\delta$ can be written directly as

$$\widetilde{F}_i^\delta(x) = \prod_{j=1}^{N} [\cos(x + \varphi_j) + K_j]$$

where

$$\varphi_j = -\frac{x_{2j-1} + x_{2j}}{2} \tag{3}$$

and

$$K_j = -\cos(x_{2j} + \varphi_j). \tag{4}$$

Clearly, each term in the product is constructed such as to vanish at two points, x_{2j-1} and x_{2j}. The Fourier series $F_i^\delta(x)$ can then be written as

$$F_i^\delta(x) = y_i \frac{\widetilde{F}_i^\delta(x)}{\widetilde{F}_i^\delta(x_i)}.$$

In order to perform the final summation

$$F(x) = \sum_i F_i^\delta(x)$$

it is convenient to expand the product form of the $F_i^\delta(x)$ functions into the standard series form and add them term-by-term. Since the product form is just successive multiplication of the $\cos(x + \varphi_j) + K_j$ terms, one can use the prescription from the Sec. 2.1 repeatedly. Formulas (1) and (2) show that each term in the product increases the degree of the Fourier series by one, thus leading to F_i^δ which has the degree N. The summation $F(x) = \sum_i F_i^\delta(x)$ does not increase the degree and therefore the final trigonometric interpolation has the correct degree N, which corresponds to $2N + 1$ parameters in the Fourier series and $2N + 1$ data points (without any freedom).

2.3 Even number of data points

Let me now assume that the total number of data points is even $2N$. The set $(x_{j,j\neq i}, 0)$ then contains $2N - 1$ points. We arbitrarily separate one of them, so we have $N-1$ pairs, and one single point to describe. Without loss of generality let the separated point be the one with the index N (the last one). The interpolation is then written

$$\widetilde{F}_i^\delta(x) = [\cos(x + \varphi_N) + K_N] \times \prod_{j=1}^{N-1} [\cos(x + \varphi_j) + K_j], \tag{5}$$

where the quantities indexed by the letter j are defined by the expressions (3) and (4) from the previous section. The interesting numbers are φ_N and K_N. One needs to chose them so, as to fulfill the two following constraints:

- describe the point $(x_N, 0)$ and
- provide a desired cutoff.

The final summation $F(x) = \sum_i F_i^\delta(x)$ inherits the cutoff from the F_i^δ functions (if done consistently), therefore one needs to focus on the cutoff for a single F_i^δ function. Here the following reasoning can be made: the highest degree term appearing in the standard form of the Fourier series after the expansion of the product (5) comes from the multiplication of the cosine terms

$$\cos(x+\varphi_N)\prod_{j=1}^{N-1}\cos(x+\varphi_j), \tag{6}$$

as seen from the formula (1): the multiplication by K (clearly) does not increase the series degree, the degree is increased when trigonometric terms are mutually multiplied. The expression (6) itself represents a complete Fourier series, I am however interested only in the highest frequency term, the one given uniquely by this expression. The following is true[1]

$$\cos(x+\varphi_N)\prod_{j=1}^{N-1}\cos(x+\varphi_j) = \frac{1}{2^{N-1}}\cos\left(Nx+\varphi_N+\sum_j\varphi_j\right) + \text{ terms of lower degree.}$$

The highest frequency term will therefore look like

$$\cos\left(Nx+\varphi_N+\sum_j\varphi_j\right) = \cos\left(\varphi_N+\sum_j\varphi_j\right)\cos(Nx) - \sin\left(\varphi_N+\sum_j\varphi_j\right)\sin(Nx).$$

So, clearly, if one desires to make a high-frequency sine cutoff (keep the cosine function) then one needs to make vanish the sine numerical pre-factor

$$-\sin\left(\varphi_N+\sum_j\varphi_j\right) = 0$$
$$\Leftarrow$$
$$\varphi_N = -\sum_j\varphi_j.$$

If one desires to make a high-frequency cosine cutoff (keep the sine function) then one asks for

$$\cos\left(\varphi_N+\sum_j\varphi_j\right) = 0$$
$$\Leftarrow$$
$$\varphi_N = \frac{\pi}{2}-\sum_j\varphi_j.$$

[1] From $\cos(a)\cos(b) = \frac{1}{2}[\cos(a+b)+\cos(a-b)]$ one has $\cos(x+\varphi_i)\cos(Mx+\varphi_j) = \frac{1}{2}\cos[(M+1)x+\varphi_i+\varphi_j] + \frac{1}{2}\cos[(M-1)x+\varphi_j-\varphi_i]$. The later equality used repeatedly leads to an accumulation (summation) of phases in the highest frequency term.

If one desires a symmetric cutoff ($a_N = b_N$) then the following applies

$$\begin{aligned}
\cos\left(\varphi_N + \sum_j \varphi_j\right) &= -\sin\left(\varphi_N + \sum_j \varphi_j\right) \\
\sin\left(\varphi_N + \sum_j \varphi_j\right) + \cos\left(\varphi_N + \sum_j \varphi_j\right) &= 0 \\
\tan\left(\varphi_N + \sum_j \varphi_j\right) + 1 &= 0 \\
\tan\left(\varphi_N + \sum_j \varphi_j\right) &= -1 \\
&\Leftarrow \\
\varphi_N + \sum_j \varphi_j &= \frac{3}{4}\pi \\
\varphi_N &= \frac{3}{4}\pi - \sum_j \varphi_j .
\end{aligned}$$

Now that we have settled the angle φ_N we can determine the value of K_N. To make $\cos(x_N + \varphi_N) + K_N$ equal to zero, one needs

$$K_N = -\cos(x_N + \varphi_N).$$

With φ_N and K_N determined, one can proceed further (expand the product (5)) to get the full interpolating Fourier series as described in previous sections 2.1 and 2.2.

3 Extending existing trigonometric interpolation

3.1 Existing interpolation describes even number of points $2N$

If the existing Fourier interpolation $F^{ex}(x)$

$$F^{ex}(x_i) = y_i, \quad 1 \le i \le 2N$$

describes an even number of data points, then the extension to an additional point is straightforward. One constructs the appropriate $\widetilde{F}^{\delta}_{2N+1}(x)$ function exactly as presented in Sec. 2.2

$$\widetilde{F}^{\delta}_{2N+1}(x) = \prod_{j=1}^{N} \left[\cos(x + \varphi_j) + K_j\right], \quad \varphi_j = -\frac{x_{2j-1} + x_{2j}}{2}, \quad K_j = -\cos(x_{2j} + \varphi_j),$$

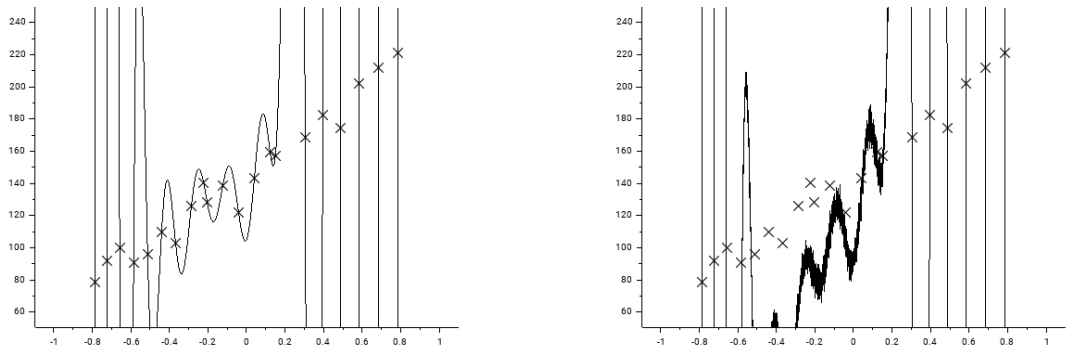
and, after expanding the product in $\widetilde{F}^{\delta}_{2N+1}$, sums both Fourier series (term-by-term)

$$F(x) = F^{ex}(x) + \alpha \widetilde{F}^{\delta}_{2N+1}(x),$$

where

$$\alpha = \frac{y_{2N+1} - F^{ex}(x_{2N+1})}{\widetilde{F}^{\delta}_{2N+1}(x_{2N+1})}.$$

Function $\widetilde{F}^{\delta}_{2N+1}(x)$ does not spoil the existing description and enables us to describe also the point (x_{2N+1}, y_{2N+1}). The original interpolation $F^{ex}(x)$ has an arbitrary high-frequency cutoff (because describing an even number of points), the cutoff liberty after our procedure disappears.



a) b)

Figure 1: a) The non-expanded product form of the trigonometric interpolation seems to be less affected by numerical errors then b) the (same) trigonometric interpolation fully expanded and summed (i.e. having the standard Fourier series form).

3.2 Existing interpolation describes odd number of points $2N-1$

In this paragraph I assume that the existing interpolation $F^{ex}(x)$ describes an odd number of points $2N-1$

$$F^{ex}(x_i) = y_i, \quad 1 \leq i \leq 2N-1.$$

The "delta" Fourier series takes form as in Sec. 2.3

$$\widetilde{F}^{\delta}_{2N}(x) = [\cos(x+\varphi_N) + K_N] \times \prod_{j=1}^{N-1} [\cos(x+\varphi_j) + K_j], \quad \varphi_j = -\frac{x_{2j-1}+x_{2j}}{2}, \quad K_j = -\cos(x_{2j}+\varphi_j).$$

The degree of $\widetilde{F}^{\delta}_{2N}(x)$ is greater then the one of $F^{ex}(x)$: the degree of $F^{ex}(x)$ is $N-1$, the degree of $\widetilde{F}^{\delta}_{2N}(x)$ is N. Therefore the highest frequency coefficients of their sum are dictated by the $\widetilde{F}^{\delta}_{2N}(x)$ series, and once more therefore, the cutoff procedure (determination of φ_N and K_N) from Sec. 2.3 applies. The result is written as

$$F(x) = F^{ex}(x) + \alpha \widetilde{F}^{\delta}_{2N}(x),$$

where

$$\alpha = \frac{y_{2N} - F^{ex}(x_{2N})}{\widetilde{F}^{\delta}_{2N}(x_{2N})}.$$

4 Discussion and summary

The presented method should be seen as an alternative method to perform a trigonometric interpolation. What is Lagrange construction to polynomials, that is this construction to Fourier series. One may ask about possible advantages of the proposed method. There is one I can mention as observation: if the non-expanded (product) form is used for $\widetilde{F}^{\delta}_{i}(x)$ functions at given $\widetilde{x}$ (i.e. each function is evaluated at $\widetilde{x}$ and the resulting numbers are summed) then the result is less sensitive to numerical errors than what one gets when expanding and summing (i.e. transforming the interpolation into the standard series form and evaluating only after). I tested it in the *SciLab* software, the (identical) data points and the results are shown in Fig. 1.

As summary I propose the summary formulas:

- Odd number of data points:

$$F(x) = \sum_{i=1}^{2N+1} \left\{ \prod_{j=1, j\neq i}^{N} \left[\cos\left(x - \frac{x_{2j-1}+x_{2j}}{2} \right) - \cos\left(x_{2j} - \frac{x_{2j-1}+x_{2j}}{2} \right) \right] \right\}.$$

- Even number of data points:

$$F(x) = \sum_{i=1}^{2N} \left\{ \left[\cos(x+\varphi_N) - \cos(x_N+\varphi_N)\right] \times \right.$$
$$\left. \times \prod_{j=1, j\neq i}^{N-1} \left[\cos\left(x - \frac{x_{2j-1}+x_{2j}}{2}\right) - \cos\left(x_{2j} - \frac{x_{2j-1}+x_{2j}}{2}\right)\right] \right\}$$

with cutoff options

$$\text{high-sine cutoff} : \varphi_N = \sum_j \frac{x_{2j-1}+x_{2j}}{2}$$
$$\text{high-cosine cutoff} : \varphi_N = \frac{\pi}{2} + \sum_j \frac{x_{2j-1}+x_{2j}}{2},$$
$$\text{symmetric cutoff} : \varphi_N = \frac{3}{4}\pi + \sum_j \frac{x_{2j-1}+x_{2j}}{2}.$$

In this case one should correctly understand that j runs over $2N-2$ values, because j is different from i and from N. The angle φ_N contains an implicit i dependence and therefore the expression $[\cos(x+\varphi_N) - \cos(x_N+\varphi_N)]$ cannot be factorized.

References

[1] https://en.wikipedia.org/wiki/Lagrange_polynomial

[2] A. Liptaj, "Progressive Fourier (or trigonometric) interpolation",
https://www.scribd.com/document/344970283/Progressive-Fourier-or-trigonometric-interpolation
http://vixra.org/abs/1704.0149

Appendix: *SciLab* code

The program serves two aims:

- Gives the reader an out-of-the-box implementation of the "delta" 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 $y_1 \neq y_N$ 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 contains two versions of the interpolation evaluation: via the "un-expanded product" form and via the transformation into the standard series form.

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 [sub_x,sub_y,om_x,om_y]=getOmittedData(x,y,pos)
    L = length(x)
    k=0

    for i=1:L-1
        if i==pos then
            k=1
        end
        sub_x(i) = x(i+k)
        sub_y(i) = y(i+k)
    end

    om_x = x(pos)
```

```
    om_y = y(pos)

endfunction

function [phi,K,nrm] = getSinglePointFnc(X,Y,om_x,om_y,f)
    L = length(X)
    nPairs = int(L/2)

    for i=1:nPairs
        phi(i) = -(X((2*i)-1)+X(2*i))/2
        K(i) = -cos(X(2*i)+phi(i))
    end

    if (2*nPairs<L) then //Odd number of points

        phiSum = 0
        for i=1:nPairs
            phiSum = phiSum + phi(i)
        end

        if f == 0 then // high sine cutoff, cosine remains
            phi(nPairs+1) = -phiSum
        elseif f==1 then// high cosine cutoff, sine remains
            phi(nPairs+1) = %pi/2 - phiSum
        else // symmetric cutoff
            num = sin(phiSum) + cos(phiSum)
            den = sin(phiSum) - cos(phiSum)
            //phi(nPairs+1) = atan(num/den)
            phi(nPairs+1)=3*%pi/4-phiSum
        end
        K(nPairs+1) = -cos(X(L)+phi(nPairs+1))
    end

    yVal = 1
    nPairs = length(phi)
    for i=1:nPairs
        yVal = yVal*(cos(om_x + phi(i)) + K(i))
    end
    nrm = om_y/yVal

endfunction

function [array_phi,array_K,array_nrm] = singPtFnArray(X,Y,f)
    L = length(X)

    for i=1:L
        [sub_X,sub_Y,om_x,om_y]=getOmittedData(X,Y,i)
        [phiArr,kArr,normalization] = getSinglePointFnc(sub_X,sub_Y,om_x,om_y,f)

        L2 = length(phiArr)
        for j=1:L2
            array_phi(i,j) = phiArr(j)
            array_K(i,j) = kArr(j)
        end
        array_nrm(i) = normalization

    end
endfunction

function [array_c,array_s]=get_CS_from_PhiK(array_phi,array_K)
    L = length(array_phi)

    for i=1:L
        array_c(i) = cos(array_phi(i))
        array_s(i) = -sin(array_phi(i))
    end
endfunction

function [f] = evSinglPtFn(phi,K,nrm,x)
    L = length(phi)
    val=1

    for i=1:L
        val = val.*(cos(x+phi(i))+K(i))
    end

    f = nrm*val
endfunction

function [f] = getValFromProduct(array_phi,array_K,array_nrm,x)
    L = length(array_nrm)
    val = 0

    for i=1:L
        val = val + evSinglPtFn(array_phi(i,:),array_K(i,:),array_nrm(i),x)
    end

    f = val
endfunction

function [cosCfs,sinCfs] = getStandardFourier(arrPhi,arrK,nrm)
    L = length(arrPhi)

    cosCfs(1)=1
    sinCfs(1)=0

    for i=1:L
        A = cos(arrPhi(i))
        B = -sin(arrPhi(i))
        K = arrK(i)

        [cosCfs,sinCfs] = trigMultiply(cosCfs,sinCfs,K,A,B)
    end

    cosCfs = nrm*cosCfs
    sinCfs = nrm*sinCfs

endfunction
```

```
function [c,s] = getFourierCfs(Phi_2D,K_2D,nrm_1D)
    L = length(nrm_1D)

    for i=1:L

        if i==1 then
            [c,s] = getStandardFourier(Phi_2D(i,:),K_2D(i,:),nrm_1D(i))
        else
            [cCfs,sCfs] = getStandardFourier(Phi_2D(i,:),K_2D(i,:),nrm_1D(i))
            c = c + cCfs
            s = s + sCfs
        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


// Program flow start
dataSet = read("data.txt",-1,2)
nDat = length(dataSet)/2
datX = dataSet(:,1)
datY = dataSet(:,2)


// Transform data to a more appropriate interval??
//datX = tsfData(0,2*%pi,1,datX,datY)
[minX,maxX,minY,maxY] = winSize(datX,datY)


  // Factorized version
[a_phi,a_K,a_nrm] = singPtFnArray(datX,datY,2)
xAx = linspace(minX,maxX,10000)
yAx = getValFromProduct(a_phi,a_K,a_nrm,xAx)
scf(1)
plot2d(datX,datY,-2,rect=[minX,minY,maxX,maxY])
plot2d(xAx,yAx,rect=[minX,minY,maxX,maxY])


   // Standard version
[c,s] = getFourierCfs(a_phi,a_K,a_nrm)
disp(" COSINE COEFS: ")
disp(c)
disp(" SINE COEFS: ")
disp(s)
disp(" NORMALIZATION COEFS: ")
disp(a_nrm)
y2_Ax = fourier(c,s,xAx)
scf(2)
plot2d(datX,datY,-2,rect=[minX,minY,maxX,maxY])
plot2d(xAx,y2_Ax,rect=[minX,minY,maxX,maxY])
```