Programming Lab I
Handout 3

Dr. Martin Vogt
martin.vogt@bit.uni-bonn.de

May 5, 2020

Deadline: May 20, 2020 **Next handout: May 12, 2020**

Dotplots

The dotplot is a matrix that gives an overview of the similarities between two sequences. The columns correspond to residues of sequence 1 and the rows to residues of sequence 2. In a simple dotplot, a matrix position (x, y) is left blank, if the residue y in sequence 1 and residue x in sequence 2 are different. On the other hand, the position (x, y) is filled, if residue y and residue x match. In the resulting dotplot, stretches of similar residues between sequence 1 and 2 show up as diagonals (in Northwest-Southeast direction). DNA also sometimes shows stretches of inversions, i.e. stretches of DNA copied back to front. These show up as diagonals from Northeast to Southwest. E.g.:

```
 |ABRACADABRACADABRA
-+------------------
A|*  * * *  * * *  *
B| *     *     *
R|  *     *     *
A|*  * * *  * * *  *
C|   *     *
A|*  * * *  * * *  *
D|     *     *
A|*  * * *  * * *  *
B| *     *     *
R|  *     *     *
A|*  * * *  * * *  *
C|   *     *
A|*  * * *  * * *  *
D|     *     *
A|*  * * *  * * *  *
B| *     *     *
R|  *     *     *
A|*  * * *  * * *  *
```

Simple dotplots often contain noise and are thus difficult to interpret. Applying filter rules during the creation of a dotplot can reduce this noise. For this purpose, a window size w and a stringency s can be introduced. Instead of considering two residues x and y individually, a small sequence of residues around each residue is considered, the so called *window*. The window is usually an odd number w and in addition to the residue itself $(w-1)/2$ residues before and after the central residue are considered. A dot will only be drawn only if at least a certain number of residues match. This number is called *stringency*. The simple dot plot above becomes a special case where $w = 1$ and $s = 1$.

Consider the two short sequences:

```
WIND_QWS
| ||  |
WQND_ERS
```

with the central residue `D` in both sequences and window size w and stringency s:

- For $w = 3$ and $s = 2$ a dot will be drawn because for `NDQ` and `NDE` there are 2 matches.
- For $w = 5$ and $s = 3$ no dot will be drawn because only two residues match in `INDQW` and `QNDER`
- For $w = 7$ and $s = 4$ a dot will be drawn because of the 4 matching positions.

Matrices in Python

Lists, dictionaries, and tuples are a number of high level general purpose data structures of Python for holding collections of elements that prove to be efficient and easy to use for a lot of practical applications in day-to-day programming. While they might not be optimal for every problem they can still be the data structure of choice considering that a) they are easy to use, b) you are already familiar with them and c) performance is not critical in a lot of situations. However, there are situations where using the 'standard' data structures of Python becomes cumbersome and inefficient. This is the case with matrices, i.e., two-dimensional arrays of numbers. In general, representing higher-dimensional data in vanilla Python is neither very efficient nor specifically intuitive to use. To alleviate this problem an external package `numpy` for handling vectors, matrices, and higher dimensional numerical data is available. Although it does not belong to the core language of Python it has become extremely popular simply because of the frequent need to handle numerical data. We will use it here to represent our matrices. First, let's see how matrices can be implemented in standard Python. One way to think about a matrix is as a list of lists of numbers. A 3x3 matrix could be written like

```
mat = [[1,2,3],[4,5,6],[7,8,9]]
```

and elements can be read using double indices, e.g. `mat[1][1]==5` or `mat[2][0]==7`, taking into account that counting starts at 0. Assignment of values also works: `mat[0][1] = -2` sets the second element of the first row to -2. This works in pretty standard fashion for small matrices, but trying to set up a matrix of arbitrary size is a little cumbersome. For instance if we would like to set up a matrix with dimensions corresponding to the length of two sequences `m=len(seqA)` and `n=len(seqB)` we could use something like:

```
mat=[]
for i in range(m):
row = [0]*n
mat.append(row)
```

or, even shorter:

```
mat=[[0]*n for _ in range(m)]
```

The dimensions of the matrix can be retrieved using `len(mat)` for the number of rows and len(mat[0]) for the number of columns.

The numpy package

The numpy package provides specialized classes for dealing with numerical data of higher dimensions. If you know the capabilities of R or Matlab you will have come to appreciate the capabilites of these languages to deal with arrays and matrices of numbers with high level statements that process all elements of a vector or matrix in a single statement without having to write explicit loops. If you are not familiar with numpy you can have a look at the tutorial on `https://docs.scipy.org/doc/numpy/user/quickstart.html` or check out the pdf at `http://www.engr.ucsb.edu/~shell/che210d/numpy.pdf`.

Programming exercises

Ex.1 *(0 pts) Matrices as lists of lists*
The previously mentioned solution for initializing matrices as lists of lists works fine. Now, consider the following modification, which tries to optimize setting up the matrix by initializing a single row of `n` 0's only once and append this row `m` times:

```
row = [0]*n # WRONG
mat=[]
for i in range(m):
mat.append(row)
```

or even shorter:

```
mat=[[0]*n]*m # WRONG
```

Why are these alternatives wrong? Demonstrate this using an example.

Ex.2 *(20 pts) The Dotplot matrix*

(a) *(4 pts) Generate dotplot matrix*
Write a function `dotplot(seqA,seqB,w,s,heading,filename)` that takes four parameters: two DNA or amino-acid sequences and two parameters `w` and `s` for the window size and stringency. The function should return a matrix where the number of rows (1st dimension) corresponds to the length of `seqA` and number of columns (2nd dimension) corresponding to the length of `seqB`. The matrix elements corresponding to positons `i` in `seqA` and `j` in `seqB` should contain 1 if at least `s` of the pairs of symbols in the windows of size `w` around position `i` and `j` match. Otherwise, it should contain 0. (*Note:* In some definitions of dotplots it is required that symbols at position `i` and `j` must always match regrardless of window size and stringency. Feel free to add this additional conditon to your implementation.)

- You can use the numpy package. E.g., `dp=zeros((len(seqA),len(seqB)),dtype=int)` creates a new two-dimensional array of the desired dimensions. Elements are accessed using double indices: e.g. `dp[1,2]` or double indexing `dp[1][2]`.

(b) *(4 pts) The Dotplot as ASCII art*
Write a function `dotplot2Ascii(dp,seqA,seqB,heading,filename)` that takes five parameters: A dotplot matrix `dp`, the two sequences from which it was generated; `heading` is a string to be used as title and `filename` is the name of an output file for the dotplot. The function should create an ASCII dotplot like the one given at the beginning of the handout. You should be able to use the function in the following way:

```
seqA="peter piper picked a peck of pickled peppers"
seqB="a peck of pickled peppers peter piper picked"
dp = dotplot(seqA,seqB,5,4)
dotplot2Ascii(dp,seqA,seqB,"Peter Piper's first dotplot","mydotplot.txt")
```

Test your program on lines from the file `peter.txt`. For the above example you can compare your result to `peters-dotplot.txt`.

(c) *(4 pts) Graphical output using matplotlib*
The ASCII output works fine for small sequences. However, for longer sequences a more graphical output would be preferred. For Python, the `matplotlib` package provides a rich interface for generating graphical outputs. The library is integrated within the python notebook. You can use something like

```
%matplotlib inline
import numpy as np
import matplotlib.pylab as plt
```

or even shorter

```
%pylab
```

which corresponds to

```
from numpy import *
from matplotlib.pylab import *
```

Feel free to check out some of the many tutorials; e.g.:

- `http://matplotlib.org/users/pyplot_tutorial.html`
- `https://realpython.com/python-matplotlib-guide/`
- `http://www.loria.fr/~rougier/teaching/matplotlib/`
- `http://jakevdp.github.io/mpl_tutorial/tutorial_pages/tut1.html`

Write a function `dotplot2Graphics(dp,hdA,hdB,heading,filename)` that takes a dotplot `dp` and labels `hdA` and `hdB` describing the two sequences as arguments to be used as labels for the x- and y-axis, and a title for the figure. `filename` should be the filename for saving the figure. The function should create a dotplot of the figure, save it to a file and display it on screen. (`filename` should be a filename with a graphics extension that `matplotlib` understands like `.png`, `.ps`, or `.pdf`.)

- Write two versions: the first version should plot the dots of the dotplot individually using an appropriate symbol, e.g. a small `+` or a small dot. The second version should use the `imshow` function of `matplotlib`, which takes the complete `dp` matrix as one argument. (Try to get the colors right.)
- Use `hdA` and `hdB` to label the axes and `heading` for the figure title.
- Ticks at the x- and y-axes should reflect the sequence positions of the respective sequences.
- Position (1,1) of the matrix can appear either in the bottom left or top left corner. Make sure that the origin is in the top left corner, so that sequence A reads from top to bottom and sequence B from left to right.

(d) *(4 pts) Write a script*
Combine all functions in a single script named `dotplot.py` the script should take a number of command line arguments:

```
python dotplot.py w s seqA seqB title output
```

The command line parameters `w` and `s` should give the window and stringency of the dotplot. `seqA` and `seqB` should be the names of two files containing sequences in FASTA format. `title` is a heading for the dotplot and `output` should be a filename to which the resulting dotplot is written. If the output filename has extension `.txt` the function `dotplot2Ascii` should be used for generating the dotplot and if it has a graphics extension (`.png`, `.ps`, or `.pdf`) use `dotplot2Graphics`. You should be able to call your program from within the notebook using:

```
%run dotplot.py 10 6 human_pax6_NM_001604.fasta mouse_pax6_NM_013627.fasta human-vs-mouse human-mouse.png
```

(e) *(4 pts) Generate some nice dotplots*
Generate dotplots for the following two pairs of sequences.

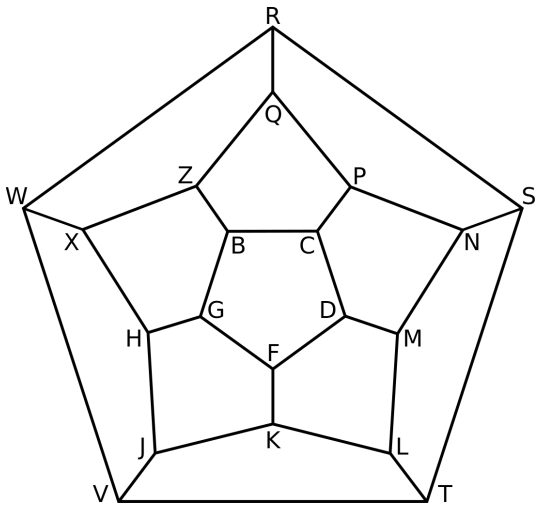
(a) `human_pax6_NM_001604.fasta` versus `mouse_pax6_NM_013627.fasta`

(b) `PAX6_HUMAN_P26367.fasta` versus `eyeless_NP_001014693.1.fasta`

Find "good" values for window size and stringency so that significant sequence similarities show up in the dotplot while the noise of random matches is reduced. Note, that one comparison is between DNA sequences while the other pair contains amino acid sequences. This should be reflected in different choices of width and stringency accordingly.

Ex.3 *(Optional: 3 pts) The Icosian Game*

This is a 19th-century puzzle invented by the renowned Irish mathematician Sir William Hamilton (1805-1865) and presented to the worls as the "Icosian Game". The game was playes on a wooden board with holes representing major world cities and grooves representing connections between them. The object of the game was to find a circular route that would pass through all the cities exactly once before returning to the starting point. Nowadays such a route is known as a Hamiltonian circle. (Taken from: A. Levitin, M. Levitin *Algorithmic Puzzles*. Oxford University Press 2011)



Write a program to find all Hamiltonian circles starting and ending at R.

Ex.4 *(Optional: 3 pts) Exact change please*

The Euro currency has 1, 2, 5, 10, 20, and 50 cent coins in addition to 1 and 2 euro coins (100 cents = 1 euro). Write a program to answer the question how many possibilities exist to get an exact change of 1 euro! Two possibilities are considered different if they differ in the number of coins of at least one denomination. The order in which the change is given does not matter. E.g., $1 \times 50\text{ct}, 1 \times 20\text{ct}, 2 \times 10\text{ct}, 4 \times 2\text{ct}, 2 \times 1\text{ct}$ would be one possibility and $4 \times 20\text{ct}, 10 \times 2\text{ct}$ another.