Scientific Computing in Python 5 – Modelling using ODE's 1

Yesterday we used a couple methods to solve ODE's. We also saw how ODE's can relate to real world problems through our panda and the cabbage equation.

In reality these two ODE's should be feeding into each other so that the population of either relies on the population of the other at any specific time. When we have a predator prey system like that it is often called Lotka Volterra models. To start with we will do our cabage panda equation where the panda growth is restricted by the number of cabbages and the cabbage growth is restricted by the number of pandas eating them.

We can get this to work though, as long as we can create our ODE's. Lets start with the carrying capacity equation but we'll add in a mortality of pandas.

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{0.5C}\right) - mN$$

Where 0.5C is a limit on the population based on the number of cabbages. A panda eat's 2 cabbages per week so the number of pandas a that can be supported is number of Cabbages/2

$$\frac{dC}{dt} = rC\left(1 - \frac{C}{K}\right) - 2N$$

Cabbages have a similar equation but a carrying capacity is decided on the amount of nutrients in the soil. We will assume for now that this is fixed at 1000 cabbages. I've adjusted the growth rate of each to make it a more stable equation.

What we now want is a way of referring to N and C within our ODE solvers so that it updates each timestep.

For this we will create an array of the equations where N = y[0] and C = y[1]

```
def pandafun(y,t):
    return np.array([((0.01*y[0])*(1 - y[0]/(0.5*y[1]))-(0.001*y[0])),
    ((0.6 * y[1]) * (1 - (y[1]/k)) - (2*y[0]))])
#we have definite our equations in an array.
# now we define our starting points in this cas 2 pandas and 100 cabages.
y0 = np.array([2,100])
#the number of timesteps to run it over
t = np.linspace(0,400,400)
#and plug it into the ODE solver
answer = odeint(pandafun,y0, t)
#pull out the number of cabbages at each step
```

```
cabs = answer[:,1]

#and the number of pandas

pands = answer[:,0]


#now plot them

plt.plot(t,cabs)

plt.plot(t,pands)
```

We use an older ode solver in this case as the new one isn't fully supported yet. We can see from this that our population of cabages quickly gets to 1000 then as the pandas increase slowly starts to drop. It will eventually hit 0 but we get boundary issues in the ODE with too many timesteps as it starts to represent incredibly tiny changes of a bounded unit population. i.e. losing 0.1 of a panda. Play around with the various parameters and see where you get crashes etc.

Change the starting number of pandas, cabbages, their growth rates or their mortalities, see if you can generate a stable system where they reach an equilibrium. This is population modelling.

If we wanted we could parralelise this and perform a parameter sweep, trialling every possible parameter combination, but this would probably need to be on a super computer!

Problem 1: If we try and run this ODE for too many time steps when the population crashes we get impossible things like sudden spikes to trillions of cabbages or pandas, this is obviously not possible and is due to errors in the way the ODE calculates things. The first problem is to add bounds to the ODEs. You want to be using if statements to prevent things going negative or above certain thresholds. This helps enforce your already present equation limits.

Challenge Problem

Problem 2: How about we tackle where our carrying capacity of cabbages is coming from? Add a third equation that describes the quality of the soil and controls the carrying capacity of the cabbages.

Cabbages are limited by the nutrient content of the soil which is dependent on the rain and temperature. Also the area available to grow in which is a constant limiting factor.

Imagine your own way these can work. Come up with your own numbers and try them out. If you like google some real parameters for all these things.