

# Practical Bioinformatics

Mark Voorhies

4/29/2011

# Our current tool set

- data (strings, floats, lists, nested lists)
- logic (if/then, try/except, for, while)
- functions (def, import, reload)

# Our current tool set

- data (strings, floats, lists, nested lists)
- logic (if/then, try/except, for, while)
- functions (def, import, reload)
- File I/O (open, csv)
- Random numbers (seed, shuffle, choice)

# Our current tool set

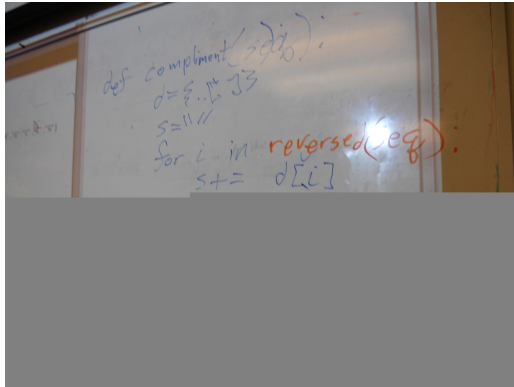
- data (strings, floats, lists, nested lists)
- logic (if/then, try/except, for, while)
- functions (def, import, reload)
- File I/O (open, csv)
- Random numbers (seed, shuffle, choice)
- Descriptive statistics (mean, pearson)



# Whiteboard Image

```
def distance(seq1, seq2):  
    d = 0  
    s = 0  
    for i in seq1:  
        s += d[i]  
    return s  
join(["A", "T", "C"])
```

# Whiteboard Image



```
def complement(s):  
    d = ''  
    for i in reversed(s):  
        d += chr(ord('0') - ord(i))
```

# Dictionaries

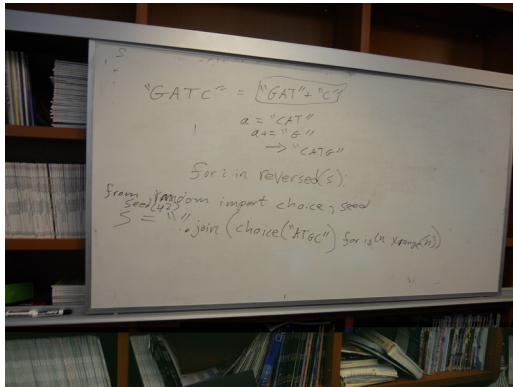
```
geneticCode = {"TTT": "F", "TTC": "F", "TTA": "L", "TTG": "L",  
               "CTT": "L", "CTC": "L", "CTA": "L", "CTG": "L",  
               "ATT": "I", "ATC": "I", "ATA": "I", "ATG": "M",  
               "GTT": "V", "GTC": "V", "GTA": "V", "GTG": "V",  
  
               "TCT": "S", "TCC": "S", "TCA": "S", "TCG": "S",  
               "CCT": "P", "CCC": "P", "CCA": "P", "CCG": "P",  
               "ACT": "T", "ACC": "T", "ACA": "T", "ACG": "T",  
               "GCT": "A", "GCC": "A", "GCA": "A", "GCG": "A",  
  
               "TAT": "Y", "TAC": "Y", "TAA": "*", "TAG": "*",  
               "CAT": "H", "CAC": "H", "CAA": "Q", "CAG": "Q",  
               "AAT": "N", "AAC": "N", "AAA": "K", "AAG": "K",  
               "GAT": "D", "GAC": "D", "GAA": "E", "GAG": "E",  
  
               "TGT": "C", "TGC": "C", "TGA": "*", "TGG": "W",  
               "CGT": "R", "CGC": "R", "CGA": "R", "CGG": "R",  
               "AGT": "S", "AGC": "S", "AGA": "R", "AGG": "R",  
               "GGT": "G", "GGC": "G", "GGA": "G", "GGG": "G" }
```



# Exercise: Transforming sequences

- 1 Write a function to return the antisense strand of a DNA sequence in 3'→5' orientation.
- 2 Write a function to return the complement of a DNA sequence in 5'→3' orientation.
- 3 Write a function to translate a DNA sequence

# Whiteboard Image



# Why compare sequences?

# Why compare sequences?

- To find genes with a common ancestor
- To infer conserved molecular mechanism and biological function
- To find short functional motifs
- To find repetitive elements within a sequence
- To predict cross-hybridizing sequences (e.g. in microarray design)
- To predict nucleotide secondary structure

**Homologs** heritable elements with a common evolutionary origin.

**Homologs** heritable elements with a common evolutionary origin.

**Orthologs** homologs arising from speciation.

**Paralogs** homologs arising from duplication and divergence within a single genome.

**Homologs** heritable elements with a common evolutionary origin.

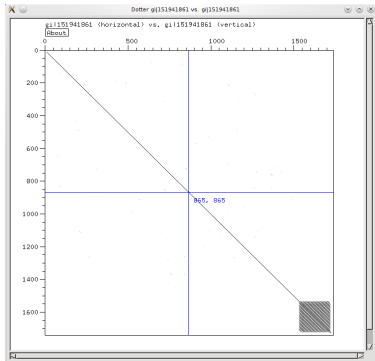
**Orthologs** homologs arising from speciation.

**Paralogs** homologs arising from duplication and divergence within a single genome.

**Xenologs** homologs arising from horizontal transfer.

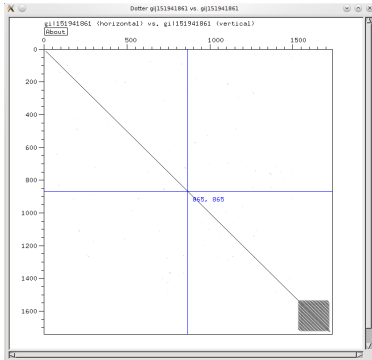
**Onologs** homologs arising from whole genome duplication.

# Dotplots





# Dotplots



If you're feeling ambitious

- 1 Given two sequences, write a dotplot in CDT format for JavaTreeView
- 2 Add a windowing function to smooth the dotplot

# Types of alignments

**Global Alignment** Each letter of each sequence is aligned to a letter or a gap (*e.g.*, Needleman-Wunsch)

**Local Alignment** An optimal pair of subsequences is taken from the two sequences and globally aligned (*e.g.*, Smith-Waterman)

# Exercise: Scoring an ungapped alignment

$s = \{$   
  " A " : { " A " : 1.0 , " T " : -1.0 , " G " : -1.0 , " C " : -1.0 } ,  
  " T " : { " A " : -1.0 , " T " : 1.0 , " G " : -1.0 , " C " : -1.0 } ,  
  " G " : { " A " : -1.0 , " T " : -1.0 , " G " : 1.0 , " C " : -1.0 } ,  
  " C " : { " A " : -1.0 , " T " : -1.0 , " G " : -1.0 , " C " : 1.0 } }  
 $\}$

# Exercise: Scoring an ungapped alignment

$s = \{$   
  " A " : { " A " : 1.0 , " T " : -1.0 , " G " : -1.0 , " C " : -1.0 } ,  
  " T " : { " A " : -1.0 , " T " : 1.0 , " G " : -1.0 , " C " : -1.0 } ,  
  " G " : { " A " : -1.0 , " T " : -1.0 , " G " : 1.0 , " C " : -1.0 } ,  
  " C " : { " A " : -1.0 , " T " : -1.0 , " G " : -1.0 , " C " : 1.0 } }  
 $\}$

$$S(x;y) = \sum_i^N s(x_i; y_i)$$

# Exercise: Scoring an ungapped alignment

$s = \{$   
  " A " : { " A " : 1.0 , " T " : -1.0 , " G " : -1.0 , " C " : -1.0 } ,  
  " T " : { " A " : -1.0 , " T " : 1.0 , " G " : -1.0 , " C " : -1.0 } ,  
  " G " : { " A " : -1.0 , " T " : -1.0 , " G " : 1.0 , " C " : -1.0 } ,  
  " C " : { " A " : -1.0 , " T " : -1.0 , " G " : -1.0 , " C " : 1.0 } }  
 $\}$

$$S(x;y) = \sum_i^N s(x_i; y_i)$$

- 1 Given two equal length sequences and a scoring matrix, return the alignment score for a full length, ungapped alignment.

# Exercise: Scoring an ungapped alignment

$s = \{$   
  " A " : { " A " : 1.0 , " T " : -1.0 , " G " : -1.0 , " C " : -1.0 } ,  
  " T " : { " A " : -1.0 , " T " : 1.0 , " G " : -1.0 , " C " : -1.0 } ,  
  " G " : { " A " : -1.0 , " T " : -1.0 , " G " : 1.0 , " C " : -1.0 } ,  
  " C " : { " A " : -1.0 , " T " : -1.0 , " G " : -1.0 , " C " : 1.0 } }  
 $\}$

$$S(x;y) = \sum_i^N s(x_i;y_i)$$

- 1 Given two equal length sequences and a scoring matrix, return the alignment score for a full length, ungapped alignment.
- 2 Given two sequences and a scoring matrix, find the offset that yields the best scoring ungapped alignment.

# Exercise: Scoring a gapped alignment

- 1 Given two equal length gapped sequences (where “-” represents a gap) and a scoring matrix, calculate an alignment score with a -1 penalty for each base aligned to a gap.

# Exercise: Scoring a gapped alignment

- 1 Given two equal length gapped sequences (where “-” represents a gap) and a scoring matrix, calculate an alignment score with a -1 penalty for each base aligned to a gap.
- 2 Write a new scoring function with separate penalties for opening a zero length gap (*e.g.*,  $G = -11$ ) and extending an open gap by one base (*e.g.*,  $E = -1$ ).

$$S_{gapped}(x;y) = S(x;y) + \sum_i^{gaps} (G + E * len(i))$$



- 1 Read chapter 3 of the BLAST book (Sequence Alignment).
- 2 Try initializing and filling in a dynamic programming matrix by hand (*e.g.*, try reproducing one of the examples from the BLAST book on paper).