PCSC Project 2017 - Data Approximation

Cleres David
Computational Science & Engineering
Email: david.cleres@epfl.ch

Lesimple Nicolas
Computational Science & Engineering
Email: nicolas.lesimple@epfl.ch

***Abstract*—This final project of the semester focused on data approximation by numerical means. The group implemented three of the most common numerical methods, namely Lagrange Polynomial functions, Least Square Regression and finally Fourier Approximation. All these tools took a training data file as an input and the mentioned numerical methods were able to generate a qualified guess of the output values for new untrained x values. Finally, by using implemented test functions, one was able to obtain the proportion of accurately interpolated values.**

I. Introduction

Traditionally, Fortran and MATLAB® have been the languages of choice for scientific computing applications. The recent development of complex mathematical modeling fields as diverse as biology, finance, and materials science, ... This has driven a need for software packages that allow computational simulations based on these models. The complexity of the underlying models, together with the need to exchange code between coworkers, has motivated programmers to develop object-oriented code. The C++ programming language is an Object Oriented (OO) language. Combining this offered modularity and clear programming style enables the programmers to minimize the introduction of errors into code by exhibiting the efficacy of classes and highlighting the main features of object-orientation. In this project, one could take advantage of the highly modular C++ language for the purpose of data approximation data using different approximation methods. This report serves as a documentation about the mentioned project and describes how to compile and execute the code. This report also contains a description of the tests that were created to assess the correctness of the program. Furthermore, this document also contains a few figures about our results with the different approximation methods. Finally, a few sentences aim to describe other methods that could have been used but, because of time restriction, were not implemented.

II. Dataset

The user was provided with different datasets containing the x and y values of the periodic functions described below. By doing so, it was possible to first interpolate the given points and then to demonstrate, with graphical interpretations, that the computations made by the programs were correct for any data set. Thanks to the modularity of the code it should be simple for the user to add more datasets if he/she wants to. In this case, it was only necessary to generate a new text file containing the points to interpolate and simply to change the name of the file to read in. The functions files with the data points that we provided were:

1) $\cos(\pi x)$ a periodic function of period 2
2) $\cos(3\pi x) + \sin(\pi x)$ a periodic function of period 2

As show by these examples, the code was thought to deal with real numbers and not with imaginary inputs.

III. Methods

Lagrange Polynomial Approximations - In numerical analysis, Lagrange polynomials are used for polynomial interpolation. For a given set of points (x_j, y_j) for $x_i \neq x_j$ for all i, j. The Lagrange polynomial is the polynomial of lowest degree that interpolates for each x_j the corresponding value y_j (i.e. the functions coincide at each point).

Given a set of k + 1 data points:

$(x_0, y_0), ..., (x_j, y_j), ..., (x_k, y_k)$ where no two x_j were the same, the interpolation polynomial in the Lagrange form was a linear combination : $L(x) := \sum_{j=0}^{k} y_j l_j(x)$ of Lagrange basis polynomials:

$$l_j(x) := \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{x - x_m}{x_j - x_m} \tag{1}$$

$$= \frac{(x - x_0)}{(x_j - x_0)} ... \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} ... \frac{(x - x_k)}{(x_j - x_k)}, \ell_j(x) \tag{2}$$

$$:= \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{x - x_m}{x_j - x_m} \tag{3}$$

$$= \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)}, \tag{4}$$

When constructing interpolating polynomials, there was a trade off between having the best fit and having a smooth well-behaved fitting function. The more data points that were used in the interpolation, the higher was the degree of the resulting polynomial. However, increasing the degree lead to more small oscillations between the data points, meaning that a high-degree interpolation may be a poor predictor of the y-values of the function between the data points, although the

accuracy at the data points will be close to perfect as shown by the tests. The given formula was implemented using a double for loop. Theses loops permitted to use the Lagrange formula on all the points given in argument and thus to interpolate our approximations. It is important to note that the interpolation was then made on a new set of x points. The result was then compared with the control data.

For example, for n=3 points,

For $n = 3$ points,

$$P(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}y_3$$
$$P'(x) = \frac{2x-x_2-x_3}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{2x-x_1-x_3}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{2x-x_1-x_2}{(x_3-x_1)(x_3-x_2)}y_3.$$

Note that the function $P(x)$ passes through the points (x_i, y_i), as can be seen for the case $n = 3$.

$$P(x_1) = \frac{(x_1-x_2)(x_1-x_3)}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x_1-x_1)(x_1-x_3)}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x_1-x_1)(x_1-x_2)}{(x_3-x_1)(x_3-x_2)}y_3 = y_1$$
$$P(x_2) = \frac{(x_2-x_2)(x_2-x_3)}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x_2-x_1)(x_2-x_3)}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x_2-x_1)(x_2-x_2)}{(x_3-x_1)(x_3-x_2)}y_3 = y_2$$
$$P(x_3) = \frac{(x_3-x_2)(x_3-x_3)}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x_3-x_1)(x_3-x_3)}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x_3-x_1)(x_3-x_2)}{(x_3-x_1)(x_3-x_2)}y_3 = y_3.$$

Least Squares Approximations - Least squares (LS) was a standard approach in regression analysis to the approximate solution of overdetermined systems, in other words, sets of equations in which there were more equations than unknowns. This method was a form of mathematical regression that found the line of best fit for a dataset, providing a visual demonstration of the relationship between the data points. Each point of data was representative of the relationship between a known independent variable and an unknown dependent variable. Nowadays, the most important application of LS is in data fitting. The best fit in the LS sense minimizes the sum of squared residuals (ie. the difference between an observed value, and the fitted value provided by a model). When the problem had substantial uncertainties in the independent variable (the x variable), then simple regression and LS methods had problems; in such cases, the methodology required for fitting errors-in-variables models may be considered instead of LS. The sum of the squares of the offsets was used instead of the offset absolute values because this allows the residuals to be treated as a continuously differentiable quantity. However, because squares of the offsets were used, outlying points could have a disproportionate effect on the fit, a property which may or may not be desirable depending on the problem at hand. LS problems fell into two categories: linear or ordinary least squares and nonlinear least squares, depending on whether or not the residuals were linear in all unknowns. The most common application of the LS method, referred to as linear or ordinary, aimed to create a straight line that minimizes the sum of the squares of the errors generated by the results of the associated equations, such as the squared residuals resulting from differences in the observed value and the value anticipated based on the model. The coefficients and summary outputs explained the dependence of the variables being tested. Problem statement: The objective consisted of adjusting the parameters of a model function to best fit a data set. A simple data set consists of n points (data pairs) where x_i was an independent variable and y_i was a dependent variable whose value was found by observation. The model function has the form $f(x, \beta)$. The goal was to find the parameter values for the model that "best" fitted the data. The fit of a model to a data point was measured by its residual, defined as the difference between the actual value of the dependent variable and the value predicted by the model:

$$r_i = y_i - f(x_i, \beta) \tag{5}$$

The least-squares method found the optimal parameter values by minimizing the sum S, of squared residuals:

$$S = \sum_{i=1}^{n} {r_i}^2. \tag{6}$$

An example of a model in two dimensions was that of the straight line. Denoting the y-intercept as β_0 and the slope as β_1, the model function was given by

$$f(x, \beta) = \beta_0 + \beta_1 x. \tag{7}$$

To solve the LS problem one needed to minimize the sum of squares which was found by setting the gradient to zero. Linear LS : A regression model was a linear one when the model comprises a linear combination of the parameters $f(x, \beta) = \sum_{j=1}^{m} \beta_j \phi_j(x)$, where the function ϕ_j was a function of x. Letting $X_{ij} = \frac{f(x_i,\beta)}{\beta_j} = \phi_j(x_i)$, one could then see that in that case the least square estimate (or estimator, in the context of a random sample), was given by $\beta = (X^T X)^{-1} X^T y$.

Indeed, in term of matrix it was possible to consider an overdetermined system

$$\sum_{j=1}^{n} X_{ij}\beta_j = y_i, (i = 1, 2, ..., m) \tag{8}$$

of m linear equations in n unknown coefficients, $\beta 1, \beta 2, \ldots, \beta n$, with m $>$ n. For a linear model as above, not all of X contained information on the data points. The first column was populated with ones, $X_{i1} = 1$, only the other columns contained actual data, and n = number of regressors + 1. This could be written in matrix form as

$$\mathbf{X}\beta = \mathbf{y} \tag{9}$$

where

$$\mathbf{X} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} \\ X_{21} & X_{22} & \cdots & X_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m1} & X_{m2} & \cdots & X_{mn} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

Such a system usually had no solution, so the goal was instead to find the coefficients β which fitted the equations "best", in the sense of solving the quadratic minimization problem

$$\hat{\beta} = argmin_{\beta} S(\beta), \tag{10}$$

where the objective function S was given by

$$S(\beta) = \left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|^2. \tag{11}$$

This minimization problem had a unique solution, provided that the n columns of the matrix X were linearly independent, given by solving the normal equations

$$(\mathbf{X}^{\mathrm{T}}\mathbf{X})\hat{\boldsymbol{\beta}} = \mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{12}$$

In the terms of implementation, this matrix was used to compute and find the coefficients β that minimized the error. To do so the Gaussian elimination was used and coded with nested for loops.

Piece Wise Approximations - In term of math, the program did exactly the same as for the two methods above. Indeed, in our code, functions that were written for Least Square and Lagrange approximation were called. The only difference was that the program did not do that for all dataset but for separated data. Indeed the data set was separated in the wanted number of intervals. Thus approximation in each decreased data set was applied and the plot was done. The plot was a good illustration of the fact that one divided the data set and that we plotted the data in a Piece Wise way.

Fourier Approximations - Nowadays, the Fourier Transform (FT) is a major tool in all technical domains, raging for image processing to telecommunication. Originally, by computing the FT of a function, the user decomposed a function of time into the frequencies that made it up. The Fourier transform of a function of time is represented in the frequency domain. However, the FT is not limited to functions of time, but in order to have a unified language, the domain of the original function is commonly referred to as the *time domain* while the domain of the FT is referred to as the *frequency domain*. For many functions, one can define an operation that reverses this: the inverse Fourier transformation of a frequency domain representation combines the contributions of all the different frequencies to recover the original function of time. The fact that one has the possibility to freely oscillate between the time and frequency domain enables the user to compute certain operation which a demanding in the time domain more easily in the frequency domain and then to switch back to the time domain to get the final output. However, in the frame of this project, it was possible to take advantage of the fact the Fourier Series are a way to represent a function as a superimposed sum of simple sine waves with different frequencies. More formally, Fourier Series decompose any **periodic function** into a superimposed sum of complex exponentials. The discrete-time Fourier transform (DFT) is a periodic function, often defined in terms of a Fourier series. The Z-transform, another example of application, reduces to a Fourier series for the important case $|z| = 1$. There were several different formulae that one could use to compute the DFT. However, as mentioned in the *dataset* paragraph, the input could only be real numbers. This made possible to simplify the formulas. Finally, the Fourier Series were given by the following formula:

$$f_n = a_0 + \sum_{k=1}^{N-1} a_k \cos\frac{n\pi k}{N} \ + b_k \sin\frac{n\pi k}{N} + a_N \cos \pi n \tag{13}$$

Using the (real-valued) formula for f_n:

$$a_k = (\frac{1}{N}) \sum_{k=-N+1}^{N} f_n \cos\frac{n\pi k}{N} \tag{14}$$

$$b_k = (\frac{1}{N}) \sum_{k=-N+1}^{N} f_n \sin\frac{n\pi k}{N} \tag{15}$$

Here N is the number of sample input data, k is the index of the k-iest Fourier coefficient and n the n-st approximation of the discrete function f.

Fast Fourier Transforms (FFT) - On the path to find a way to get the Fourier coefficients, a code to perform FFT was written. During the implementation of the Fourier approximation part, it was useful to compare the FFT values and the ones given by the Fourier coefficient. Since it was nice to see on the same graph the values of the function in the time and in the frequency domain, the decision was made to keep the FFT part in the code even if it was not used for approximation purposes but more as an additional information piece about the input data. Furthermore, the FFT is widely used tool in numerical approximation of integrals. To compute the FFT, two different methods could be used by the program depending on the nature of the input. Indeed, when one observed that the input length was a power of two, the Radix 2 or Cooley–Tukey FFT Algorithm code, which is the most commonly used to compute FFTs, was used whereas in all the others cases, in which the input was of arbitrary size Bluestein's algorithm was used. The advantage of performing different algorithm depending on the characteristics of the input was to be able to decrease the complexity of the code. Bluestein's FFT algorithm or chirp-z-transform algorithm could be used to compute prime-length DFTs in $\mathcal{O}(N \lg N)$ operations. The mathematical details about Bluestein's algorithm are shown below.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} \qquad k = 0, \ldots, N-1.$$

Fig. 1: FFT Algorithm

$$\begin{aligned} X_{N_2 k_1 + k_2} &= \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{N_1 n_2 + n_1} e^{-\frac{2\pi i}{N_1 N_2} \cdot (N_1 n_2 + n_1) \cdot (N_2 k_1 + k_2)} \\ &= \sum_{n_1=0}^{N_1-1} \left[e^{-\frac{2\pi i}{N} n_1 k_2} \right] \left(\sum_{n_2=0}^{N_2-1} x_{N_1 n_2 + n_1} e^{-\frac{2\pi i}{N_2} n_2 k_2} \right) e^{-\frac{2\pi i}{N_1} n_1 k_1} \end{aligned}$$

Fig. 2: Cooley–Tukey FFT Algorithm

$$
\begin{aligned}
X(k) &= \sum_{n=0}^{N-1} x(n) W^{-kn} W^{\frac{1}{2}(n^2+k^2)} W^{-\frac{1}{2}(n^2+k^2)} \\
&= W^{-\frac{1}{2}k^2} \sum_{n=0}^{N-1} \left[x(n) W^{-\frac{1}{2}n^2} \right] W^{\frac{1}{2}(k-n)^2} \\
&= W^{-\frac{1}{2}k^2} (x_q * w_q)_k,
\end{aligned}
$$

Fig. 3: Bluestein's FFT Algorithm

IV. Usage

In order to be able to easily share the code between the two members of the group, the version control system GitHub has been used. To access to the repository containing the program one simply has to open a terminal window and to type $gitclonehttps : //github.com/dcleres/PCSC2017\_Group5.git$". By doing so the user is able to download the project to the newly generated folder $PCSC2017\_Group5$. The program could be used simply by importing the project repository in CLion and to run the project. Once the project was launched the user had the possibility to choose between executing the program by using manual inputs into the terminal (by hitting 1) or using the information saved in the config.dat file (by hitting 2). However, both ways of doing lead to the same functionalities. For the user that chose the manual entries the usage was straightforward and guided by the content on the terminal window which explicitly mentioned how the user could first choose between using $\sin \pi x$ or $\sin$ x+ $\cos 3\pi$x as the function to interpolate. In the next step, the user was asked to hit 1. to perform a Least Squares data approximation, type 2. to appreciate the graphs of Fourier data approximation, 3. to compute the Lagrange polynomial data approximation, 4. for the PieceWise Least Squares data approximation and finally 5. for the PieceWise Lagrange data approximation. The config.dat file contained four lines. The first field, labeled with *Approximation Method*, should contain a number between 1 and 5. In this line, the user could choose between the five possible approximation techniques. Moreover, the second field was used to specify which function the user wanted to interpolate. The third and fourth fields were not necessarily used by all the approximation methods. For instance, the Fourier approximation did not need these last two lines so one could give random numbers in these containers (as long as there were some values). This was also true for the Lagrange approximation code. The other methods, like Least Squares and Piece Wise Lagrange, needed the first three lines in order to specify the degree of the polynomial function to interpolate. Finally, Least Squares approximation by pieces needed all the four lines. In this specific case, all four fields were necessary since the method needed to know the degree and the number of intervals to compute. By doing all the steps as mentioned in the two previous paragraphs all the programs should work well and give a qualified approximation of the function. During the execution of the program, the user had to follow the instructions on the terminal. However, it is important to know the pipeline of execution, meaning that once everything was entered the program took a few seconds to compute the answer. Following to this step, a graph appeared on the screen for 20 seconds. Once the first graph disappeared the testing part of the program was launched and showed some graphs which only appear 5 seconds to increase the execution speed of the program. In normal execution, 5 graphs should successively appear (corresponding to each interpolation method). However the different with the graphs before was that this time the graphs only compared the expected data with the interpolation results. As a final step, once the graphs disappeared the terminal gave the last information about the accuracy of the model by displaying of the accuracy of the predictions in comparison to the real data (with a tolerance of $\epsilon = 0.2$).

V. Results

LS Approximations Plot of the default points and the approximations points using the LS method: Using the LS

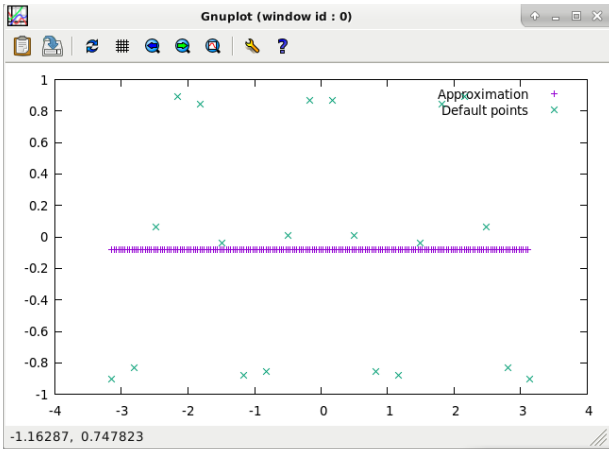

Fig. 4: Least Square approximation of degree 1 with f(x)= $\cos \pi$x

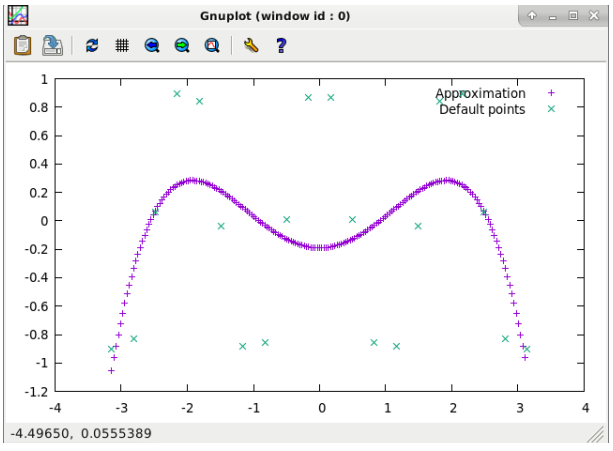

Fig. 5: Least Square approximation of degree 4 with f(x)= $\cos \pi$x

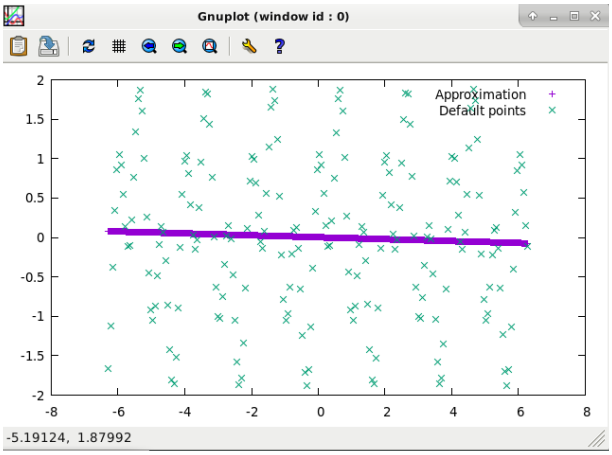

Fig. 6: Least Square approximation of degree 1 with f(x)= sin x+ cos3πx

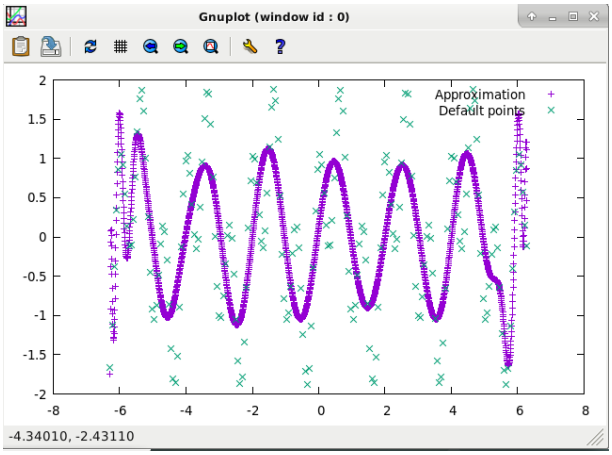

Fig. 7: LS approximation of degree 50 with f(x)= sinx +cos3πx

approximation, once the program was launched, the user had to choose the degree of LS she/he wanted. By choosing the degree, the user was able to influence the shape of the curve to interpolate. For instance in fig.4 and fig.6 the user chose to a curve degree 1 (straight line): this was the linear approximation. The resulting approximation was a straight line through the middle of the oscillations. This was logical due to the fact that the function tried to approximate was a periodic function with a line. For fig.5, the approximation was bad because the user chose a lower degree than needed to perform an appropriate approximation. Indeed, as the function has seven minima, meaning that the degree must be equal or higher than seven in order to get a good interpolation. Fig.7 was an example of the power of LS approximation once an appropriate degree was given.

Lagrange Approximations Fig.8 showed the Lagrange approximation using the whole given data set. Considering that the approximation curve and control curve were very close to each other, it was able to demonstrated the power of the Lagrange approximation. Considering the large number

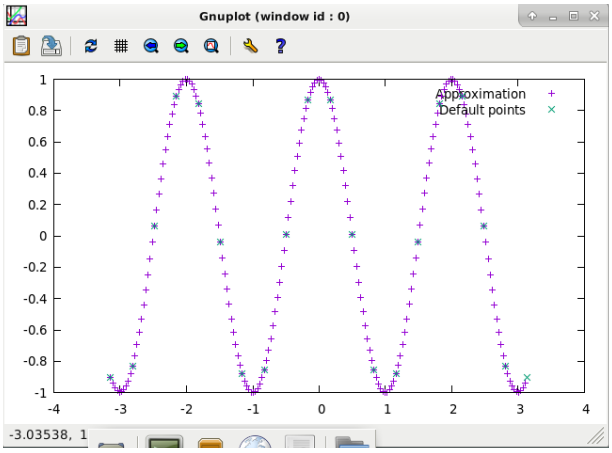

Fig. 8: Lagrange approximation with f(x)= cosπx

number of points to interpolated and the special shape of the function, the approximation was highly satisfactory. In the practice, th user only had just to select the Lagrange method.

Piece Wise Least Square Approximations In addition, the

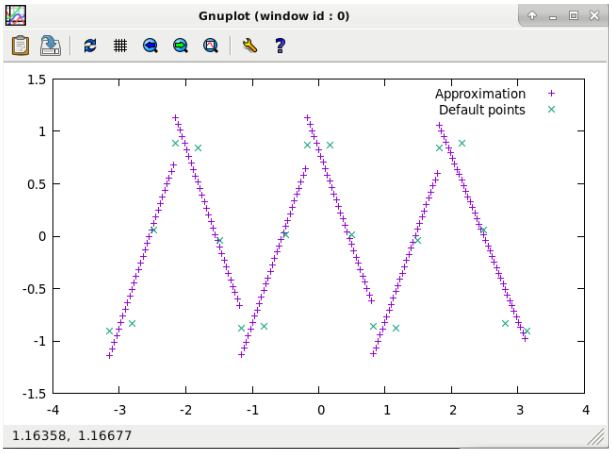

Fig. 9: Piece Wise Least Square approximation of degree 1 with 6 intervals with f(x)= cosπx

Piece Wise Least Square approximation was coded. Fig.9 show the linear piece wise approximation and Fig.10 show a piece wise polynomial approximation using Least Square method. Theses two approximations were really accurate. which is quite accurate. To make this figure, the user need to enter the degree of Least Square and the number of intervals he desire. Fig.11 illustrated that Piece Wise Least Square method is a powerful one when you chose the good parameters.

Piece Wise Lagrange Approximations Fig.12 and Fig.13 demonstrate the great accuracy of the Piece Wise Lagrange method. During the testing changing the number of intervals never changed the accuracy.

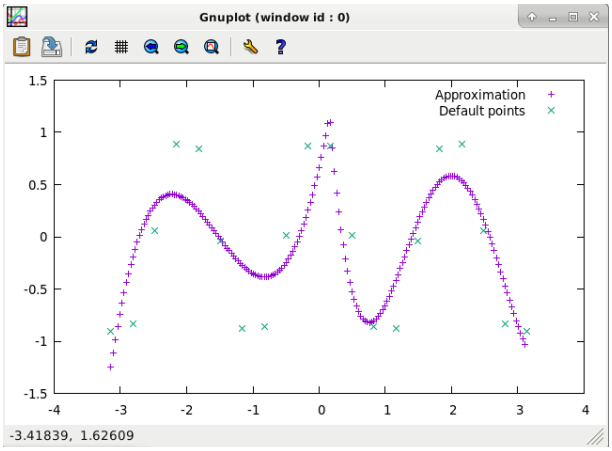

Fig. 10: Piece Wise Least Square approximation of degree 4 with 2 intervals with f(x)= cosπx

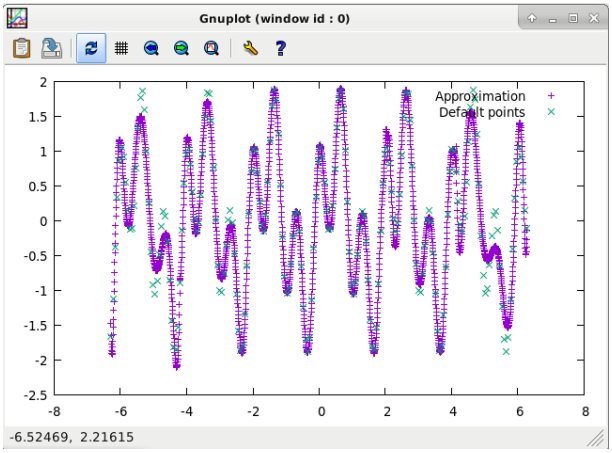

Fig. 11: Piece Wise Least Square approximation of degree 8 with 20 intervals with f(x)=cosπx + sin3πx

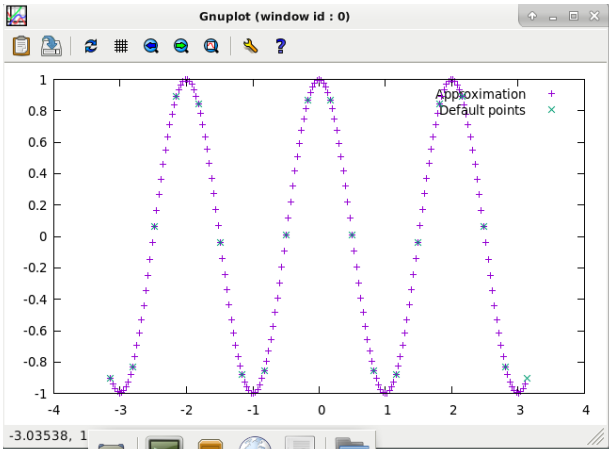

Fig. 12: Piece Wise Lagrange approximation with 4 intervals with f(x)=cosπx

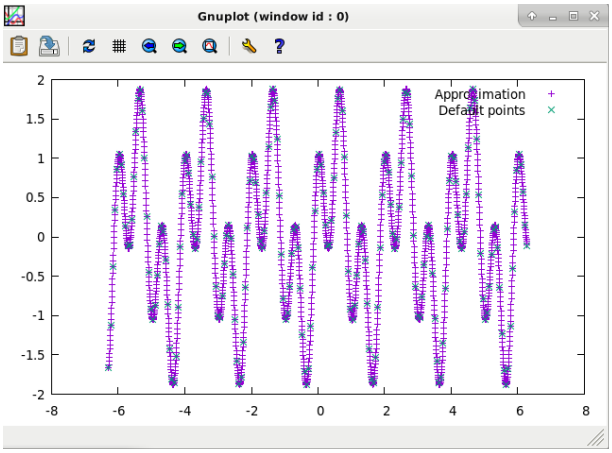

Fig. 13: Piece Wise Lagrange approximation with 8 intervals with f(x)= sinπx+cos3πx

VI. Discussion

The result section permitted to prove partially that the used approximations where correct simply by looking at the plots of the interpolated data versus the real data. Hence, this represented the first test. It was very practical to be able to plot the data since without wasting too much time the user could immediately see if estimation was good. The plots obtained were performed by an open source program called GNU-plot. However, one could not directly use GNU plot directly with the computed values in the C++ program. Fortunately, an interface class, to make the link between the program and GNU-plot was available on the Internet. The code of this interface was not optimal so the group chose to spend some time to change a few things (taking arguments by constant references, ...) in order to optimize the runtime. Furthermore, the second mean to assess the correctness of the code was to print the computed coefficient on the terminal for the single polynomial approximation, Piece-wise polynomial approximation with no derivative continuity and Least square approximation. Finally, some tests were also performed by computing the different regression on MatLab and to compare the obtained results with the computed data of the C++ code. This final, way of testing was more difficult since the approximation never gave the exact same output so the code had to be able to tackle this kind of problem by accepting a small range of fluctuation between the ”theoretical” MatLab output and the C++ output.

VII. Possible Improvements

Due to time limitation, it was difficult to do all the things that the groups wanted to accomplish. Here is a list of some elements that could have been done with more time. First, more mathematical ways of making approximation would have been performed. (Chebyshev approximation, Remez's algorithm...). When the user asked for a degree that was too high in comparison to the optimal degree to approximate the data, the methods had resolution problems and the interpolated

points were not as accurate as for the optimal degree's solution at the borders of the intervals. One could have thought about a way to prevent this kind of problem by computing several interpolations with a different degree and to take the answer that minimized the errors with the real input data and the interpolated data, this would guarantee to always give an optimal solution. There also exist alternative implementations for the resolution method (in addition to the Gaussian elimination that was made). It would have been possible to make tests to choose which one of the solvers was the most appropriate to a given situation. Furthermore, only real values could be interpolated. It could have been interesting to extend the models to the complex inputs. Finally, the last thing was the provide solution worked only for unidimensional datasets. By using the generalization of the formulas to higher degrees it would have been possible to approximate any data in higher dimensions.

VIII. Appendix

References

[1] Jeff Cogswell, D. Ryan Stephens, Jonathan Turkanis, C. D. (2009). C++ Cookbook. O'Reilly Media.

[2] Pitt-Francis, Joe, Whiteley, J. (2012). Guide to Scientific Computing in C++. Springer-Verlag London.

[3] Wikipedia. (2017). Fourier Series. Retrieved from $https://en.wikipedia.org/wiki/Fourier_series$

[4] Wikipedia. (2017). FFT. Retrieved from $https://en.wikipedia.org/wiki/Fast_Fourier_transform$

[5] Wikipedia. (2017). Lagrange. Retrieved from $https://en.wikipedia.org/wiki/Lagrange_polynomial$

[6] Wikipedia. (2017). Least Squares. Retrieved from $https://en.wikipedia.org/wiki/Least_squares$

IX. MERRY CHRISTMAS AND HAPPY NEW YEAR