Assignment 3

Introduction

Interpolation is the process of fitting a curve through a set of data. This can be done in various ways, for which the required smoothness is of importance. For example, linear regression could be used to fit a straight line through the data. Or the method of 'smoothing' to ensure a smooth curve through the data set. Piece-wise linear function can be used to fit linear functions to segments of the curve, which can be seen as gluing straight lines together. This is definitely the easiest option, but it is not smooth. Splines is another option, which consists of piece-wise gluing 3rd order polynomials that agree up to the 2nd derivative. For this assignment the focus will be on exponential curve fitting, which uses the least square method to find the best coefficients. The least square method minimizes the sum of the squares of the residuals of the points from the curve.[1]

Syntax

This function uses three input arguments, the data for which a curve has to be fitted, in the form of a $(m \times 2)$ matrix, a vector of initial guesses for the exponents – this has to be a $1 \times n$ matrix – and the maximum number of function evaluations of the used `fminsearch` function. The last argument is not required and, if not present, will default to 200 times the amount of input lambdas.
The output are two vectors, displaying the calculated optimal exponents (λ_i) and the calculated optimal constants (c_i). Furthermore, the residue is displayed as well. Finally, a picture is produced of the data and the calculated fit, $f(x)$, through the data points.

Description of the Method

The data are fit with a function of the following form:

$$f(x) = C_1 e^{\lambda_1 x} + C_2 e^{\lambda_2 x} + \ldots + C_n e^{\lambda_n x} = y$$

We can rewrite this as a set of linear equations like so:

$$f(x) = \underbrace{\begin{bmatrix} e^{\lambda_1 x_1} & \ldots & e^{\lambda_n x_1} \\ \ldots & \ldots & \ldots \\ e^{\lambda_1 x_M} & \ldots & e^{\lambda_n x_M} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} C_1 \\ \ldots \\ C_n \end{bmatrix}}_{C} = y$$

Let us name the matrix A and the vector of constants C, we can now say $AC = y$. Since x and y are given, and the λ values are calculated by the function using the

[1]van den Berg, J.B., Planqué, B., 11 January 2017, Lecture Notes on Numerical Mathematics

initial guess, only the constants are unknown. We can rewrite the formula to $C = A \backslash y$. The constants can thus be computed as everything is else known.
We measure how well the resulting line fits by the residual. This is given by $R(\lambda) = \|AC - y\|$, the lower the better. It is uncommon that the initial guess of lambdas has the best fit, thus we need to try different values of λ until the residue is minimised. This can be done using the MATLAB function `fminsearch`. This method takes two input variables: a function to minimise and an initial guess. The function does take more options, as it is an advanced function, but we only used the maximum amount of function evaluations – which can be chosen by the user in our method.
We want to minimise the residue. If we write this out in full form we get the following:

$$g(\underline{\lambda}) = \|AC - y\| = \left\| \underbrace{\begin{bmatrix} e^{\lambda_1 x_1} & \dots & e^{\lambda_n x_1} \\ \dots & \dots & \dots \\ e^{\lambda_1 x_M} & \dots & e^{\lambda_n x_M} \end{bmatrix}}_{A} \underbrace{\left(\begin{bmatrix} e^{\lambda_1 x_1} & \dots & e^{\lambda_n x_1} \\ \dots & \dots & \dots \\ e^{\lambda_1 x_M} & \dots & e^{\lambda_n x_M} \end{bmatrix} \backslash\, y \right)}_{C(=A\backslash y)} - y \right\|$$

The vectors of λ is the input, the variables x and y are known. This function is easy to implement in MATLAB, as it is built for vector operations. Combining the function `exp` for exponents, and the pair of vectors, gives us the following MATLAB implementation for matrix A:

$$\texttt{exp}\left(\begin{bmatrix} x_1 \\ \dots \\ x_m \end{bmatrix} \begin{bmatrix} \lambda_1 & \dots & \lambda_n \end{bmatrix} \right)$$

The rest of the function g is trivial to implement (see Appendix code line 27).

Example and Illustration

Applying interpolation onto the first given data set with initial guesses being $\lambda_1 = -1, \lambda_2 = -2, \lambda_3 = -4$ for the λs gives us the output below and Figure 1 (see next page).

```
>> exp_lin_comb_fit(data1, [-1 -2 -4])
constants : 1.9345      3.0411     0.28064
lambdas   : -0.327742     -1.93149     -51.5458
residue   : 6.2465
```

As can be seen, the values of the constants (C) are, $C_1 = 1.9345, C_2 = 3.0411, C_3 = 0.28064$, the values of the λs are $\lambda_1 = -0.327742, \lambda_2 = -1.93149, \lambda_3 = -51.5458$, the found residue is evaluated to be $R = 6.2465$. This would result in the following function:

$$f(x) = 1.9345e^{-0.327742x} + 3.0411e^{-1.93149x} + 0.28064e^{-51.5458x}$$

The MATLAB function plots the given data set in black dots, and the fitted curve in a red line. As can be concluded from Figure 1 this line is a close fit to the data.

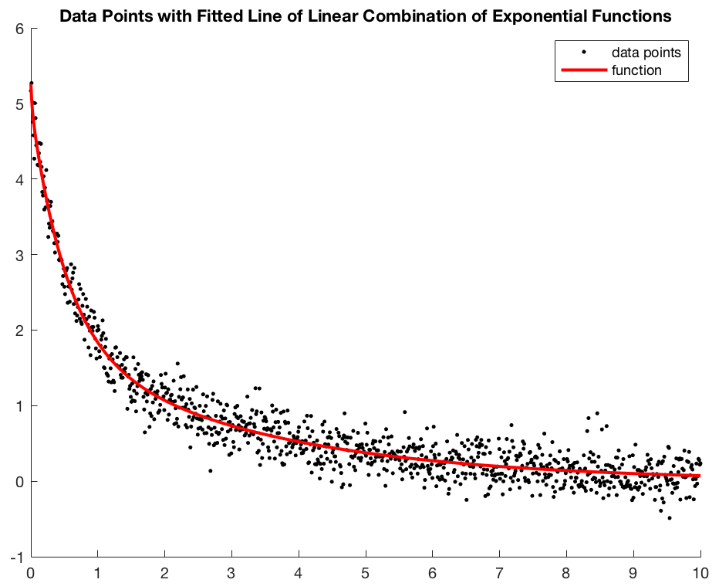

Figure 1: Graphical Representation of Interpolation on the First Data Set with Initial Guesses $\lambda_1 = -1, \lambda_2 = -2, \lambda_3 = -4$.

Advantages and Drawbacks

Some of the advantages of using the least square method to fit a curve to data is that it is very easy to use and to understand. Furthermore, calculations are very fast. There are a few limitations to using this function. First of all, the program doesn't work if the initial guesses for the λs is given as a $M \times 1$ matrix, it only works if this is given as a $1 \times M$ matrix, this is because these are incorrect dimensions for the matrix multiplication used in this method.
Furthermore, this function is very dependent on the input from the user for its success probability. The user has to decide how many λs to give and the values of these λs. For example, it was found that if bad initial guesses for the λs were given, this would not give an accurate fit to the data. This is displayed in Figure 2 and 3.

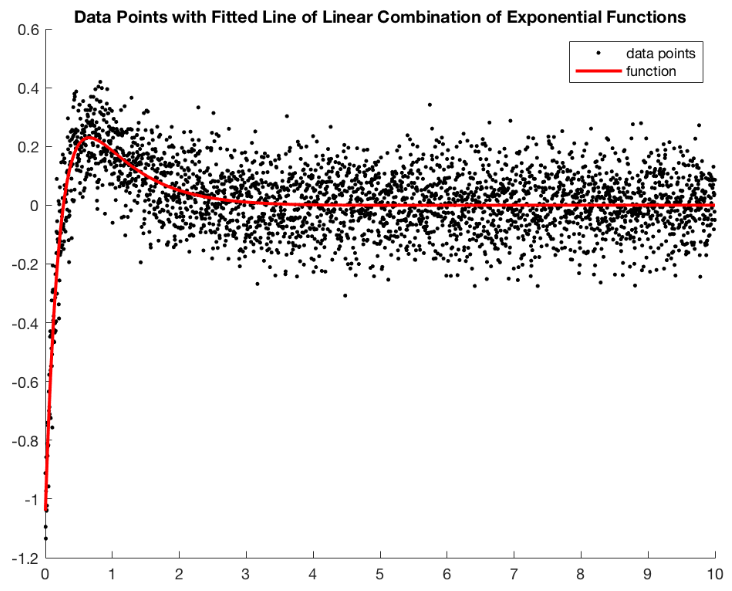

Figure 2: Graphical Representation of Interpolation on the Third Data Set with Initial Guesses $\lambda_1 = -1, \lambda_2 = -2, \lambda_3 = -4$

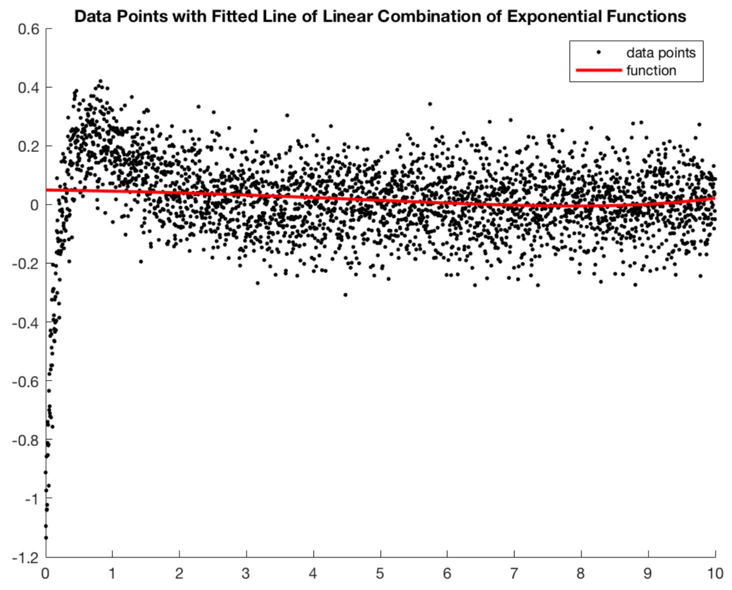

Figure 3: Graphical Representation of Interpolation on the Third Data Set with Initial Guesses $\lambda_1 = 1, \lambda_2 = 2, \lambda_3 = 4$

For the user it can be trial an error to get a good fit at first with a given guess. This can not be fixed by the program. However, if a reasonable input is given and the program will find a solution, the output does not have to be close to the input variables (see the input at Example and Illustration.

Questions

It is possible to give multiple λs as input for the function, but which value is optimal? Of how many e-powers should the function consist to give a reasonable output? We tested this with the following λ values: $\lambda_1 = -1, \lambda_2 = -2, \lambda_3 = -3, \lambda_4 = -4, \lambda_5 = -5$. For every dataset we collected the residue with the n input λs. The results can be found below.

	`data1`	`data2`	`data3`
1 variable	8.6751	10.8317	6.5838
2 variables	6.2533	4.425	5.5047
3 variables	6.2465	4.4035	5.5042
4 variables	6.2442	4.4014	5.5016
5 variables	6.2441	4.4013	5.4999

Table 1: Residue with Different Combinations of Variables λ and Data

We can see from Table 1 that with more variables the residual is decreasing, which is logical, as with more variables it can be estimated more precisely. With just three input variables the dataset `data1` and `data2` show a good fit, comparable to that of five inputs. With `data3` only two variables seem to be needed for a well fitted plot, one variables does not just cut it (see Figure 4 and 5). The rest of the variables, or e-powers, can thus be 'hidden', while still remaining a reasonable fit. This gives us the following equations for every data set:

$$\texttt{data1:}\quad f(x) = 1.9345e^{-0.327743x} + 3.0411e^{-1.93149x} + 0.28064e^{-51.5459x}$$

$$\texttt{data2:}\quad g(x) = 2.3592e^{-2.11864x} + 3.2302e^{-5.84746x} + 4.4084e^{-11.48x}$$

$$\texttt{data3:}\quad h(x) = 0.87397e^{-1.4253x} + -1.9103e^{-4.3181x}$$

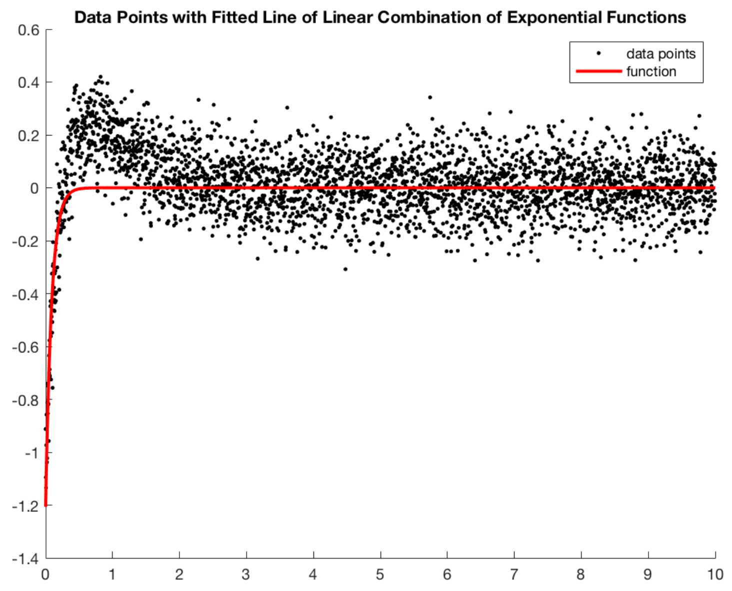

Figure 4: Graphical Representation of Interpolation on the Third Data Set with Initial Guesses $\lambda_1 = -1$

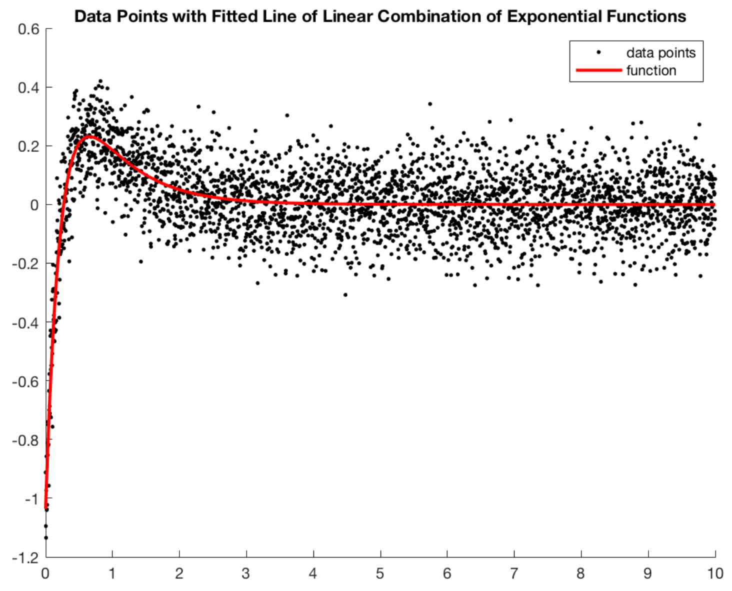

Figure 5: Graphical Representation of Interpolation on the Third Data Set with Initial Guesses $\lambda_1 = -1, \lambda_2 = -2$

Performance

The function was written with performance in mind. The code is short and concise and uses a minimal amount of external functions. It was chosen to first print the output and then plot the figure, as MATLAB tends to hang before plotting any figure. This immediately gives the user an output, instead of having to wait on the figure. Only `fminsearch` was available for use in this exercise, so it was not possible to compare performance to other methods of finding the optimal combination of lambda values.

Appendix: Function Implementation

```matlab
function [] = exp_lin_comb_fit(data, initial_guess_lambdas, max_evals)
%exp_lin_comb Fit a linear combination of exponential functions to data.
%   EXP_LIN_COMB_FIT(DATA, INITIAL_GUESS_LAMBDAS, MAX_EVALS) tries to fit a
%   line on the input DATA by creating a combination of exponential
%   functions. A function is found by using the INITIAL_GUESS_LAMBDAS
%   vector as start variables of the function, and then finding optimal
%   values in the neighbourhood of this input.
%   After method completion, a graph will be shown with the data points and
%   fitted line printed. The console will return the constants, lambdas
%   and residue of the found function.
%   The option MAX_EVALS is optional, if nothing is input it will default
%   to the default value of 200 times the amount of variables in the guess.

% Set default values. Default MaxFunEvals of fminsearch is
% 200*numberOfVariables per documentation.
if nargin < 3
    max_evals = 200*length(initial_guess_lambdas);
end

% Input data is an (m x 2) matrix. With columns 1 and 2 being x and y
% respectively.
x = data(:, 1);
y = data(:, 2);

% We want to minimise residue, so the least squares function should be on
% ||AC-y||, with A being the vector of all e-powers and C being A\y.
least_squares_function = @(lambdas) norm(exp(x*lambdas) * (exp(x*lambdas)\y) - y);

% With the initial guess, try to find lambdas where the residue is the
% smallest with fminsearch.
best_found_lambdas = fminsearch(least_squares_function, initial_guess_lambdas, ...
    optimset('MaxFunEvals', max_evals));

% With the found lambdas, calculate A, C and the residue.
A       = exp(x*best_found_lambdas);
C       = A\y;
```

```
residue = norm(A*C-y);

% The fit line has the x values of the input and the y values A*C.
x_line = x;
y_line = A*C;

% Print out all the values.
disp(['constants : ' num2str(C')                ]);
disp(['lambdas   : ' num2str(best_found_lambdas)]);
disp(['residue   : ' num2str(residue)           ]);

% Plot the data as black dots and the fit line as red line.
hold on;
plot(x,      y,      'k.');
plot(x_line, y_line, 'r-', 'LineWidth', 2);
title('Data Points with Fitted Line of Linear Combination of Exponential Functions');
legend('data points', 'function');
hold off;
```