Exercise 2.9

Filename: *crossing_2_graphs.m*
It was needed to write a program, which shows the intersection point of two graphs. Implemented as a function. Input values are number of points (N) and the error (e). Output variable is the intersection point. To go through the whole interval the “for”-loop from 1 to N is used. Then the “if”-statement is used to check if the difference of the functions is less than the needed error. The achieved logical value by that, allows us to write a value into the variable, inside the ‘’if’’-statement body, which, as a result, is an approximate intersection point. As more the number of points, as better the intersection point is going to be approximated.

Exercise 2.15

Filename: *linear_interpolation.m*
It was needed to create an interpolation function. Function takes as an input the array of points and the needed time point, where the function needs to be interpolated. Time array (from 0) is created considering the length of the input array. The “for”-loop is used for iterating in terms of number of points. “if” statement checks the condition so that our t lays in the interval. Then the formula of linear interpolation is used (Figure 1).

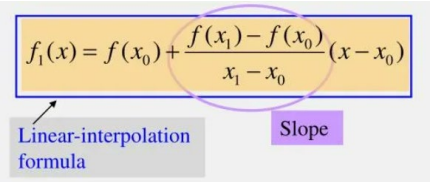

Figure 1 – Linear-interpolation formula

The lower term (x1 – x0) is dropped, which is in our case is the difference between next and previous time step, which is always equal to 1 minute. X in the right parenthesis is our time point, where we want to create the interpolated value. The results of interpolation with initial data of y = [4.4, 2, 11, 21.5, 7.5], with corresponding t = [0, 1, 2, 3, 4] for t = 2.5 and t = 3.1 are shown at the Figure 2. Red circles are the interpolated points. They lay on the line between the corresponding previous and the next point, so by that it can be stated, that the main idea of interpolation is achieved.

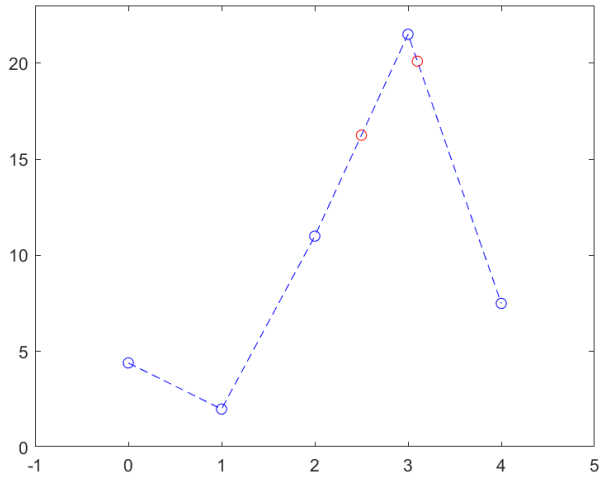

Figure 2 – Interpolation for time t = 2.5 and t = 3.1

Exercise 2.18

Files: *sinesum.m, test_sinesum.m, plot_compare.m, err.m, trial.m, fit_sines.m*
Main task idea is to create a function, that approximates some mathematical function by the sum of sines.
a) *sinesum.m*
Two-dimensional calculation of the sum of functions is implemented via “for”-loops. The result is saved in variable. The testing is performed next.
b) *test_sinesum.m*
Testing the sinesum function with values of t = [-pi/2 pi/4] and b = [4 -3]. As a result, the values of -4.000 and 0.1716 are obtained. Hand calculation shows the same values, so, the function works quite correct.
c) *plot_compare.m*
Function implements the plot comparison of resulted function approximation with the real function. This function is going to be used later.
d) *err.m*
Function implements the calculation of the approximation error by values comparison.
e) *trial.m*
Main idea of this function is to calculate an error by interactively providing it with b coefficients. So, that gives us an opportunity to use the trial & error method to get the needed values, that will satisfy us in our function approximation. “While” loop is used to create a continuous loop, that we can exit, by typing special character (provided by “if”-statement).
f) *fit_sines.m*
This function is the result of all the functions, that were mentioned earlier. Main idea – automation of the process of finding the “best” b-coefficients, that’ll give us the smallest approximation error. It is done by performing the 3-dimensional “for” loop search with b value comparison to the initial ones (“if”-statement). For analyzing function f = t/pi value of the smallest error is 5.49218 with corresponding b-coefficients as 0.6, -0.3 and 0.2. The analyzed function and sine approximation of it are shown at the Figure 3.

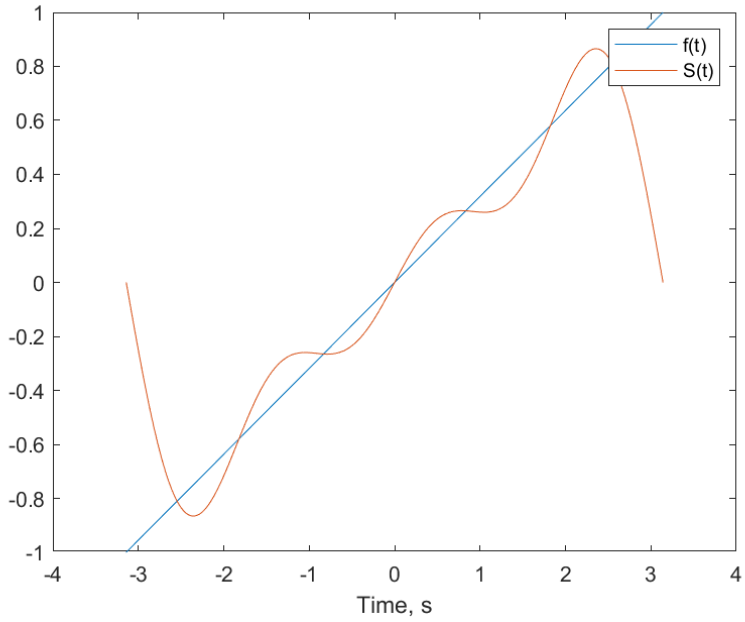

Figure 3 – original and approximated functions comparison