Making animations with Python

The following page contains a few examples of animations you can make with Python:

http://matplotlib.org/1.5.3/examples/animation/index.html

We will look at two types of animations: a preexisting sequence of images using the method **ArtistAnimation** and animating an object calculated in a function using the method **FuncAnimation**. Both are methods of the **matplotlib.animation** package. Their respective documentation can be found there:

http://matplotlib.org/api/animation_api.html

Note that for each example below, you can either display your animation with **plt.show()** or save it in some movie format (I use mp4).
The following script generates a sequence of 50 images containing a normal random distribution of values on a grid 30x30:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

fig = plt.figure()

ims = []
for add in np.arange(50):
    base = np.random.randn(30,30)
#     ims.append((plt.imshow(base),))
#     ims.append((plt.pcolor(base),))
    ims.append((plt.pcolormesh(base),))

imani = animation.ArtistAnimation(fig, ims, interval=50,
repeat=False)
imani.save('random.mp4')
```

Commenting out the last statement and replacing it with a **plt.show()** will play the animation live. The method **ArtistAnimation** plays the sequence **ims**, called also an *Artist*. The sequence **ims** contains the individual images as a Python list, note that you can build an Artist sequence from different plotting calls (**pcolor**, **imshow** and **pcolormesh** in the examples above).

The method **FuncAnimation** allows you to create an animation from a function that can generate and update a plot. The plot could be any graphic object in Python. In the examples below I will show you how it works for a histogram, a simple line and a scatter plot.

The script below shows an animation of 20 frames, each frame is a histogram of 100 random numbers generated from a normal distribution:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

number_of_frames = 20
n_data_perframe = 100
data = np.random.randn(number_of_frames,n_data_perframe)

def update_hist(num, data):
#    plt.cla()
    print(num)
    plt.hist(data[num,:])

fig = plt.figure()
hist = plt.hist(data[0,:])

animation = animation.FuncAnimation(fig, update_hist,
frames=number_of_frames, repeat=False,fargs=(data, ) )
```

Inside `FuncAnimation`, the iteration over the number of frames is passed in `frames`. The function `update_hist()` generates a new histogram based on a new set of data passed with `fargs`. Check online what the other arguments are/can be. A new histogram is generated at each function call, but it is possible to freeze the plot environment for all calls as shown in other examples below.

Project 2 (Due date Tuesday November 7th 12p.m.)

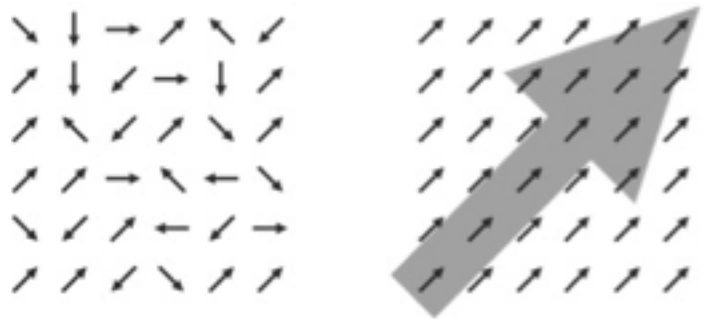
Some materials exhibit a natural magnetization. The mechanism behind this behavior is called **ferromagnetism**:

https://en.wikipedia.org/wiki/Ferromagnetism

In ferromagnetic materials, the individual electron magnetic moments μ tend to align relative to their closest neighbor, which creates a macroscopic magnetic field that is the source of natural magnets. The mathematical formulation of this phenomenon was first done using the so-called Ising model:

https://en.wikipedia.org/wiki/Ising_model

The Ising model describes the interaction between electron spins in a very simple way. The basic idea is the following: to simplify things, electrons are placed on a regular lattice, a grid. The difference between a ferromagnetic and non-magnetic material could be visualized as (left is non-magnetized material, right is ferromagnetic material, each little arrow represents an electron spin, which is aligned with the electron little magnetic moment μ):



We clearly see how macroscopic magnets can occur when all electron;s magnetic moments μ are aligned. To simplify the picture even more, let's assume that each electron at site "i" has a spin σ_i which can be up (σ_i =+1) or down (σ_i=-1), i.e. only two possible orientations. The energy H(σ) of the electrons network σ is modeled as:

$$H(\sigma) = -\sum_{i,j} J_{ij}\sigma_i\sigma_j$$

where the sum i,j is over all the close pairs (or close neighbors), σ_i is the spin of electron "i" and σ_j is the spin of electron "j". The quantity J_{ij} describes the interaction between nearby electrons, and to simplify we can take J_{ij}=1. An equilibrium state for the spins network is obtained when energy is minimized, but according to statistical physics, the probability P(σ) for the system to be in an equilibrium state σ at temperature T is given by:

$$P(\sigma) = \frac{e^{-\frac{H(\sigma)}{T}}}{Z_T}$$

where Z_T is called the partition function, and it is the sum over all possible configurations:

$$Z_T = \sum_{\sigma} e^{-\frac{H(\sigma)}{T}}$$

The partition function plays the role of a normalization constant for the probability P(σ) and you need not to worry about it. It is interesting to know that thermodynamic variables of the system (e.g. total energy, free energy, entropy, etc…) can be expressed as derivatives of Z_T.

For this project, you will create a spins network at some temperature T and simulate how its configuration evolves with time until it reaches equilibrium. The evolution of the system is entirely driven by the ratio of probabilities of two configurations σ and σ':

$$\frac{P(\sigma')}{P(\sigma)} = e^{-\frac{H(\sigma')-H(\sigma)}{T}}$$

This is the only quantity you will have to calculate in order to decide if a configuration change is happening or not.

Before we get into the details of the implementation, I want to discuss why this model was particularly important: back at the time, before WWII, it was known that ferromagnetic materials would loose their magnetic property above a certain temperature called the Curie temperature T_c (https://en.wikipedia.org/wiki/Curie_temperature). The resolution of the Ising model in two dimensions (which is the topic of your project) proved the existence of the Curie temperature, and it provided a convincing physical and mathematical explanation of ferromagnetism described as a phase transition (https://en.wikipedia.org/wiki/Phase_transition). The Ising model was considerably developed later on and it gave rise to many fundamental branches of physics, related to quantum field theory, phase transitions, critical phenomena and much more. This is why this model is of highly historical importance!

Practical implementation:

1- Create a grid of size 50x50. Each grid cell is an "electron", you will randomly assign a value +1 or -1 for each cell, which represents the electron spin orientation (up or down). This is you initial electron spins configuration σ. Choose a value for the temperature T.

2- Calculate the energy H(σ) of the configuration setup in 1-. You will use and **periodic boundary conditions**, which we wrap around the states at the boundary (edge). So that the right edge is immediately left of the left edge and the bottom edge is immediately above the top edge.

3- To calculate the next possible configuration, you will randomly pick an electron and flip its spin. This creates a possible new configuration σ'. In order to decide if this new configuration is accepted by the system or not, you first have to calculate the new energy H(σ'). If H(σ') < H(σ) you always accept the new configuration because it means the new energy is lower than the previous one. However, if the new energy H(σ') is higher than H(σ), then you only accept the transition only with the probability:

$$p = e^{-\frac{H(\sigma')-H(\sigma)}{T}}$$

4- Repeat steps 2 and 3 until you reach convergence (see **objectives**).

5- Once convergence is achieved you will calculate the average magnetic moment of the whole network $\mathcal{M}$ from:

$$\mathcal{M} = \mu \sum_i \sigma_i$$

Objectives: the goal of this project is to re-discover the Curie temperature T_c and see how ferromagnetism develops spontaneously from the Ising model. Once your code is working, you will calculate the average magnetic moment $\mathcal{M}$ for the following system temperatures (which are given in some arbitrary units):

T=[0.01,0.1,1.,2.,3.,4.,5.,10.,100.]

For each temperature, you will run your code 5 times in order to get 5 independent measurements of $\mathcal{M}$. For each of these 5 moment values, you will only keep the largest one and generate a plot of this maximum $\mathcal{M}$ value as function of temperature. You should be able to see the phase transition and approximately estimate the Curie temperature.

For step 4, you will assume that convergence is achieved after repeating steps 2 and 3 600000 times.

You will take μ=1 and J_{ij}=1.
Your code should be called `ising.py` and it should generate the plot max($\mathcal{M}$) as a function of temperature automatically. This plot should be called `Tcurie.pdf`.
In addition, you will generate 3 movies that show the time evolution of the spins network for 3 different temperatures: T=[0.1 ,2.5, 100]. You will name these movies

`temp_0.1.mp4`, **`temp_2.5.mp4`** and **`temp_100.mp4`** respectively. There is no need to include the 600000 time step in the movie, you can just include one network image every thousand.