Computer Assignment
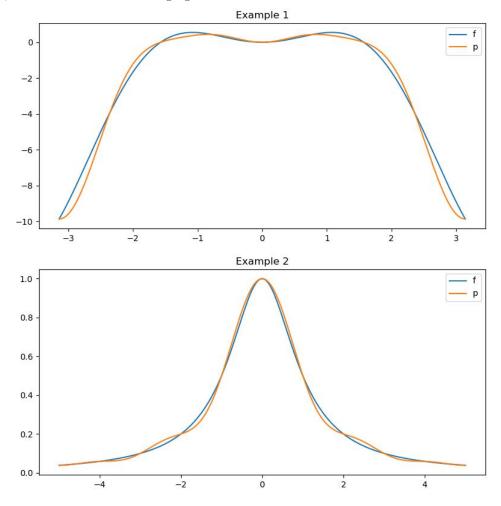
The program which does the required is submitted via KLMS along with this document. Given the two endpoints of the interval `a` and `b`, the step size `h`, and a function `f`, the program computes the trigonometric interpolating polynomial by first computing the coefficients `beta` $= [\beta_k]$ of the phase polynomial using the function `fft` provided by `numpy` package, and then computing the coefficients `A` $= [A_j]$, `B` $= [B_j]$ using the relations between β_k and A_j, B_j. The computed interpolating polynomials are printed out, as the following figure.

As indicated above, the result say that for example the trigonometric interpolating polynomial for the first function as coefficients

$$A_0 = -4.2121, \quad A_1 = -4.0095, \quad A_2 = 2.4674, \quad A_3 = -0.92528, \quad A_4 = -0.72269,$$
$$B_1 = 0, \qquad\qquad B_2 = 0, \qquad\qquad B_3 = 0.$$

The original function f and the interpolating polynomial p plotted together, for the two functions given, is shown as in the following figure.

Since both functions given are even functions, we expect that $B_j = 0$ for all j, and indeed that is the case as we can see in the console output.

For the first function the trigonometric interpolating polynomial is nice, but not great. Our guess is that the nature of f, having an M-shaped graph with a very shallow valley, is not well captured with only a few cosine functions. However increasing the number of points in the abscissa shows an improvement in the quality of interpolation.

What is really interesting is the wellness of trigonometric interpolating polynomial interpolating the Runge function. Ordinary polynomials show terrible performance when interpolating the Runge function. But in contrast trigonometric interpolating polynomials well interpolate even the Runge function. Thus, we observe that trigonometric interpolating polynomials have some advantages that ordinary interpolating polynomials do not possess.