Algorithms for Bioinformatics

2018/2019

Pairwise Sequence Alignment

Pedro G. Ferreira
[d**CC**] @ Faculty of Sciences University of Porto

Outline

- Comparing Sequences
- Visualize alignments
- Alignments as an optimisation problem
  - Problem definition and complexity
  - Substitution matrices
  - Gap penalties
  - Implementing objective function
- Global Alignment Algorithms with Dynamic Programming
  - Needleman-Wunsch
- Local Alignment Algorithms with Dynamic Programming
  - Smith-Waterman
- Class MatrixNum

Inferring function and Phylogenetics

- Bioinformatics provides tools to help researchers to infer the function of the genes and the respective proteins. It relies on the assumption that biological sequences with high similarity share similar functions.

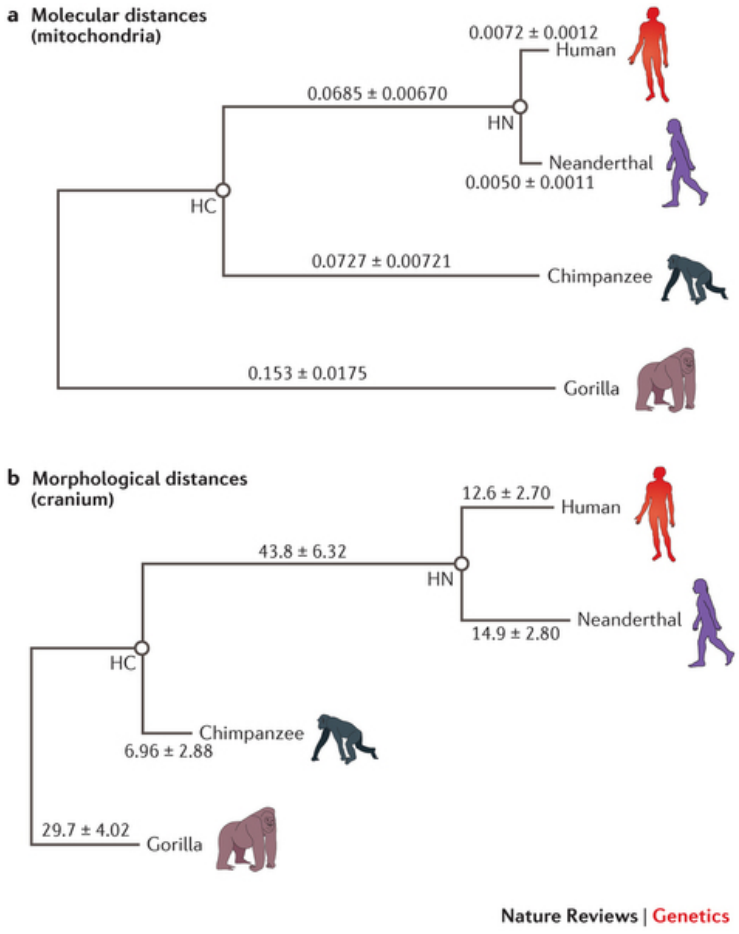
Phylogenetics is the division of biology that studies evolutionary divergence and relationship between organisms, based on two important important concepts:

- **Similarity.** Measures the resemblance and differences between organisms without taking into account any contextualization.

- **Homology.** Investigates the common ground between the organisms and if they share any ancestral characteristics. Find the point in the evolutionary tree that they started to diverge.

- Sequences with high degree of similarity have high probability of being homologous and share similar function. The higher the similar the higher the probability.

Phylogenetics



Depending on the nature of the characters different evolutionary rates can be observed:

- Molecular sequences have a nearly constant rate in these close species.
- Morphological characteristics evolve in a different way.

Bayesian molecular clock dating of species divergences in the genomics era
Mario dos Reis, Philip C. J. Donoghue & Ziheng Yang
Nature Reviews Genetics 17, 71–80 (2016) doi:10.1038/nrg.2015.8

Sequence Similarity


```
Seq1: TATACTTCAGGAACTAATTCTGAAGCATCA

Seq2: TATACTAAAGGAACTAATGCTAAAGCATGA
Seq3: TATCCTCCAGGCATTATTCTTGGACCTTCT
```

- Given Sequence 1, which of the other two sequences is most similar?
  - Sequence similarity can be measured by counting the number of symbols that you need to change in one sequence (query) in order to transform it in the other sequence (target).
  - The distance between two sequences allows to quantify their similarity and how related their biological function may be. We expect that similar sequences will have similar functions.
  - This measure of distance also provides an indication of homology which helps us to know the evolutionary relationship between sequences.
  - Sequence similarity is the basis of phylogenetics studies.

Sequence Similarity

- Distance can be calculated by:
  - Hamming distance
  - Edit distance

- Hamming distance is a dissimilarity measure that corresponds to minimum number of symbols that are different between the two sequences.

```
Q:ATTACGAT
    | | |
T:ATCAGGTT
```

```
Hamming distance is 3
```

- This is a limited distance measure since it requires sequences to have the same length, can not perform edit operations.

- Distance between sequences of different length must be calculated with edit distance.

Sequence Similarity

- The edit distance between the query and the target sequence corresponds to the minimum number of editing operations (and associated costs) to transform one sequence into the other.

- Operations can be:
  - **Substitution** by replacing one symbol by the other

```
Q: ATTACG
T: ATCACG
```

```
T → C
```

  - **Deletion** by deleting one symbol in the query sequence.

```
Q:ATTACG
T:ATTAG
```

```
Q:ATTACG
T:ATTA-G
```

```
C → _
```

  - **Insertion** by inserting one symbol in the query sequence.

```
Q:ATTACG
T:ATTACCG
```

```
Q:ATTAC-G
T:ATTACCG
```

```
_ → C
```

Sequence Alignment

- Sequence alignment is a way of arranging the biological sequences to identify regions of similarity. Such regions may indicate a functional, structural, or evolutionary relationships between the sequences, that can be either DNA, RNA or protein.

- From a computational view an alignment is a procedure to align similar symbols keeping the same order.

- Alignments may be classified as either **global** or **local**.

- A global alignment aligns two sequences from beginning to end, aligning each letter in each sequence only once. An alignment is always produced. Indicated for sequences that share significant similarity over most of their extents.

- A local alignment maximizes the alignment of the parts of the sequences that share similarity. Finds the best aligned subsequence. An alignment may not be produced if no sufficient similarity is found.

- Global alignments are performed with the *Needleman–Wunsch* algorithms and local alignments with the *Smith–Waterman* algorithm.

Sequence Alignment

- There are many ways to align two sequences.

```
Q: CTGTCGCTGCACG
T: TGCCGTG
```

```
global:
CTGTCGCTGCACG
-TG-C-C-G--TG
```

```
local:
CTGTCGCTGCACG
-TG-CCGTG----
```

```
local:
CTGTCGCTGCACG
-TGCCG-T----G
```

- We need a way to obtain the biological meaning of an alignment. A scoring system can be used to evaluate and quantify the quality of the alignment. Possible scoring system:

    - 1 for a match between symbols
    - 0 for a mismatch

Sequence Alignment

```
CGAGGCACAACGTCA
|||  |||   ||||||
CGATGCAAGACGTCA

Score:12
```

```
ATTGGACAGCAATCAGG
|      ||   |      |
ACGATGCAAGACGTCAG

Score:5
```

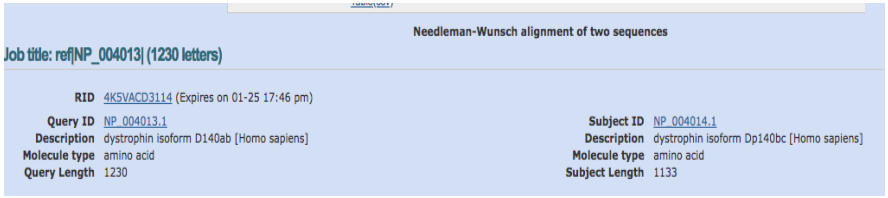
- Simple alignment score do not take into account the physico-chemical properties of amino-acids or the biological sequence information such as the structural and evolutionary relationship between the amino-acids.

- Given the properties of the amino-acids in proteins some mismatches are more acceptable than others. Substitution matrices give a score for each substitution of one amino-acid by another.

Sequence alignment

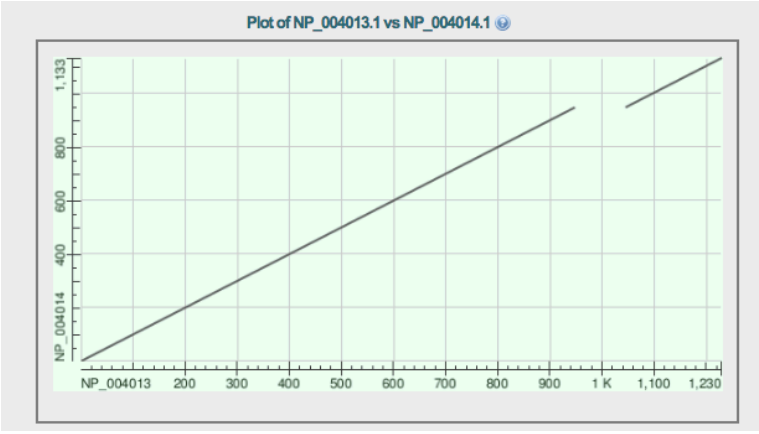
Example: NP_004013 with NP_004014 are dystrophin isoforms, but the first sequence is missing about 100 residues starting at residue 948 (some exons have been spliced out of the corresponding mRNA).

https://blast.ncbi.nlm.nih.gov/




```
NW Score = 5763
Identities = 1133/1230 (92%), Positives = 1133/1230 (92%), Gaps = 97/1230 (8%)

Query  1     MPSSLMLEVPALADFNRAWTELTDWLSLLDQVIKSQRVMVGDLEDINEMIIKQKATMQDL  60
             MPSSLMLEVPALADFNRAWTELTDWLSLLDQVIKSQRVMVGDLEDINEMIIKQKATMQDL
Sbjct  1     MPSSLMLEVPALADFNRAWTELTDWLSLLDQVIKSQRVMVGDLEDINEMIIKQKATMQDL  60

Query  61    EQRRPQLEELITAAQNLKNKTSNQEARTIITDRIERIQNQWDEVQEHLQNRRQQLNEMLK  120
             EQRRPQLEELITAAQNLKNKTSNQEARTIITDRIERIQNQWDEVQEHLQNRRQQLNEMLK
Sbjct  61    EQRRPQLEELITAAQNLKNKTSNQEARTIITDRIERIQNQWDEVQEHLQNRRQQLNEMLK  120

Query  121   DSTQWLEAKEEAEQVLGQARAKLESWKEGPYTVDAIQKKITETKQLAKDLRQWQTNVDVA  180
             DSTQWLEAKEEAEQVLGQARAKLESWKEGPYTVDAIQKKITETKQLAKDLRQWQTNVDVA
Sbjct  121   DSTQWLEAKEEAEQVLGQARAKLESWKEGPYTVDAIQKKITETKQLAKDLRQWQTNVDVA  180

Query  181   NDLALKLLRDYSADDTRKVHMITENINASWRSIHKRVSEREAALEETHRLLQQFPLDLEK  240
             NDLALKLLRDYSADDTRKVHMITENINASWRSIHKRVSEREAALEETHRLLQQFPLDLEK
Sbjct  181   NDLALKLLRDYSADDTRKVHMITENINASWRSIHKRVSEREAALEETHRLLQQFPLDLEK  240

Query  241   FLAWLTEAETTANVLQDATRKERLLEDSKGVKELMKQWQDLQGEIEAHTDVYHNLDENSQ  300
             FLAWLTEAETTANVLQDATRKERLLEDSKGVKELMKQWQDLQGEIEAHTDVYHNLDENSQ
Sbjct  241   FLAWLTEAETTANVLQDATRKERLLEDSKGVKELMKQWQDLQGEIEAHTDVYHNLDENSQ  300

Query  301   KILRSLEGSDDAVLLQRRLDNMNFKWSELRKKSLNIRSHLEASSDQWKRLHLSLQELLVW  360
             KILRSLEGSDDAVLLQRRLDNMNFKWSELRKKSLNIRSHLEASSDQWKRLHLSLQELLVW
Sbjct  301   KILRSLEGSDDAVLLQRRLDNMNFKWSELRKKSLNIRSHLEASSDQWKRLHLSLQELLVW  360

Query  361   LQLKDDELSRQAPIGGDFPAVQKQNDVHRAFKRELKTKEPVIMSTLETVRIFLTEQPLEG  420
             LQLKDDELSRQAPIGGDFPAVQKQNDVHRAFKRELKTKEPVIMSTLETVRIFLTEQPLEG
Sbjct  361   LQLKDDELSRQAPIGGDFPAVQKQNDVHRAFKRELKTKEPVIMSTLETVRIFLTEQPLEG  420

Query  421   LEKLYQEPRELPPEERAQNVTRLLRKQAEEVNTEWEKLNLHSADWQRKIDETLERLQELQ  480
             LEKLYQEPRELPPEERAQNVTRLLRKQAEEVNTEWEKLNLHSADWQRKIDETLERLQELQ
Sbjct  421   LEKLYQEPRELPPEERAQNVTRLLRKQAEEVNTEWEKLNLHSADWQRKIDETLERLQELQ  480
```

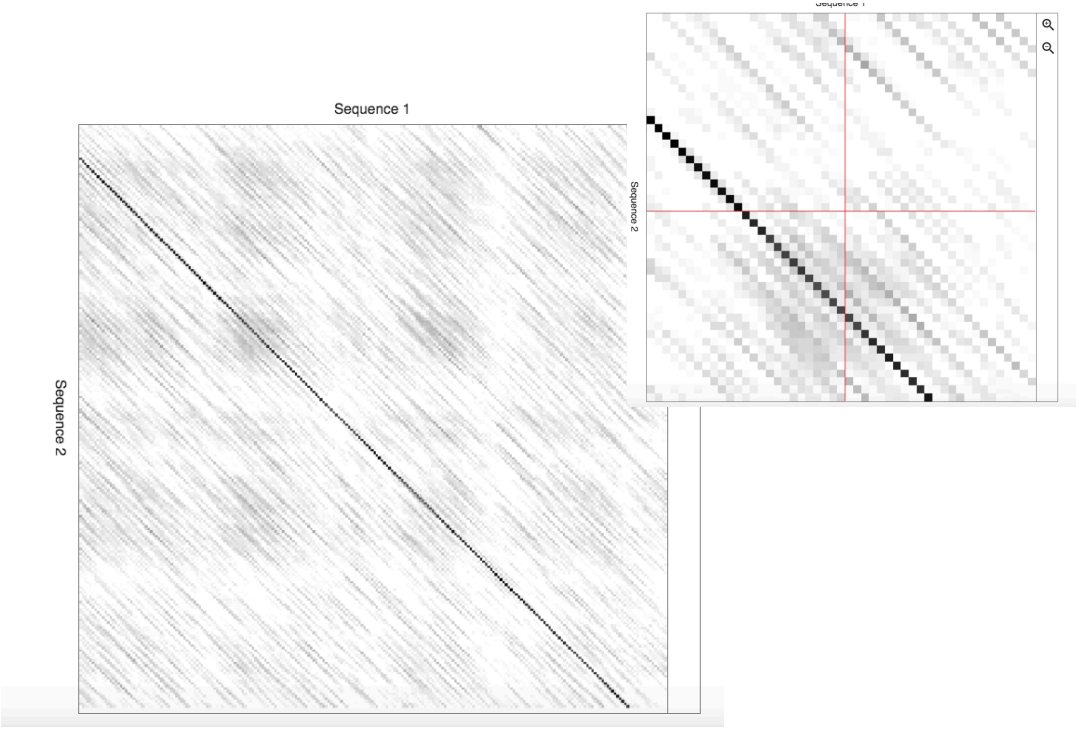
. . .

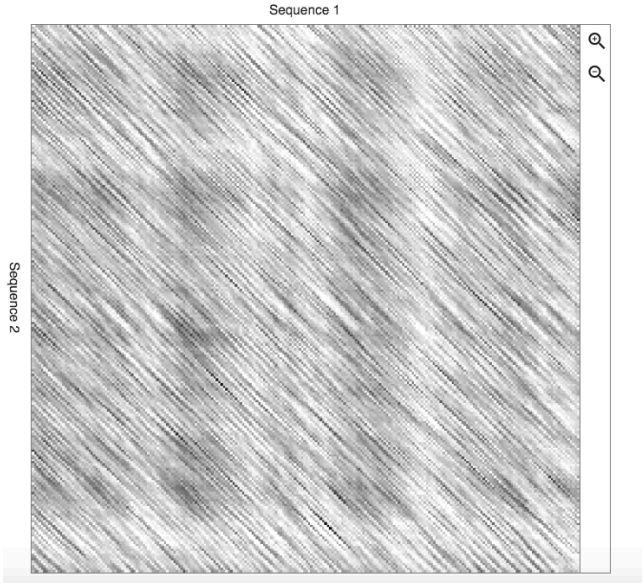



```
Query  781   SIQIPRQLGEVASFGGSNIEPSVRSCFQFANNKPEIEAALFLDWMRLEPQSMVWLPVLHR  840
             SIQIPRQLGEVASFGGSNIEPSVRSCFQFANNKPEIEAALFLDWMRLEPQSMVWLPVLHR
Sbjct  781   SIQIPRQLGEVASFGGSNIEPSVRSCFQFANNKPEIEAALFLDWMRLEPQSMVWLPVLHR  840

Query  841   VAAAETAKHQAKCNICKECPIIGFRYRSLKHFNYDICQSCFFSGRVAKGHKMHYPMVEYC  900
             VAAAETAKHQAKCNICKECPIIGFRYRSLKHFNYDICQSCFFSGRVAKGHKMHYPMVEYC
Sbjct  841   VAAAETAKHQAKCNICKECPIIGFRYRSLKHFNYDICQSCFFSGRVAKGHKMHYPMVEYC  900

Query  901   TPTTSGEDVRDFAKVLKNKFRTKRYFAKHPRMGYLPVQTVLEGDNMETPASSPQLSHDDT  960
             TPTTSGEDVRDFAKVLKNKFRTKRYFAKHPRMGYLPVQTVLEGDNMET
Sbjct  901   TPTTSGEDVRDFAKVLKNKFRTKRYFAKHPRMGYLPVQTVLEGDNMET----------  948

Query  961   HSRIEHYASRLAEMENSNGSYLNDSISPNESIDDEHLLIQHYCQSLNQDSPLSQPRSPAQ  1020
Sbjct  ----------------------------------------------------------------

Query  1021  ILISLESEERGELERILADLEEENRNLQAEYDRLKQQHEHKGLSPLPSPPEMMPTSPQSP  1080
                          NLQAEYDRLKQQHEHKGLSPLPSPPEMMPTSPQSP
Sbjct  949   ----------NLQAEYDRLKQQHEHKGLSPLPSPPEMMPTSPQSP  983

Query  1081  RDAELIAEAKLLRQHKGRLEARMQILEDHNKQLESQLHRLRQLLEQPQAEAKVNGTTVSS  1140
             RDAELIAEAKLLRQHKGRLEARMQILEDHNKQLESQLHRLRQLLEQPQAEAKVNGTTVSS
Sbjct  984   RDAELIAEAKLLRQHKGRLEARMQILEDHNKQLESQLHRLRQLLEQPQAEAKVNGTTVSS  1043

Query  1141  PSTSLQRSDSSQPMLLRVVGSQTSDSMGEEDLLSPPQDTSTGLEEVMEQLNNSFPSSRGH  1200
             PSTSLQRSDSSQPMLLRVVGSQTSDSMGEEDLLSPPQDTSTGLEEVMEQLNNSFPSSRGH
Sbjct  1044  PSTSLQRSDSSQPMLLRVVGSQTSDSMGEEDLLSPPQDTSTGLEEVMEQLNNSFPSSRGH  1103

Query  1201  NVGSLFHMADDLGRAMESLVSVMTDEEGAE  1230
             NVGSLFHMADDLGRAMESLVSVMTDEEGAE
Sbjct  1104  NVGSLFHMADDLGRAMESLVSVMTDEEGAE  1133
```

Visual Alignment

- The dotlet tool compares sequences by the diagonal plot method.





- Sequences with higher similarity show a darker diagonal.
- Lack of diagonal indicates low similarity.

https://dotlet.vital-it.ch/

Visual Alignment

- In their simplest forms, dotplot matrices place dots in the cells where the characters in the two sequences coincide.

- Write a function that given the number of rows and columns creates a numerical matrix.

Visual Alignment

- In their simplest forms, dotplot matrices place dots in the cells where the characters in the two sequences coincide.

```
1. def create_mat(nrows, ncols):
2.     """ create matrix given dimensions: number of rows and columns filled wit
   h zeros"""
3.     mat = []
4.     for i in range(nrows):
5.         mat.append([])
6.         for j in range(ncols):
7.             mat[i].append(0)
8.     return mat
```

Visual Alignment

- In their simplest forms, dotplot matrices place dots in the cells where the characters in the two sequences coincide.

- Write a function that given two sequences creates a dot plot by filling the matrix with 1s where there is a match and 0s where there is a mismatch.

- Write a function that given the dotplot matrix and the two input sequences does the pretty printing of the matrix.

Visual Alignment

- In their simplest forms, dotplot matrices place dots in the cells where the characters in the two sequences coincide.

```
1. def dotplot(seq1, seq2):
2.     """ basic dotplot algorithm: fills with ones coincident characters """
3.     mat = create_mat(len(seq1), len(seq2))
4.     for i in range(len(seq1)):
5.         for j in range(len(seq2)):
6.             if seq1[i] == seq2[j]:
7.                 mat[i][j] = 1
8.     return mat
```

```
1. def print_dotplot(mat, s1, s2):
2.     """ prints dotplot """
3.     import sys
4.     sys.stdout.write(" " + s2+"\n")
5.     for i in range(len(mat)):
6.         sys.stdout.write(s1[i])
7.         for j in range(len(mat[i])):
8.             if mat[i][j] >= 1:
9.                 sys.stdout.write("*")
10.            else:
11.                sys.stdout.write(" ")
12.        sys.stdout.write("\n")
```

Visual Alignment

- The algorithm can be noise. In order to reduce the noise, filter alignments by a minimum of matches. Apply a scanning window around each position and count the number of matching characters; if they pass a *stringency* threshold keep the alignment.

```
1. def extended_dotplot (seq1, seq2, window, stringency):
2.     """ extended dotplot with window and stringency parameters """
3.     mat = create_mat(len(seq1), len(seq2))
4.     start = int(window/2)
5.     for i in range(start,len(seq1)-start):
6.         for j in range(start, len(seq2)-start):
7.             matches = 0
8.             l = j - start
9.             for k in range(i-start, i+start+1):
10.                if seq1[k] == seq2[l]: matches += 1
11.                l += 1
12.                if matches >= stringency: mat[i][j] = 1
13.    return mat
14.

15.def test():
16.    s1 = "CGATATAGATT"
17.    s2 = "TATATAGTAT"
18.    mat1 = dotplot(s1, s2)
19.    print_dotplot(mat1, s1, s2)
20.
21.    print("")
22.    mat2 = extended_dotplot(s1, s2, 5, 4)
23.    print_dotplot(mat2, s1, s2)
```

Sequence Alignment

- In pairwise sequence alignment we try to arrange two sequences so that the number of matching characters is maximised. The order of the characters in the sequences cannot be altered.

- There are **many solutions** for a given input. Thus, this is an **optimisation** problem where we try to find the best solution among the possible ones.

- In order to select the best solution we need to define a proper *objective function*.

Input: two sequences (in a well defined alphabet) and an objective function.

Output: optimal pairing of the characters in each sequence; gaps are placed in each sequence to maximise the objective function.

Sequence Alignment

- The problem is highly complex. Consider for simplicity and as an example two sequences of length *n*.
- The maximum length of the whole alignment is *2n,* obtained by placing gaps all over one of the sequences.
- The total number of solutions is $\binom{2n}{n} = \frac{(2n)!}{n!^2}$ approximately 120 billion solutions!
- Brute-force approaches will not work.

Sequence Alignment

- The objective function attributes a value for each of the solutions. This is also called *score*.

- In this case we have a maximisation problem where we want to find the highest value for the most similar sequences and the lowest values for the most distant or less similar sequences.

- As we have seen before we can use a simple scheme of given 1 to a match and 0 to a mismatch. This will be an additive objective function. From the biological point of view this may not be the most appropriate function.

- Mutations typically affect adjacent positions. It would be important to discriminate between **matches**, **mismatches** and **gaps**. While the first contributes to a better alignment (positive contribution) this is not the case for the other two (negative contribution).

Sequence Alignment

- For DNA/RNA typical values include:
  - Match: +2 or +3
  - Mismatch and gap: -3 or -2

- For protein sequences we have a different alphabets. Aminoacids share physico-chemical properties that allow to group them. So a mismatch between R and A is very different than a mismatch between R and K.

- *Substitution matrices* capture the different scores between each amino acid pair. It assigns a value for each pair of possible symbols in the alphabet (either amino acid or nucleic acids) excluding gaps.

- For proteins they are calculated based on the probability of finding a given pair of aligned amino-acids in a good alignment over the probability of obtaining such pairing by chance.

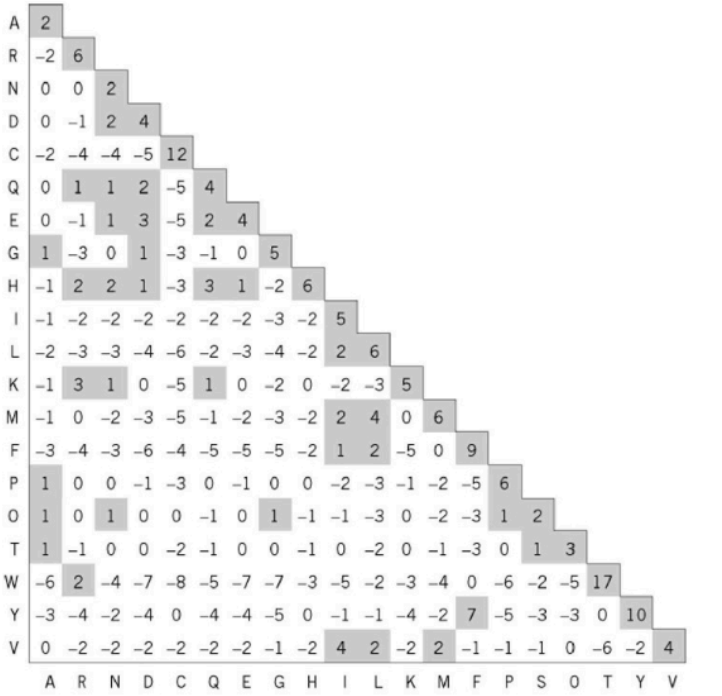
- From the biological point of view the introduction of a gap has a bigger impact. A *gap penalty function* defines how to penalize a gap or series of gaps.

PAM Matrix

- PAM. Point accepted mutation is a measure of the rate at which point mutations that substitute one amino acid residue for another. Was introduced by Dayhoff et al. to quantify the amount of evolutionary change in a protein sequence. It was based on the alignment of 1572 protein sequences from 71 families.

- One PAM unit is the amount of evolution which will change, on average, 1% of amino acids in a protein sequence.

- The PAM matrix provides the score of the substitution of one amino-acid by another.



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	O	T	Y	V
A	2																			
R	-2	6																		
N	0	0	2																	
D	0	-1	2	4																
C	-2	-4	-4	-5	12															
Q	0	1	1	2	-5	4														
E	0	-1	1	3	-5	2	4													
G	1	-3	0	1	-3	-1	0	5												
H	-1	2	2	1	-3	3	1	-2	6											
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5										
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6									
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5								
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6							
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9						
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6					
O	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2				
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3			
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17		
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

http://what-when-how.com/molecular-biology/point-accepted-mutation-molecular-biology/

BLOSUM

- BLOSUM (BLOcks SUbstitution Matrix) matrix is a substitution matrix used for sequence alignment of proteins. It is based on frequencies of substitutions in blocks of local alignments in related proteins (BLOCKS database).

- Each matrix is tailored to a particular evolutionary distance.

- Example: the scores in the BLOSUM62 matrix are derived using sequences sharing no more than 62% identity.

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

www.memim.com

Affine gap penalty

- Typical gap penalties have values ranging from -12 to -7 in protein sequences and -3 or -2 in DNA/RNA.

- The gap opening should have a larger penalty. The *affine gap penalty* model give a different penalty for the opening of the gap and for its extension:

  - *affine_gap(len) = g + r*len* , where *g* is the cost of gap opening, *r* the cost of each gap and *len* the extension of the gap (in symbols).

Objective function

- The objective function takes into account both substitution matrix and gap penalty.

- Develop a function to implement a substitution matrix given the alphabet, a value for match and a value for mismatch. Hint: use a dictionary.

Objective function

- The objective function takes into account both substitution matrix and gap penalty.
- Develop a function to implement a substitution matrix given the alphabet, a value for match and a value for mismatch.

```
1. def create_submat (match, mismatch, alphabet):
2.     """ substitution matrix as dictionary """
3.     sm = {}
4.     for c1 in alphabet:
5.         for c2 in alphabet:
6.             if (c1 == c2):
7.                 sm[c1+c2] = match
8.             else:
9.                 sm[c1+c2] = mismatch
10.         return sm
```

```
1. sm = create_submat(1, 0 , "ACGT")
2. print (sm)
```

Objective function

- In practice for protein substitution matrices it is easier to read from a table on an existing file. *blosum62.mat* provided in the class. Assume the matrix is symmetric and first row contains the symbols in the alphabet.

```
1. def read_submat_file (filename):
2.     """read substitution matrix from file """
3.     sm = {}
4.     f = open(filename, "r")
5.     line = f.readline()
6.     tokens = line.split("\t")
7.     ns = len(tokens)
8.     alphabet = []
9.     for i in range(0, ns):
10.        alphabet.append(tokens[i][0])
11.    for i in range(0,ns):
12.        line = f.readline();
13.        tokens = line.split("\t");
14.        for j in range(0, len(tokens)):
15.            k = alphabet[i]+alphabet[j]
16.            sm[k] = int(tokens[j])
17.    return sm
```

Objective function

- Write a function that provides the score of a column alignment, i.e. between characters *c1* and *c2*. Assume a constant gap penalty *g* and substitution matrix *sm*.

Objective function

- Write a function that provides the score of a column alignment or position, i.e. between characters *c1* and *c2*. Assume a constant gap penalty *g* and substitution matrix *sm*.

```
1. def score_pos (c1, c2, sm, g):
2.     """score of a position (column)"""
3.     if c1 == "-" or c2=="-":
4.         return g
5.     else:
6.         return sm[c1+c2]
```

Objective function

- Write a function that calculates the score of aligning two sequences *seq1* and *seq2*, given a gap *g* and matrix *sm*.

Objective function

- Write a function that calculates the score of aligning two sequences *seq1* and *seq2*, given a gap *g* and matrix *sm*.

```
1. def score_align (seq1, seq2, sm, g):
2.     """ score of the whole alignment """
3.     res = 0;
4.     for i in range(len(seq1)):
5.         res += score_pos (seq1[i], seq2[i], sm, g)
6.     return res
```

Objective function

- Write the test code for the alignments:

```
-CAGTGCATG-ACATA
TCAG-GC-TCTACAGA
```

Match = 2 and Mismatch = -2; gap = -3

```
LGPSSGCASRIWTKSA
TGPS-G--S-IWSKSG
```

Substitution matrix = blosum62.mat; gap = -8

Objective function

- Write the test code for the alignments:

```
1. def test_DNA():
2.     sm = create_submat(2,-2,"ACGT")
3.     seq1 = "-CAGTGCATG-ACATA"
4.     seq2 = "TCAG-GC-TCTACAGA"
5.     g = -3
6.     print(score_align(seq1, seq2, sm, g))
7.
```

```
8. def test_prot():
9.     sm = read_submat_file("blosum62.mat")
10.    seq1 = "LGPSSGCASRIWTKSA"
11.    seq2 = "TGPS-G--S-IWSKSG"
12.    g = -8
13.    print(score_align(seq1, seq2, sm, g))
```

Affine gap score

```
1.  def score_affinegap (seq1, seq2, sm, g, r):
2.      res = 0
3.      ingap1 = False
4.      ingap2 = False
5.      for i in range(len(seq1)):
6.          if seq1[i]=="-":
7.              if ingap1: res += r
8.              else:
9.                  ingap1 = True
10.                 res += g
11.         elif seq2[i]=="-":
12.             if ingap2: res += r
13.             else:
14.                 ingap2 = True
15.                 res += g
16.         else:
17.             if ingap1: ingap1 = False
18.             if ingap2: ingap2 = False
19.             res += sm[seq1[i]+seq2[i]]
20.     return res
21.
22.
23. def test_prot():
24.     sm = read_submat_file("blosum62.mat")
25.     seq1 = "LGPSSGCASRIWTKSA"
26.     seq2 = "TGPS-G--S-IWSKSG"
27.     g = -8
28.     r = -2
29.     print(score_align(seq1, seq2, sm, g))
30.     print(score_affinegap(seq1, seq2, sm, g, r))
```

Dynamic Programming for Global Alignment

- *Dynamic Programming* is a general-purpose class of optimisation algorithms based on divide-and-conquer strategy.
- The optimal alignment of two sequences is based on the optimal alignment of its sub-sequences.
- Two sequences A and B of size n and m:
  - $A = a_1 a_2 \ldots a_n$
  - $B = b_1 b_2 \ldots . b_m$
  - Build matrix S where rows indices are elements from A and columns indices are elements from B
  - S has n+1 and m+1 dimensions
  - Add extra column and extra row in the beginning to represent gaps in all positions of the sequences.
- Fill matrix S so that in each position the element indicates the score of the optimal alignment of the sub-sequences of A and B, considering all the symbols from the beginning until the character represented in the row from A and column from B.
- $S_{i,j}$ represents the alignment of $a_1 \ldots a_i$ and $b_1 \ldots b_j$

Dynamic Programming for Global Alignment

- Main ideia: fill matrix S cell by cell, using the value of adjacent cells to reach the target cell.
- This means using the previously calculated scores for smaller alignment to calculate the score of the current alignment.
- Recurrence relation:
  - $S_{i,j} = \max(S_{i-1,j-1} + sm(a_i, b_j), S_{i-1,j} + g, S_{i,j-1} + g)$ **for all $0 < i <= n$ and $0 < j <= m$**
  - *sm(c1,c2)* - substitution score between c1 and c2; g is constant gap penalty.

- Matrix is filled left to right and top to bottom. To calculate $S_{i,j}$ you need to have calculated: $S_{i-1,j}$ $S_{i,j-1}$ $S_{i-1,j-1}$.
- The recurrence relation states that:
  - An alignment is obtained by composition of other alignments.
  - Add an extra column to the previous alignment and compute the new score by adding the previous score with the one from the added column.

Dynamic Programming for Global Alignment

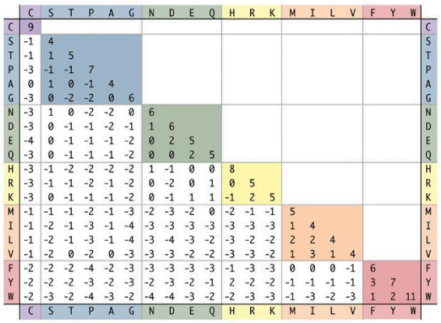
- Three ways to add columns to previous alignment (based on recurrence formula):
  - Add column with symbols from both sequences;
  - Add symbol from first sequence and gap;
  - Add gap and symbol from second sequence;
- Initialize the matrix:
  - Fill first row and column: multiply the number of columns or rows by g; $S_{i,0} = i * g$

S matrix for the Needleman-Wunsch

Substitution matrix BLOSUM62 and g = -8

		Sequence B				
		b_1	b_2	b_3	b_4	b_5
	gap	H	G	W	A	G
	gap	0	-8	-16	-24	-32	-40
a_1	P	-8	-2	-10	-18	-25	-33
a_2	H	-16	0	-4	-12	-20	-27
a_3	S	-24	-8	0	-7	-11	-19
a_4	W	-32	-16	-8	11	3	-5
a_5	G	-40	-24	-10	3	11	9

Sequence A



Example:

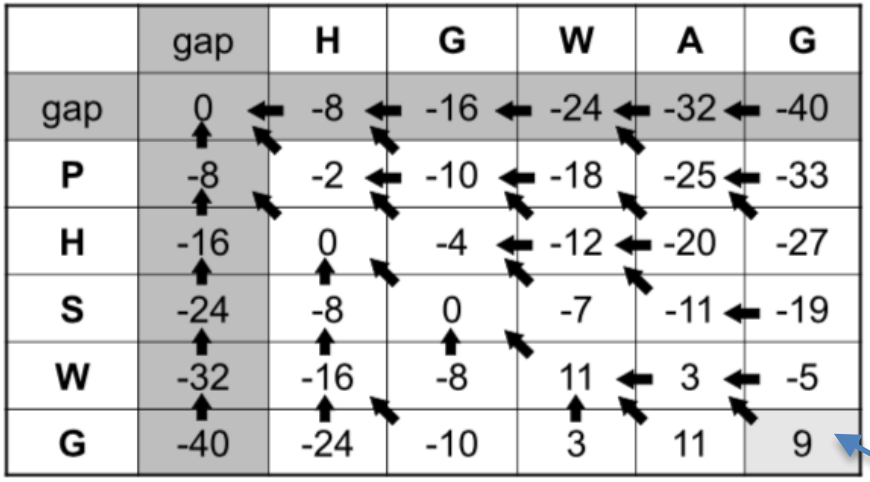
$S_{1,1} = max\ (\ S_{0,0} + sm(\text{“H”,“P”})\ ,\ S_{0,1} + g,\ S_{1,0} + g\ ) = max(\ 0\text{-}2,\ -8\text{-}8,\ -8\text{-}8) = -2$

Best alignment score

From Bioinformatics Algorithms, Rocha & Ferreira

Dynamic Programming for Global Alignment

- The previous procedure is sufficient to calculate the score of the best alignment but not the alignment it self. We need to keep track of the decisions that define the path to reach from the beginning of the sequences to the best alignment.
- Keep a matrix called trace-back (T) to keep all the three possible (3 symbol alphabet) moves at each cell. From T we can recover the alignment. Start from the lower-right cell and trace-back to upper-left cell. Moves:
  - Diagonal: add characters from both sequences.
  - Vertical: add character from first sequence and gap to the second.
  - Horizontal: add gap from the first sequence and character from the second.



From Bioinformatics Algorithms, Rocha & Ferreira

S and T matrices for the Needleman-Wunsch algorithm

Dynamic Programming for Global Alignment

- The process of recovering the optimal alignment from the trace-back information in the Needleman-Wunsch.

	gap	H	G	W	A	G
gap	0	-8	-16	-24	-32	-40
P	-8	-2	-10	-18	-25	-33
H	-16	0	-4	-12	-20	-27
S	-24	-8	0	-7	-11	-19
W	-32	-16	-8	11	3	-5
G	-40	-24	-10	3	11	9

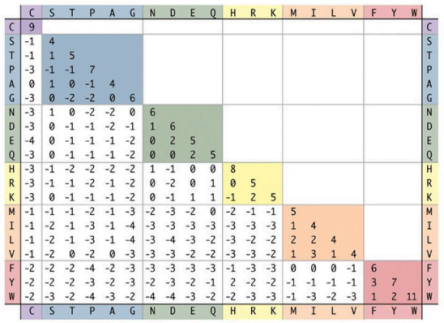
Best alignment:

```
P H S W - G
- H G W A G
```

Score of the best alignment:

-8 + 8 + 0 + 11 -8 + 6 = **9**

From Bioinformatics Algorithms, Rocha & Ferreira



Needleman-Wunsch Algorithm

- Fill matrices S and T (1 for diagonal, 2 for vertical and 3 for horizontal).

```
1.  def needleman_Wunsch (seq1, seq2, sm, g):
2.      """Global Alignment"""
3.      S = [[0]]
4.      T = [[0]]
5.      # initialize gaps in rows
6.      for j in range(1, len(seq2)+1):
7.          S[0].append(g * j)
8.          T[0].append(3)
9.      # initialize gaps in cols
10.     for i in range(1, len(seq1)+1):
11.         S.append([g * i])
12.         T.append([2])
13.     # apply the recurrence to fill the matrices
14.     for i in range(0, len(seq1)):
15.         for j in range(len(seq2)):
16.             s1 = S[i][j] + score_pos (seq1[i], seq2[j], sm, g);
17.             s2 = S[i][j+1] + g
18.             s3 = S[i+1][j] + g
19.             S[i+1].append(max(s1, s2, s3))
20.             T[i+1].append(max3t(s1, s2, s3))
21.     return (S, T)
22.
23. def max3t (v1, v2, v3):
24.     """Provides the integer to fill in T"""
25.     if v1 > v2:
26.         if v1 > v3: return 1
27.         else: return 3
28.     else:
29.         if v2 > v3: return 2
30.         else: return 3
```

Needleman-Wunsch Algorithm

- Recover the optimal alignment from T and A and B.
- Starts bottom right corner.
- Diagonal symbol in T adds symbols form A and B to the alignment.
- A vertical cell in T leads to move to previous low and adds to the alignment a symbol from A in the respective row and gap from B. Vice-versa if horizontal cell.

```
1. def recover_align (T, seq1, seq2):
2.     # alignment are two strings
3.     res = ["", ""]
4.     i = len(seq1)
5.     j = len(seq2)
6.     while i>0 or j>0:
7.         if T[i][j]==1:
8.             res[0] = seq1[i-1] + res[0]
9.             res[1] = seq2[j-1] + res[1]
10.            i -= 1
11.            j -= 1
12.        elif T[i][j] == 3:
13.            res[0] = "-" + res[0]
14.            res[1] = seq2[j-1] + res[1]
15.            j -= 1
16.        else:
17.            res[0] = seq1[i-1] + res[0]
18.            res[1] = "-" + res[1]
19.            i -= 1
20.    return res
```

Needleman-Wunsch Algorithm


```
1. def print_mat (mat):
2.     for i in range(0, len(mat)):
3.         print(mat[i])
4.
5. def test_global_alig():
6.     sm = read_submat_file("blosum62.mat")
7.     seq1 = "PHSWG"
8.     seq2 = "HGWAG"
9.     res = needleman_Wunsch(seq1, seq2, sm, -8)
10.    S = res[0]
11.    T = res[1]
12.    print("Score of optimal alignment:", S[len(seq1)][len(seq2)])
13.    print_mat(S)
14.    print_mat(T)
15.    alig = recover_align(T, seq1, seq2)
16.    print(alig[0])
17.    print(alig[1])
```

Exercises

- Apply the Needleman-Wunsch to the following sequences and parameters:
  - S1: TACT
  - S2: ACTA
  - g = -3
  - Match = 3
  - Mismatch = -1

Calculate matrices S and T and the optimal alignment and its score. Check if there are alternative optimal alignments.

- Write a program in Python, using the previous function to confirm your results.

Dynamic Programming for Local Alignment

- In the local alignment we are interested in the best partial alignment of the sub-sequences from the two sequences, which maximize the objective function (score).

- The *Smith-Waterman* is the algorithm for local sequence alignment. This algorithm reformulates the recurrence relation used in the DP.

- If the best of the three alternatives leads to a negative score, this means that in that position we need to restart the alignment from this position onwards, ignoring the previous parts of the sequences.

- Recurrence relation:
  - $\mathbf{S_{i,j} = max(S_{i-1,j-1} + sm(a_i, b_j), S_{i-1,j} + g, S_{i,j-1} + g, 0)}$ **for all** $\mathbf{0 < i <= n}$ **and** $\mathbf{0 < j <= m}$
  - *sm(c1,c2)* - substitution score between c1 and c2; g is constant gap penalty.

Dynamic Programming for Local Alignment

- Initialization of matrix S: first row and column filled with zero.
- The best alignment can occur in any cell of the matrix, corresponding to the highest score value in S.
- Multiple alternative best alignments may occur.

	gap	H	G	W	A	G
gap	0	0	0	0	0	0
P	0	0	0	0	0	0
H	0	8	0	0	0	0
S	0	0	8	0	1	0
W	0	0	0	19	11	3
G	0	0	6	11	19	17

Examples:

$S_{1,1} = max (S_{0,0} + sm(\text{“H”},\text{“P”}) , S_{0,1} + g, S_{1,0} + g, 0) = max(0\text{-}2, \text{-}8\text{-}8, \text{-}8\text{-}8, 0) = 0$

$S_{4,3} = max (S_{3,2} + sm(\text{“H”},\text{“P”}) , S_{3,3} + g, S_{4,2} + g, 0) = max(8+11, 0\text{-}8, 0\text{-}8, 0) = 19$

From Bioinformatics Algorithms, Rocha & Ferreira

Dynamic Programming for Local Alignment

- The trace-back matrix T contains four possible values: three previous values (diagonal, vertical, horizontal) and an extra value to the cases where the alignment is terminated (correspond to cells in S with 0).
- The recovery process starts on the cell with highest value and proceeds until the upper left corner is reached.

	gap	H	G	W	A	G
gap	0	0	0	0	0	0
P	0	0	0	0	0	0
H	0	8	0	0	0	0
S	0	0	8	0	1	0
W	0	0	0	19	11	3
G	0	0	6	11	19	17

Best alignments:

H S W
H G W

H S W G
H G W A

From Bioinformatics Algorithms, Rocha & Ferreira

Smith-Waterman Algorithm

- Fill matrices S and T (1 for diagonal, 2 for vertical and 3 for horizontal and 0 for termination).

```
1.  def smith_Waterman (seq1, seq2, sm, g):
2.      """Local alignment"""
3.      S = [[0]]
4.      T = [[0]]
5.      maxscore = 0
6.      # first row filled with zero
7.      for j in range(1, len(seq2)+1):
8.          S[0].append(0)
9.          T[0].append(0)
10.     # first column filled with zero
11.     for i in range(1, len(seq1)+1):
12.         S.append([0])
13.         T.append([0])
14.     for i in range(0, len(seq1)):
15.         for j in range(len(seq2)):
16.             s1 = S[i][j] + score_pos (seq1[i], seq2[j], sm, g);
17.             s2 = S[i][j+1] + g
18.             s3 = S[i+1][j] + g
19.             b = max(s1, s2, s3)
20.             if b <= 0:
21.                 S[i+1].append(0)
22.                 T[i+1].append(0)
23.             else:
24.                 S[i+1].append(b)
25.                 T[i+1].append(max3t(s1, s2, s3))
26.                 if b > maxscore:
27.                     maxscore = b
28.     return (S, T, maxscore)
```

Smith-Waterman Algorithm

- Define a function to find the cell in the matrix mat with max value. Return the respective row and column.

Smith-Waterman Algorithm


```
1. def max_mat(mat):
2.     """finds the max cell in the matrix"""
3.     maxval = mat[0][0]
4.     maxrow = 0
5.     maxcol = 0
6.     for i in range(0,len(mat)):
7.         for j in range(0, len(mat[i])):
8.             if mat[i][j] > maxval:
9.                 maxval = mat[i][j]
10.                maxrow = i
11.                maxcol = j
12.    return (maxrow, maxcol)
```

Smith-Waterman Algorithm


```
1. def recover_align_local (S, T, seq1, seq2):
2.     """recover one of the optimal alignments"""
3.     res = ["", ""]
4.     """determine the cell with max score"""
5.     i, j = max_mat(S)
6.     """terminates when finds a cell with zero"""
7.     while T[i][j]>0:
8.         if T[i][j]==1:
9.             res[0] = seq1[i-1] + res[0]
10.            res[1] = seq2[j-1] + res[1]
11.            i -= 1
12.            j -= 1
13.        elif T[i][j] == 3:
14.            res[0] = "-" + res[0];
15.            res[1] = seq2[j-1] + res[1]
16.            j -= 1
17.         elif T[i][j] == 2:
18.             res[0] = seq1[i-1] + res[0]
19.             res[1] = "-" + res[1]
20.             i -= 1
21.    return res
```

Smith-Waterman Algorithm


```
1. def test_local_alig():
2.     sm = read_submat_file("blosum62.mat")
3.     seq1 = "PHSWG"
4.     seq2 = "HGWAG"
5.     res = smith_Waterman(seq1, seq2, sm, -8)
6.     S = res[0]
7.     T = res[1]
8.     print("Score of optimal alignment:", res[2])
9.     print_mat(S)
10.    print_mat(T)
11.    alinL= recover_align_local(S, T, seq1, seq2)
12.    print(alinL[0])
13.    print(alinL[1])
```

Exercises

- Apply the Smith-Waterman to the following sequences and parameters:
  - S1: ANDDR
  - S2: AARRD
  - g = -8
  - Match = 3
  - Mismatch = -1

Calculate matrices S and T and the optimal alignment and its score. Check if there are alternative optimal alignments.

- Write a program in Python, using the previous function to confirm your results.

Exercises

- Calculate the identity function between two sequences. It should return a value of 0 to 1, where 1 is seq1 == seq2. You can derive this by the global alignment with a match score of 1 and mismatch or gap with 0 score. Normalize by the length of the longest sequence.
- The function should receive two sequences and the alphabet.

Exercises

- Calculate the identity function between two sequences. It should return a value of 0 to 1, where 1 is seq1 == seq2. You can derive this by the global alignment with a match score of 1 and mismatch or gap with 0 score. Normalize by the max sequence length.
- The function should receive two sequences and the alphabet.

```
1. def identity(seq1, seq2, alphabet = "ACGT"):
2.     sm = create_submat(1,0,alphabet)
3.     S,_ = needleman_Wunsch(seq1, seq2, sm, 0)
4.     equal = S[len(seq1)][len(seq2)]
5.     return equal / max(len(seq1), len(seq2))
```

Exercises

- Calculate the *edit distance* between two sequences. As we have seen before, edit distance is the minimum number of operations required to transform one sequence into the other using insertions, deletions and substitutions. In this case matches will have a 0 score and -1 for gaps and mismatches.

Exercises

- Calculate the *edit distance* between two sequences. As we have seen before, edit distance is the minimum number of operations required to transform one sequence into the other using insertions, deletions and substitutions. In this case matches will have a 0 score and -1 for gaps and mismatches.

```
1. def edit_distance(seq1, seq2, alphabet = "ACTG"):
2.     sm = create_submat(0, -1, alphabet)
3.     S = needleman_Wunsch(seq1, seq2,sm,-1)[0]
4.     res = -1*S[len(seq1)][len(seq2)]
5.     return res
```

Numerical Matrices Class

- Develop a class to keep and handle numeric matrices. The matrix should be represented as a list of lists.
  - The attributes:
    - Number of rows (*nrow*)
    - Number of columns (*ncol*)
    - List of lists to represent the matrix (*mat*)
  - Constructor:
    - Receive *nrow* and *ncol* and create an empty *mat*
  - Methods
    - Read and write a specific index *(i,j)*;
    - Calculate the sum of the values in mat;
    - Sum an value to all elements in mat;
    - Retrieve the max and the min value in mat;
    - Retrieve the indices of the mac and the min value in mat;
    - Check if the matrix is square (same number of rows and cols).
    - Multiply all elements by a value returning another matrix.