**1.0 SMITH-WATERMAN ALGORITHM**

The Smith-Waterman algorithm (implemented in algs.py) takes as input a substitution matrix (indicating the score/penalty for aligning/misaligning two amino acids), two protein sequences, and a gap-penalty scheme (including a penalty for opening a gap and for extending it).

Two matrices are created: the scoring matrix, indicating the maximum possible score of a particular cell given the state of the previous maximum score, and a state matrix. The state matrix stores the source of a cell’s score, indicating whether it was from an alignment, from an insertion, or from a deletion (in practice, whether it comes from the diagonal, from the left, or from the above).

This file also includes a function for traceback, which returns the best alignment based on the state matrix created above.

**1.1 BLOSUM50 SCORES**

The following scheme was implemented to determine the optimal gap opening and extension penalties for the BLOSUM50 matrix:

for gapO in range(1,21):

for gapE in range(1,6):

# find maximum value at a True Positive Rate (TPR) of 0.7

t = get\_cutoff(base\_dir, gapO, gapE, 0.7, blosum)

# calculate the negative scores

neg\_scores = […]

# Calculate false positive rate

FP = sum(i > t for i in neg\_scores)

FPR = FP/neg\_count

(Exact code can be found in \_\_main\_\_.py, under the heading PART I Question 1.)

The optimal gap penalty was selected based on the combination that yielded the smallest FPR at a TPR of 0.7. The following (opening, extension) pairs all yielded an optimum of 0.24: (6, 5) (7, 3) (8, 2). Full parameter results can be found in /output/gapScores.txt.

**1.2 COMPARING Substitution matrices**

The optimal substitution matrix with gap opening and extension penalties at (6,5) was BLOSUM50, as shown in the ROC curve for all matrices below:



At a TPR of 0.7, the following alignment scores were found for each matrix:

|  |  |
| --- | --- |
| Matrix | FPR at TPR=0.7 |
| BLOSUM50 | 0.24 |
| BLOSUM62 | 0.30 |
| MATIO | 0.38 |
| PAM100 | 0.32 |
| PAM250 | 0.32 |

**1.3 Normalizing by Sequence Length**

The maximum alignment scores were divided by the length of the shortest sequence in each comparison. These scores were then used to calculate the ROC curve shown below.



The normalized scores appear to hover around y = x line, indicating that the ability to distinguish between true and false positives is mostly dependent on sequence length. The longer a sequence is, the more likely it is to get a higher maximum hit. This analysis seems to suggest that either my implementation of the algorithm and/or the way that these positive and negative hits were determined was heavily reliant on sequence length.

**2.0 Optimization ALGORITHM**

To optimize the substitution matrix, I implemented a random permutation algorithm that modifies a single substitution by a random step (integers from -5 to +5, excluding 0). If the new matrix has a better objective score (given by the sum of TPR at a selection of FPR), then the new matrix is accepted and returned to the beginning to be modified again. The algorithm is implemented in optimize.py.

In order to reduce computational overhead, a subset of protein pairs was used to optimize the substitution matrix. These pairs were the 15 lowest-scoring positive pairs and the 15 highest-scoring negative pairs. Since our objective function optimizes for acceptance of all true positives at all false positive rates, these pairs are intuitively the most influential when designing a substitution matrix.

Notably, this random permutation method worked better than an attempt at the Downhill Simplex algorithm (see align/dead\_simplex.py, saved for posterity). This may be because the time and memory requirements of the Simplex calculations restricted me to a relatively small number of maximum iterations.