Python practical 1

Prof Guy-Bart Stan*
Department of Bioengineering & Imperial College Centre for Synthetic Biology, Imperial College London, London SW7 2AZ, United Kingdom

I. INTRODUCTION

In these instructions, Python code is in *italics*. The data files required for parts II and IV can be found on Blackboard: `https://bb.imperial.ac.uk`. Copy these files to your working directory, e.g., your desktop or wherever you like. All your tasks are highlighted in **bold font**. You are encouraged to work out the best way of presenting the information yourself. The Python library Matplotlib provides you with much more powerful tools for graphical display than spreadsheet programs, although you may find Matplotlib's tools more diffcult to use at the beginning.

In addition to me, there are teaching assistants in the class room who can provide assistance and guidance.

For other exercises specifically related to modelling gene expression and gene regulation, have a look at `http://openwetware.org/wiki/Imperial_College/Courses/2010/Synthetic_Biology/Computer_Modelling_Practicals`.

For more in depth coverage of modelling techniques, you are referred to my course "Modelling in Biology" which is available on my research webpage: `http://www.bg.ic.ac.uk/research/g.stan/#Lecture_Notes` at the section Lecture Notes.

II. EXPLORATORY DATA ANALYSIS

We will start with an introductory exercise to familiarise you with reading data into Python.

Epithelial cells form skin-like cell layers in order to provide a selective barrier between the interior of tissue or organs and their exterior. Signalling is rich: cells can communicate with their neighbors via physical forces, adhesion receptors, and signalling molecules, and change cell morphology and genetic programmes in response. The following data analysis is based on images of a cultured epithelial cell monolayer, obtained from confocal microscopy of cells for which adhesion receptor E-cadherin was stained (Fig. 1A). The images were pre-analysed to approximate cells as polygons of different classes, i.e., number of cell neighbours (Fig. 1B). For instance, cells in such a layer are often hexagonally shaped, corresponding to polygons of class 6. We will consider such data from control and mutant cells, and analyse the statistics of these data. The file *Epithelia_polygon_data.xls* provided to you contains the number of cells with different polygon classes from three repeat experiments (exp1, exp2, and exp3). Spreadsheet 1 contains the control data, and spreadsheet

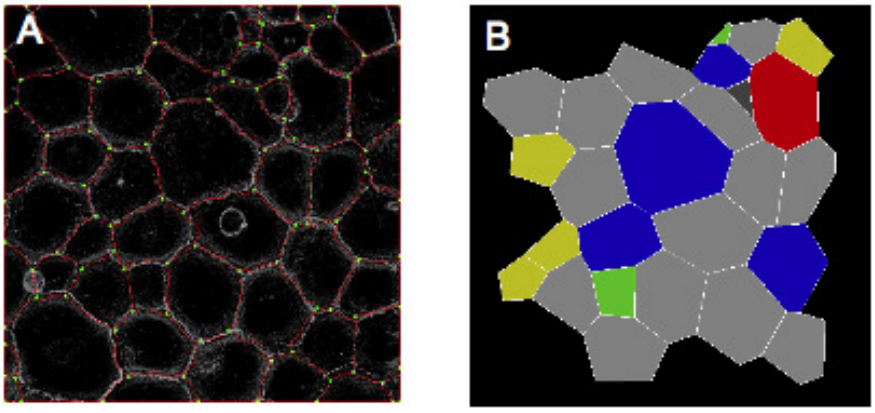

Figure 1: Epithelial cells. (A) E-cadherin stained cells. (B) Cells are approximated as polygons. Colors represent different polygon classes, e.g., gray cells are hexagons.

* E-mail: g.stan@imperial.ac.uk

2 contains the mutant data.

Question 1: Pandas is a Python library for data manipulation and analysis. Pandas is particularly useful to manipulate data stored in tables with labelled rows and columns — called dataframe. In the interactive mode, import pandas (*import pandas as pd*), then load the control data with *csheet = pd.read_excel('Epithelia_polygon_data.xls', sheet_name='Control')*. Display the matrix of control data by typing *print(csheet)*. Describe what you see (dimension and organization of the table/matrix) and compare with what you see when opening the same file with Excel. Do the same for the mutant data using (*sheet_name='Mutant'*).

Question 2: For the control data, obtain the vector of polygon classes with *C1=C[0:7]['n'])* as well as the matrix of frequencies for the polygon classes for the three experiments by typing *C2=C([0:7],[1,2,3])*. What do these Pandas expressions mean? Calculate the average frequencies of polygon classes with *nc=np.mean(C2,1)* (1 stands for average along the 1st dimension of matrix C2, there is a zeroth dimension before the 1st); and their associated standard deviations with *sc=np.std(C2,1,ddof=1)*. The 'ddof' tells the software to weight by N and not (N-1).

Plot a histogram summarising this information. The histogram should be in red with black error bars.

```
plt.style.use('ggplot')
x_pos = [i for i, _ in enumerate(mc)]

plt.bar(C1, mc, color='red', yerr=sc,
        align='center',
        ecolor='black',
        linewidth=2.0,
        capsize=3)

plt.xlabel("Polygon Class n")
plt.ylabel("Control")
plt.grid(False)
plt.show()
```

Repeat for mutant data using a blue colour for the histogram bars.

III. SOLVING HIGHER-ORDER DIFFERENTIAL EQUATIONS

The van der Pol oscillator was originally "discovered" by the Dutch electrical engineer and physicist Balthasar Van der Pol. Van der Pol found stable oscillations, now known as limit cycles, in electrical circuits employing vacuum tubes. His findings were reported in the September 1927 issue of Nature. The van der Pol equation has a long history of being used in both the physical and biological sciences. For instance, in biology, Fitzhugh and Nagumo used the equation as a model for action potentials of neurons. The equation has also been utilised in seismology to model the two plates involved during a geological fault.

Rewrite the Van der Pol second order differential equation (ODE)

$$\frac{d^2y(t)}{dt^2} - \left(1 - y^2(t)\right)\frac{dy(t)}{dt} + y(t) = 0, \tag{1}$$

in terms of a system of first-order ODEs.

This should give you:

$$\begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = \left(1 - y_1^2\right) y_2 - y_1 \end{cases}$$

with $y_1(t) = y(t)$ and the notations $\dot{y}_1 = \frac{dy_1}{dt}$, $\dot{y}_2 = \frac{dy_2}{dt}$.

We want to numerically integrate this ODE using Scipy in Python for the following conditions:

- $y_1(0) = 0$, $y_2(0) = 2$;
- numerical integration interval $[0, 20]$.

To numerically integrate this system of ODE we will use the SciPy ODE solver *RK45*. This is similar to the Matlab numerical solver ODE45.

Question 1: Write a function to describe the right-hand side of the ODEs (*def vdp1(t,y)*).

Question 2: Write some code to solve the ODE and plot the results $y_1(t)$ and $y_2(t)$ in a graph of y vs t. You can *import scipy.integrate* then get the solution with

```
scipy.integrate.solve_ivp(vdp, t_span=(0,20), y0=(0,2), method='RK45', rtol = 1e-6)
```

Hint: see lecture notes.

IV. OPTIONAL EXERCISE: FITTING MODEL TO DATA

The bacterial chemotaxis pathway, shown in Fig. 2, allows cells to sense and swim towards nutrients using a biased random walk of runs (straight swimming) and tumbling (random reorientation) of cells. This pathway includes chemoreceptors, which can be active or inactive. Ligand binding turns the pathway off, leading to straight swimming of the cells. Slow adaptation via receptor methylation increases the receptor activity and the probability of tumbling back to the pre-stimulus value again. The files *Am1_L.txt*, *Am2_L.txt*, and *Am3_L.txt* are data files given to you, containing dose-response curves of the receptor activity (column 2) and standard deviation (column 3) as a function of the ligand concentration (column 1) for three different methylation levels.

Question 1: You can load a data file into a numpy array with *am1_data = np.genfromtxt('Am1_L.txt')*. Load each data file, and plot with red, green, and blue colors.

```
# This listing is incomplete - fill in the gaps
am1\_data = np.genfromtxt('Am1_L.txt')
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
x1=am1_data[:,0] # conc
y1=am1_data[:,1] # response
ax.scatter(x1,y1,c='red',label='am1')
ax.scatter(x2,y2,c='blue',label='am2')
ax.legend()
plt.show()
```

Convince yourself that this does not look informative and redo the plot with a logarithmic x axis (*ax.set_xscale('log')*). Add the axis labels *Activity* (*ax.set_ylabel('Activity')*) and *Ligand concentration (M)* (*ax.set_xlabel('Ligand concentration (M)')*) to the y- and x-axis, respectively.

Question 2: The receptor activity is given by

$$A = \frac{a}{1 + e^{N\left(E + \log\left(\frac{1+\frac{L}{K_{off}}}{1+\frac{L}{K_{on}}}\right)\right)}}$$

where a is an overall amplitude factor, N is the receptor complex size, E is the methylation energy, K_{on} is the ligand disscociation constant in the on-state of the receptors, and K_{off} is the ligand dissociation constant in the off-state of the receptors.

This example code will help you fit the parameters to the model given the data and plot the result.

```
def predict_activity(L,x):
    """ Given ligand concentrations L, and parameters x, predict activities A"""
    a,N,E,Koff,Kon = x
    A = np.divide(a, (1+ np.exp(N* (E + np.log( (1 + np.divide(L,Koff))/(1 + np.divide(L,Kon)))))))
    return A

def loss(x,L,data):
    """squared difference between measured and predicted activity, for an array of data, and parameters
     x)"""
    diff = np.sum( (predict_activity(L,x)-data)**2)
    return diff

#initial guess
x0_guess=np.array([2,3,-2,0.00005,0.0005])

# minimize provides a unified interface to all of scipy's solvers.
# We'll use the Nelder-Mead solver, we also need to hand it x1 and y1
res = scipy.optimize.minimize(loss, x0_guess,method='Nelder-Mead',args=(x1,y1))
x_fit = res['x'] # fitted parameter values

logfig = plt.figure()
logax = logfig.add_subplot(1, 1, 1)
logax.set_xscale('log')
logax.scatter(x1,y1,c='red',label='am1')
```

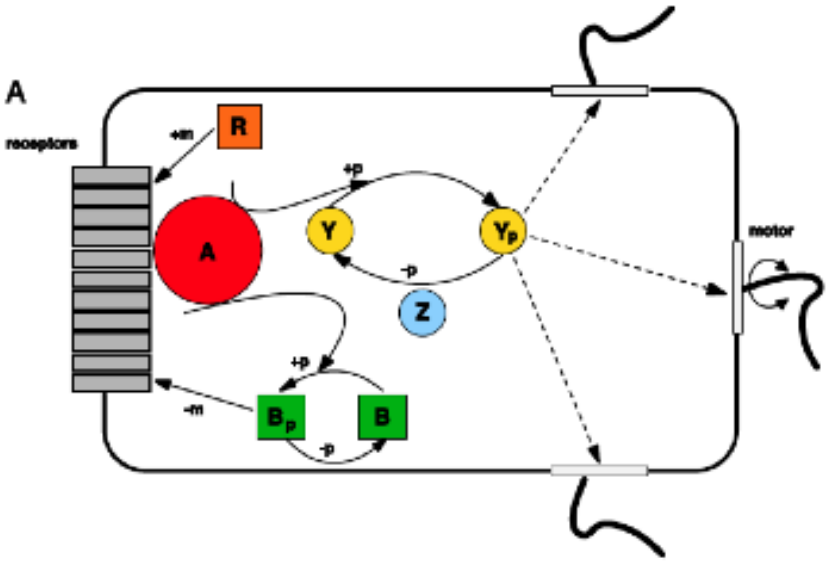

Figure 2: Chemotaxis pathways in *Escherichia coli*.

```
logax.plot(x1, predict_activity(x1,x_fit)) # plot predictions from parameters
logax.set_xlabel('Ligand Concentration (M)')
logax.set_ylabel('Activity')
logax.legend()
plt.show()
```

After minimisation, *x_fit* provides the best fit for the parameters.

Question 3: Compare data and fit by plotting the predictions against the data for Am1, Am2, and Am3. Save the plots and make a table of fitting parameters for each dose-response curve, i.e., for *Am1_L*, *Am2_L*, and *Am3_L*. Try different initial guesses.

Hint: see lecture notes.