Assignment III
Numerical Methods

T.J. Oosterhuis

April 2, 2017

Exercise I

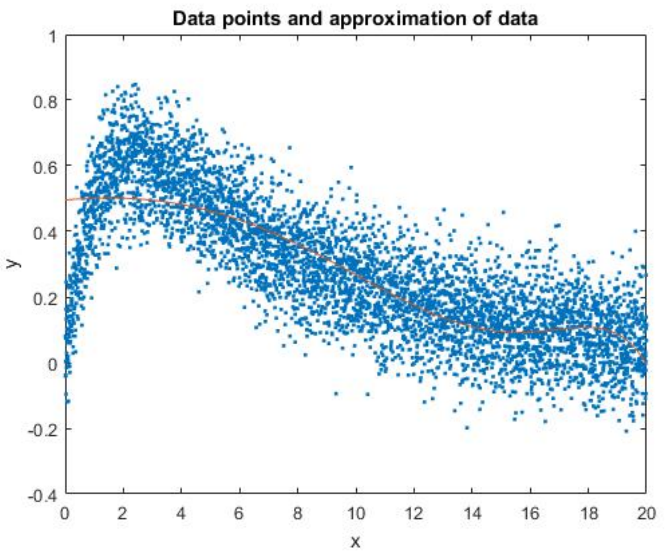
The function Oosterhuis_assignment3_exercise1 uses the least squares of residuals method to give a function that is an approximation of the data. The input is a bivariate dataset containg n points $(x_i; y_i)$ and a vector of initial gusses for $\lambda_1, .., \lambda_m$. An example are the arguments 'data3' and the vector $\begin{bmatrix}\lambda_1 & \lambda_2 & \lambda_3 & \lambda_4\end{bmatrix} = \begin{bmatrix}0.1 & 0.2 & 0.3 & -0.2\end{bmatrix}$, where data3 is a bivariate dataset. The output of the function with these inputs is a vector $\lambda = \begin{bmatrix}-0.13 & 0.84 & -0.86 & -0.39\end{bmatrix}$, a vector $C = \begin{bmatrix}0.32 & -0.11 & 0.01 & 0.35\end{bmatrix}$ and third the sums of the squared residues which is 59.90.

The functions gives an approximation for the solution of the equation $A * C = y$, where A is a matrix dependent on lambdas and x (the independent variable), C are some constants and y is the dependent variable. So the equation that's being solved is

$$\begin{bmatrix}\exp(\lambda_1 x_1) & \exp(\lambda_2 x_1) & \dots & \exp(\lambda_m x_1) \\ \exp(\lambda_1 x_2) & \exp(\lambda_2 x_2) & \dots & \exp(\lambda_m x_2) \\ \dots & & & \\ \exp(\lambda_1 x_n) & \exp(\lambda_2 x_n) & \dots & \exp(\lambda_m x_n)\end{bmatrix} * \begin{bmatrix}C_1 \\ C_2 \\ \dots \\ C_m\end{bmatrix} = \begin{bmatrix}y_1 \\ y_2 \\ \dots \\ y_n\end{bmatrix}.$$

The solution for lambda and C is construted such that $\sum_{i=1}^{n}(C_1 \exp(\lambda_1 x_i) + \dots + C_m \exp(\lambda_m x_i) - y_i)^2$ is minimal.

The reliability and precision of the computed lambdas, values voor C and the residue depends on the user input. When the initial guesses for lambdas are 'bad', the function can return a approximation for the data that fits the data quite bad. This is for example the case when for the dataset data3 the initial guesses for the lambdas don't contain a negative value. The plot of such an result is shown in the figure below. We see the sum of the squared residues is 76.10, which is also much higher than in the example above when a negative intial guess was used for one of the lambdas.



Figures 1 to 4 show the solution of the least square approximation along with the data points for four different data sets. This output was produced by first plotting the data and then plotting the independent variables against the solution of $A * C$, where the matrix A and the vector C both depent on the new values for the lambdas.

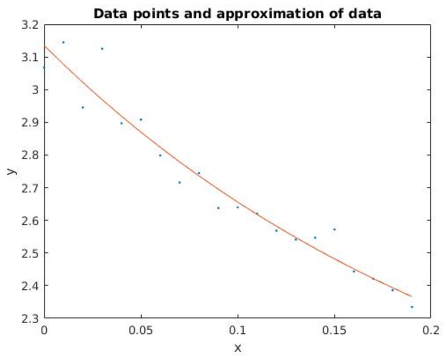


Figure 1: Data and least square approximatiotn for 20 data points.

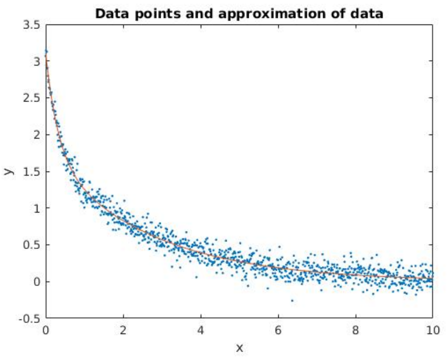


Figure 2: Data and least square approximatiotn for dataset 'data1'.

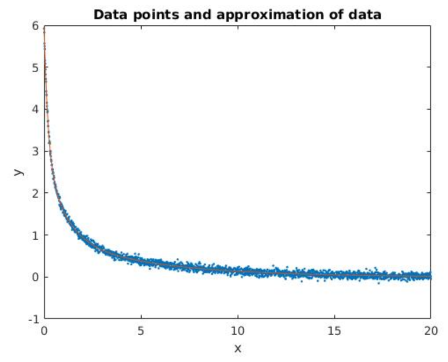


Figure 3: Data and least square approximatiotn for dataset 'data2'.

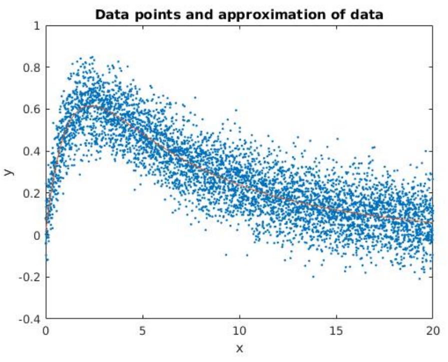


Figure 4: Data and least square approximatiotn for dataset 'data3'.

Exercise II

The function Oosterhuis_assignment3_exercise2 takes an n number of points. From this points a figure is made by connection the points using interpolation such that a closed curve C is constructed. The function should be called without arguments, a figure with the xy-plane instead will pop up where the user can click the points which will be the coordinates. As an example, the figure of a hand is drawn. The result of the points and the connecting lines is shown in figure 5.

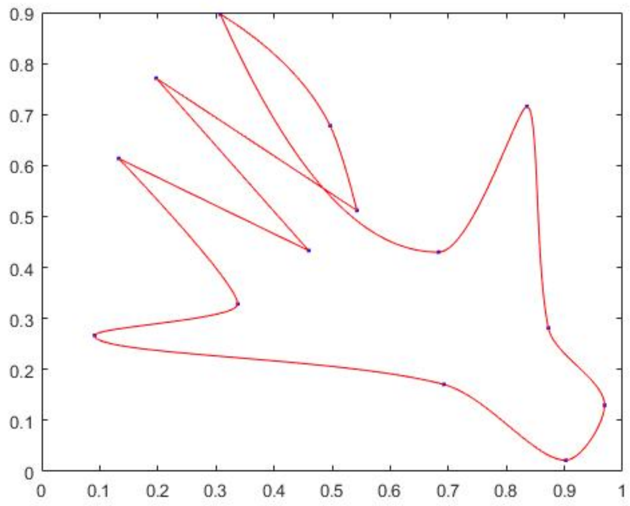


Figure 5

To determine the length of C, the cumulative sum of the square root of the elementswise differences of x squared plus the elementwise differences of y squared are taken. Using interpolation between every points (x_i, y_i) and (x_{i+1}, y_{i+1}) a line is constucted. The points along with the interpolated lines are plotted. The coordinates of the center of mass of the figure are equal to $\big((\int_\Omega x)/(\int_\Omega 1); ((\int_\Omega x)/(\int_\Omega 1)\big)$. The area of the enclosed domain can be calculated via Greens theorem: $\oint_{\delta\Omega}[f_1 dx + f_2 dy] = \int_\omega [\frac{\delta f_2}{\delta x} - \frac{\delta f_1}{\delta y}]dxdy$. Since the area doesn't depend on f_1, f_2, they can be chosen and are chosen in such an way the intergal is easy to calculate: $f_1 = 0, f_2 = x$. When these are filled in this gives $\int_\Omega [\frac{\delta f_2}{\delta x} - \frac{\delta f_1}{\delta y}]dxdy = \int_\Omega 1 dxdy$.

An example of the output of the function is showsn above in 5. The picture was produced by taking the points on the xy-plane as defined by the user. The points are stored in two vectors, one with the x-component and one with the y-component of every number. To make a closed curve, both vectors get an extra point at the end of the vector, being the same element as the first one.

Exercise III

The functions Oosterhuis_exercise3_assignment3_2 calculates the discrete Fourier transform or the inverse discrete Fourier transform of a vector. The function takes zero or a one, a zero indicates the discrete Fourier transform should be calculated and a one indicates its inverse should be calculated. The second function input is a vector of length 2^n. This is the vector of which the (inverse) discrete Fourier transform will be calculated. As an example the function is called with the arguments zero and the vector $x = [0; 0.707; 1; 0.707; 0; -0.707; -1; -0.707]$. The elements of this vector are values from the $\sin(x), x \in [0, 2\pi]$. The output of the function is the vector of the same length as x.

The function uses recursion. When the vector is called with a vector of length one, the function output will be just this vector. When the function is called with a vector of length 2^n, the function will make two vectors: a vector of the even elements and a vector of the odd elements of the original vector. From both these vectors the Fourier transform is calculated using the function Oosterhuis_exercise3_assignment3_2 again with as arguments the new vector and the one or zero unchanged. Note that this is the step where the recursion takes place. After the discrete Fourier transform of the vectors with the even and the odd elements are computed, we call them y_even respectively y_odd. The vector y_even is multiplied with $\omega = \exp(-2\pi i/N) * v$, where v a is the vector $\begin{bmatrix} 0 \\ 1 \\ \dots \\ N \end{bmatrix}$ and N is the length of both the vectors y_even and y_odd. A vector with two elements is created: $y = \begin{bmatrix} y_{odd} + \omega y_{even} \\ y_{odd} - \omega y_{even} \end{bmatrix}$. This vector is the function output and will be the Fourier transform of the given vector. The inverse discrete Fourier transform is calculated the same way, only using $\omega = \exp(2\pi i/N) * \begin{bmatrix} 0 \\ 1 \\ \dots \\ N \end{bmatrix}$, so without the minus sign in the exponent.

The function does not work whenever the length of the input vector is not a power of two, since at a certain moment in the recursion the length of the vector that needs to be splitted is not even. Another restriction of the function is the fact that it uses the discrete Fourier transform, not the continuous Fourier transform. Hence a function is only being evaluated at a finite number of points, giving an error.

As an example the discrete fourier transform of the function $\sum_{n=1}^{k} n \cos(n\omega t), \omega = 20\pi$ is computed. This function is plotted in figure 6. The discrete Fourier transform of this function is shown in figure 7. The discrete Fourier transform is plotted in the fequency domain and shows for each given frequency how much of the signal lies within that frequency. The figure of the discrete Fourier transform was produced by evaluating the time function on 128 points between in the interval $0; 0.5]$. From this vector the discrete Fourier transform was computed. This gives a vector of complex numbers. For every complex number the magnitude is calculated. To make the frequencies one sided this was only done for the first half of the vector, the other elements are deleted so this results in a vector of length 64. To scale the amplitudes, every elements in this vector is devided by 128, the number of oberservations. This final vector of length 64 is plotted against a vector with the different frequencies, in this case 1 to 64 with stepwidth one.

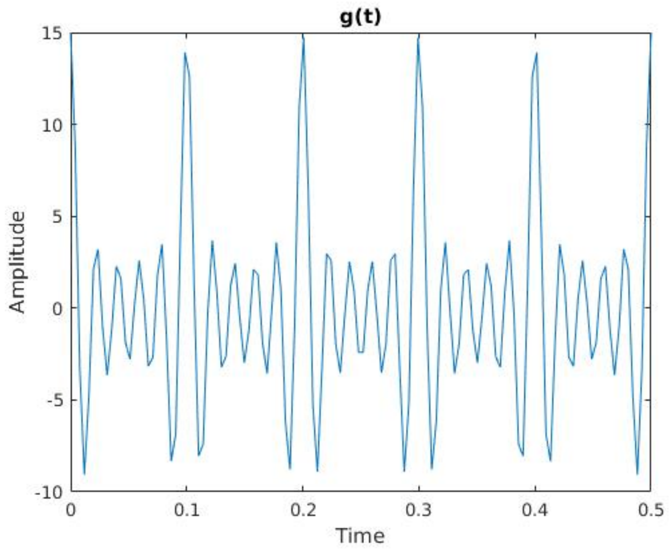


Figure 6: $g(t) = \sum_{n=1}^{k} n\cos(n\omega t), \omega = 20\pi$

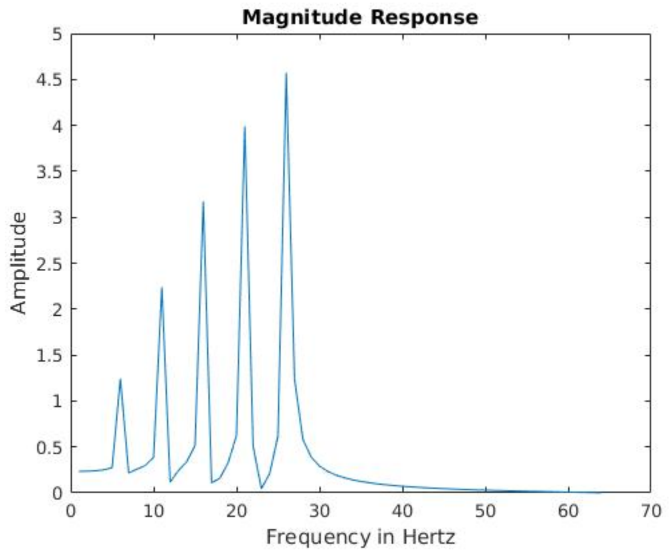


Figure 7: Discrete Fourier transform of $g(t)$ in frequency domain.

For large n, the function is not very fast. For the example above, where the function was evaluated at 128 points, it took about 40 seconds to make the plot. We take a look at the formula for the discrete Fourier transform $F_m = \sum_{k=1}^{N} f_k \omega^{(k-1)(m-1)}$. Since ω is unknown these are all linear equations so computing the Fourier transform is just matrix multiplication. This implies the number of operations needed is $O(N^2)$, which is way to much to use it for larger datasets. To reduce the number of computations $F_f(m)$ is splitted in a function on the even elements and a function on the odd elements of the input vector: $F_f(m) = F_{f_{odd}}(m) + \omega_{N/2}^{m-1} F_{f_{feven}}(m)$. This allows to interpret $F_{f_{odd/even}}$ as twice repeated vectors. Using this equation the number of operations will reduce to $O(N\log_2 N)$

The convolution of two functions f and g evaluated at 2^n number of points can be computed by taking the inverse Fourier transform of the product of the Fourier transform of both functions. This method is used in Oosterhuis_assignment3_exercise3_3. This function can take two function handles and will evaluate these functions at eight number of points or the function can take vectors which consist of the evaluation of a function in 2^n number of points.