Predator-prey relationships

PHYS 246 class 10

https://lkwagner.github.io/IntroductionToComputationalPhysics/intro.html

Announcements/notes

```
from google.colab import drive
drive.mount('/content/drive')
!cp /content/drive/MyDrive/Colab\ Notebooks/Dynamics.ipynb ./
!jupyter nbconvert --to HTML "Dynamics.ipynb"
```

- 'Markov chains' is due tonight
- There are updates as of 11 AM today. Make sure you get the current version.
- April 17th -- project proposals due
- May 13th NOON -- slides and ipynb file due

Q: For the parameters above, is the first derivative of bunnies or foxes positive (which initially increases)?

A:

Q: Suppose you were to reverse the initial conditions, so that there are more predators than prey at the beginning. How would the answer to the above change (you can try it!)?

A:

Q: What is the relationship between the maximum in bunnies vs foxes? In phase, out of phase by 180 degrees, something else?

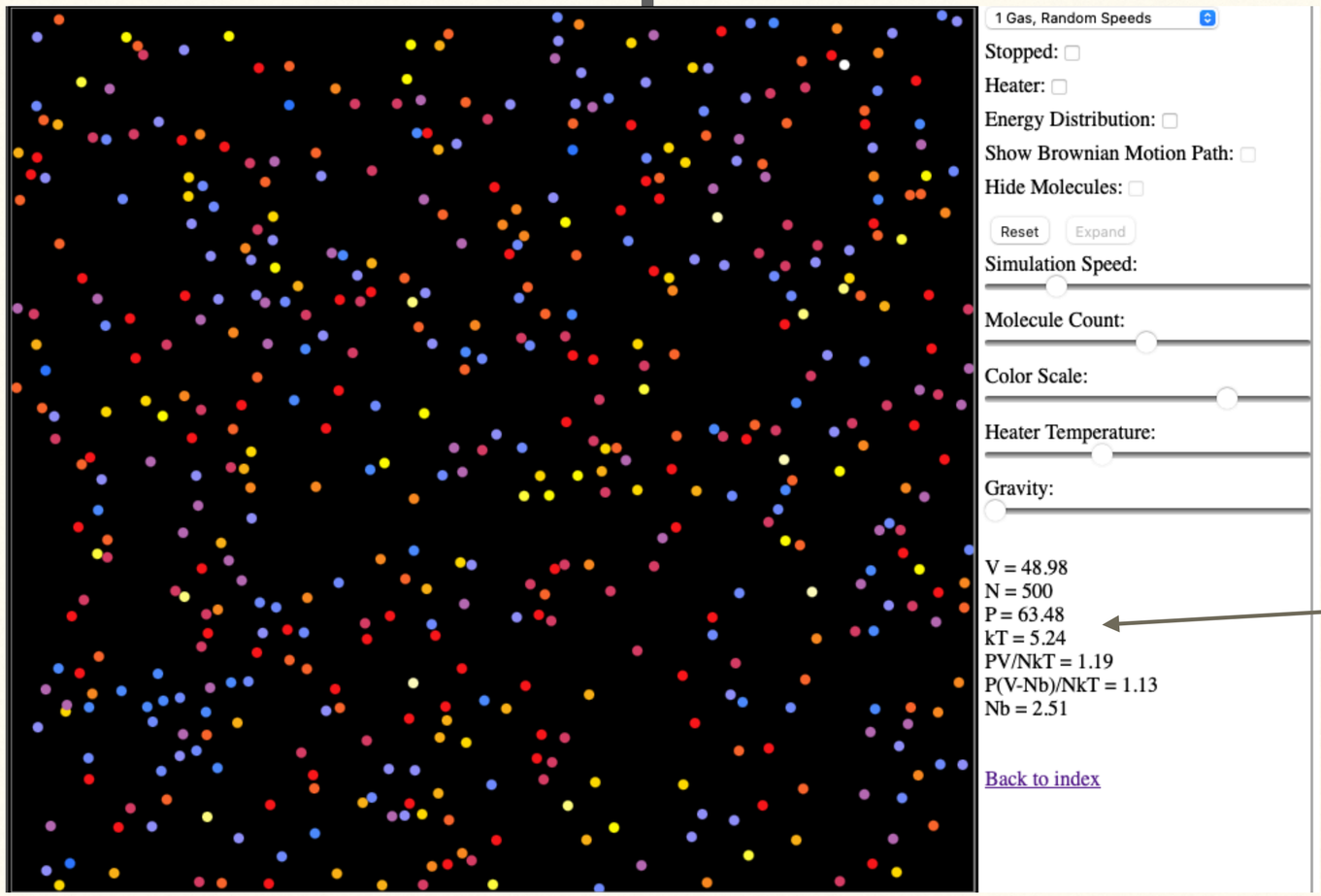
A:

Project description

This assignment is fairly open-ended, but make sure that it is achievable. A relatively small project that gets completed well is better than an ambitious one that you don't finish. We also know that sometimes you will encounter issues that necessitate a change of plans from your original proposal. That's ok, just make sure your reasoning for changing the plan is (briefly) documented. Some examples that might help you choose your scope (please don't make it too derivative of these; this is a time to show your creativity):

- Re-implement an expensive calculation we did so that it runs on the GPU (JAX can be part of this), and benchmark how much speed increase you get.
- Do a full simulation of the solar system including all 8 (9??) planets.
- Look into the SIR model that was used for Covid. Compare to predator-prey.
- Improve your ML model for galaxy classification to above 80% accuracy. You will need a better neural network for this.

General concept: discrete versus continuous models



Thermodynamics: averaged quantities, smooth

Statistical mechanics
-- discrete, fluctuations

Lotka-Volterra equations: continuous

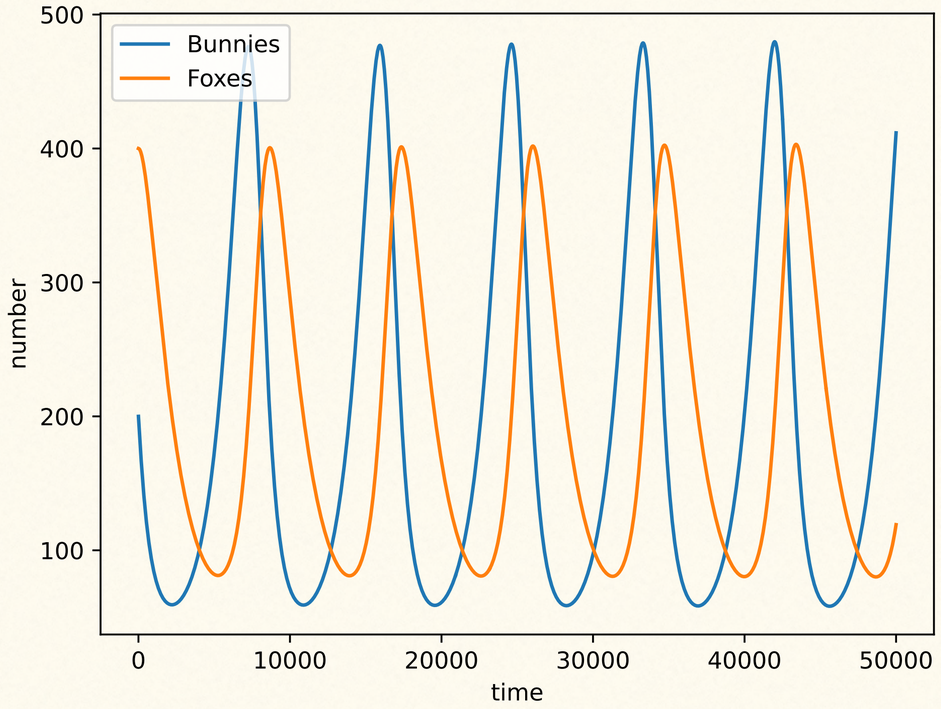
$$\frac{dB(t)}{dt} = \alpha B(t) - \beta F(t)B(t)$$

$$\frac{dF(t)}{dt} = DF(t)B(t) - \gamma F(t)$$

α : reproduction rate of bunnies
γ : death rate of foxes
β : how often bunnies get eaten by foxes
D : how many foxes reproduce

Basic idea: foxes are machines that turn bunnies into more foxes.

Time-stepping Lotka-Volterra



$$\frac{dB(t)}{dt} = \alpha B(t) - \beta F(t) B(t)$$

$$B(t + \Delta t) \simeq B(t) + \left(\frac{dB}{dt} \right) \Delta t$$

$$B(t + \Delta t) \simeq B(t) + (\alpha B(t) - \beta F(t) B(t)) \Delta t$$

Stochastic method

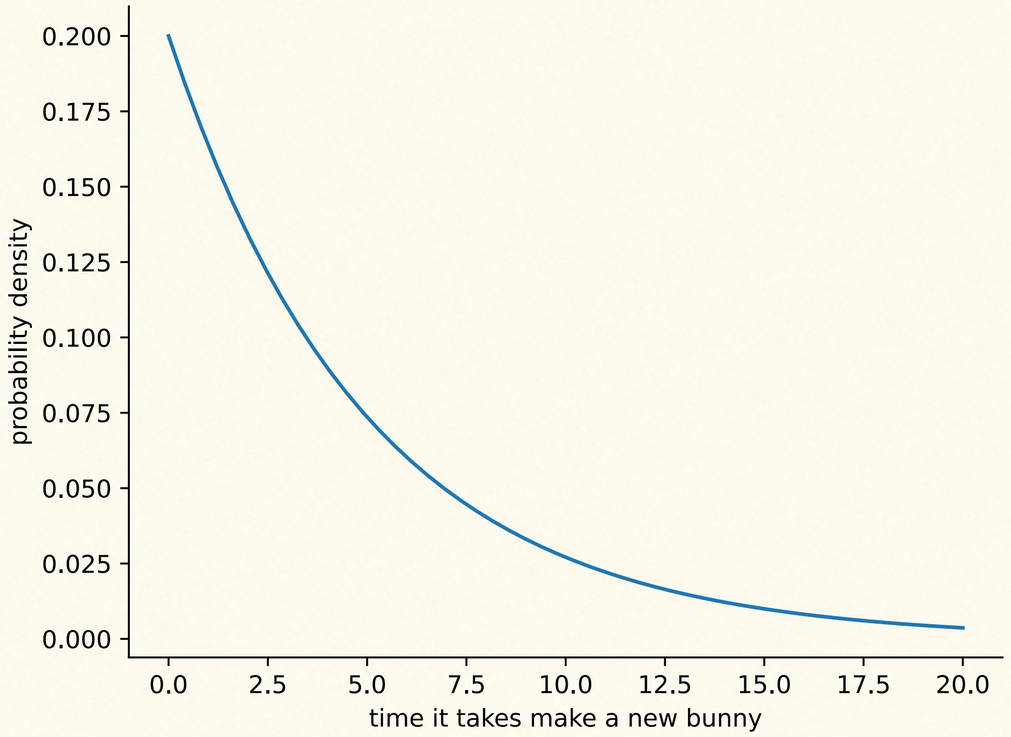
Lotka-Volterra equations are averaged 'mean-field' equations. They are easy to solve and simulate but average over a bunch of detail.

B and F are floats, they really represent sort of an average number. So you can have 36.232 bunnies in expectation.

Reality has fluctuations! We can incorporate those by doing a harder calculation..

Exponential distribution

Model $\rho(t) = \alpha e^{-\alpha t}$



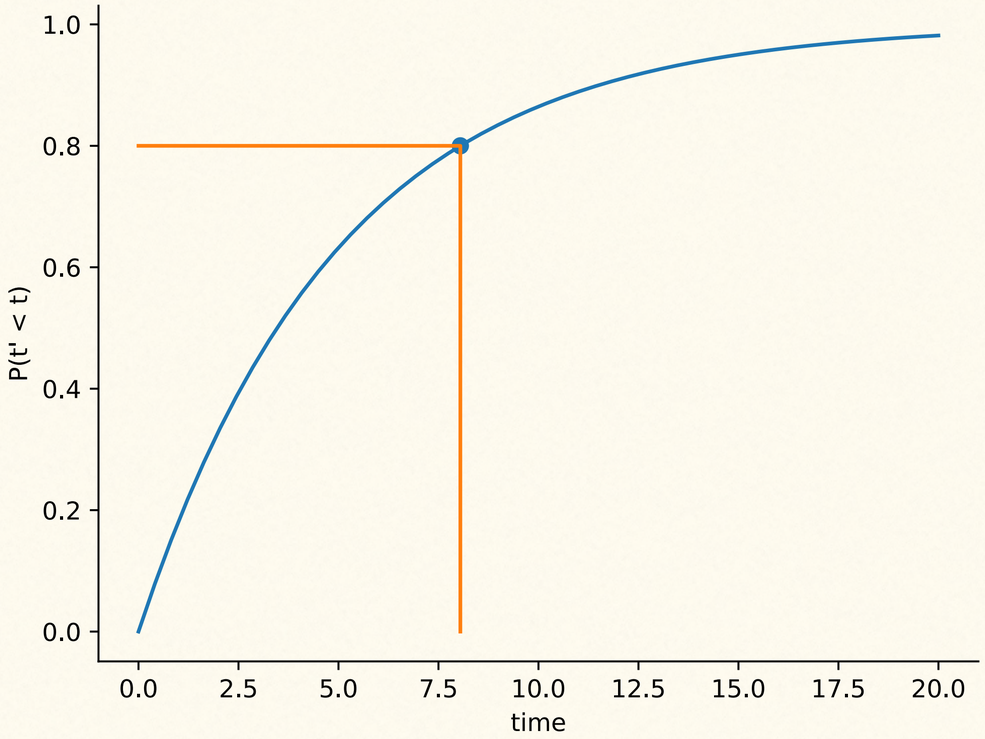
These processes are **memoryless**: if you know nothing happens up to time s, then you 'restart'

Proof:

$$P(T > s + t \,|\, T > s) = \frac{P(T > s + t \text{and} T > s)}{P(T > s)}$$

$$= \frac{e^{-\alpha(s+t)}}{e^{-\alpha s}} = e^{-\alpha t} = P(T > t)$$

Sampling from an exponential distribution



$$\int_0^t \alpha e^{-\alpha t'} dt' = 1 - e^{-\alpha t}$$

Generate random number χ

$$t' = -\ln(1-\chi)/\alpha = -\ln(\chi)/\alpha$$

is sampled from the exponential.

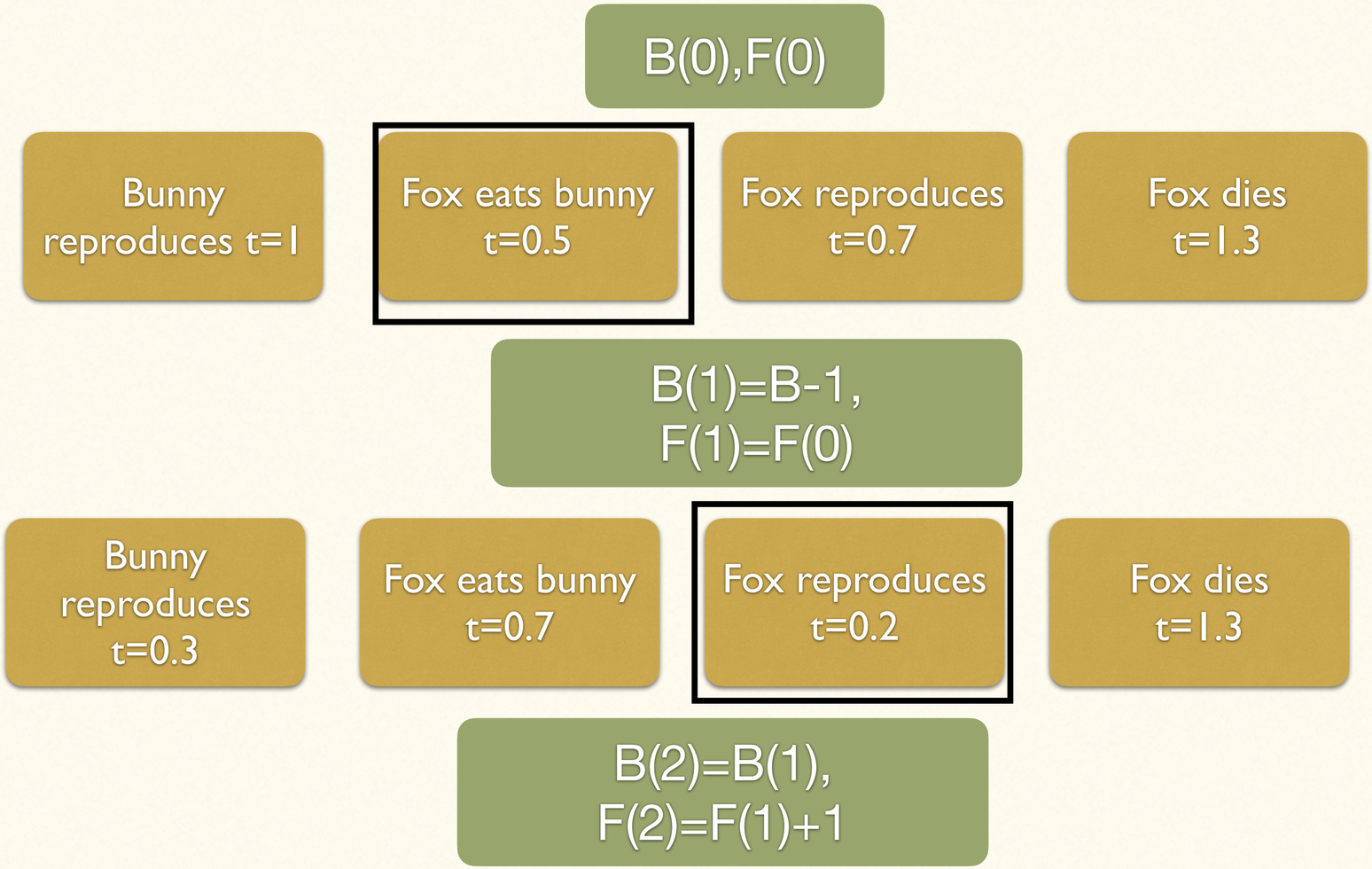
Continuous time Monte Carlo

You have a bunch of Poisson processes with rate r_i.

As long as nothing changes, each would happen at time $t_i = -\ln(\chi_i)/r_i$.

Algorithm:

* the action that would happen first happens first.
* Then the rates change
* Memoryless -> recompute all rates and find the first action again



Hint for doing Monte Carlo

You might be tempted to do something like this:

```
t1 = -np.log(np.random.rand())/(alpha*B)
t2 = -np.log(np.random.rand())/(gamma*F)

if t1 < t2: # and t1 < t3 and t1 < t4
  B+=1
elif t2 < t3: # and t2 < t4
  F-=1
```

But this scales better and is less likely to be buggy.

```
changes = [(1,0), (0,-1)]
rates = np.array([alpha*B, gamma*F])
ts = -np.log(np.random.rand(2))/rates
choose = np.argmin(ts)
B+=changes[choose][0]
F+=changes[choose][1]
time += ts[choose]
```