Random Walk
Lecture 8

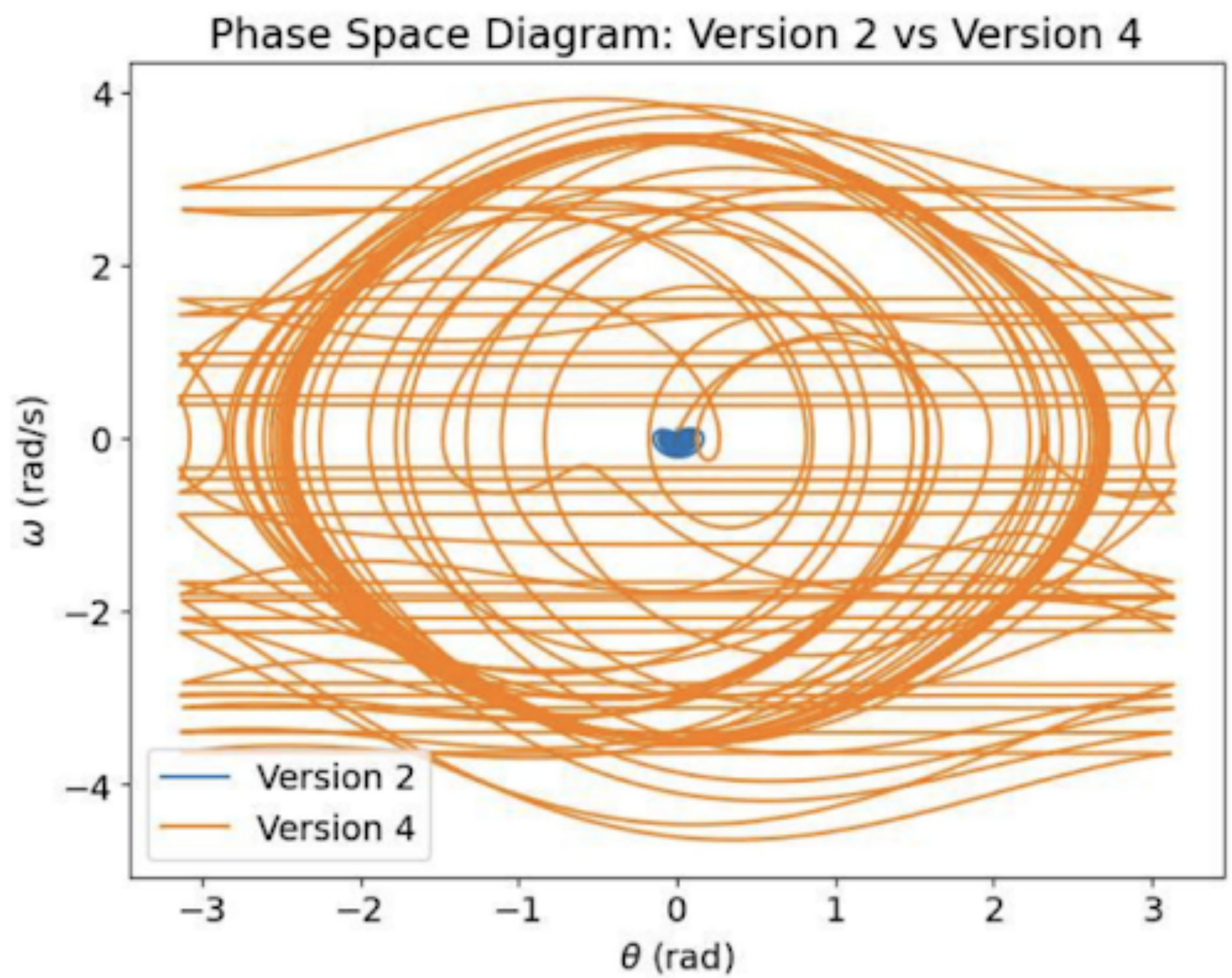
•

PHYS 246 class 8
Fall 2025
J Noronha-Hostler
https://jnoronhahostler.github.io/IntroductionToComputationalPhysics/intro.html

Announcements/notes

- Final project discussion (see email soon!)
- No offices from JNH next week but Surkhab and Maxwell still have them!
- Students in class today received extra credit!
- Plots

Can't see version 2

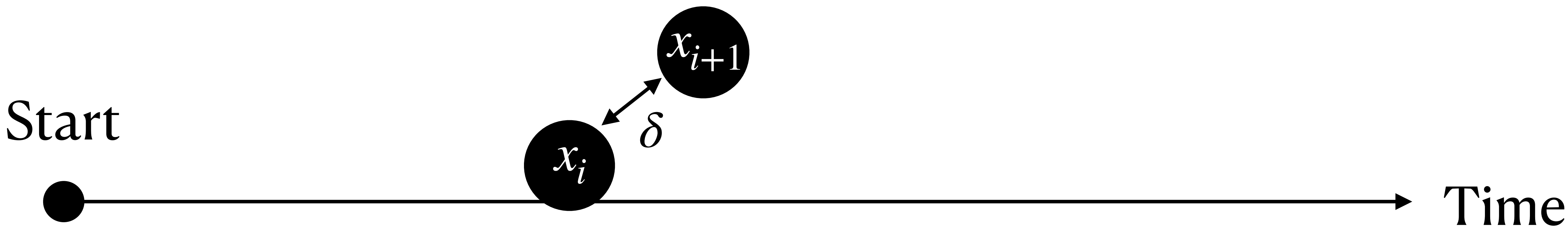


Stochastic processes

"pattern that may be analyzed statistically but may not be predicted precisely"

- Where students sit on class
- How dust or pollen moves through air or a liquid
- Leaves falling from trees
- Financial markets
- Monte Carlo methods (statistical physics, quantum mechanics, particle physics, ...)
- Training machine learning models

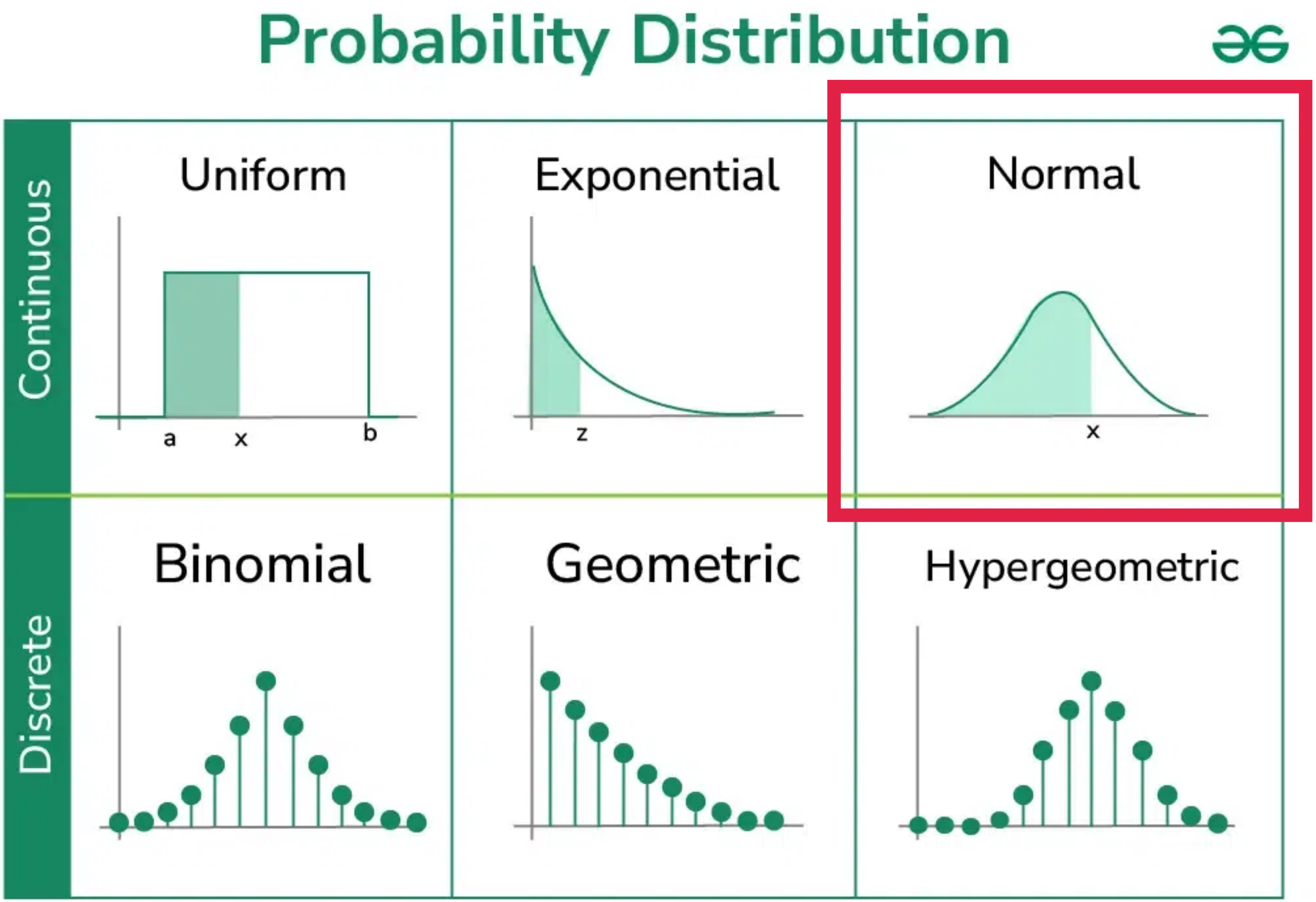
Taking a random walk



- Each step in the walk $x_i = x_{i+1} + \delta$ where δ is a random step
- δ depends on the problem:
  - Coin flipping $\delta = \pm 1$
  - Walk $\delta = [x_{min}, x_{max}]$ (depends on person's stride)
- Easy way to do this: $\delta = \sigma N[0,1]$ Sample from 0 to 1, but renormalize by σ as needed

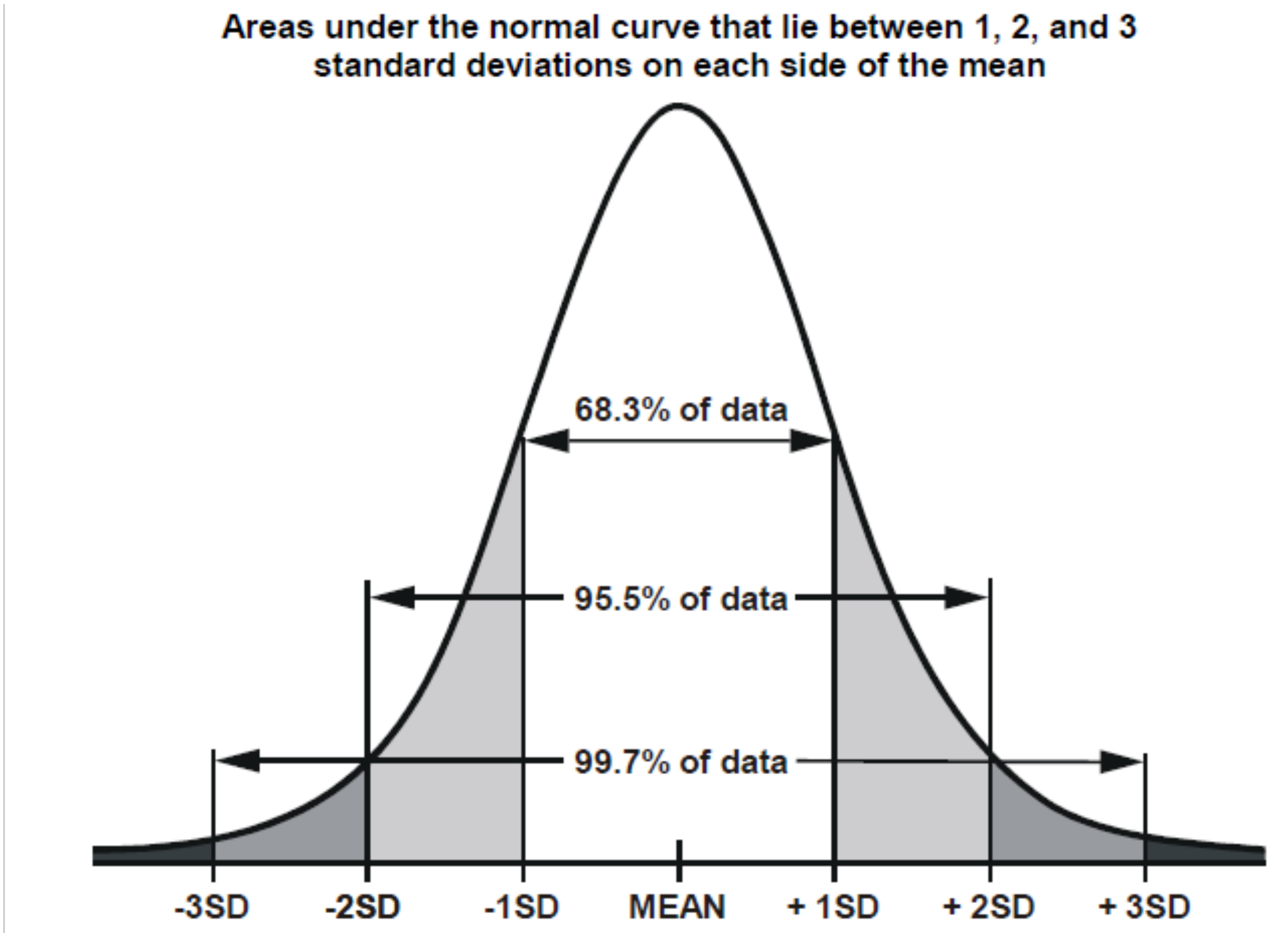
How do we sample randomly?

Types of probability distributions



Normal distribution or "Bell curve"

Mean vs standard deviation



Then

Mean, variance etc

Moments of a distribution: $\langle x^n \rangle = \frac{1}{N_{samp}} \sum_{i}^{N_{samp}} x_i^n$

Useful moments for this class today

Mean $n = 1$, so $\langle x \rangle = \frac{1}{N_{samp}} \sum_{i}^{N_{samp}} x_i$

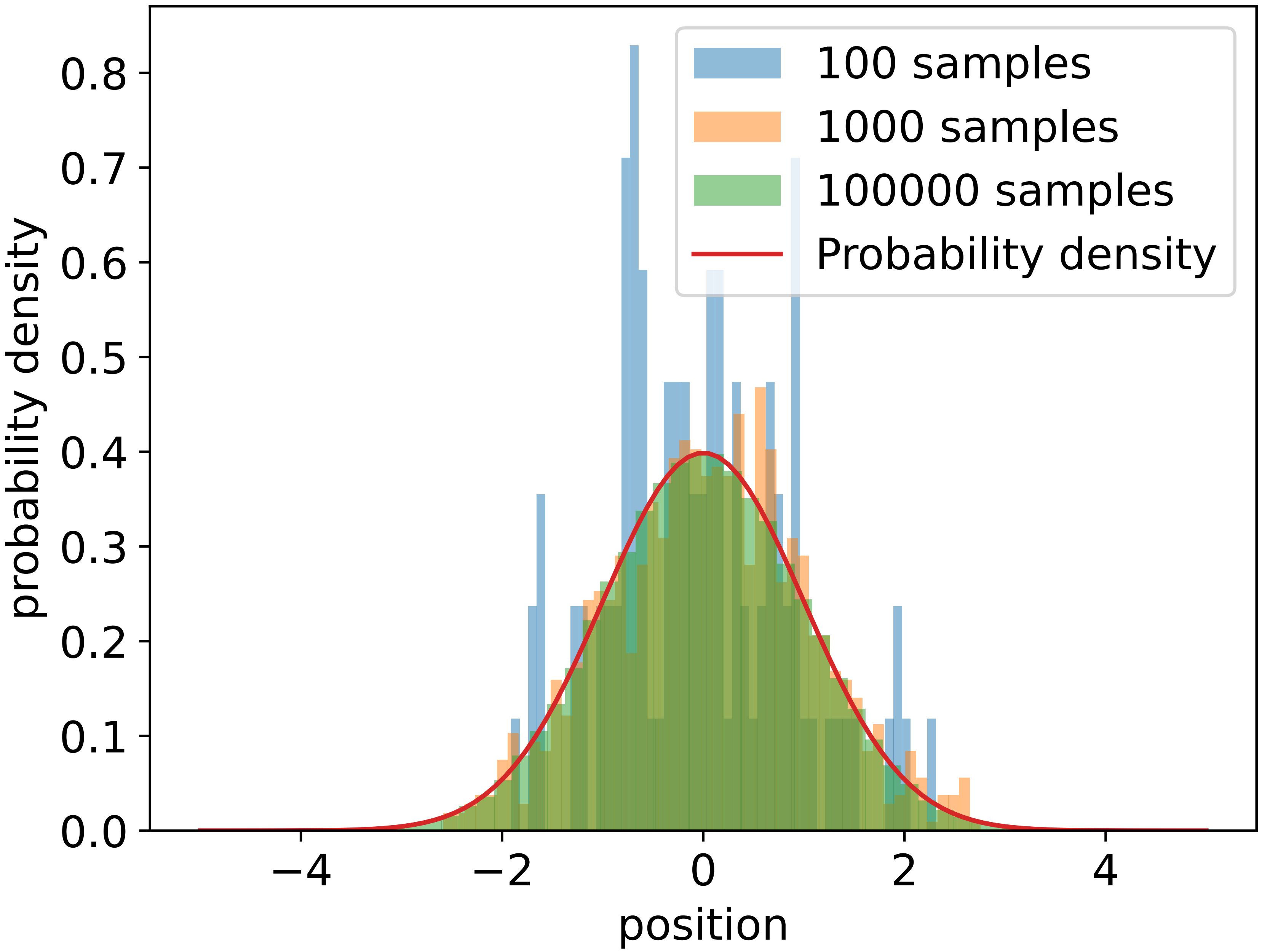
Then, for $n = 2$, $\langle x^2 \rangle = \frac{1}{N_{samp}} \sum_{i}^{N_{samp}} x_i^2$

Variance: $var = \langle x^2 \rangle - \langle x \rangle^2$

Standard deviation: $\sigma = \sqrt{\langle x^2 \rangle - \langle x \rangle^2}$

Histograms and probability distributions

Probability distribution vs samples



Probability density:

$\rho(x) \geq 0$ and $\int \rho(x)dx = 1$

$\int_a^b \rho(x)dx$ proportion of samples (on average) in the range [a,b]

A histogram plots the proportion of samples in the range [a,b]

Histograms and probability distributions

```
for nsample in [100, 1000, 100000]:
    samples = np.random.randn(nsample)
    plt.hist(samples, bins=50, density=True, alpha = 0.5, label=f'{nsample} samples')
x = np.linspace(-5, 5, 100)
y = np.exp(-x**2/2)/np.sqrt(2*np.pi)
plt.plot(x, y, label="Probability density")
plt.xlabel("position")
plt.ylabel("probability density")
plt.legend()

plt.savefig("hist_vs_density.pdf", bbox_inches='tight', transparent = True)
```

np.random.randn generates samples from a "normal" probability distribution centered at 0 (mean=0), within 1 standard deviation

Numerical Integration

Riemann sums

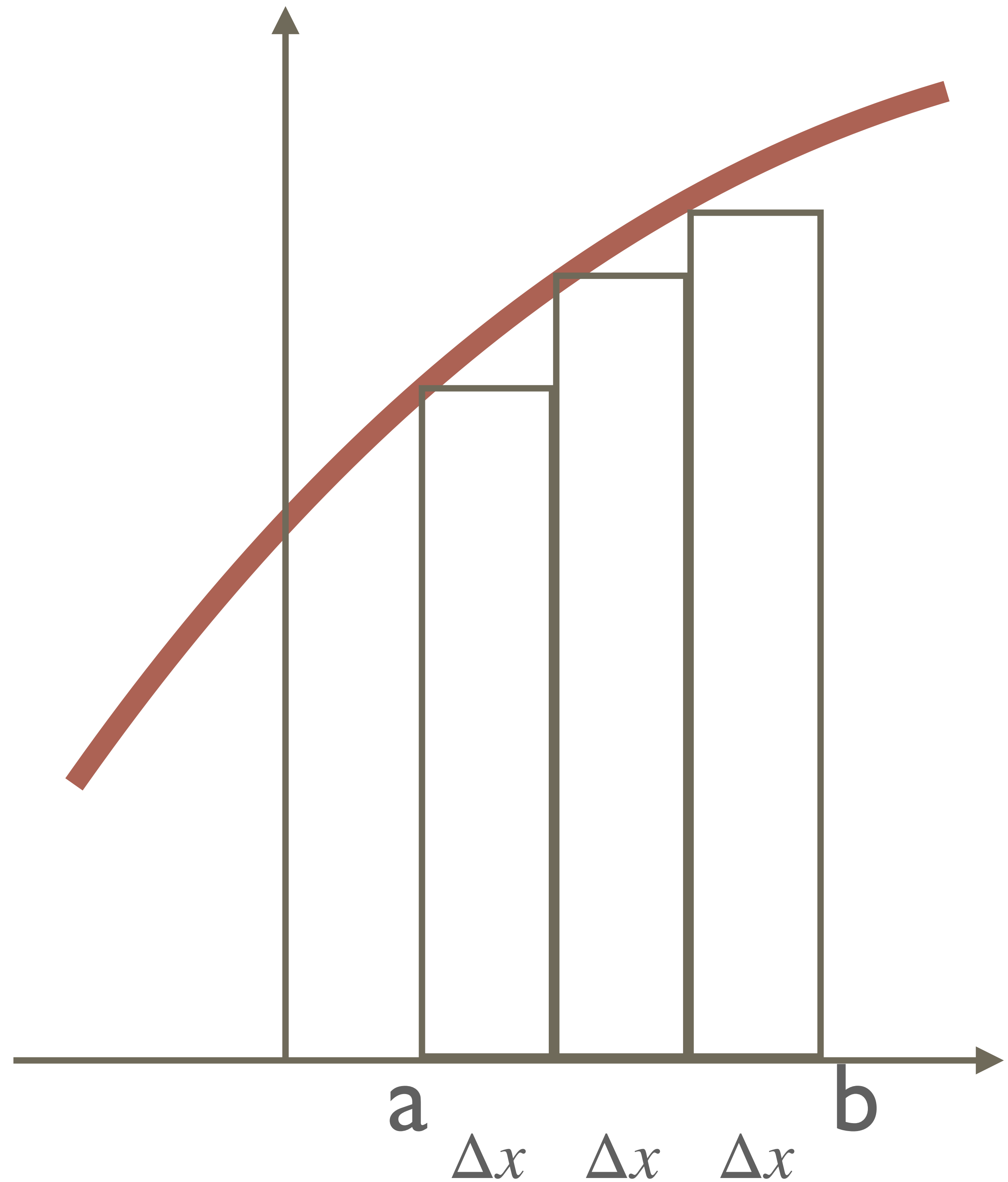
Riemann sums

$$\int_a^b \rho(x)dx \simeq = \sum_i \rho(x_i)\Delta x$$

Sample variance should match the histogram variance

$$\frac{1}{N}\sum x_i^2 \simeq \int x^2 \rho(x)dx$$

You should be able to confirm this!



Diffusion Equation

Tells us how the probability density changes in time.

$$\frac{\partial \rho}{\partial t} = D\frac{\partial^2 \rho}{\partial x^2}$$

You can solve this analytically but it's pretty easy to do numerically too.

$$\rho(x, t + \Delta t) = \rho(x) + \Delta t\frac{\partial \rho(x, t)}{\partial t} = \rho(x) + \Delta t D\frac{\partial^2 \rho(x, t)}{\partial x^2}$$

Very similar to Euler integration but you propagate an entire function instead of just a position.

The density computed this way should match the histograms of the simulations you did!

Forward differences

Second-order numerical derivatives

First-order derivatives: $f'(x_i) = \dfrac{f(x_{i+1}) - f(x_i)}{\Delta x}$

Second-order derivatives: $f''(x_i) = \dfrac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{(\Delta x)^2}$

Can use np.roll(list, shift) to help $\quad list \equiv \{-3,4,12,16,3,0,7\}$

Then, np.roll(list,-2) gives $\{12,16,3,0,7,-3,4,\}$

np.roll(list,1) gives $\{7,-3,4,12,16,3,0\}$

They shift your list (wraps extra numbers)

We don't want periodic boundary conditions, so set wrapped numbers to 0!

Hints

For the large sample sizes (100,000, 10,000), make sure to test for small numbers of walkers first.

np.roll will move the probability density +/- one grid point -> makes it easy to take second derivative.

Inverting the loop

```
for i in range(t):
    walkers += 6*np.random.randn(N)
```

memory: nwalkers

Independent samples, can all be generated at the same time!

Generating walks per sample

```
walks = np.array([randomWalk(1000, 6) for i in range(10)])
```

memory: 1001*nwalkers