Symbulate: Probability Simulation in Python

Kevin Ross
Cal Poly

Joint work with Dennis Sun (Cal Poly and Google)

Research supported by the Bill and Linda Frost Fund

May 30, 2019

CAL POLY

1. Motivation
2. Symbulate Gallery
3. Symbulate Mechanics
4. Symbulate in the Classroom

CAL POLY

1. Motivation
2. Symbulate Gallery
3. Symbulate Mechanics
4. Symbulate in the Classroom

CAL POLY

Pedagogical Goals for Probability and Simulation

- Conceptual understanding
- Multivariable thinking
- Problem solving
- Active learning

Pedagogical Goals for Probability and Simulation

- Conceptual understanding
- Multivariable thinking
- Problem solving
- Active learning

Proposal:

- Simulate **everything**
  - Tactile, in class
  - Technology, with Symbulate
- Use lots of visuals

CAL POLY

A Simple Dice Rolling Example

Suppose we roll two fair, six-sided dice. Let X be the sum and Y the maximum of the rolls. What is $\mathrm{Cov}(X, Y)$?

What's wrong with the following `R` code to approximate the answer?

```
x <- replicate(10000, sum(sample(1:6, size=2, replace=T)))
y <- replicate(10000, max(sample(1:6, size=2, replace=T)))
cov(x, y)
```

A Simple Dice Rolling Example

Suppose we roll two fair, six-sided dice. Let X be the sum and Y the maximum of the rolls. What is $\mathrm{Cov}(X, Y)$?

What's wrong with the following `R` code to approximate the answer?

```
x <- replicate(10000, sum(sample(1:6, size=2, replace=T)))
y <- replicate(10000, max(sample(1:6, size=2, replace=T)))
cov(x, y)
```

Correct `R` code:

```
xy <- replicate(10000, sample(1:6, size=2, replace=T))
x <- apply(xy, 2, sum)
y <- apply(xy, 2, max)
cov(x, y)
```

Shortcomings of `R` (and other simulation languages)

- `R` does not prevent you from writing code that makes no statistical sense
- The (correct) code does not resemble the language of probability, e.g., `apply(xy, 2, max)`
- Programming a simulation involves several levels of code:
  - Defining the probability simulation
  - Summarizing simulation output
  - Constructing visualizations

Shortcomings of `R` (and other simulation languages)

- `R` does not prevent you from writing code that makes no statistical sense
- The (correct) code does not resemble the language of probability, e.g., `apply(xy, 2, max)`
- Programming a simulation involves several levels of code:
  - Defining the probability simulation
  - Summarizing simulation output
  - Constructing visualizations

As a result:

- Students must learn two languages: the language of probability and the language of coding simulations

CAL POLY

Symbulate

Symbulate is a Python library for specifying simulations from probability models.

```
P = BoxModel([1, 2, 3, 4, 5, 6], size=2, replace=True)
X = RV(P, sum)
Y = RV(P, max)
(X & Y).sim(10000).cov()
```

Symbulate

Symbulate is a Python library for specifying simulations from probability models.

```
P = BoxModel([1, 2, 3, 4, 5, 6], size=2, replace=True)
X = RV(P, sum)
Y = RV(P, max)
(X & Y).sim(10000).cov()
```

Define a probability space.

Symbulate

Symbulate is a Python library for specifying simulations from probability models.

```
P = BoxModel([1, 2, 3, 4, 5, 6], size=2, replace=True)
X = RV(P, sum)
Y = RV(P, max)
(X & Y).sim(10000).cov()
```

Define a probability space.

A random variable is a function defined on a probability space.

Symbulate

Symbulate is a Python library for specifying simulations from probability models.

```
P = BoxModel([1, 2, 3, 4, 5, 6], size=2, replace=True)
X = RV(P, sum)
Y = RV(P, max)
(X & Y).sim(10000).cov()
```

Define a probability space.

A random variable is a function defined on a probability space.

Simulate 10000 draws from the joint distribution of X and Y.

Symbulate Prevents Common Code Mistakes

```
X = RV(BoxModel([1, 2, 3, 4, 5, 6], size=2, replace=True), sum)
Y = RV(BoxModel([1, 2, 3, 4, 5, 6], size=2, replace=True), max)
(X & Y).sim(10000).cov()
```

Symbulate Prevents Common Code Mistakes

```
X = RV(BoxModel([1, 2, 3, 4, 5, 6], size=2, replace=True), sum)
Y = RV(BoxModel([1, 2, 3, 4, 5, 6], size=2, replace=True), max)
(X & Y).sim(10000).cov()
```

```
Exception:  Random variables must be defined on the
same probability space.
```

Language of Probability and Simulation

- Components of a probability model
  - **Probability space**
  - Related **events**, and in particular, **conditioning** on events
  - **Random variables** or **stochastic processes** defined on the probability space, possibly via *transformations*

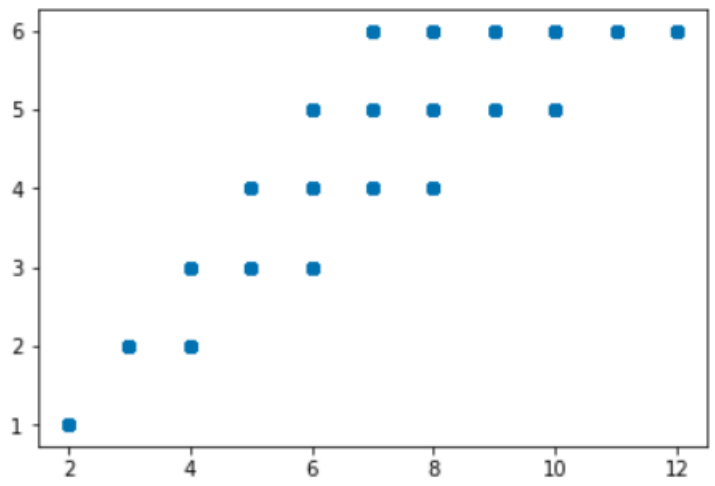
Language of Probability and Simulation

- Components of a probability model
  - **Probability space**
  - Related **events**, and in particular, **conditioning** on events
  - **Random variables** or **stochastic processes** defined on the probability space, possibly via *transformations*
- Steps in a simulation
  - Define a probability model
  - Simulate realizations of objects, possibly with conditioning
  - Summarize and visualize simulation output

Some Features of Symbulate

- Syntax is consistent with the language of probability
- Probability spaces, random variables, etc., are abstract objects that can be manipulated
- Realizations generated using `.sim()`
- Can simulate from conditional distributions using `|`
- Universal function `.plot()` automatically determines an appropriate plot

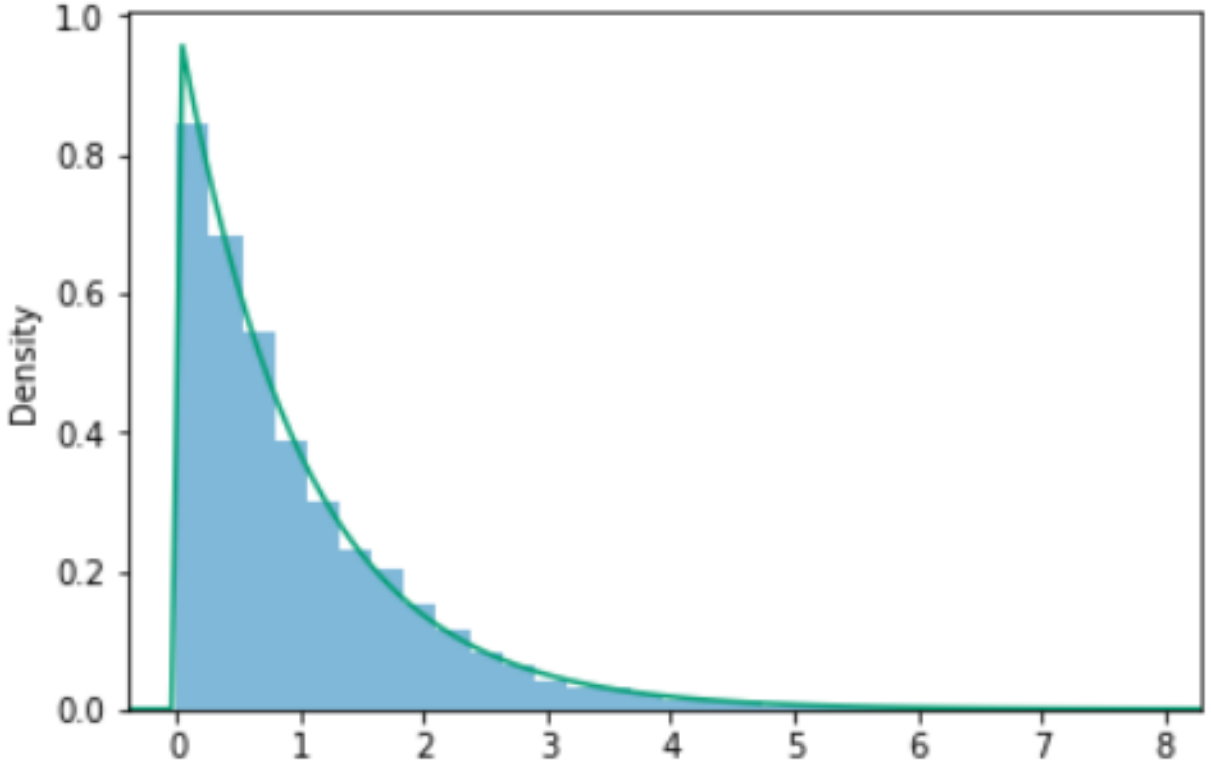
```
P = BoxModel([1, 2, 3, 4, 5, 6], size=2, replace=True)
X = RV(P, sum)
Y = RV(P, max)
(X & Y).sim(10000).plot()
```



CAL POLY

Transforming a Random Variable

```
U = RV(Uniform(0, 1))
X = -log(1 - U)
X.sim(10000).plot()

Exponential(1).plot()
```



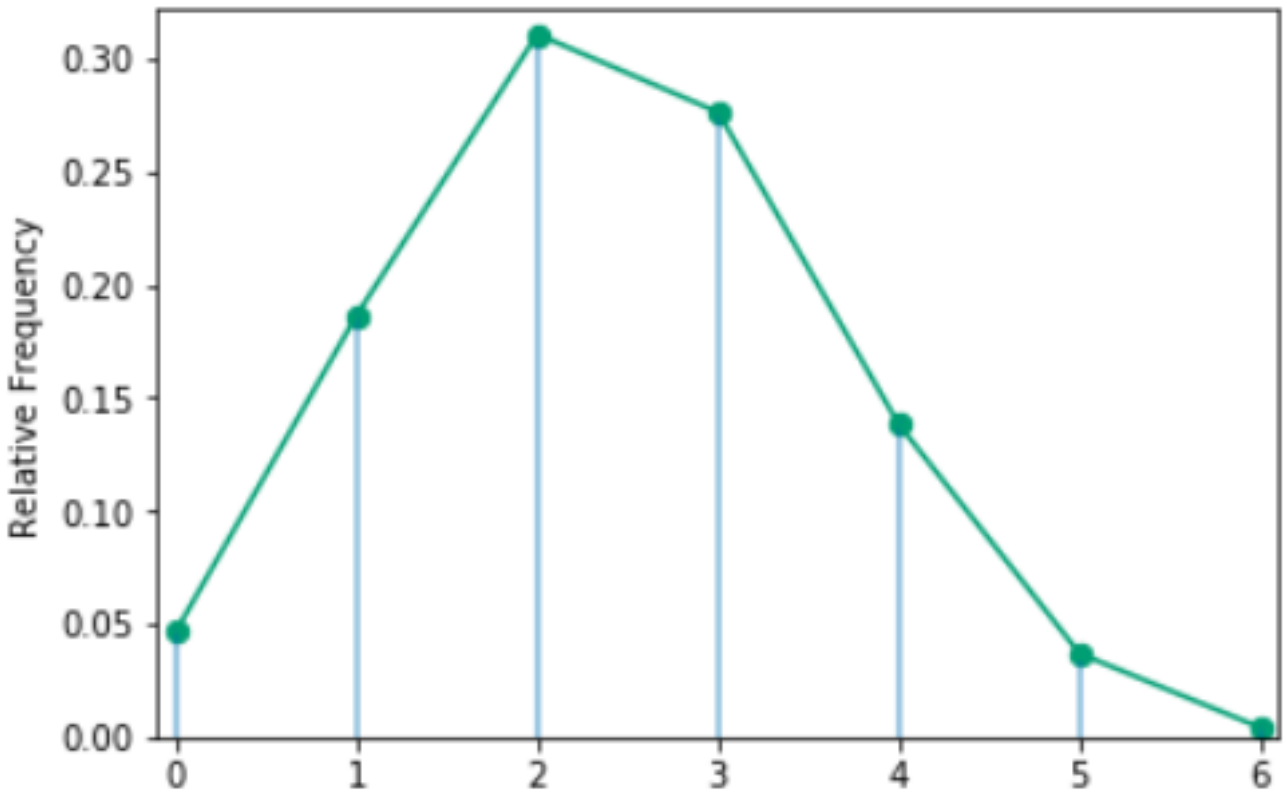
Simulating from a Conditional Distribution

```
X, Y = RV(Poisson(2) * Poisson(3))
(X | ((X + Y) == 6) ).sim(10000).tabulate()
```

Outcome	Value
0	475
1	1852
2	3108
3	2770
4	1378
5	377
6	40
Total	**10000**

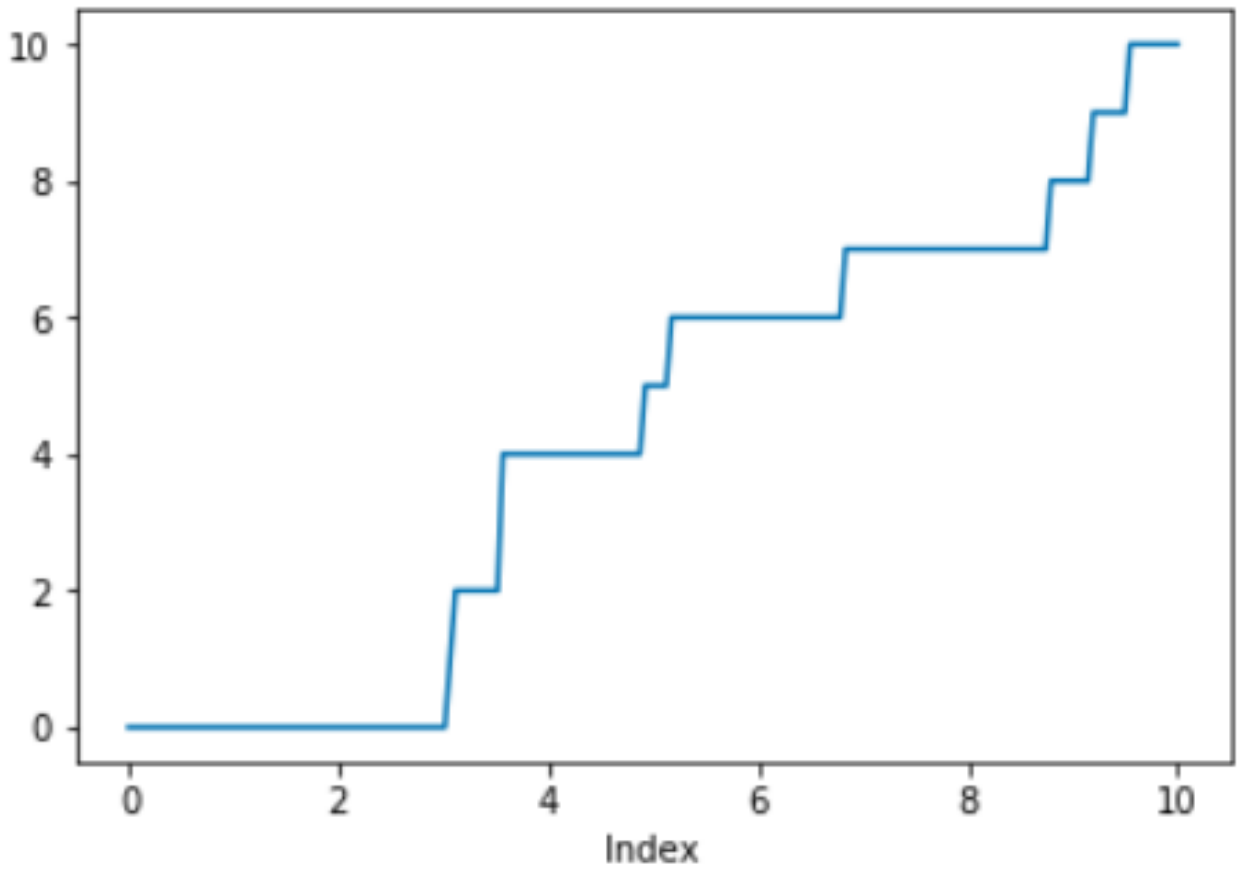
Simulating from a Conditional Distribution

```
X, Y = RV(Poisson(2) * Poisson(3))
(X | ((X + Y) == 6) ).sim(10000).plot()

Binomial(6, 2/5).plot()
```



Sample Path of a Poisson Process

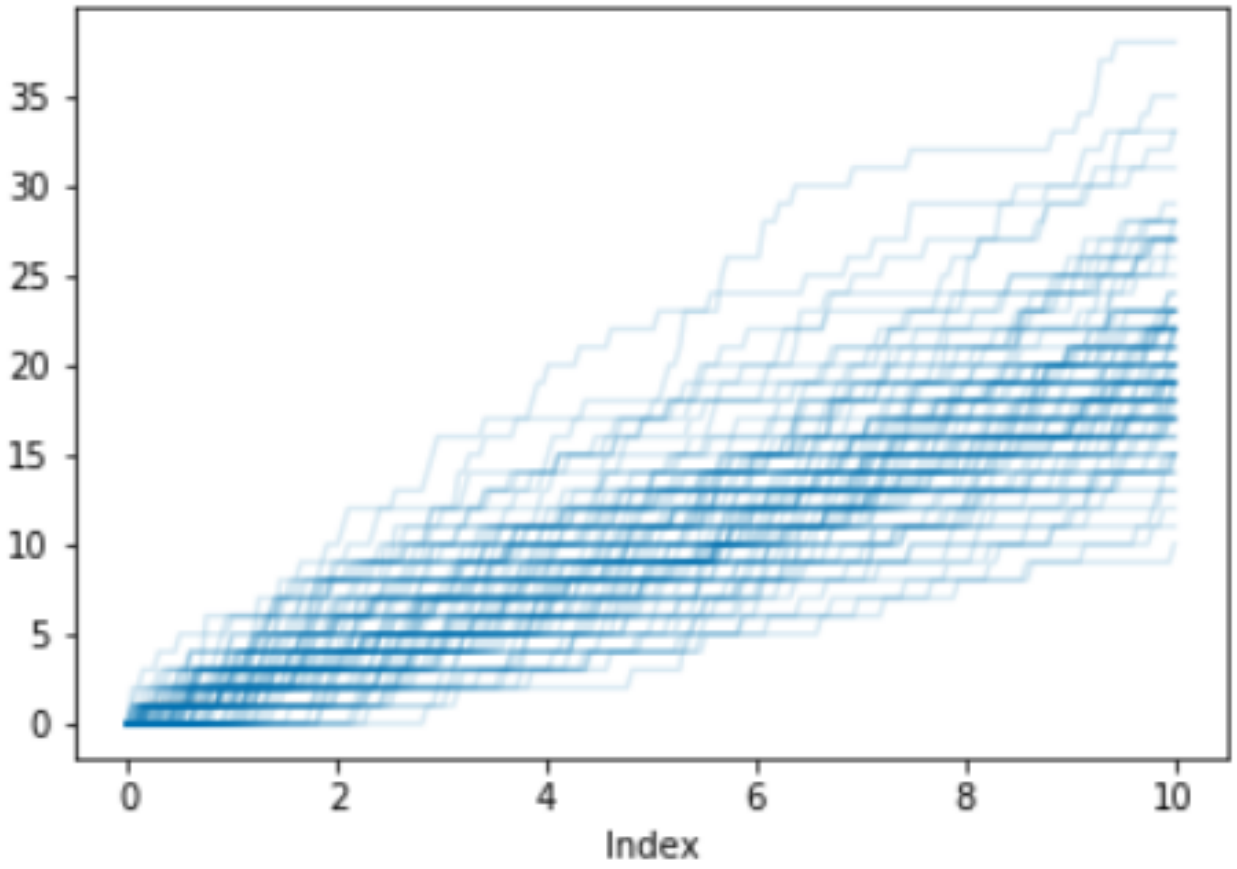
```
N = PoissonProcess(rate=2)
N.sim(1).plot()
```



CAL POLY

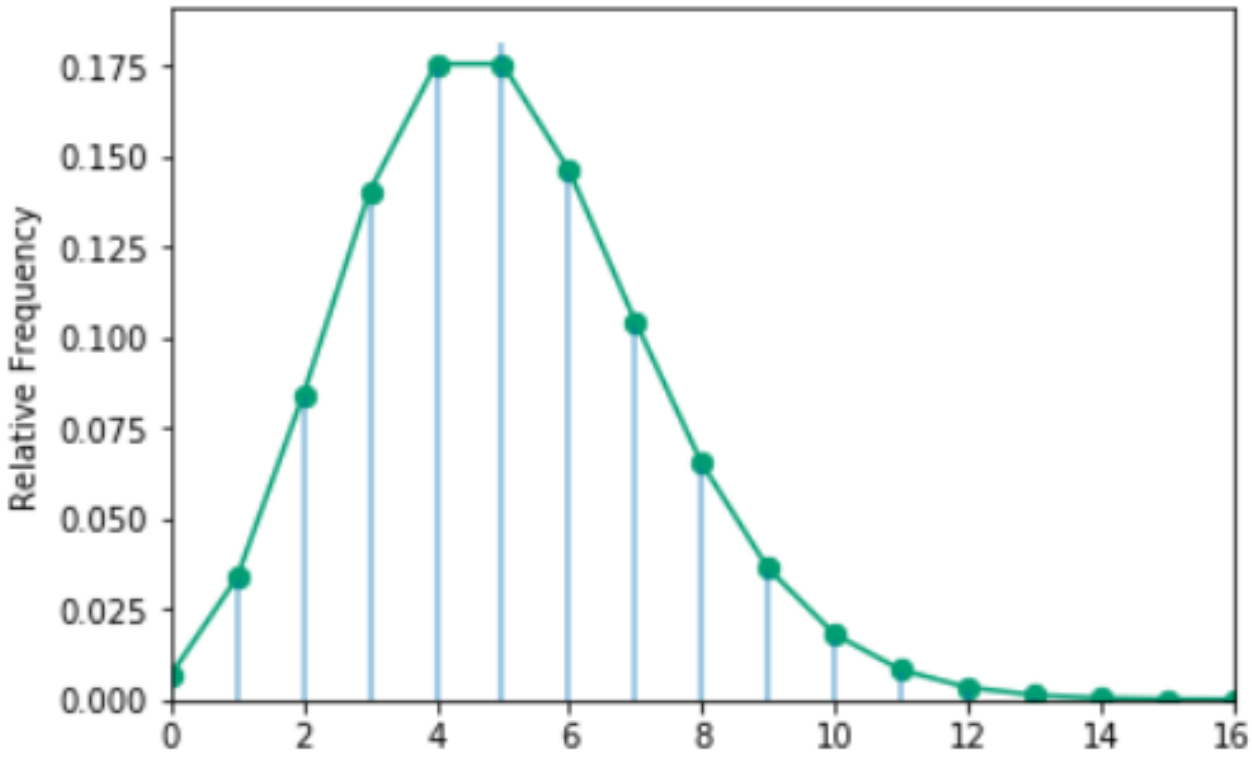
Many Sample Paths of a Poisson Process

```
N = PoissonProcess(rate=2)
N.sim(100).plot()
```



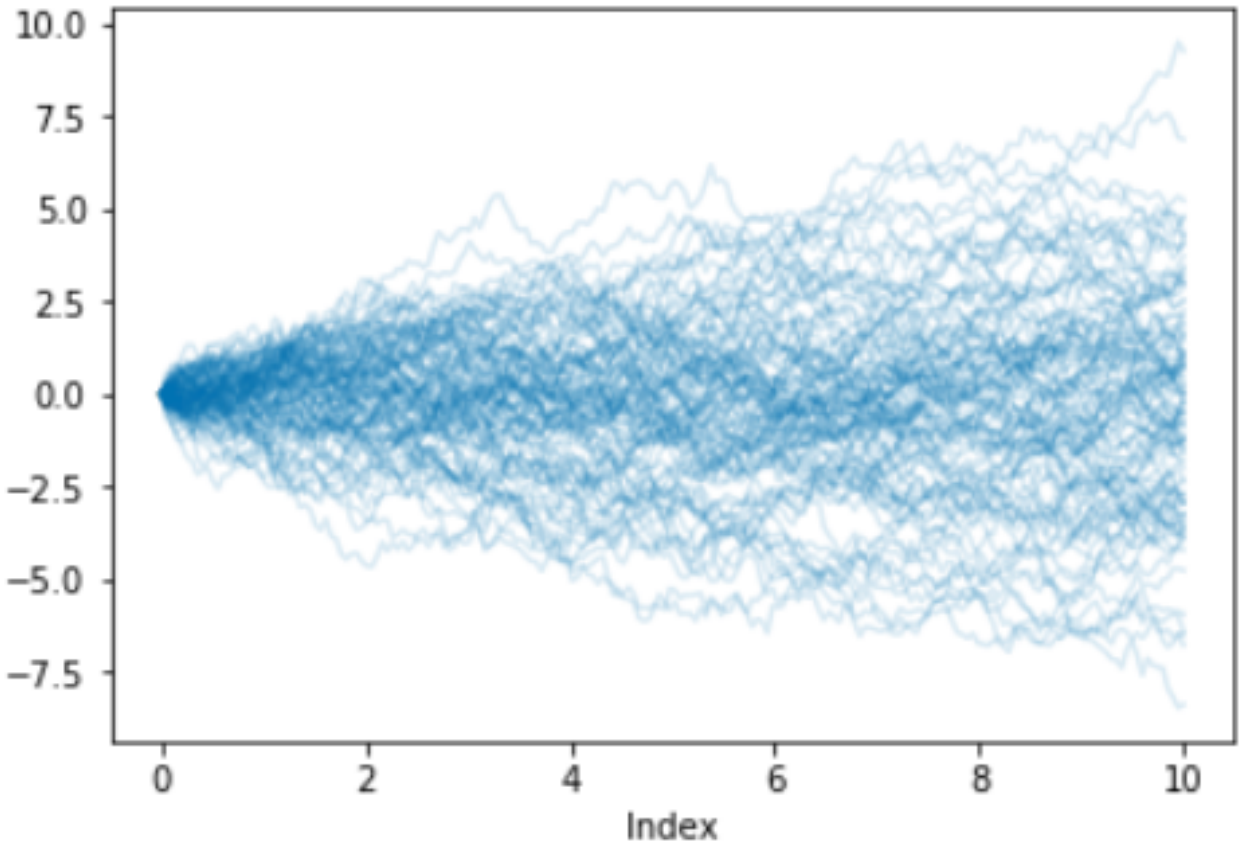
Poisson Process: Marginal Distribution

```
N = PoissonProcess(rate=2)
N[2.5].sim(10000).plot()

Poisson(5).plot()
```



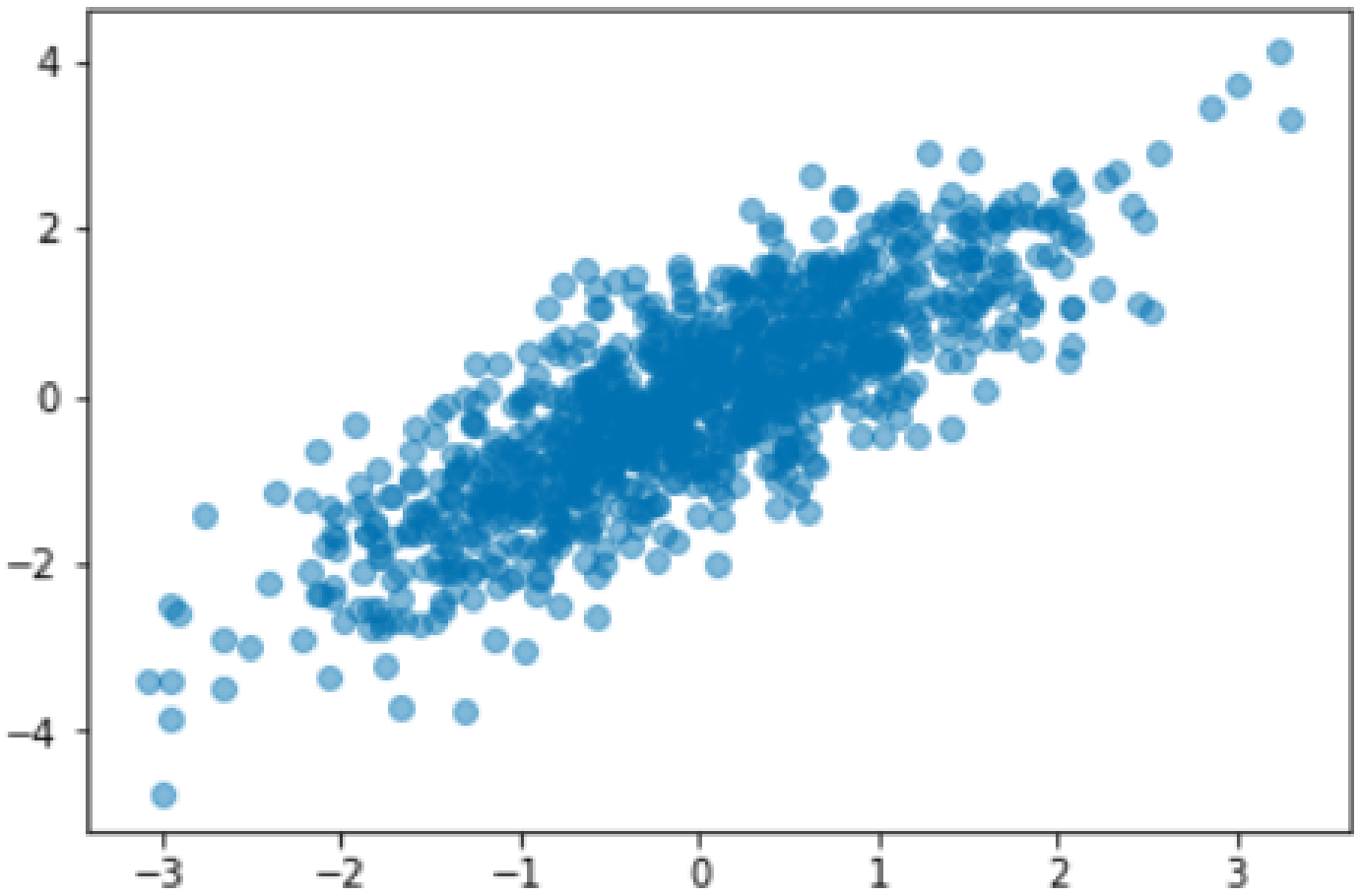
Brownian Motion Sample Paths

```
W = BrownianMotion(drift=0, scale=1)
W.sim(100).plot()
```



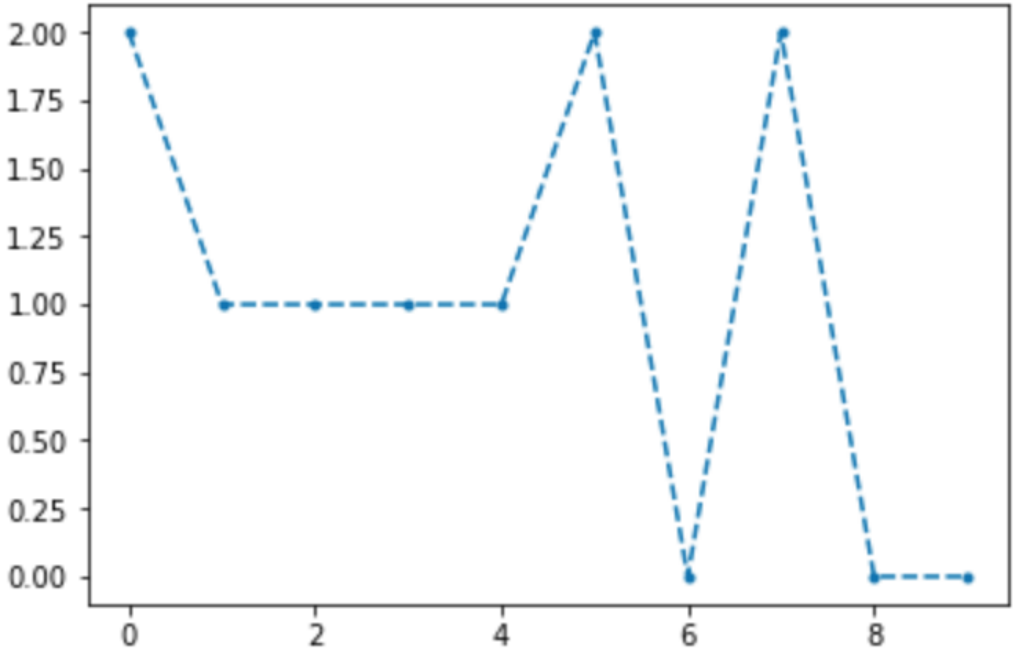
Brownian Motion: Joint Distribution

```
(W[1] & W[1.5]).sim(1000).plot()
```



Discrete Time Markov Chains

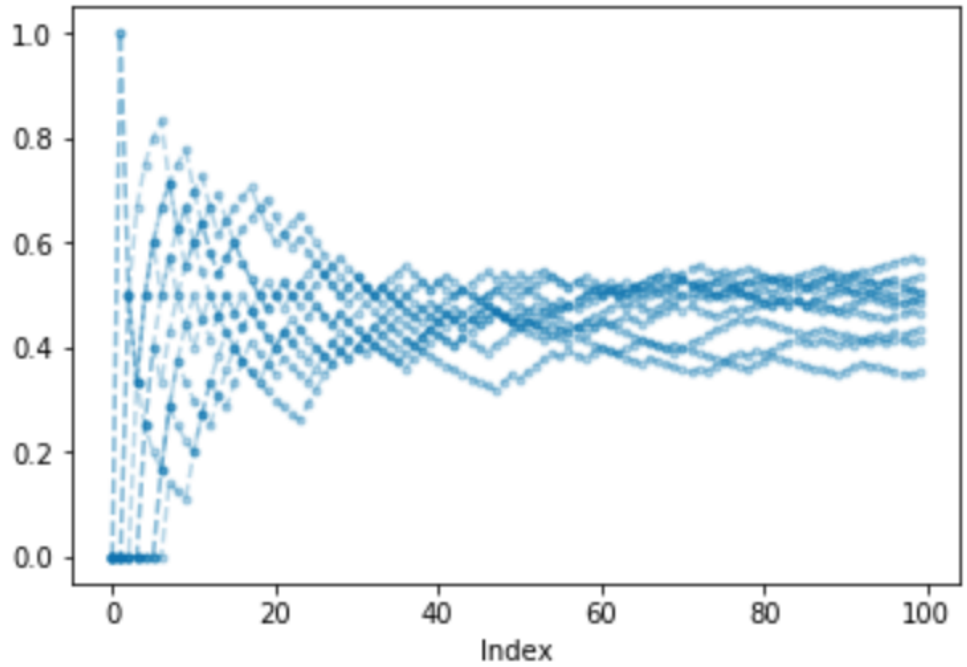
```
T = [[0.2, 0.4, 0.4],
     [0.3, 0.5, 0.2],
     [0.6, 0.4,   0]]
pi0 = [1/3, 1/3, 1/3]

P = MarkovChainProbabilitySpace(transition_matrix=T,
                                initial_dist=pi0)
X = RV(P)
X.draw().plot()
```



CAL POLY

Markov Chains: Cumulative Fraction of Time Spent in State

```
Y = RandomProcess(P)
Y[0] = 0
for i in range(1, 100):
    Y[i] = X[0:i].apply(count_eq(1)) / i

Y.sim(10).plot(tmax=100)
```

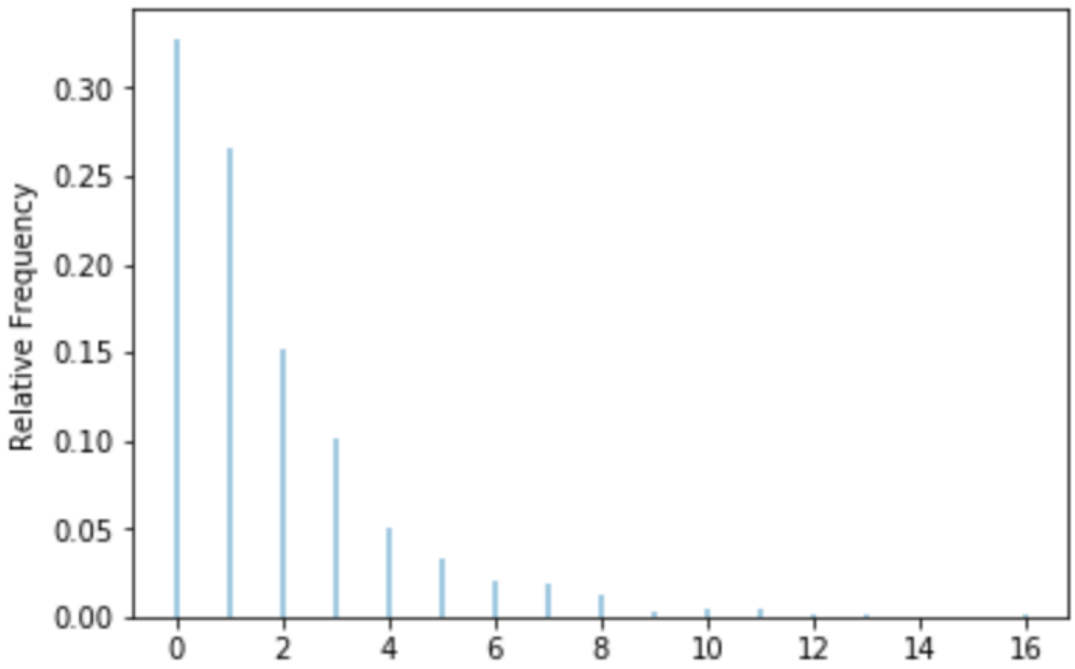


CAL POLY

Markov Chains: Stopping Time

Stopping Time (e.g., time to hit state 1)

```
def first_time_in_state_1(chain):
    for t, state in enumerate(chain):
        if state == 1:
            return t

T = X.apply(first_time_in_state_1)
T.sim(1000).plot()
```



CAL POLY

Markov Chains: Stopping Time

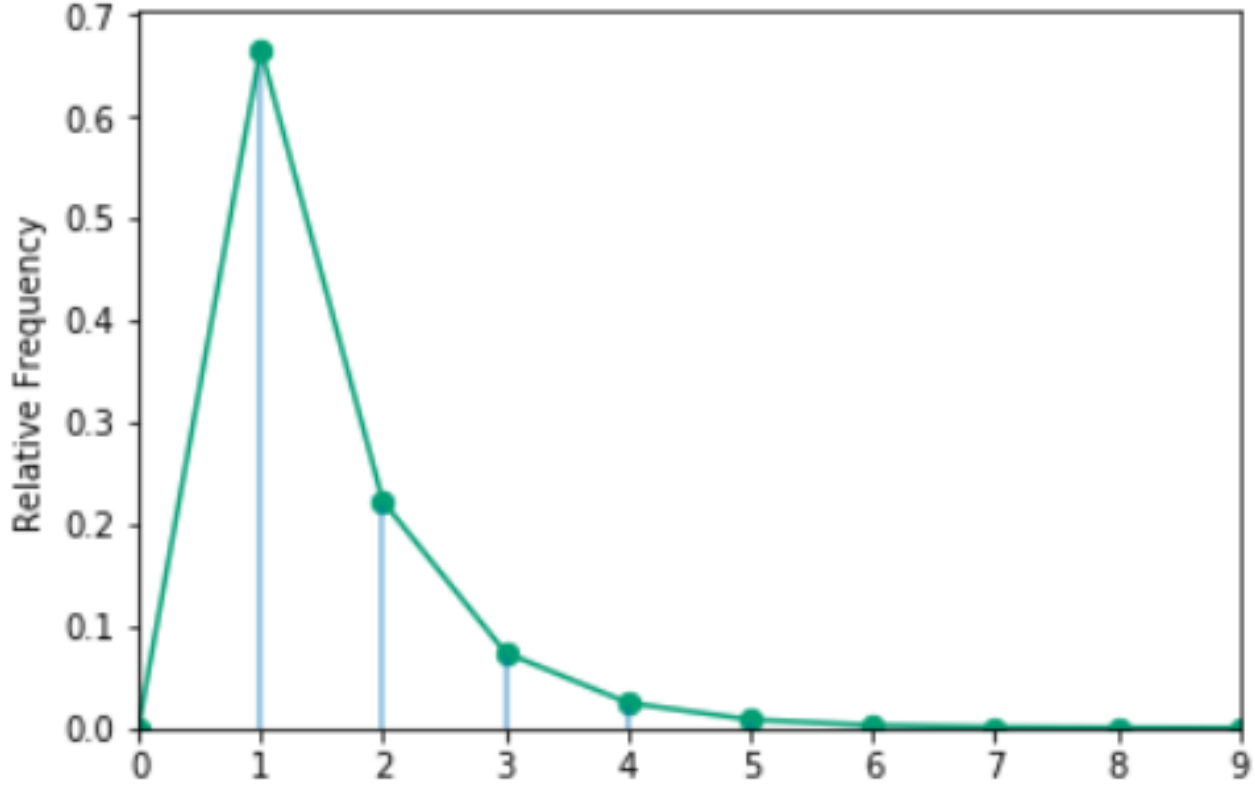
We can evaluate processes at stopping times, too.

```
X[T].sim(100)
```

Index	Result
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
...	...
99	1

Poisson Process at Independent Exponential Time

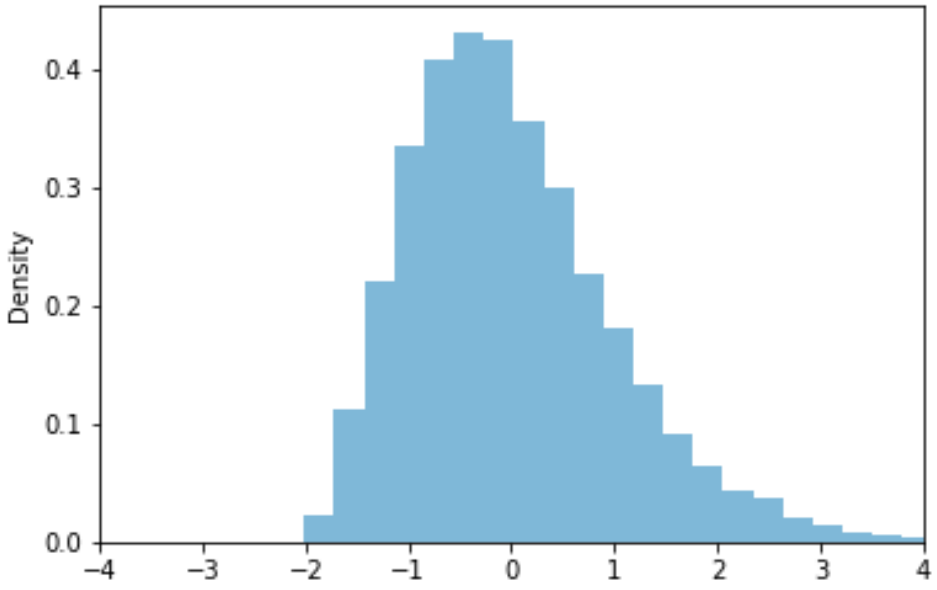
```
N = PoissonProcess(rate=1)
T = RV(Exponential(rate=2))
N, T = AssumeIndependent(N, T)
(N[T] + 1).sim(10000).plot()
Geometric(2/3).plot()
```



Jupyter Widgets

```
population = Exponential(1)

def CLT(n):
    RV(population ** n, mean).sim(10000).standardize().plot()
    plt.xlim(-4, 4)
    plt.show()

from ipywidgets import interact
import ipywidgets
interact(CLT, n=ipywidgets.IntSlider(min=1, max=50, step=1, value=1));
```

n 5

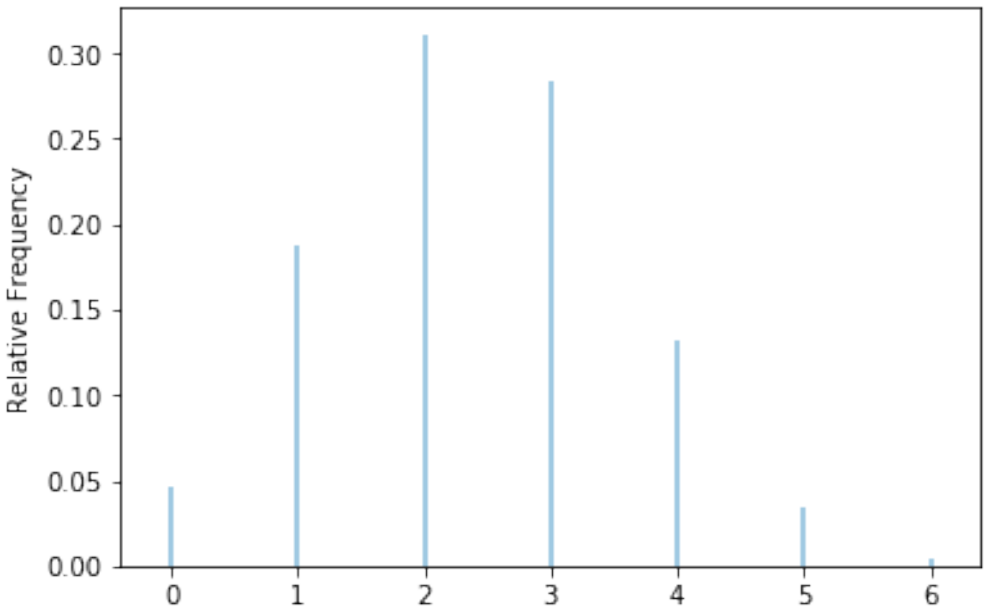


CAL POLY

CAL POLY

Probability World

```
X = RV(Poisson(2))
Y = RV(Poisson(3))
X, Y = AssumeIndependent(X, Y)
Z = (X | (X + Y == 6))
z = Z.sim(10000)
z.plot()
```

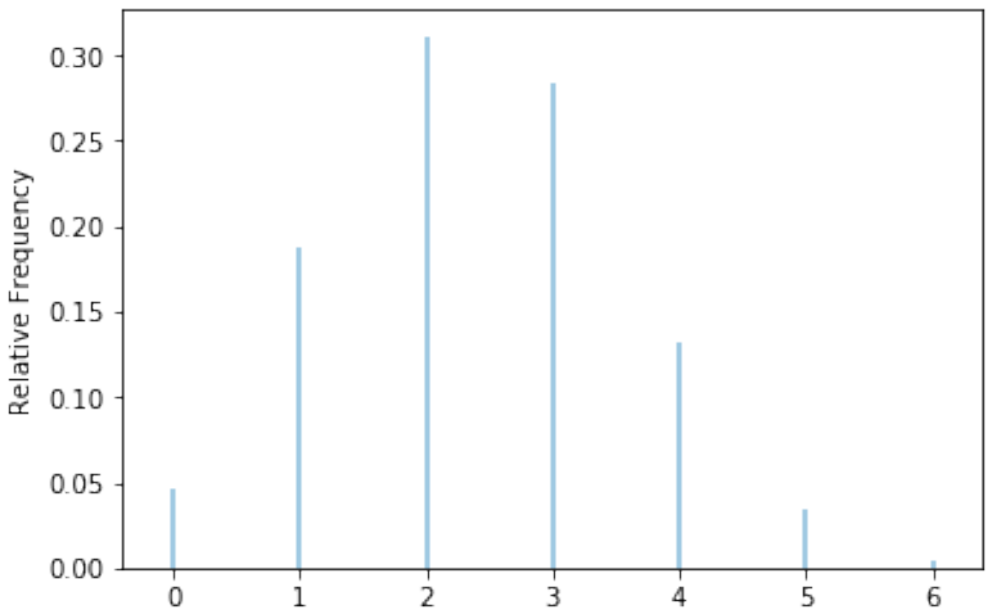


Probability World

```
X = RV(Poisson(2))
Y = RV(Poisson(3))
X, Y = AssumeIndependent(X, Y)
Z = (X | (X + Y == 6))
z = Z.sim(10000)
z.plot()
```

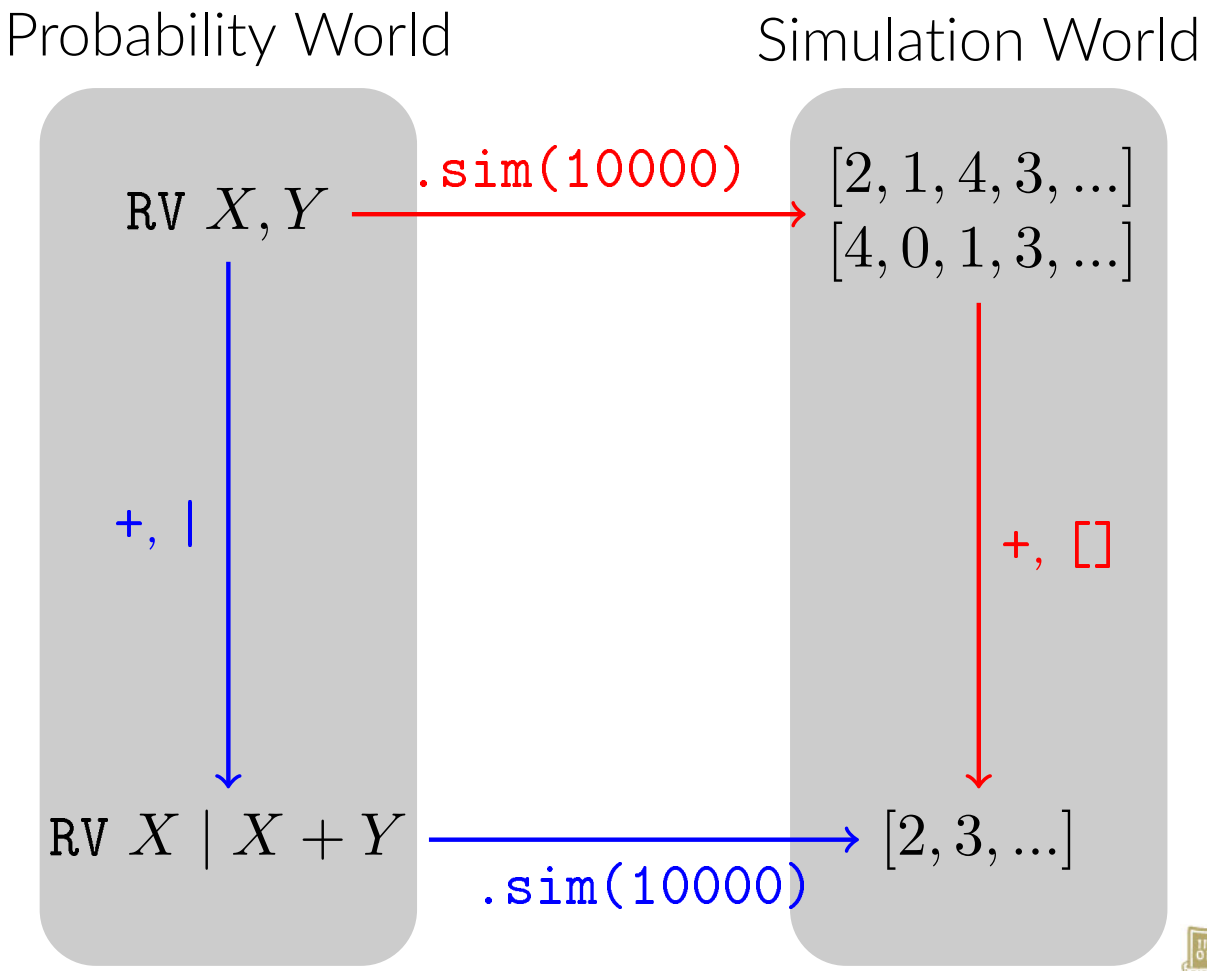
Simulation World

```
X = RV(Poisson(2))
Y = RV(Poisson(3))
X, Y = AssumeIndependent(X, Y)
x, y = (X & Y).sim(10000)
z = x[x + y == 6]
z.plot()
```



Probability vs. Simulation Worlds

For every operation in probability world (e.g., addition and conditioning), there is an equivalent operation in simulation world.



Common Mistakes in Simulation World?

In simulation world, it is easier to write code that makes no statistical sense....

```
X = RV(Poisson(2))
Y = RV(Poisson(3))
X, Y = AssumeIndependent(X, Y)
x = X.sim(10000)
s = (X + Y).sim(10000)
x[s == 6]
```

Common Mistakes in Simulation World?

In simulation world, it is easier to write code that makes no statistical sense....

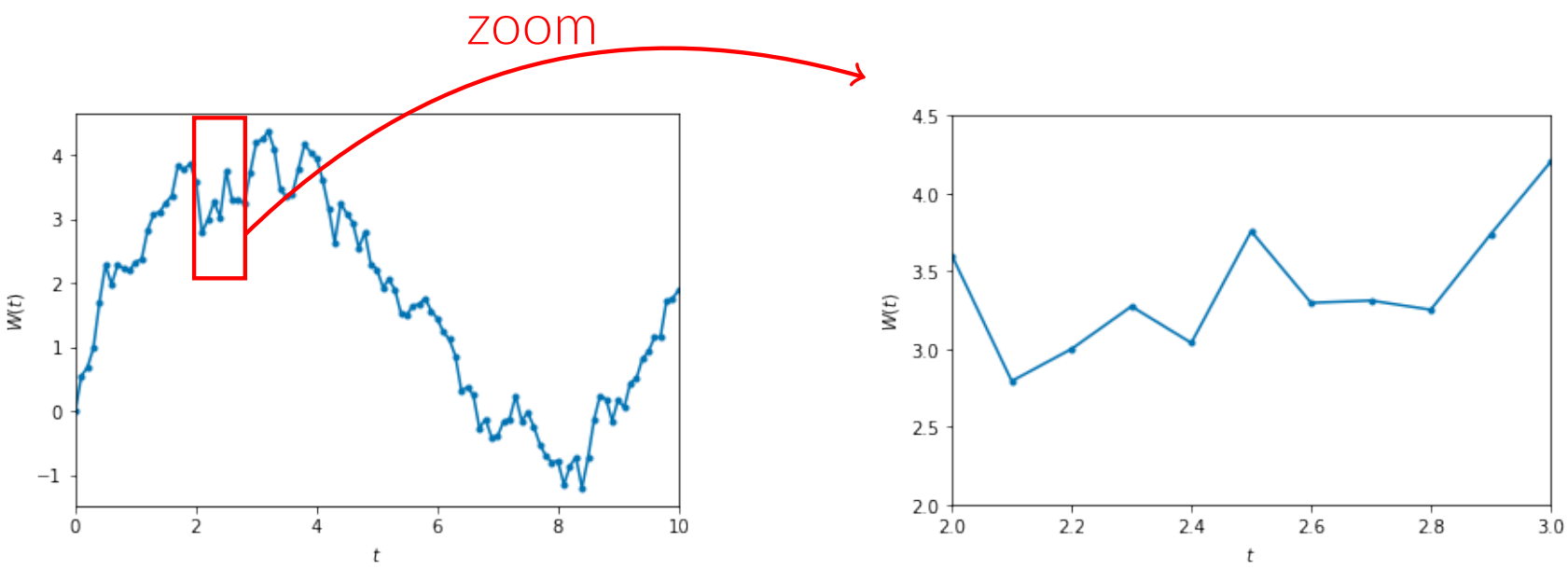
```
X = RV(Poisson(2))
Y = RV(Poisson(3))
X, Y = AssumeIndependent(X, Y)
x = X.sim(10000)
s = (X + Y).sim(10000)
x[s == 6]
```

```
Exception:  In order to filter one set of Results by
another, they must come from the same simulation.
```

...but Symbulate will refuse to perform those operations.

How to Simulate Brownian Motion?

The usual approach is to discretize and simulate a random walk.



The problem is that the resolution is fixed. If we zoom in, then the graph is no longer an accurate representation of the process.

Brownian Motion at Any Resolution?

Each realization w of Brownian motion $\{W(t)\}$ is a function of time.

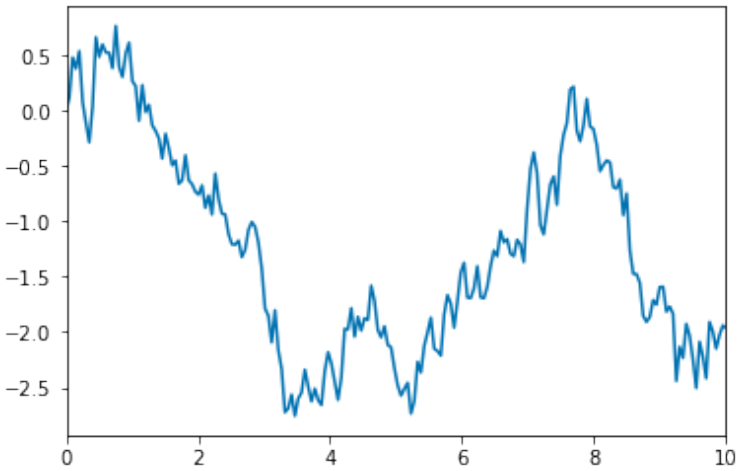
```
In [2]: W = BrownianMotion(drift=0, scale=1)
        w = W.draw()
        w

Out[2]: [continuous-time function]
```

We should be able to evaluate this function at *any* time t.

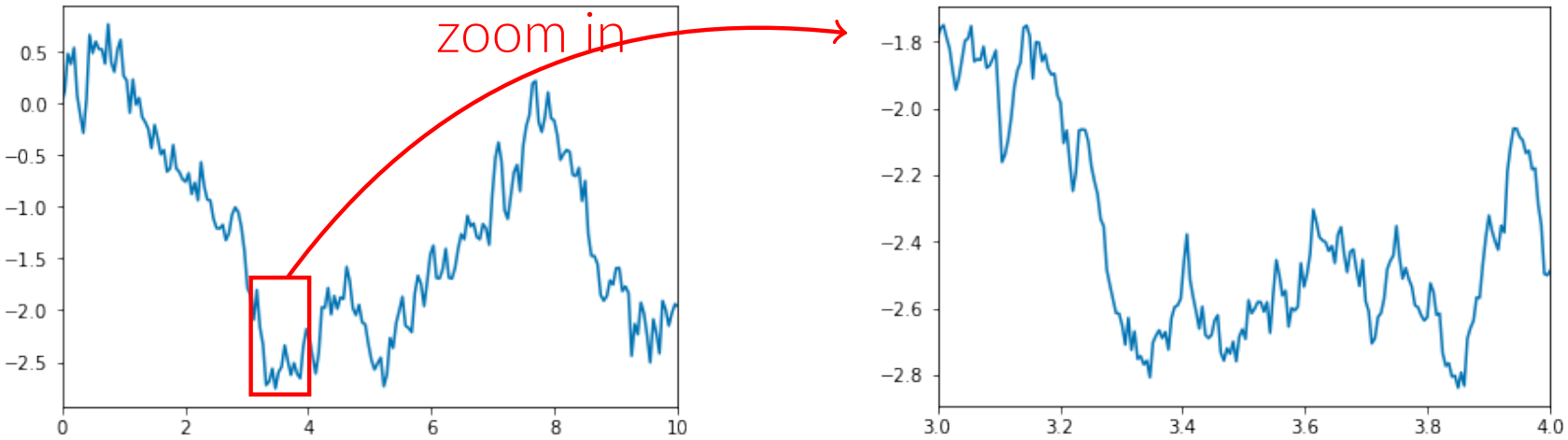
```
In [3]: w(0), w(1), w(pi)

Out[3]: (0, 0.23008791936358794, -1.7661172141876391)
```

We can't just interpolate to get these values because, even though sample paths of Brownian motion are continuous, they are nowhere differentiable.

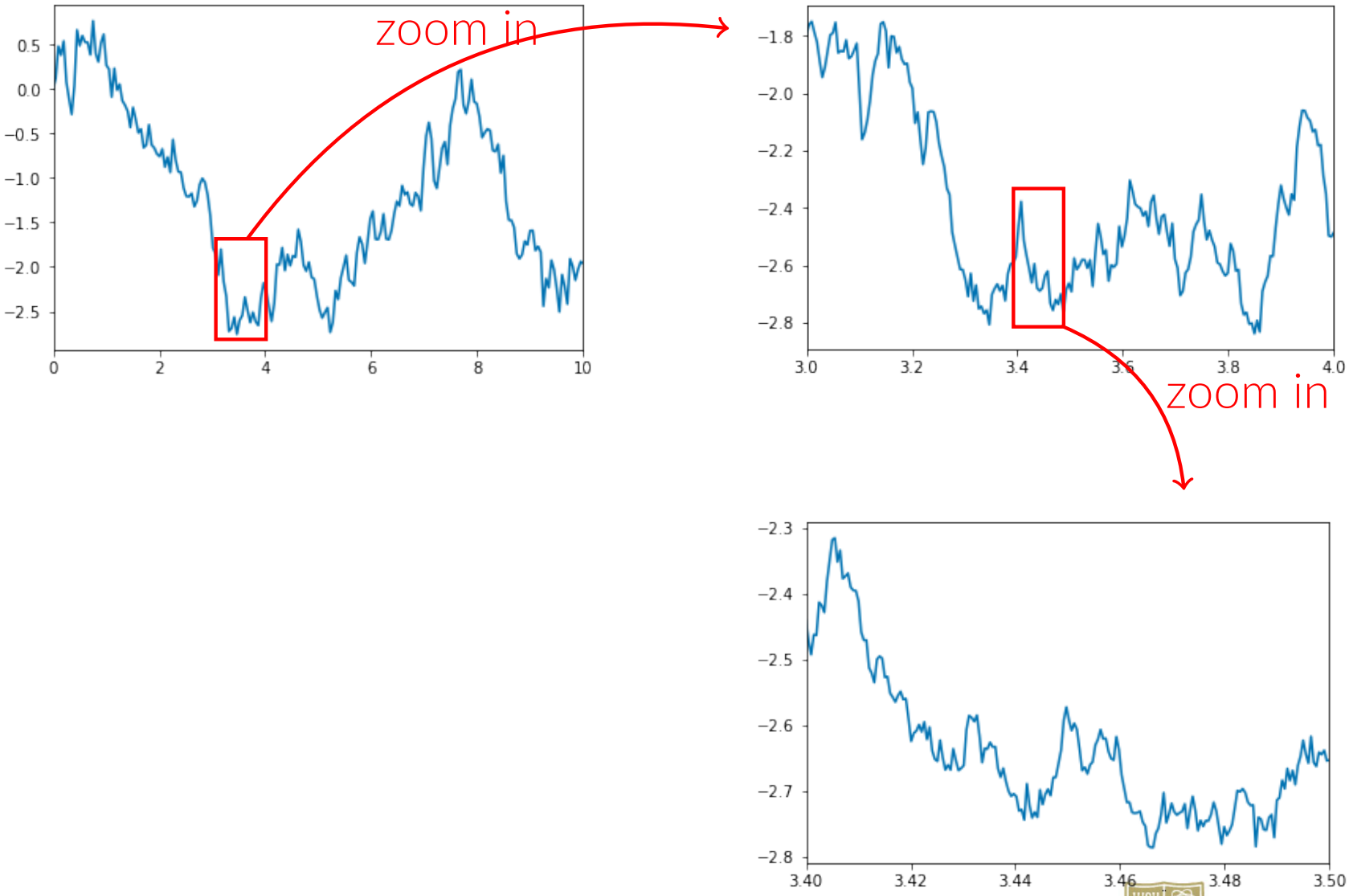
Brownian Motion at Any Resolution?



Brownian Motion at Any Resolution?

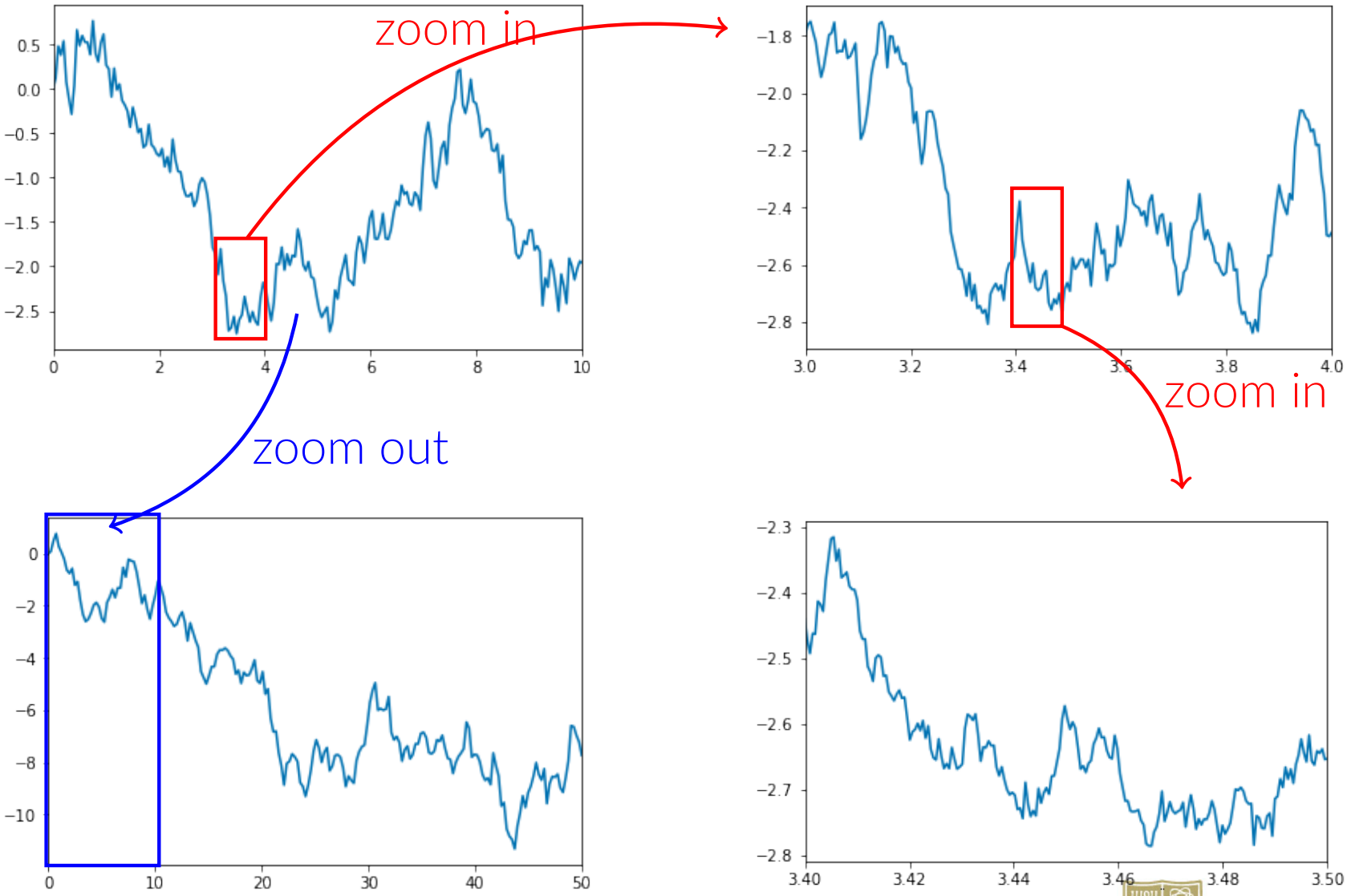


Brownian Motion at Any Resolution?



CAL POLY

Brownian Motion at Any Resolution?



CAL POLY

A Peek Under the Hood

How is Symbulate able to generate a sample path w that can be evaluated at any time t and yet remains consistent across repeated evaluations?

Problem: Brownian motion is nowhere differentiable, so infinitely many values are needed to represent the entire sample path.

A Peek Under the Hood

How is Symbulate able to generate a sample path w that can be evaluated at any time t and yet remains consistent across repeated evaluations?

Problem: Brownian motion is nowhere differentiable, so infinitely many values are needed to represent the entire sample path.

Solution: lazy evaluation

- Only generate the value of $w(t^*)$ as needed. Store the values of t^* and $w(t^*)$ in $\mathbf{t}$ and $\mathbf{w}$, respectively.
- All values are generated from the conditional distribution

$$W(t^*) \mid \{W(\mathbf{t}) = \mathbf{w}\} \sim \text{Normal}\left(\Sigma_{t^*,\mathbf{t}}\Sigma_{\mathbf{t},\mathbf{t}}^{-1}\mathbf{w}, t^* - \Sigma_{t^*,\mathbf{t}}\Sigma_{\mathbf{t},\mathbf{t}}^{-1}\Sigma_{\mathbf{t},t^*}\right).$$

A Peek Under the Hood

How is Symbulate able to generate a sample path w that can be evaluated at any time t and yet remains consistent across repeated evaluations?

Problem: Brownian motion is nowhere differentiable, so infinitely many values are needed to represent the entire sample path.

Solution: lazy evaluation

- Only generate the value of $w(t^*)$ as needed. Store the values of t^* and $w(t^*)$ in $\mathbf{t}$ and $\mathbf{w}$, respectively.
- All values are generated from the conditional distribution

$$W(t^*) \mid \{W(\mathbf{t}) = \mathbf{w}\} \sim \text{Normal}\left(\Sigma_{t^*,\mathbf{t}}\Sigma_{\mathbf{t},\mathbf{t}}^{-1}\mathbf{w}, t^* - \Sigma_{t^*,\mathbf{t}}\Sigma_{\mathbf{t},\mathbf{t}}^{-1}\Sigma_{\mathbf{t},t^*}\right).$$

Symbulate uses a similar approach for other infinite dimensional models, e.g., Poisson processes

CAL POLY

CAL POLY

Symbulate in the Classroom

We have used Symbulate in two undergraduate probability courses:

1. **Probability and Random Processes for Engineers**:
   A first course in probability (and for most, their last) for EE majors, which covers probability up through Gaussian processes and the Wiener-Khinchin theorem. Students had previous exposure to Python. Course required simulation in Symbulate only.

Symbulate in the Classroom

We have used Symbulate in two undergraduate probability courses:

1. **Probability and Random Processes for Engineers**:
   A first course in probability (and for most, their last) for EE majors, which covers probability up through Gaussian processes and the Wiener-Khinchin theorem. Students had previous exposure to Python. Course required simulation in Symbulate only.
2. **Introduction to Probability and Simulation**
   A first course in probability for statistics and data science majors, which covers probability up through the Central Limit Theorem. Students had previous exposure to Python, some to R. Course required simulation in both Symbulate and another language.

CAL POLY

How We Used Symbulate

- Instructors used Symbulate in lecture to demo concepts
- Students used Symbulate in weekly lab meetings
- Students were required to use Symbulate on homework
- Symbulate syntax and usage was assessed on exams

Survey of Students

At the end of the course, we asked students to fill out an anonymous survey about their experience with Symbulate:

	Course	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree
Visualizing simulation results in graphs facilitated my understanding of probability concepts.	1	39%	50%	10%	1%	0%
	2	47	47	6	0	0
Performing and analyzing simulations **using Symbulate** facilitated my understanding of probability concepts.	1	24	55	14	5	1
	2	26	50	20	4	0
The syntax of Symbulate facilitated my understanding of the "language of probability".	1	20	37	33	10	0
	2	11	38	36	11	4
In general, the use **of Symbulate** facilitated my understanding of probability concepts.	1	22	59	12	5	1
	2	15	55	22	5	4

CAL POLY

Survey of Students

	Course	Symbulate	Python	Matlab	R	Other
If you had to do it over, which one of the following would best represent the software you would prefer to use?	1	78%	8%	9%	3%	3%
	2	51	19	0	23	8

- Students taking probability as a terminal course (i.e., Course 1) overwhelmingly preferred Symbulate.
- Students taking probability as a gateway course (i.e., Course 2) still preferred Symbulate, but many wanted more practice with R and Python.

Interested in Symbulate?

- Try it out! To install, just run

  ```
  pip install symbulate
  ```

  at a terminal. (You will need Python and the usual scientific computing stack, e.g., Numpy and Scipy. If you don't have this, download Anaconda.)
- The project is open source. Check it out on Github:

  `http://www.github.com/dlsun/symbulate`
- Look for our paper: "Symbulate: Simulation in the Language of Probability", *Journal of Statistics Education* (2019).

Thank you!

Kevin Ross (`kjross@calpoly.edu`)

CAL POLY