# COMP 5970/6970-004
# Computational Biology: Genomics and Transcriptomics
# Lecture notes 7: 2/3/2022

Haynes Heaton

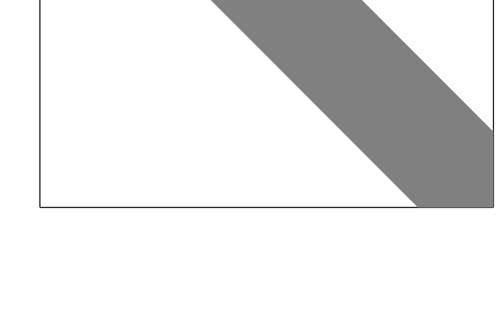
Spring, 2022

## Lecture Objectives

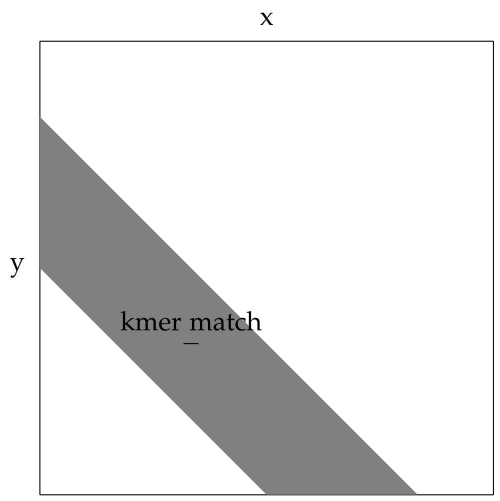
- Optimizations for alignment
- Alignment scoring
- Sequence similarity and differences
  - Molecular clock
  - Phylogenetic trees
- Dotplots and their patterns

## Optimizations for sequence alignment

All of the sequence alignment algorithms we have so far discussed take $O(n^2)$ time and space complexity. For large sequences, this becomes infeasible. One way this is generally sped up, is to only fill out part of the dynamic programming matrix. This is known as **banded** or **striped** sequence alignment. For instance, in global alignment the part of the score matrix we will out could look like this.



The width of the band is usually a parameter $b$. And with some index manipulation, the values can be packed into the top of the matrix thus making this linear in both time and space. Of course this will not guarantee the optimal alignment, but as long as the band is wide enough to encompass the optimal alignment with high probability, it is sufficient. Of course, the problem is then with overlap alignment and local alignment, how do we determine where to center th band. This is done via exact matching. We make a hash map of sequences of length $k$ in one sequence to the location in that sequence. We then go through sequences of length $k$ in the other sequence and if they share this k-mer, then we can anchor the alignment band on these shared exact match sequences.



And you can imagine this band being anchored by many kmer matches at different locations, making the band follow the key areas of sequence similarity.

## 1 Alignment scoring

So far we have used a very simple scoring system. But there are many different scoring systems one could use. Instead of a set value for match, mismatch, and indel, we could have different values for different types of matches and mismatches. We could build a substitution matrix giving different scores for each combination of alignments.

|   | A | C | G | T | - |
|---|---|---|---|---|---|
| A |   |   |   |   |   |
| C |   |   |   |   |   |
| G |   |   |   |   |   |
| T |   |   |   |   |   |
| - |   |   |   |   |   |

Because some substitutions are more likely that others. There are two different categories of these nucleotides, **purines** (A and G) and **pyrimidines** (C and T). It is much more likely for purines to mutate between one another and pyrimidines to mutate between each other than purine to pyrimidine or vice versus. These purine to purine mutations and pyrimidine to pyrimidine mutations are known as **transition** mutations and cross purine/pyrimidine mutations are called **transversion** mutations. And when dealing with RNA sequencing, C->T and A->G are more likely than the other possible transition mutations. Likewise, in protein alignment, some amino acids are more likely to substitute for each other than others. One reason for this is that amino acids may have similar DNA codons to one another, so fewer DNA mutations are required to make the substitution. Another reason is functional similarity. Some amino acids are hydrophobic wheras other proteins are hydrophilic. If a hydrophilic amino acid substitutes for a hydrophobic amino acid, it may dramatically change the shape of the protein likely killing its functionality. Thus evolutionarily we are likely to see more hydrophobic to hydrophobic substitutions. Another way we can change the scoring is to have different penalties for opening an indel versus extending it. This is termed **affine**

**gap** scoring and is used because the physical processes that create indels are either single events that cause multi-base indels or are more likely to extend and indel that was already created.

### 1.1 Probabalistic scoring

It would be good to have our scoring system have its basis in a statistical model. Our hypotheses could be that these sequences are related (by evolution or functionally) versus them occurring randomly. The random likelihood is the simplest. To see two bases aligned under a random background, it is just the product of the probability $q$ of seeing each base.

$$p(x,y|R) = \prod_i q_x i \prod_j q_y j \tag{1}$$

The alternative match model requires some knowledge of how similar related sequences are expected to be. For any two aligned bases, a joint probability $p_{ab}$ can represent the probability bases a and b were both derived from some common ancestor c.

$$p(x,y|M) = \prod_i p_{x_i y_i} \tag{2}$$

We could then use Bayes theorem

$$p(M|x,y) = \frac{p(x,y|M)p(M)}{p(x,y|M)p(M) + p(x,y|R)p(R)} \tag{3}$$

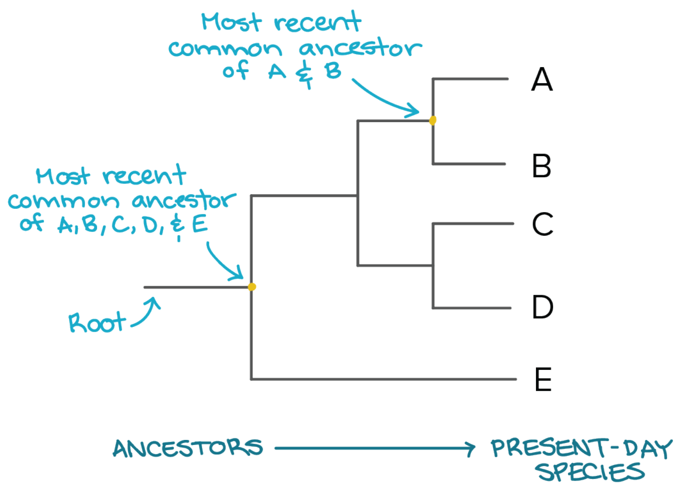
And to make this score additive in the alignment matrix, we take the log probability.

The downside of this model is that the log of the posterior probability is always negative which doesn't fit well with our local alignment paradigm of restarting a new alignment at every negative score. In this case, we want a score that is positive when it is more likely a match and negative with it is more likely random. The log odds ratio of the likelihood of a match vs the likelihood of random fits this well.

$$s(x,y) = log(\frac{p(x,y|M)}{p(x,y|R)}) \tag{4}$$

## 2 Sequence similarity

Under a **neutral hypothesis** in which mutations happen at random, the level of sequence similarity tells us something about how recently two sequences were derived from a common ancestor. This is termed the **Molecular clock**. But even if mutations happen at random, which mutations we observed is biased by evolution. We only observe sequences that exist and in order to exist, it must persist and/or replicate. And because some mutations are deleterious, we often see a lower mutation rate than the true mutation rate. But sometimes there are evolutionary pressures, especially when there is an environmental change for instance, where we observe a higher mutational rate than normal. So the times estimated by this molecular clock can be vastly incorrect at times. Still, it is a reasonable assumption that given that species a and b with more genetic similarity than species a and c, a and b have a more recent common ancestor than a and c. And people use this assumption to build phylogenetic trees. The leaves of a phylogenetic tree are generally the modern day species, the branches represent when they diverged from a common ancestor and the length of the lines usually represents an estimate of the time since a common ancestor.


Most recent common ancestor of A & B
Most recent common ancestor of A, B, C, D, & E
Root
A
B
C
D
E
ANCESTORS
PRESENT-DAY SPECIES

## 3 Dotplots

Dotplots are another common way to visually compare sequences. Before DNA sequencing technologies were developed there were protein sequencing technologies. Today the DNA sequencing technologies are many orders of magnitude cheaper and higher throughput than protein sequencing, but early sequence analysis was done on proteins. Because there are 20 different amino acids, one way people visualized similarities and differences between amino acid sequences was to have a heatmap of the two sequences in which cells for which both proteins had the same amino acid were black and otherwise were white. This could be done between two different proteins, or of a protein versus itself to see off-diagonal self similarity sometimes indicating functional units such as structures that cross a cell membrane. For DNA, because there are only 4 bases, doing the same thing would be very noisy as it is quite likely to see two of the same bases. So instead we consider multiple bases at once, and make a dot at locations where two sequences share a sequence of length $k$ and we call these sequences kmers. These dotplots have a few different patterns that emerge from different types of sequence similarity and differences.

```
import numpy as np
import pandas as pd
import scipy
from plotnine import *

## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/reticulate/python/rpytools/loader.py:3
```

Lets create some sequence to compare.

```
def random_sequence(n):
    return("".join(np.random.choice(["A","C","G","T"], n)))
```

and make some mutations for another sequence to compare it to.

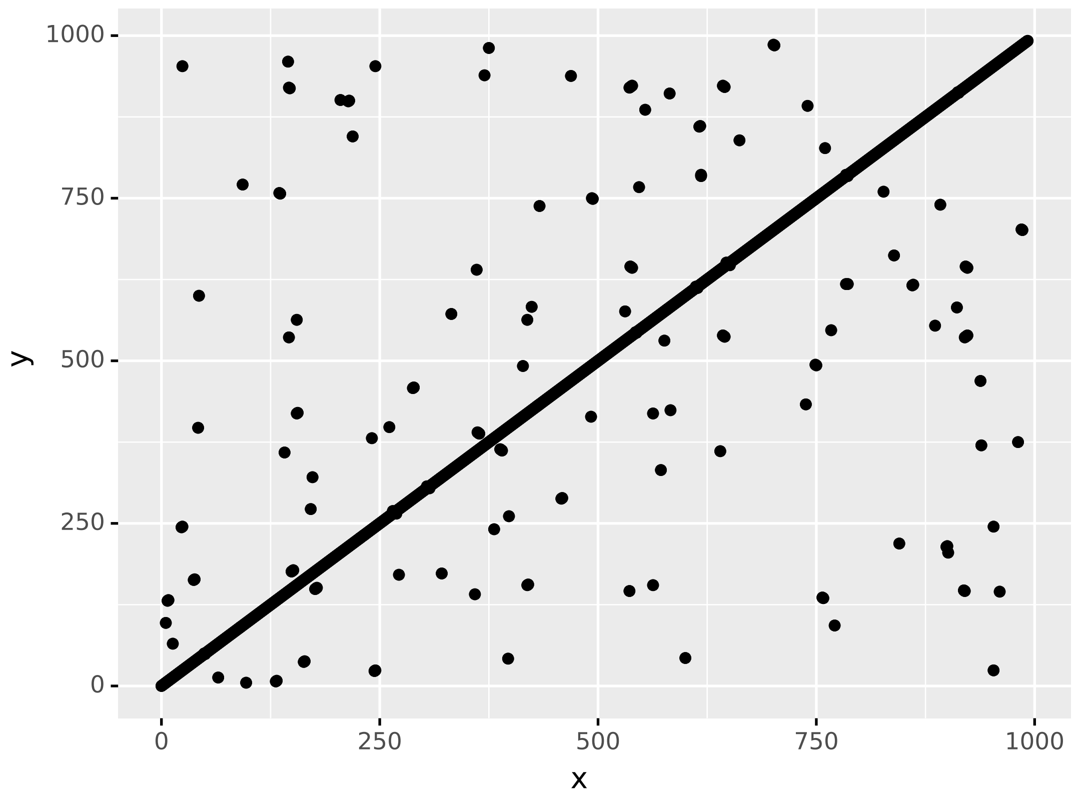
```
def mutate(s, snp_rate, indel_rate):
    x = [c for c in s]
    i = 0
    while i < len(x):
        if np.random.random() < snp_rate:
            x[i] = random_sequence(1)
        if np.random.random() < indel_rate:
            length = np.random.geometric(0.5)
            if np.random.random() < 0.5: # insertion
                x[i] = x[i] + random_sequence(length)
            else:
                for j in range(i,i+length):
                    if j < len(x):
                        x[j] = ""
                    i += 1
        i += 1
    return("".join(x))
```

```
s1 = random_sequence(1000)
s2 = mutate(s1,0.1,0.1)
```

```
def dotplot(s1, s2, k):
    s1_kmers = {}
    for i in range(len(s1)-k):
        kmer = s1[i:i+k]
        reverse_kmer = kmer[::-1]
        kmer_locs = s1_kmers.setdefault(kmer, [])
        kmer_locs.append(i)
        kmer_locs = s1_kmers.setdefault(reverse_kmer, [])
        kmer_locs.append(i)
    s1_locs = []
    s2_locs = []
    for i in range(len(s2)-k):
        kmer = s2[i:i+k]
        reverse_kmer = kmer[::-1]
        if kmer in s1_kmers:
            for j in s1_kmers[kmer]:
                s1_locs.append(j)
                s2_locs.append(i)
        if reverse_kmer in s1_kmers:
            for j in s1_kmers[reverse_kmer]:
                s1_locs.append(j)
                s2_locs.append(i)
    return((s1_locs, s2_locs))
```

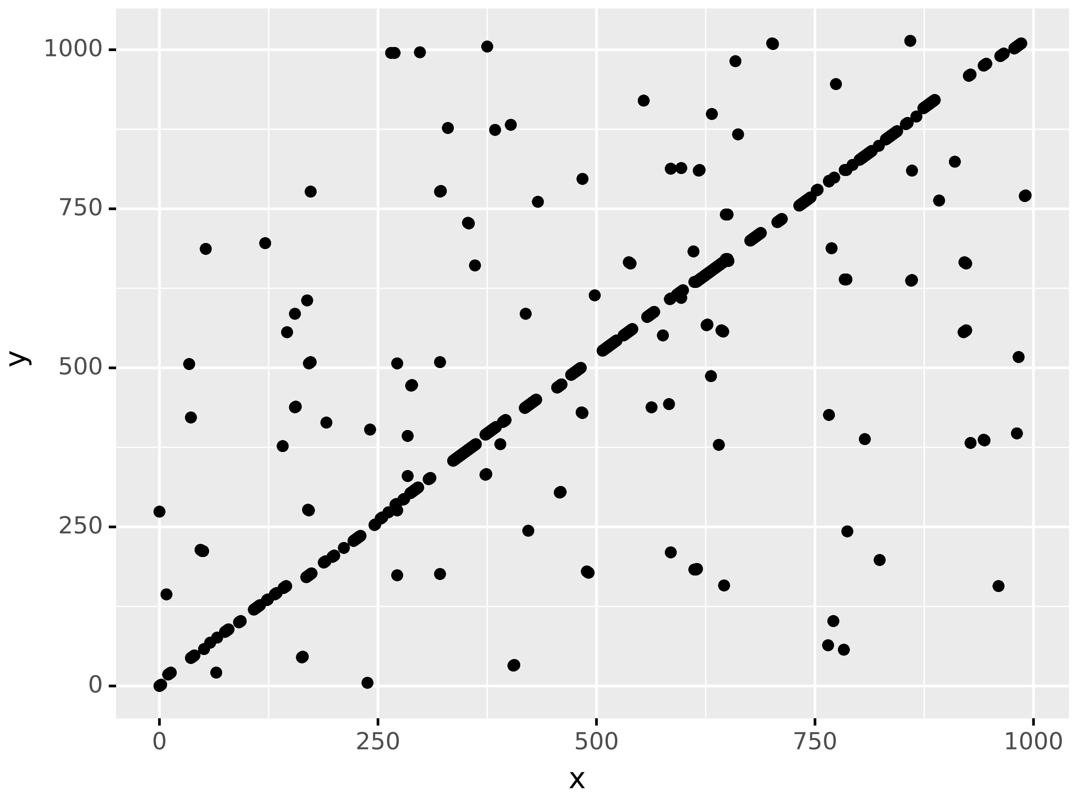
Here is an example of a self dotplot

```
(s1_locs, s2_locs) = dotplot(s1,s1,7)
df = pd.DataFrame({'x':s1_locs,'y':s2_locs})
ggplot(df)+geom_point(aes(x='x',y='y'))

## <ggplot: (8761583096140)>
```



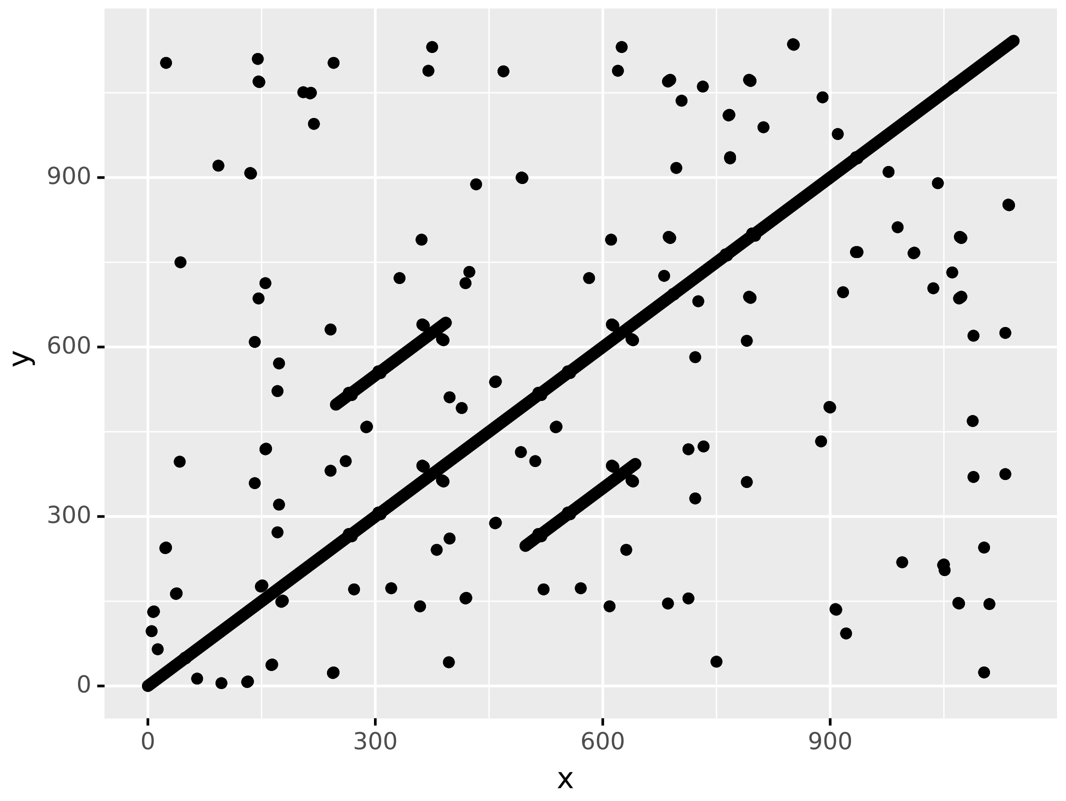
And a sequence vs a different but related sequence.

```
(s1_locs, s2_locs) = dotplot(s1,s2,7)
df = pd.DataFrame({'x':s1_locs,'y':s2_locs})
ggplot(df)+geom_point(aes(x='x',y='y'))

## <ggplot: (8761595324453)>
```



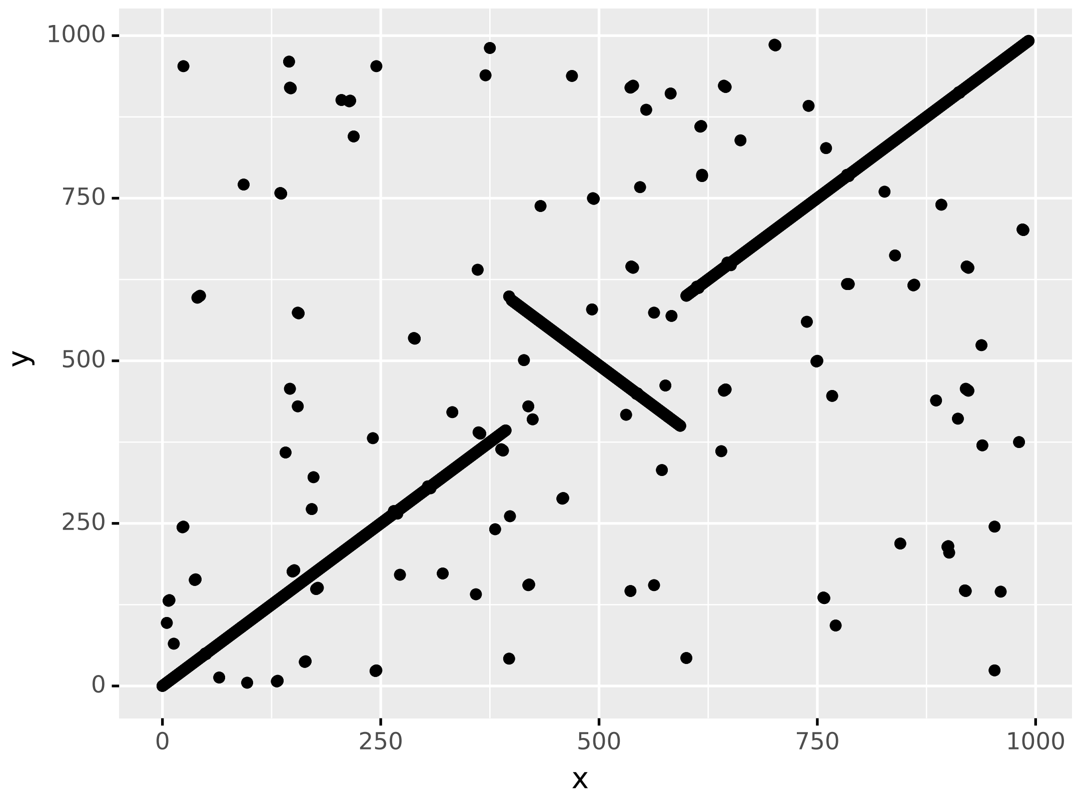
If a sequence has repeat and you compare it to itself, off-diagonals appear.

```
def repeat(s, n):
    loc1 = int(0.25*len(s))
    loc2 = int(0.5*len(s))
    rep = s[loc1:loc1+n]
    return("".join([s[0:loc2],rep, s[loc2:]]))
s2 = repeat(s1,150)
(locs1, locs2) = dotplot(s2,s2,7)
df = pd.DataFrame({'x':locs1,'y':locs2})
ggplot(df)+geom_point(aes(x='x',y='y'))

## <ggplot: (8761592172855)>
```



And if you compare a sequence to another sequence with an inversion in it, you get the opposite slope in that segment.

```
def inversion(s,n):
    loc = int(0.4*len(s))
    inversion = s[loc:loc+n]
    return("".join([s[:loc], inversion[::-1], s[loc+n:]]))
s2 = inversion(s1,200)
(locs1, locs2) = dotplot(s1,s2,7)
df = pd.DataFrame({'x':locs1,'y':locs2})
ggplot(df)+geom_point(aes(x='x',y='y'))

## <ggplot: (8761595363274)>
```



You can also get translocations of large sequences from one location to another. Pairwise alignment in the form of Needleman-Wunsch and the other variants of dynamic programming sequence alignment do not allow for inversions. Local alignment would pick up translocations only if you did Viterbi from multiple locations. So it is good to be aware of what types of genetic alterations your sequence comparison methods allow for and which they don't.