Task 1: Dynamic Time Warping

Part A (Code)

**Implement DTW:** Write a program that implements the DTW algorithm. The algorithm should return the DTW matrix and the alignment cost (i.e., bottom-right cell in DTW matrix).

Part B (Written)

**Describe DTW:** In your own words, explain what DTW is.

**Explain how DTW differs from simple sequence matching:** Highlight the unique aspects of DTW that allow it to handle variations in speed and timing.

**Detail the initial setup of the cost matrix:** Describe what each cell represents and how the initial values are set.

**Describe the process of calculating the optimal path through the cost matrix:** Explain the criteria used to determine the path (e.g., minimising total cost).

Part C (Written)

**Recurrence Relation:** Explain the recurrence relation used in DTW, describing how it facilitates the dynamic programming approach to find the minimal distance.

Part D (Code)

Dynamic Time Warping (DTW) is an effective method for aligning temporal sequences that vary in speed or timing. However, without constraints, DTW can produce alignments that stretch or compress the sequences excessively, potentially leading to unrealistic results. To address these issues, in Part D we introduce boundary constraints.

As shown in the figure below, a boundary constraint limits the alignment path to a window around the diagonal, with a **window\_size = 3**. This means that only cells up to a distance of 3 to the right and left of the diagonal are considered in the alignment path. Both the diagonal cells (in grey) and those to the left and right of the diagonal (in black) are included in the computation. Cells outside this window (in white) are excluded, ensuring that DTW only aligns points within a reasonable distance, maintaining plausible alignments.

A black and grey squares

Description automatically generated

**Implement DTW with Boundary Constraints**

Modify the standard DTW algorithm to include boundary constraints that limit the alignment to a specified window around the diagonal of the cost matrix. This window will define the maximum allowable distance in time that corresponding points on each sequence can have.

* **Implementation Details:**
  + **Window Size:** Introduce a parameter **window\_size** that controls the width of the boundary around the diagonal. For each cell **(i, j)** in the DTW matrix, limit the alignment path to remain within this window band to maintain plausible matches.
  + **Matrix Calculation:** Adjust the DTW matrix computation to consider only the elements within the window constraints for each matrix cell. This involves modifying the loops that fill in the DTW matrix to skip calculations for elements outside the specified window.
  + **Output**: The algorithm should return the DTW matrix and the alignment cost (i.e., bottom-right cell in DTW matrix).

Part E (Written)

**Computational complexity:** Discuss the computational complexity of the algorithm implemented in Part D (i.e., DTW with boundary constraints). Is it different from the algorithm in Part A (i.e., standard DTW)? State and explain the recurrence relation.

Part F (Code)

While boundary constraints introduced in Part D prevent unrealistic alignments, they may also restrict the algorithm's ability to find the most accurate alignment under certain conditions. Rigid constraints can potentially exclude optimal paths, particularly in sequences with varying densities or irregular pacing.

Part F aims to overcome this limitation by capping the total length of the warping path (i.e., the total number of steps (insertions, deletions or matches) in the alignment path) instead of imposing rigid constraints between individual points. This approach allows the DTW algorithm to adapt more naturally to the given data (i.e., sequences), providing flexibility in alignment while maintaining control over the path complexity.

To better understand how this constraint affects alignments, the figure below illustrates how cumulative alignment costs are calculated across varying path lengths (within a defined max\_path\_length). Each subplot represents the alignment costs at a specific path length, starting from shorter paths (left) to longer paths (right). The alignment costs are computed for each cell by considering cumulative costs from the previous path length. For instance, a cell at position (i, j) in the matrix for a longer path length could have its cost updated based on the closest neighbours to the top, left, and top-left diagonal (i.e., insertion, deletion, match) of cell at position (i, j) from the matrix of the previous path length. This cumulative process allows DTW to track alignment paths step-by-step, capturing variations in data while preventing the warping path from exceeding the allowed total path length.

A diagram of a graph

Description automatically generated

**Implement DTW with Total Path Length Constraint**

Modify the standard DTW algorithm to include a constraint that limits the total number of steps in the warping path. This constraint ensures a balance between maintaining sequence integrity and allowing enough flexibility for natural variations in the data.

**Implementation Details:**

* **Path Length Parameter:** Introduce a parameter **max\_path\_length** that controls the maximum allowable total steps in the alignment.
* **Output:** The algorithm should return the alignment cost. See *Path Extraction Hint* below.

**Implementation Hints:**

* **Iterative Logic Hint:** Consider how to evaluate alignment paths of varying lengths, from 1 up to the maximum path length allowed.
  + *How can an iterative approach capture cumulative costs effectively for each possible path length?* By iterating over incremental path lengths starting from 1, the algorithm can build solutions step-by-step and capture cumulative costs effectively for each possible path length.
* **Matrix Calculation Adjustments Hint:** Implement the DTW calculation so that it only accumulates paths within the allowed total number of steps. Use checks at each matrix cell to ensure that the path count remains within the valid indices of both sequences and does not exceed the current path length (see Figure above).
* **Data Structure Adjustment Hint:** So far, we have been tracking minimum alignment costs. Now, consider that you also need to ensure that the total steps in the alignment path do not exceed a given maximum.
  + *Is a 2D matrix still sufficient to keep track of the alignment costs while maintaining a constraint on the total number of steps?* Probably not. It seems like we need an additional dimension for the path length. What about a 3D matrix to explicitly capture alignment costs across different path lengths, or a 2D matrix where each cell is a dictionary to store cumulative costs for multiple path lengths?
* **Path Extraction Hint:** Consider how to identify the minimum cumulative cost among all feasible path lengths. If using a 3D matrix, compute the minimum cost across the third dimension at the last indices of the two sequences i.e., **(n, m)**. If using a 2D matrix with a dictionary structure, ensure that the cumulative cost is extracted across all possible path lengths.

Part G (Written)

**Computational complexity:** Discuss the computational complexity of the algorithm implemented in Part F (i.e., DTW with total path length constraint). Is it different from the algorithm in Part A (i.e., standard DTW)? Explain the recurrence relation.