Sequence Alignment
**similarity:** the degree of resemblance between two sequences.
**identity:** the state of possessing the same subsequence. One often quantifies the percent identity between two sequences.
**homology:** the state of sharing a common evolutionary origin.
**orthology:** The state of being homologous sequences that arose from a common ancestral gene during speciation.
**paralogy:** The state of being homologous sequences that arose from a common ancestral gene from gene duplication.
Sequence alignment is the process of arranging the characters of a pair of sequences such that the number of matched characters is maximized. We can describe the alignment between two sequences with the following notation:

GCGTAACACGTGCG--
   |   ||| |||||
AC--AACCCGTGCGAC

The vertical bars “|”, or pipes, represent matching characters. Gaps, indicated by the dash “–” are inserted in between characters in place of missing characters to optimize the number of matches. It is critical that sequence alignments are viewed in a monospace font, such as Courier, so that the width of characters don’t offset the alignment.
Consider an ungapped alignment of the sequences $x$ and $y$:

\[
\begin{array}{cccccc}
| & | & | & | & | & | \\
\end{array}
\]

Let’s work out a scoring system for this, and then later introduce gaps.
For a random model, each of the characters $x[i]$ and $y[i]$ at each position $i$ is just randomly selected according to their frequency in the database

$$P(x, y|Random) = \prod_{i=1}^{n} p_{x[i]} p_{y[i]}$$

Alternatively, if the sequences have a common ancestor, we could also define a probability $q_{a,b}$ as describing the frequency of occurrence of substitutions in the database between two characters $a$ and $b$.

$$P(x, y|Ancestor) = \prod_{i=1}^{n} q_{x[i], y[i]}$$
It is important to note that the probability $q_{a, b}$ can be expressed as a product of the frequency of occurrence of $a$ times the conditional probability of a character mutating to $b$ given that it was $a$. Therefore,

$$q_{a, b} = P(a)P(b|a) = P(a)P(a \rightarrow b)$$
It is important to note that the probability \( q_{a,b} \) can be expressed as a product of the frequency of occurrence of \( a \) times the conditional probability of a character mutating to \( b \) given that it was \( a \). Therefore,

\[
q_{a,b} = P(a)P(b|a) = P(a)P(a \rightarrow b)
\]

Furthermore, we can describe the system as (time reversible) if and only if

\[
q_{a,b} = P(a)P(b|a) = P(b)P(a|b) = q_{b,a}
\]
If we combine our two probabilistic models, we can compute a odds ratio

\[
\frac{P(x, y|\text{Ancestor})}{P(x, y|\text{Random})} = \frac{\prod_{i=1}^n q_{x[i],y[i]}}{\prod_{i=1}^n p_{x[i]}p_{y[i]}}
\]

and by taking a logarithm, we can define a score as a log-odds ratio, which is more convenient for our purpose because it turns the product into a sum

\[
S = \log \left( \frac{P(x, y|\text{Ancestor})}{P(x, y|\text{Random})} \right) = \log \left( \frac{\prod_{i=1}^n q_{x[i],y[i]}}{\prod_{i=1}^n p_{x[i]}p_{y[i]}} \right) = \sum_{i=1}^n \log \left( \frac{q_{x[i],y[i]}}{p_{x[i]}p_{y[i]}} \right)
\]

The last terms give us our similarity matrix terms \( S_{a,b} \) defined as

\[
S_{a,b} = \log \left( \frac{q_{a,b}}{p_ap_b} \right)
\]
Substitution Matrices
A Point Accepted Mutation Matrix (PAM Matrix), is was the first systematically defined matrix for scoring the similarity of peptide sequences (Dayhoff 1978)

First we define a “Mutation Matrix” $M$ that describes the probability of mutating from $a$ to $b$.

$$M_{ab} = P(a \to b) = P(b|a).$$
If our database has $n_a$ occurrences of amino acid $a$, then we define our normalized frequency $p_a$ of that amino acid as

$$p_a = \frac{n_a}{N}$$

where $N$ is the total number of amino acids in the database. Because $\sum_a n_a = N$, the probabilities $p_a$ are normalized.
We can’t measure how much time has transpired between mutations in the sequences of our database. Therefore, we have to talk about how many substitutions have taken place as a percentage.

These mutation matrices are scaled such that there is 1\% amino acid change:

\[
\sum_{a=1}^{20} p_a M_{aa} = 0.99
\]

Therefore, this equation scales the mutation rates, and gives us restrictions on the protein alignments that we can consider in our database.
The actual PAM matrix that is used in scoring is defined as

\[ PAM_1(a, b) = \log \left( \frac{p_a q_{a,b}}{p_a p_b} \right) = \log \left( \frac{M_{a,b}}{p_b} \right) \]
To get more distant relationships, we can raise the matrix $M = M_1$ to a power.

$$M_n = (M_1)^n$$

Consider $M^2$:

The result of this matrix product gives the probability of mutating from $a$ to $c$ in twice the amount of time defined by the matrix $M$, and considering all intermediate amino acids $b$.

$$\sum_b M_{ab}M_{bc} = \sum_b P(a \rightarrow b)P(b \rightarrow c)$$
BLOSUM Matrices
BLOSUM is actually a series of matrices BLOSUM-\(x\), where they are built from alignments that are at least \(x\)% identical. These alignments come from the Blocks Database, which is still available at \texttt{http://blocks.fhcrc.org}.
Consider a single column of one of these Block alignments

\[
\begin{align*}
\ldots & \text{L}\ldots \\
\ldots & \text{L}\ldots \\
\ldots & \text{L}\ldots \\
\ldots & \text{I}\ldots \\
\ldots & \text{I}\ldots \\
\ldots & \text{I}\ldots \\
\ldots & \text{V}\ldots \\
\ldots & \text{V}\ldots
\end{align*}
\]

In this example, there are 3L, 2 Is, and 2 Vs. We wish to compute the number of pairs $f_{ij}$ for each pair of amino acids $i$ and $j$. 
Consider a single column of one of these Block alignments

\[ \ldots L \ldots \]
\[ \ldots L \ldots \]
\[ \ldots L \ldots \]
\[ \ldots I \ldots \]
\[ \ldots I \ldots \]
\[ \ldots V \ldots \]
\[ \ldots V \ldots \]

When the amino acid is the same, for example \( f_{LL} \) is the number of pairs that can be selected from \( n_L \) items, or \( \binom{n_L}{2} \), where \( n_L \) is the number of occurrences of the amino acid L. When the amino acids are different, such as \( f_{IL} \), the value is computed as the product of the number of occurrences of each amino acid, like \( f_{IL} = n_L \times n_I = 3 \times 2 = 6 \). To avoid double counting, we can only consider cases when \( i < j \).
These frequencies are normalized to make a $q$ matrix, with terms defined as

$$q_{ij} = \frac{f_{ij}}{\sum_{i=1}^{20} \sum_{j=1}^{i} f_{ij}}$$

Where the second sum in the denominator just goes up to $i$ to count half of the symmetric matrix $f_{ij}$ to avoid double-counting pairs.
We can get the normalized frequencies of each amino acid from these quantities by

\[ p_i = q_{ii} + \frac{1}{2} \sum_{j \neq i} q_{ij} \]

where the 1/2 term is because there is a probability of 1/2 that a random amino acid selected from the pairs corresponding to the \( q_{ij} \) pairs is \( i \).
Next we want to compute the probability \( e_{ij} \) of selecting the amino acids \( i \) and \( j \) by random.

\[
e_{ij} = \begin{cases} 
p_ip_j, & \text{if } i = j \\
p_ip_j + p_jp_i = 2p_ip_j, & \text{if } i \neq j \end{cases}
\]

To understand this equation, imagine selecting a pair of amino acids at random from the database. If they are the same, then the probability of that happening is just \( p_ip_i = p_i^2 \). If they are different, then there are two ways that could be selected: first \( i \) then \( j \), and first \( j \) then \( i \).
We can access values of a given substitution matrix using Biopython and the module Bio.SubsMat. For example, the most commonly used substitution matrix and the default for NCBI protein BLAST is BLOSUM62. We can print the values of the matrix or access a particular term by the following:

```python
>>> from Bio.SubsMat import MatrixInfo
>>> S = MatrixInfo.blosum62
>>> S['Q','Q']
5
>>> S['W','Q']
-2
```
If you play around with this matrix (actually it is a dictionary that takes a pair of characters as keys) you will realize that some pairs are stored, but others are not and return errors when one tries to access them. For example, \( S[\text{'W'},\text{'Q'}] \) is stored, but not \( S[\text{'W'},\text{'Q'}] \). To get around this, an additional function can be created:

```python
>>> def getScore(a,b,S):
...     if (a,b) not in S:
...         return S[b,a]
...     return S[a,b]
...
>>> getScore('Q','W',S)
-2
```
In addition to a scoring matrix, we also need to define penalties for gaps. The most common gap penalty is the linear gap penalty, defined as

$$c_L(d) = Gd,$$

which is just proportional to the length $d$ of the gap by a parameter $G < 0$. A more complicated approach is an “affine gap penalty”, which penalizes opening a gap by one parameter, and extending the gap by another parameter. For example, such a gap penalty can be defined by

$$c_A(d) = G + (d - 1)E$$

which includes a gap open parameter $G$ and a gap extension parameter $E$. In practice, an affine gap penalty is much more difficult to compute.
Dynamic Programming Algorithms
Dynamic Programming

One of the first attempts to align two sequences was done by Vladimir Levenstein in 1965, called “edit distance”, and now is often called Levenshtein Distance. The edit distance is defined as the number of single character edits necessary to change one word to another. Initially, he described written texts and words, but this method was later applied to biological sequences. One of the most commonly used algorithms for computing the edit distance is the Wagner-Fischer algorithm, a Dynamic Programming algorithm. Dynamic Programming optimally phrases the full problem as the optimal solution to the smaller pieces (sub-problems).
Dynamic Programming

Let’s consider aligning the nucleotide sequences $x = \text{CAGCTAGCG}$ and $y = \text{CCATACG}$. 
Let’s consider aligning the nucleotide sequences $x = \text{CAGCTAGCG}$ and $y = \text{CCATACG}$.

For Needleman-Wunsch, let’s define a matrix $F$, such that the terms $F_{i,j}$ correspond to the score of aligning the subsequences $x[1..i]$ and $y[1..j]$. We proceed from the upper left of this matrix at $F_{0,0}$, and fill in the matrix as we move from left to right and from top to bottom.
Dynamic Programming

Let’s consider aligning the nucleotide sequences \( x = \text{CAGCTAGCG} \) and \( y = \text{CCATACG} \). For Needleman-Wunsch, let’s define a matrix \( F \), such that the terms \( F_{i,j} \) correspond to the score of aligning the subsequences \( x[1..i] \) and \( y[1..j] \). We proceed from the upper left of this matrix at \( F_{0,0} \), and fill in the matrix as we move from left to right and from top to bottom. When computing the terms of the matrix \( F \), we need to define a set of boundary conditions, namely that the score is 0 at the boundaries so that \( F_{i,0} = i \times G \) and \( F_{0,j} = j \times G \) for \( 1 \leq i \leq |x| \) and \( 1 \leq j \leq |y| \).
\[ F_{i,j} = \max \begin{cases} F_{i-1,j} + G \\
F_{i,j-1} + G \\
F_{i-1,j-1} + S_{x[i],y[j]} \end{cases} \]  

skip a position of \( x \)  
skip a position of \( y \)  
match/mismatch  

(1)
**Scoring Matrix**

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>C</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
<td>-7</td>
<td>-8</td>
<td>-9</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table**: A dynamic programming matrix to compute the score for Needleman-Wunsch Alignment.
Figure: Each term of the matrix is computed using the recursion relation from the term above, to the left, and diagonally above-left.
<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
<td>-7</td>
<td>-8</td>
<td>-9</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
<td>-7</td>
</tr>
<tr>
<td>C</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>A</td>
<td>-3</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>T</td>
<td>-4</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>A</td>
<td>-5</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>C</td>
<td>-6</td>
<td>-4</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>-7</td>
<td>-5</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table**: A dynamic programming matrix to compute the score for Needleman-Wunsch Alignment.
Table: A traceback matrix for Needleman-Wunsch.
Finally, we have the result of the alignment. Here is the result of the Needleman-Wunsch alignment. As expected, because it is a global alignment, the full sequence is included and begins and ends on the first and last positions.

```
CAGCTAGCG
```

```
| || ||
```

```
C−CATA−CG
```