Numerical Methods
Assignment 03

Georgios Christos Chouliaras (2592496)

April 2, 2017

Exercise 1

The function

The function `Chouliaras_assignment3_exercise1(data,g)` fits a linear combination of exponential functions to data.
The inputs are a (m x 2) matrix which consists of the `data` and a vector of initial guesses for the exponents `g`.
The output consists of a vector `l_best` which includes the calculated optimal exponents l_i, a vector `c_best` with the calculated optimal constants C_i as well as the residue. Furthermore, the function produces a picture of the fitted function and the data.
As an example, in order to fit a linear combination of exponentials to `data1` data set of the `expo-examples.mat` file using as vector of initial guesses `g = [-1 -2 -3 -4 -5]` one should type: `[l_best,c_best,residue] = Chouliaras_assignment3_exercise1(data1, [-1 -2 -3 -4 -5])`

The method

The function that is used to fit the data is of the form $f(x) = C_1 e^{\lambda_1 x} + C_2 e^{\lambda_2 x} + \ldots + C_n e^{\lambda_n x}$. The number of exponents n is determined by the length of the initial guesses vector `g`. The function determines the optimal exponents λ_i and constants C_i in order for function f to be the best fit of the data. Here the best fit refers to the least squares, in other words minimizing the $|Ax - b|^2$ (the Euclidian norm).

The procedure is as follows. An anonymous function is created in which the matrix A is constructed. Then for fixed $\lambda = (\lambda_1, \ldots, \lambda_n)$ the optimal coefficients C_i are calculated using the least square method. Next, using the `fminsearch` function the residue is minimized $residue = ||Ax - b||^2$ and the best exponents λ_i are calculated. By minimizing the residue which is the distance between the points and the function, the function that produces the best fit is found. Finally, the fitted function along with the data are plotted. It should be noted that in `fminsearch` we use the squared norm for the residue, since it seems more natural choice than minimizing a square root, i.e. $||x||_2 = \left(\sum_i^n |x_i|^2\right)^{1/2}$ is the 2-norm, so we get the square in order to get rid of the square root.

Discussion & Illustration

It should be noted that the quality of the fit is affected in a great extent by the vector of the initial guesses. A bad choice of initial guess vector can lead to a low quality fit which does not follow the form of the data.

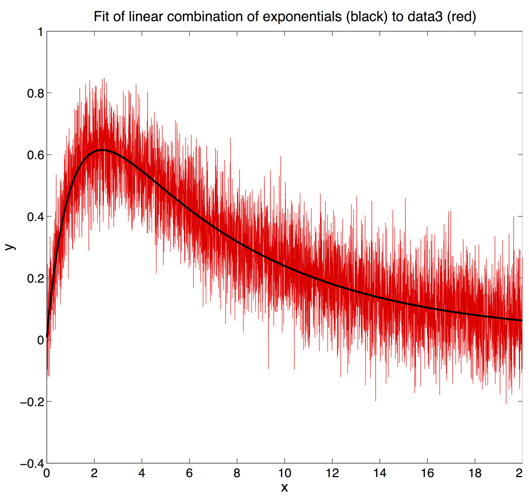
In order to identify the hidden exponents in the signals several different lengths for the initial guess vector were tested. To achieve this, a function `efficiency_assignment3_exercise1(data)` was built which stores the outputs of `Chouliaras_assignment3_exercise1(data, g)` for each data set contained in `expo-examples.mat` as well as for initial guess vectors `g` of variable length from 1 to 20. In this way we can find how the residue changes when the length of the initial guess vector increases.
For `data1` as data set the initial guesses were created by `linspace(0,1,z)` for z indicating the length of the initial guess vector. The results for the first four iterations can be seen in table 1.

z	residue
1	20.6597
2	10.1358
3	10.1357
4	10.1296

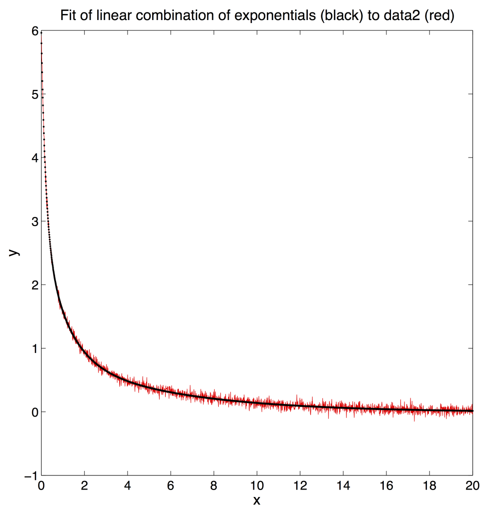
Table 1: The calculated residue for `data1` using vector of initial guesses with variable length indicated by z.

In table 1 can be seen that the residue decreases significantly after increasing the vector length from 1 to 2, however after this point the reduction in the residue is very small. Furthermore, it was observed that for $z = 3$ one of the 3 C_i coefficients is almost zero, while for $z > 3$ the coefficients started to have very large values, something that indicates overfitting. In the case that $z = 3$ the residue is 10.13, the optimal coefficients and exponents are $C_{best} = [-9.29 \cdot 10^{-4}, 1.39, 1.72]$ and $\lambda_{best} = [0.019, -3.14, -0.36]$ and as it can be seen, the first coefficient is almost zero, while the next two are significant. It can be seen that the λ which corresponds to the insignificant C_1 is also much smaller than the rest. The exponents corresponding to these significant coefficients are the two *hidden* exponents in the signal and their values are those corresponding to the λ_{best} vector of exponents. A plot of the fitted function on `data1` can be seen in figure 1c.
For `data2` the initial guess vector of the form `linspace(-4,0,z)` was used and it was found that the residue does not improves significantly after $z = 4$ and also there is a similar behavior in the coefficients' values. For vector of initial guesses with length 4 the residue is 5.06, the optimal coefficients are $C_{best} = [2.96, 2.06, 0.93, -0.008]$ and the optimal exponents $\lambda_{best} = [-4.97, -0.96, -0.18, -6.73 \cdot 10^{-4}]$. In this case there are three hidden exponents and are these that correspond to the significant coefficients C_1, C_2, C_3. Again the exponent corresponding to the insignificant C_4 is much smaller compared to the other exponents. The fit of the function to `data2` can be seen in figure 1b.
For `data3` again the the initial guess vector is of the form `linspace(-4,0,z)` and after checking the residues for z from 1 to 20 it was found that for vector of guesses with length 3 the residue is 51.9 and it does not improve significantly by increasing z. Again for larger values of z the coefficients take very large values due to overfitting. For $z = 3$ the optimal coefficients are $C_{best} = [-1.01, 1.01, 0.0095]$ and the optimal exponents $\lambda_{best} = [-0.94, -0.15, 0.005]$ which means that there are 2 hidden exponents which correspond to the significant coefficients C_1, C_2 while the 3rd coefficient is almost zero. The fit of the function to `data3` can be seen in figure 1a.

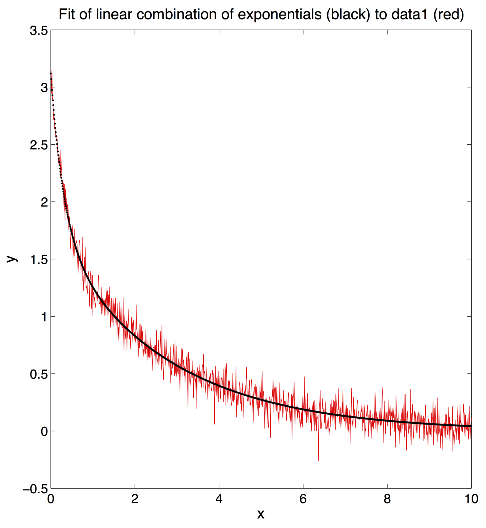
It is observed that in all figures the functions fit the data smoothly.



(a) Fitted function f to `data3`



(b) Fitted function f to `data2`



(c) Fitted function f to `data1`

Figure 1: Figures of the fitted functions f and the data sets `data1, data2, data3`.

Exercise 2

The function

The function `Chouliaras_assignment3_exercise2` determines the center of mass of a domain enclosed by an interpolated curve.
There is no input or output, but the function produces a message which instructs the user to put his hand on the rectangle on the screen and outline the circumference of his hand with mouse clicks. The user can either press *Yes* and proceed or *Cancel* to cancel the procedure. After the user has outlined his hand on the screen, the function interpolates the input points with splines to create a nice hand shape and then calculates the center of mass. Finally, the function produces a picture of the input points, the interpolated curve, the area inside the curve and a mark which indicates the center of mass.
The syntax of the function is the following: `Chouliaras_assignment3_exercise2`

The method

The function uses `ginput` in order for the user to outline his hand and then stores the input points in two vectors (x_i, y_i) for $i = 1, \dots, n$ where n is the length of the input vectors. Then it checks for consecutive clicks on the same point and deletes the duplicate points. Furthermore, in order for the area to be closed, the last elements of vectors x, y are set equal to the first element of the corresponding vectors, in other words $x_1 = x_{n+1}$ and $y_1 = y_{n+1}$. In order to interpolate the curve the arc-length parametrization is used with parameter $t : t_1 = 0$ and $t_{i+1} = t_i + \sqrt{(x_{i_1 - x_i})^2 + (y_{i+1} - y_i)^2}$. In this way a vector t of length n is produced and using the function `spline` we put splines through the points (t_i, x_i) and (t_i, y_i).
In order to calculate the center mass we do the following. The center of mass of a set $\Omega \subset \mathbb{R}^2$ has coordinates $(\int_\Omega x)/(\int_\Omega 1)$ and $(\int_\Omega y)/(\int_\Omega 1)$. In order to convert these two-dimensional integrals to one-dimensional we can use Green's theorem: $\oint_{\partial\Omega}[f_1 dx + f_2 dy] = \int_\Omega \left[\frac{\partial f_2}{\partial x} - \frac{\partial f_1}{\partial y}\right] dxdy$.
In order to calculate the center of mass coordinates we need to choose f_1 and f_2 in a way such as the difference $\frac{\partial f_2}{\partial x} - \frac{\partial f_1}{\partial y}$ to be equal to x, y and 1. Due to this, we conclude with 3 different choices for f_1 and f_2. For $\int_\Omega x$ we choose $f_1 = 0$ and $f_2 = \frac{x^2}{2}$, for $\int_\Omega y$ we choose $f_1 = -\frac{y^2}{2}$ and $f_2 = 0$ and for $\int_\Omega 1$ we take $f_1 = 0$ and $f_2 = x$. The aforementioned integrals are computed using the function `trapz` which makes use of the trapezoidal rule and with these, the center of mass is calculated.

Discussion & Illustration

In figure 2 can be seen a picture produced by the function. The red marks are the mouse clicks of the user which correspond to the input points (x_i, y_i), the black line is the interpolated curve, the cyan area is the area inside the curve and the x mark is the center of mass.

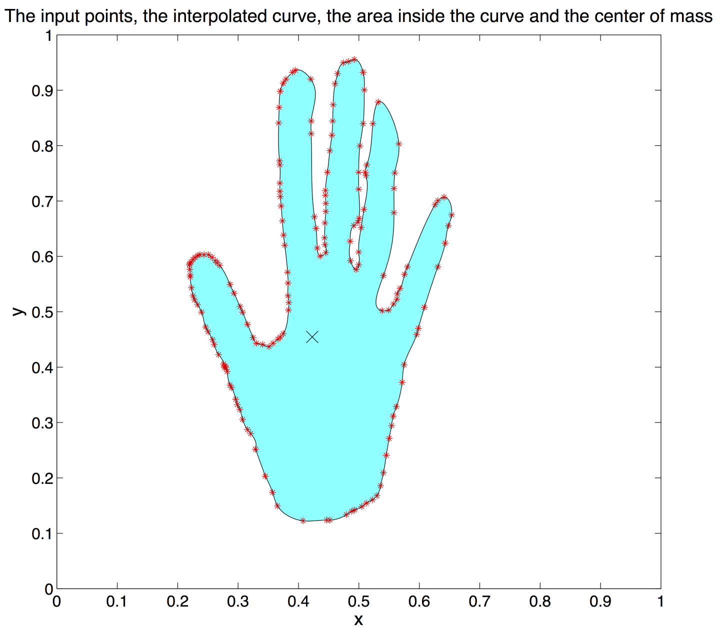


Figure 2: Figure of the input points (red marks), the interpolated curve (black line), the area inside the curve (in cyan) and the center of mass (X mark).

Exercise 3

The function

The function `Chouliaras_assignment3_exercise3(v,type,count)` calculates the discrete Fourier transform and the inverse transform of a vector.
The input consists of a vector `v` of length 2^n and a variable `type` which indicates whether the normal or inverse transform needs to be computed. The input variable `type` takes values `'nor'` for normal Fourier transform and `'inv'` for inverse transform. The input variable `count` is an optional input which denotes the initial length of the vector `v` and it is only used in the computation of the inverse FFT.
The output consists of a vector which contains the calculated FFT or inverse FFT.
As an example, in order to calculate the inverse FFT of the vector `v = [1,2,3,4,5,6,7,8]` one should type: `fft = Chouliaras_assignment3_exercise3([1,2,3,4,5,6,7,8],'nor')`

The method

The Finite Fourier Transform is used to extract frequency from periodic signals. Given a function f which is periodic on $[0, L]$, where L is the period, we can write it as a sum of exponentials $f(t) = \sum_{k\in\mathbb{Z}} c_k e^{\frac{2\pi ik}{L}t}$ where the Fourier coefficients c_k are found by $c_k = \frac{1}{L}\int_0^L f(t)e^{-\frac{2\pi ik}{L}t}dt$. We can numerically approximate Nc_{k-1} by turning the integral into a sum using the following formula:

$Nc_{k-1} = \sum_{j=1}^{N} f_j e^{-\frac{2\pi i}{N}(k-1)(j-1)} \equiv F_k$ where $f_j = f(t_j)$. Calculating $F_1, \ldots, F_N$ using matrix-vector multiplications takes $O(N^2)$ operations, however we can reduce the number of operations to $O(NlogN)$ by recombining the odd $(F_{f(1:2:N)})$ and even $F_{f(2:2:N)}$ elements in an iterative scheme where at each iteration the vector is halved, constructing the solution from the previous half-solutions. To achieve that, we call the function recursively using the following scheme (in symbolic notation): $F_{f(1:N)} = [F_{f(1:2:N)} + \omega^{(0:\frac{N}{2}-1)} * F_{f(2:2:N)}; F_{f(1:2:N)} - \omega^{(0:\frac{N}{2}-1)} * F_{f(2:2:N)}]$, where $\omega = e^{\frac{-2\pi i}{N}}$ and N is assumed to have length 2^p. The process ends when the first level is reached, in other words when the vector has length 1.
The inverse Fourier transform is calculated by $f_j = \frac{1}{N} \sum_{k=1}^{N} F_k e^{\frac{2\pi i}{N}(j-1)(k-1)}$. It needs to be noted that in this formula N in is the initial length of the vector, and in order to store it in our recursive algorithm, an extra optional input variable `count` is used.

Discussion & Illustration

In order to validate the algorithm, it was tested against the built in `fft` and `ifft` matlab functions. The results of functions were identical, verifying the efficiency of the function.

An important aspect of the program is the speed. Hence, in order to measure how fast the algorithm computes the FFT, several different lengths for the input vector were tested and the running time was recorded. The results of these tests for input vector of length 2^n are can be seen in figure (3). In this figure it is observed that the running time is almost 0 until $n = 8$ which means a vector length equal to 2^8 and after this it starts increasing at an exponential rate.

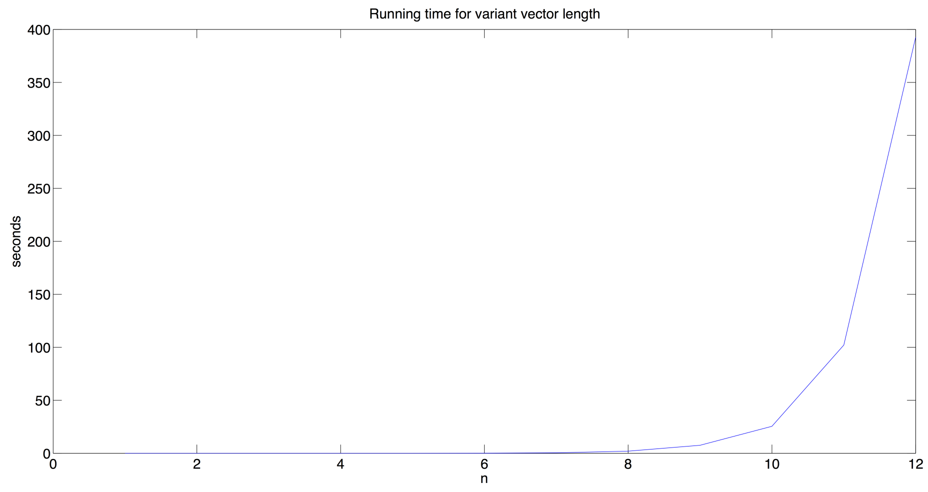
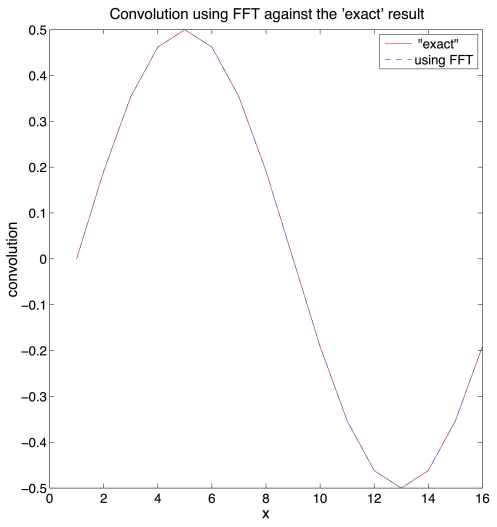


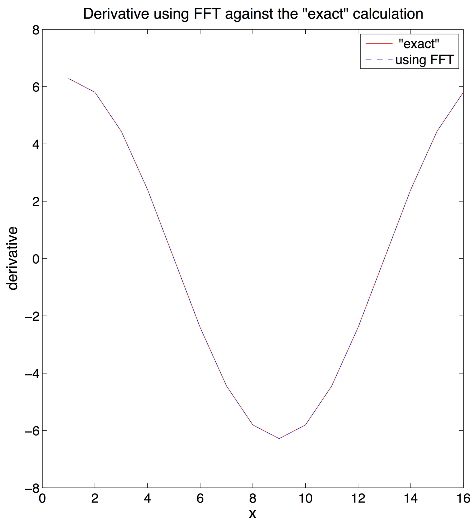
Figure 3: Running time in seconds, for increasing vector length 2^n.

Next, we define two approximately periodic functions f and g: $f(x) = sin(2\pi x)$ and $g(x) = cos(2\pi x)$ for which $L = 1$. Then, we define a vector `v` of length $N = 2^4$ and in order to determine the Fourier coefficients c_k for function f we do the following: we sample the function f on a uniformly space domain and we calculate the Fourier transform of the function evaluated at the points of the vector. Then, the coefficients $c_k(f)$ are computed by dividing the former result by N. We do exactly the same for function g and we obtain the coefficients $c_k(g)$. Then we can calculate the Fourier coefficients of the convolution using the formula $c_k(f \star g) = Lc_k(f)c_k(g)$. Now we can get the convolution of the functions by multiplying the coefficient of the convolution by N (since we divided by N before, to get the coefficients) and then we apply the inverse Fourier transform. In order to evaluate the result we calculate the convolution in paper as follows: $(f \star g)(x) = \int_0^L f(y)g(x-y)dy = \int_0^1 sin(2\pi y)cos(2\pi(x-y))dy = sin(\pi x)cos(\pi x)$. We compare the results by plotting the convolution calculated by hand against the result using FFT as it can be seen in figure 4a and it is observed that the results are identical, hence the convolution is calculated appropriately. It has to be noted here that the "exact" result is not really exact, as it was calculated using the matlab functions `sin` and `cos` which are also approximations, however it is referred as "exact" to be distinguished by the FFT-based result.

Next, we also want to compute the first and second derivatives of f using the discrete Fourier transforms. The fourier coefficients of the first and second derivative can be computed using the formulas $c_k(f') = \frac{2\pi}{L}ikc_k(f)$ and $c_k(f'') = -\left(\frac{2\pi}{L}\right)^2 k^2 c_k(f)$. Therefore the values f' and f'' can be obtained by taking the inverse FFT of the aforementioned Fourier coefficients. It has to be noted that since we assume that the input vector is of length 2^n we know that N is even, and hence we use the appropriate ordering as it is stated in the lecture notes. It is important to note here that when applying the inverse FFT to the coefficients we must also multiply by N, since we divided by N to get the coefficients. The code concerning the calculation of the convolution and the derivatives can be found in the additional matlab file `determine_assignment3_exercise3.m`.

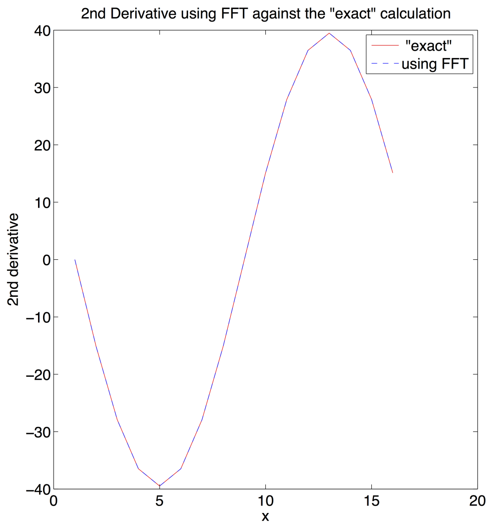
In order to check if the derivatives are computed appropriately, we can compute the derivatives of f in paper as follows: $sin(2\pi x)' = 2\pi cos(2\pi x)$ and $sin(2\pi x)'' = -4\pi^2 sin(2\pi x)$. Now we can compute the derivatives of the function evaluated at the vector `v` using matlab and plot them against the computed derivatives using FFT in order to evaluate the results. The plot of the first and second derivatives using the FFT against the derivatives using the paper calculation can be seen in figures 4b and 4c. The functions are identical and hence the computation of the derivatives is correct.





(a) Plot of convolution with the aid of FFT against the result calculated in paper.

(b) Derivative of $sin(2\pi x)$ against the derivative using FFT.



(c) 2^{nd} Derivative of $sin(2\pi x)$ against the derivative using FFT.

Figure 4: Comparison "exact" against "FFT-based" convolution and derivatives.