# Predator-Prey
# Lecture 10

**PHYS 246 class 10**
**Fall 2025**
**J Noronha-Hostler**

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

# Announcements

- I'll need to leave early this week and next due to some family constraints.
- I changed some notation's on today's assignment earlier today, please use the current version!
- 

# Final Guidance

- Teams up to 4 people
- Small coding project
- Presentations ~5 minutes
- Graded on:
  - Presentation skills: Intro, methodology, results, conclusions
  - Working code
  - Shared work responsibilities
- Date/Time: Wednesday 17 December at 1:30 PM
- Location: **32 psychology**

# Fox $F(t)$ vs bunnies $B(t)$

## Dynamics over time

- Prey reproduce: each prey produces $\frac{\Delta B(t)}{B(t)} = \alpha \Delta T$ offspring

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

- Predators die: each prey dies as $\frac{\Delta F(t)}{F(t)} = -\gamma \Delta t$

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

- Predators eat prey:
  - Rate Predators run into prey: $F(t)B(t)$
  - Prey gets eaten: $\Delta B(t) = -\beta F(t)B(t)$
  - Predators eats prey, not hungry, will reproduce: $\Delta F(t) = \delta F(t)B(t)$

# Fox $F(t)$ vs bunnies $B(t)$

## Dynamics over time

- Prey reproduce: each prey produces $\dfrac{\Delta B(t)}{B(t)} = \alpha \Delta T$ offspring $\dfrac{dB(t)}{dt} = \alpha B(t)$
- Predators die: each prey dies as $\dfrac{\Delta F(t)}{F(t)} = -\gamma \Delta t$
  $\dfrac{dF(t)}{dt} = -\gamma F(t)$
- Predators eat prey:
  - Rate Predators run into prey: $F(t)B(t)$
  - Prey gets eaten: $\dfrac{dB(t)}{dt} = -\beta F(t)B(t)$
  - Predators eats prey, not hungry, will reproduce: $\dfrac{dF(t)}{dt} = \delta F(t)B(t)$

# Combine all equations

**Entire ecosystem**

- Change of prey: $\dfrac{dB(t)}{dt} = \alpha B(t) - \beta F(t)B(t)$
- Change of predators: $\dfrac{dF(t)}{dt} = \delta F(t)B(t) - \gamma F(t)$
- Requires initial conditions: $F_0 = F(0), \quad B_0 = B(0)$
- Defined by set or parameters: $\left\{\alpha, \beta, \delta, \gamma\right\}$
- Called Lotka-Volterra equations

# Stationary points

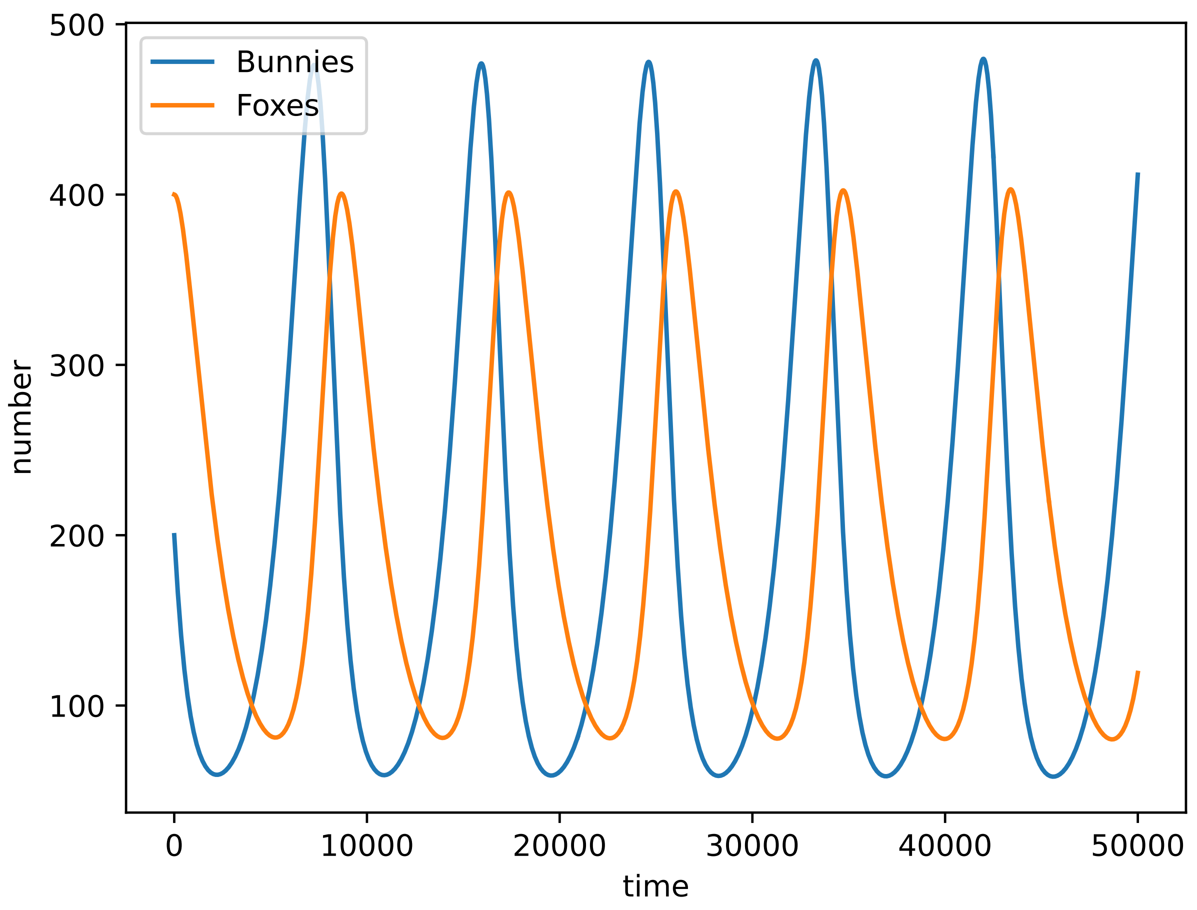
## Places when total numbers don't change?

Original equation

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

Euler method

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




Plug in original equation

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

# Stochastic Method

Fractions of bunnies!

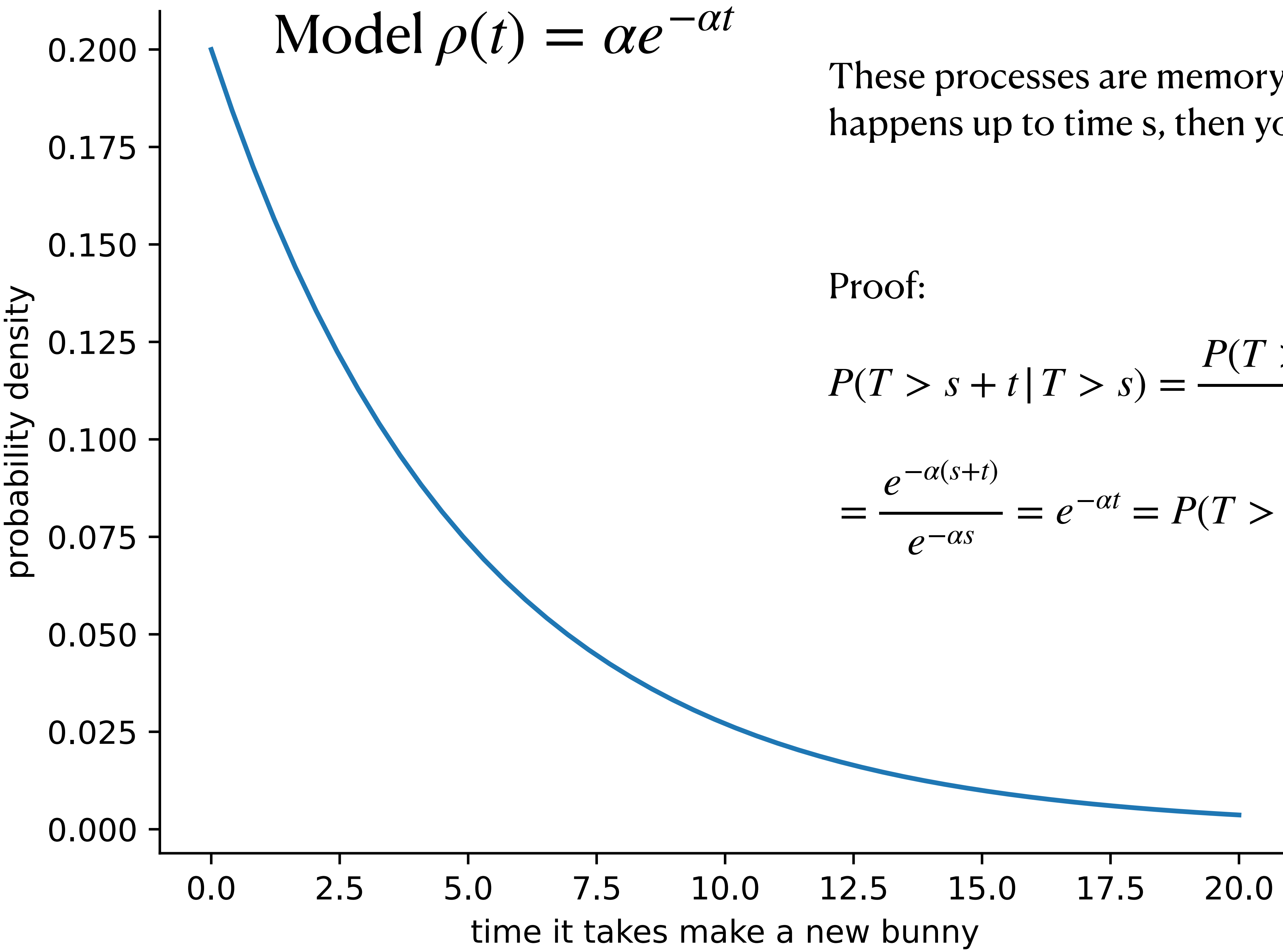
Lotka-Volterra equations are averaged 'mean-field' equations. They are easy to solve and simulate but average over a bunch of detail.

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

## Average time behavior (Poisson Process)



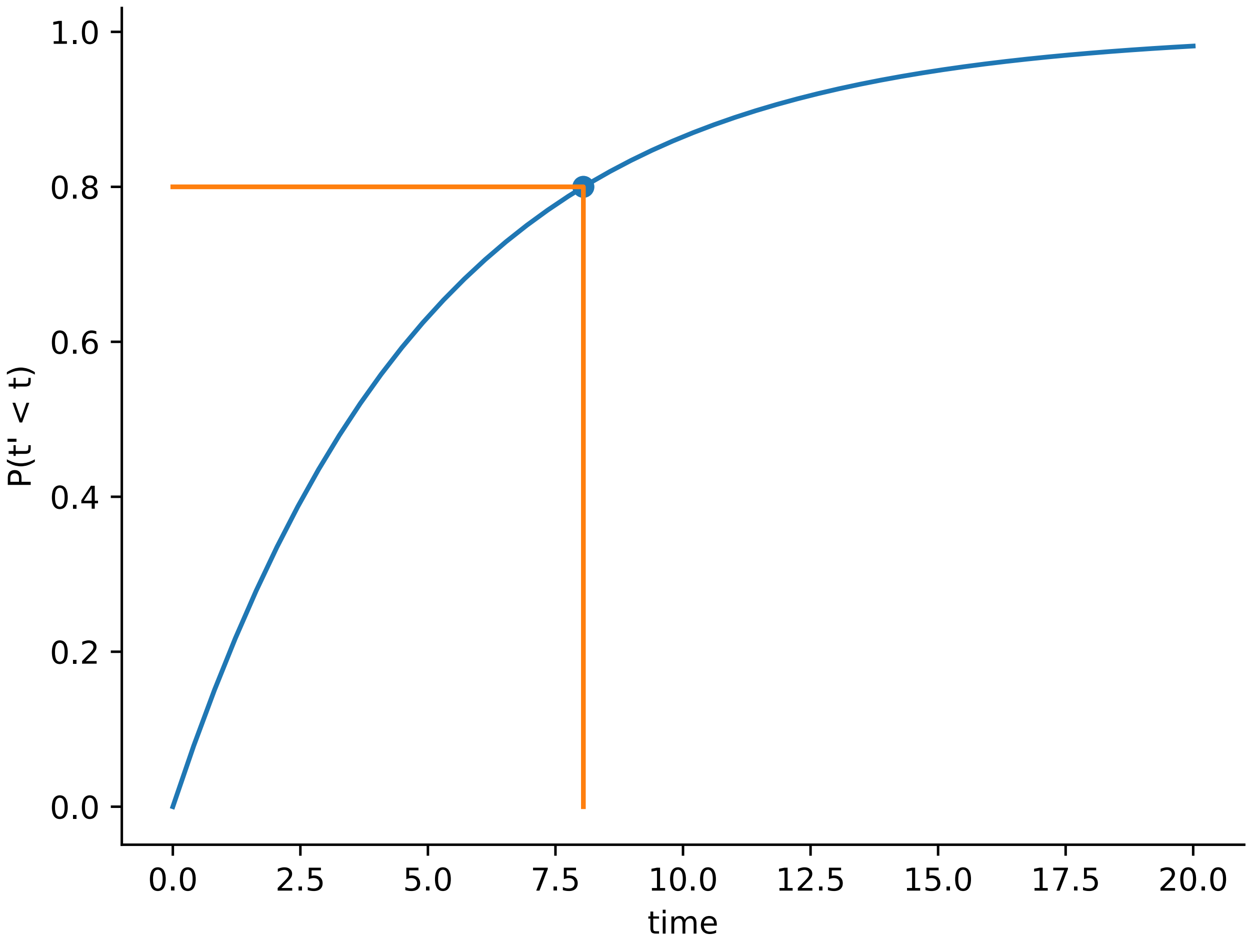

These processes are memoryless: if you know nothing happens up to time s, then you 'restart'

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




Find $\Delta t$,
time to make first bunny!

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

Generate random number $\chi$

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

is sampled from the exponential.

# Continuous time Monte Carlos

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

# Continuous time Monte Carlo

B(0),F(0)

Bunny reproduces

Fox eats bunny t=0.5

Fox reproduces

Fox dies t=1.3

B(1)=B-1,

Bunny reproduces

Fox eats bunny t=0.7

Fox reproduces

Fox dies t=1.3

F(2)=F(0)+1,

# Hints for doing a Monte Carlo

```
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
```

Instead of this

Choses which event happens first

```
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]
```

Try this