Interpolation is the operation of fitting a curve to a set of points. That is, given a set of points $\{(x_1, y_1), \dots, (x_k, y_k)\}$, interpolation finds a function f such that $f(x_1) = y_1, \dots, f(x_k) = y_k$. The graph of f passes through all the given points.

Throughout this problem, we will assume that arithmetic operations on real and complex numbers take constant time.

(a) (optional) How would you fit a polynomial to *any* $2n + 1$ points in $\Theta(n^2)$ time?

(b) How would you fit a polynomial to $2n + 1$ values at *specifically* the $(2n + 1)$-th roots of unity in $\Theta(n \log n)$ time?

(c) Suggest how to fit a *Laurent polynomial*

$$\sum_{k=-n}^{n} c_k x^k,$$

instead of a polynomial

$$\sum_{k=0}^{2n} c'_k x^k.$$

The true power of this technique comes from replacing the variable in a polynomial expression with a complex exponential.

(d) Show how by setting $x = e^{it}$, we can rewrite

$$\sum_{k=-n}^{n} c_k x^k$$

in the form of a *trigonometric polynomial*

$$A_0 + \sum_{k=1}^{n} A_k \cos(kt + \phi_k),$$

where all the coefficients A_k and ϕ_k are complex numbers.

(e) Therefore describe how to fit a trigonometric polynomial to $2n+1$ *equally spaced* points $0, t, 2t, \dots$ in $\Theta(n \log n)$ time.

(f) How can we use the trigonometric polynomial representation to interpret a Laurent polynomial as a superposition of several sinusoids?

- What distinguishes the sinusoids arising from each term of the trigonometric polynomial?
- What do the coefficients A_k and ϕ_k represent?

(g) Suggest how this might be applied to:

(i) processing audio to remove unwanted low pitch sound (such as wind);

(ii) analysing time series in finance or climatology;

(iii) (optional, hard) remove the *aliasing* effect that arises when an image is reconstructed from samples;

(iv) (optional, very hard) compressing an image using JPEG.

Rubric.

(a) State a method to find any algebraic expression (not necessarily the coefficient representation) for the fitted polynomial. The question is optional, so you can delve into as much or as little as you like.

(b) State a method to convert from this particular value representation to the coefficient representation.

(c) Discuss the differences in fitting a 'polynomial' with negative exponents as compared to a regular polynomial.

(d) Perform the substitution and continue manipulating the expression until you reach the desired form.

(e) Describe how to transform the input in this scenario into inputs suitable for the algorithm developed in the previous parts, and how to 'undo' this transformation to recover the solution.

(f) It's sufficient to just directly answer the two dot points.

(g) Only a very cursory explanation is needed here. We don't expect you to know the ins and outs of these applications, but they are starting points for your own investigation into perhaps the most important modern algorithm.

- This task will form part of the portfolio.
- Ensure that your argument is clear and keep reworking your solutions until your lab demonstrator is happy with your work.

Solution.

(a) We can assume that we the points are sorted by their real components and then their complex components if needed, which will take $O(n \log n)$ time. We can then form a Newton Polynomial by creating a divided difference table, which should take $O(n^2)$.

(b) Since we are using the roots of unity, we can use the IFFT to convert from the frequency domain to the coefficient domain. Both the FFT and IFFT run in $O(n \log n)$ time.

(c) To fit a Laurent polynomial, we define a system of linear equations $A\boldsymbol{c} = \boldsymbol{y}$ where $A \in \mathbb{C}^{(2n+1)\times(2n+1)}$ and $A_{ij} = x_i^{j-n}$, $\boldsymbol{c} \in \mathbb{C}^{2n+1}$ is our vector of desired coefficients, and $\boldsymbol{y}$ is the vector of our known output values. The inclusion of negative powers in the Laurent polynomial means that the matrix A now has entries corresponding to both positive and negative powers of x_i. However, the structure of the matrix remains similar to that of a standard polynomial, with each row corresponding to a different point and each column corresponding to a different power of x.

(d) By setting $x = e^{it}$, we have

$$\begin{aligned}\sum_{k=-n}^{n} c_k x^k &= c_0 + \sum_{k=1}^{n} \left[c_k x^k + c_{-k} x^{-k}\right] \\ &= c_0 + \sum_{k=1}^{n} \left[c_k e^{ikt} + c_{-k} e^{-ikt}\right] \\ &= c_0 + \sum_{k=1}^{n} \left[(c_k + c_{-k}) \cos kt + i(c_k + c_{-k}) \sin kt\right].\end{aligned}$$

Now, we can express each of these term as a single cosine with a phase shift. We will let A_k be the magnitude and ϕ_k to be the phase angle, each for the k-th term. That is, we want

$$A_k e^{i\phi_k} = c_k + c_{-k} \text{ and } A_k e^{-i\phi_k} = c_k - c_{-k}.$$

We multiply both equations to eliminate the ϕ_k term, giving us $A_k^2 = (c_k + c_{-k})(c_k - c_{-k}) \Rightarrow$

$A_k = \sqrt{(c_k + c_{-k})(c_k - c_{-k})}$. Then, we simply have

$$e^{i\phi_k} = \frac{c_k + c_{-k+}}{A_k} = \frac{c_k + c_{-k+}}{\sqrt{(c_k + c_{-k})(c_k - c_{-k})}}.$$

With $A_0 = c_0$, we have written our Laurent polynomial as a trigonometric polynomial

$$A_0 + \sum_{k=1}^{n} A_k \cos(kt + \phi_k).$$

(e) To fit a trigonometric polynomial to $2n+1$ equally spaced points $0, t, 2t, \ldots$, we first use the FFT to compute the coefficients c_k in $\Theta(n \log n)$ time. Then, we convert these coefficients to the trigonometric form A_k and ϕ_k as described in part (d). The values of t are chosen such that the points are equally spaced over one period of the trigonometric polynomial. For example, if the period is T, we can choose $t = \frac{T}{2n}$, ensuring that the points cover the entire period. To recover the values of the trigonometric polynomial at these points, we evaluate the polynomial at each point using the computed coefficients.

(f) Each term of the trigonometric polynomial corresponds to a sinusoidal component, distinguished by their frequency determined by k. The amplitude A_k can be thought of as the value that defines the impact of the component on the shape of the Laurent polynomial. ϕ_k reporesents the phase shift of the components, which determines the starting position of the sinusoid. The principle of superposition states that the total output in a linear system is equal to the sum of the responses from each individual input. The total Laurent polynomial is simply the sum of each of these sinusoidal components that create the same shape.

(i) If we convert our audio signal to the frequency domain from the time domain, we can represent it as a trigonometric polynomial. We then need to identify the low-pitched sounds (or the frequencies associated with the noises we are eliminating) and simply remove these from the polynomial, which we can then simply convert back to the time domain.

(ii) In the realm of finance, we would be able to identify periodic trends by analysing component amplitude to identify the most powerful frequencies. For example, we could forecast stock prices and interest rates. In climatology, we can follow a similar idea of pattern recognition in the weather and conditions, including long-term climate cycles like El Niño and La Niña.

(iii) Aliasing occurs when a signal is sampled at a rate that is insufficient to capture its highest frequency components accurately. When converting this signal into a trigonometric polynomial, we can identify the frequencies that are in the signal and identify potential high-frequencies that might have caused the aliasing. From here, you just need to apply a filter that attenuates or completely removes any component.

(iv) JPEG compression starts with converting the image from RGB to YCbCr color space, which separates luminance (you can think of this as brightness) and chrominance (two colour layers where each represent a number on a linear scale between two primary colours - all four colours span colour space) components. The chrominance components are often downsampled to reduce data, usually in frequencies that humans are less sensitive to seeing. The image is then divided into 8×8 pixel blocks, and each block undergoes a Discrete Cosine Transform (DCT) to convert spatial data into frequency components. These components are quantised by dividing them by a quantisation matrix and rounding to integers, with higher frequencies (less visually significant) being reduced more aggressively. In simple terms, this matrix is used to remove frequencies depending on the aggression compression. These quantised blocks are then further compressed using run-length encoding to compress any sequences of zeros. Finally, Huffman encoding is used for further lossless compression of the data. The compression process involves both lossy (downsampling, quantisation) and lossless (run-length encoding, Huffman encoding).