Part B

Describe DTW:

In your own words, explain what DTW is.

Dynamic Time Warping is an algorithm that tries to align one sequence, A, to another, B, that differs in the time dimension in the most optimal way. This is achieved through 3 major operations: Merging, Insertion and Deletion. This warps/distorts the shape of one sequence and is represented by a matrix that can show the operations applied and the minimum “cost” (the minimum number of units involved with each operation) to align all the points from the starting point to a specific matrix cell.

Explain how DTW differs from simple sequence matching:

Highlight the unique aspects of DTW that allow it to handle variations in speed and timing.

It is a surefire method that tells the optimum choices that need to be made and can even track the order based from the starting point. Given the matrix form in which the algorithm is performed, you are not limited to observation of the optimal order of warping operations but also from any point in the matrix cell where you can look at a certain instance or scenario. For example, it can find the optimal order of warping operations given, certain operations already operated on the sequences. This flexibility enables DTW to handle sequences with varying lengths in the time dimension (or speed/frequency of recorded data points) since it’s not dependent on the nature of the recorded data (as long as there are data points recorded in both sequences).

Detail the initial setup of the cost matrix:

Describe what each cell represents and how the initial values are set.

Each cell represents the minimum “cost” of the operations to align the number of recorded data points in sequence A (e.g. the number of rows) to also the number of recorded data points in sequence B (e.g. the number of columns), both starting at the very first point (indexed as 1 in rows and 1 in columns) in addition to the absolute distance between the current point in sequence A (the current row of the cell) and current point in sequence B (the current column of the cell). The only cells that do not represent this are in the row and column indexed as 0 in the matrix. In row 0, column 0, this cell is set to the value of 0 and the rest of the values in row 0 and in column 0 are initialised to infinity for comparison purposes so that when evaluation cells row 1 and column 1, there is something to compare to (given that there is no point before the first point of either sequence that is subject of analysis).

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).

The cost matrix first needs to be initialised as explained above. Then, the minimum cost of the operations to align the number of recorded data points in sequence A (e.g. the number of rows) to also the number of recorded data points in sequence B (e.g. the number of columns) is calculated by adding the absolute distance between the current point in sequence A (the current row of the cell, e.g. i) and current point in sequence B (the current column of the cell, e.g. j) with the minimum of the cells e.g. i - 1, j and i, j - 1 and i -1 and j - 1 which represent the sets where either it uses the previous point in A and current point in B, the current point in A and previous point in B or both the previous points in A and B. When it uses the previous point in A and current point in B, this represents Insertion. When it uses the current point in A and previous point in B, this represents Deletion. When it uses both the previous points in A and B, this represents a Match. This process is done for all cells where these 3 cells already exist until the last points in A and B (the opposite side of the matrix to cell 0, 0). The value in the last calculated cell thus represents the overall minimum cost of aligning the two sequences and by tracing back through the minimum of the 3 cells used to calculate a cell, you will be able to see the optimal path through the cost matrix. In DTW, its choice is to evaluate each row before moving to the next in a systematic and orderly way.

Part C

Recurrence Relation:

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

The equation used to determine the value in each cell is reliant on the minimum of previous cells in such a way that each new cell adds on its own personal cost to the current minimum cost needed to get to that selection of points cumulatively (one from each sequence) in the most optimal path. This is highly characteristic of dynamic programming as it systematically works through all the paths in a way where it can be simpler to imagine it as two pointers on each sequence, each deciding to move or stay to the next point in a sequence if more efficient, however DTW calculates all possibilities by using this conceptual idea in a matrix. Through the systematic process of DTW, it reduces the possibility of unnecessary calculations but splits the whole operation into small manageable subproblem cases.

Part E

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.

For Part D, the computational complexity of the initialisation is O($n + m$) (where n is the length of sequence A and m is the length of sequence B, ignoring initialisation of cell 0, 0). When populating the cells within the boundaries (let k be the window width), the precise calculation is $(nm - (n - k) \cdot (m - k))$ which simplifies to $k(n + m - k)$, all the cells that are populated within boundaries. This “cell-populating” stage is the only differing part when comparing to the computational complexity of Part A, since Part D’s boundaries restrict the number of cells populated. The computational complexity of Part A for the “cell-populating” stage is nm. When comparing, we can see that the computational complexity of Part D will always be less as if $n < m$ so that $k \leq n < m$ and from observation that the computational complexity of Part D grows as k grows, when k is its maximum n, the computational complexity of Part D would be $n(n + m - n) = mn$, which is the computational complexity of Part A. Thus, the computational complexity of Part D will be faster than or equal to Part A since Part D is restricted within the boundaries.

The recurrence relation for populating the matrices works by calculating the addition of the cost of aligning the elements in sequenceA$[i - 1]$ and sequenceB$[j - 1]$ and the minimum cost among the neighbouring cells $(i - 1, j - 1)$, $(i, j - 1)$, and $(i - 1, j)$ for each cell (i, j). Thus the overall recurrence relation equation equation can be described as:
$T(i, j) = A(i - 1) - B(j - 1) + min(T(i, j - 1), T(i - 1, j), T(i - 1, j - 1)$

Part G

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.

For Part F, the computational complexity of the algorithm is nmh, where n is the length of sequence A and m is the length of sequence B and h is the maximum path length, as each cell of the 3D matrix is accessed once. This is different to Part A, which has a computational complexity of nm and since $h > 1$, Part A will have a faster computational complexity compared to Part F.

The recurrence relation for populating the matrices works by calculating the addition of the cost of aligning the elements in sequenceA$[i - 1]$ and sequenceB$[j - 1]$ and the minimum cost among the neighbouring cells $(i - 1, j - 1, k - 1)$, $(i, j - 1, k - 1)$, and

$(i-1,\ j,\ k-1)$ in the previous layer of the 3D matrix for each cell $(i,\ j,\ k)$, starting from $(1,1,1)$ which depend on the initialised values in the layer $(i,\ j,\ 0)$. Thus the overall recurrence relation equation can be described as:

$$T(i,\ j,\ k)\ =\ A(i-1) - B(j-1) + min(\ T(i,\ j-1,\ k-1), T(i-1, j,\ k-1),\ T(i-1,\ j-1,\ k-1)$$