

Random Numbers

TOUR OF ACCOUNTING

OVER HERE
WE HAVE OUR
RANDOM NUMBER
GENERATOR.



www.dilbert.com scottadams@aol.com

NINE NINE
NINE NINE
NINE NINE



10/25/01 © 2001 United Feature Syndicate, Inc.

ARE
YOU
SURE
THAT'S
RANDOM?



THAT'S THE
PROBLEM
WITH RAN-
DOMNESS:
YOU CAN
NEVER BE
SURE.

Random Variables are Useful

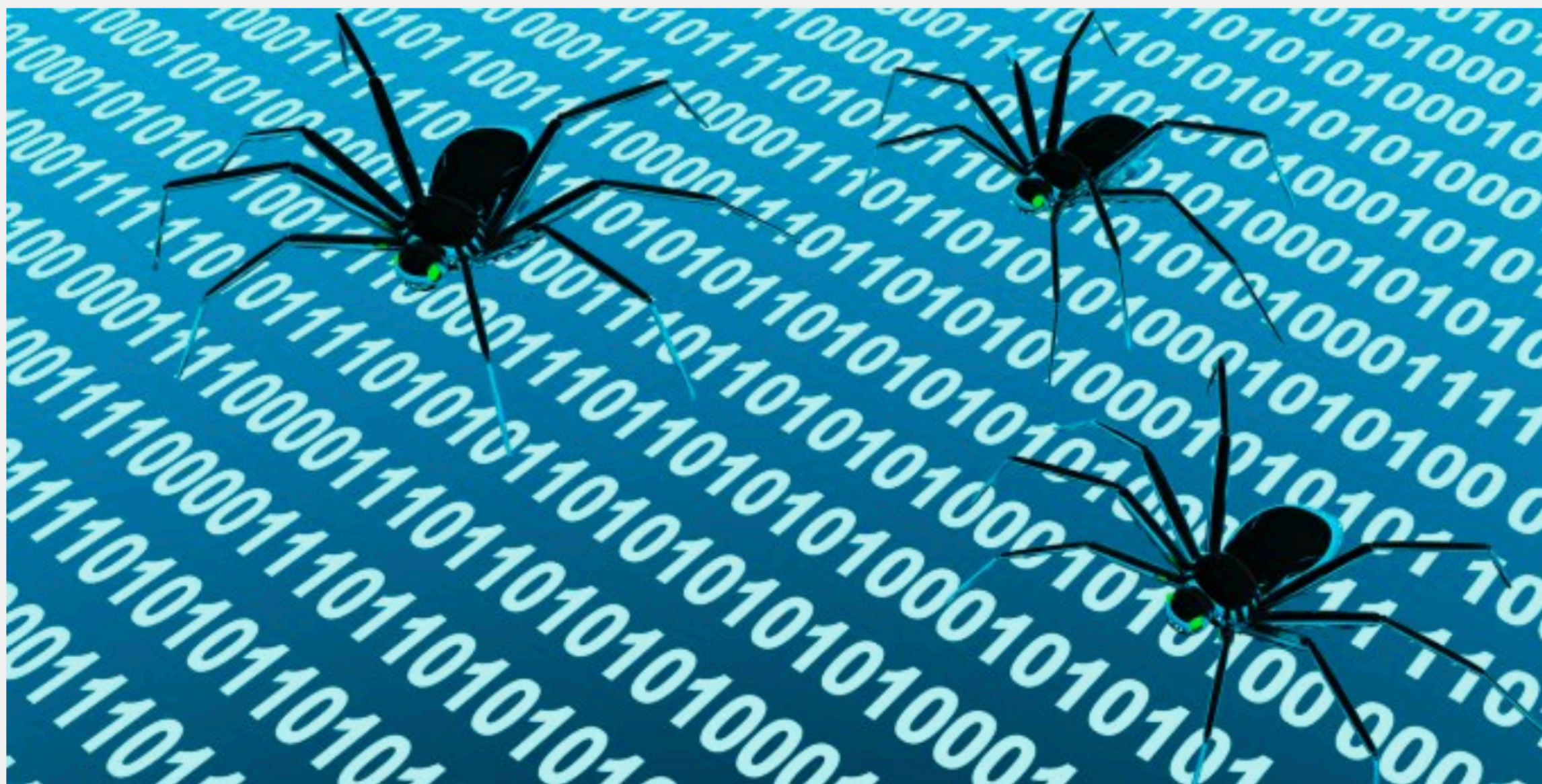
- We've seen we needed to generate lots of random variables for MCMC
- It is very common to not be able to calculate things analytically. Common to do some sort of Monte Carlo simulation.
- As you try to simulate rare events and/or have correlations, the quality of the random number generator (RNG) can be critical.
- There are MANY old, broken ones that don't pass statistical tests. (One of the NR authors spent an extra year in grad school due to a bad RNG).
- Please never use the built-in RNGs in e.g. C unless you really don't care about your answer.

OXFFFFFF EVERY TIME IS OXDEADBEEF —

How a months-old AMD microcode bug destroyed my weekend [UPDATED]

AMD shipped Ryzen 3000 with a serious microcode bug in its random number generator.

JIM SALTER - 10/29/2019, 7:00 AM



Adobe

Enlarge / Ryzen 3000's RDRAND function—what should be a high-quality pseudo-random number generator—just returns 0xFFFFFFFF every time, until its microcode is patched.

PRNG

- Computers very bad at generating truly random numbers
- Instead, starting from some state (*seed*), generate another number that is statistically uncorrelated from previous ones. These are pseudo-random noise generators (PRNGs).
- Usual output is uniformly distributed integer, possibly rescaled to be a float between 0,1.
- `np.random.rand(om)` will return floats from 0,1.
- Important - starting from same state, PRNGs will produce same sequence. If you want reproducibility, make sure you know/set initial state (`np.random.seed`, or fancier)
- Virtually all fancier random numbers work based off of uniform PRNG. *Don't write your own! Do check you're using a good one!*
- Modern implementations very fast - often ~dozen operations. Will affect how we make choices in more complicated situations.

Hardware PRNGs

- On unix-like systems (including mac), random generators built in.
- Entropy comes from environmental noise, e.g. timing jitters
- Can get hardware random seed with e.g.:
`f=open('/dev/random')`
`seed=np.fromfile(f,'uint32',1)`
`np.random.seed(seed)`
- linux also has `/dev/hwrng` (not there on mac)
- not sure about windows...

Non-Uniform

- We'll assume you have access to a good uniform PRNG.
- Let's say you want to simulate the waiting time for a radioactive decay. Distribution proportional to $\exp(-t)$.
- Can we remap a uniform deviate into an exponential one?
- What is the probability that we have to wait longer than time t ? $\exp(-t)$.
- If I told you my waiting time was longer than 30% of the samples, what was it? Well, $\exp(-t)=0.3$, or $t=-\log(0.3)$
- I know that, represented as a probability vs. number of events, my waiting time is a uniform deviate. i.e. half the time, I wait longer than 50% of samples, 10% of the time I wait longer than 90% of the samples etc.
- So, replace 30% by uniform, and $t=-\log(\text{rand})$ should be exponentially distributed.

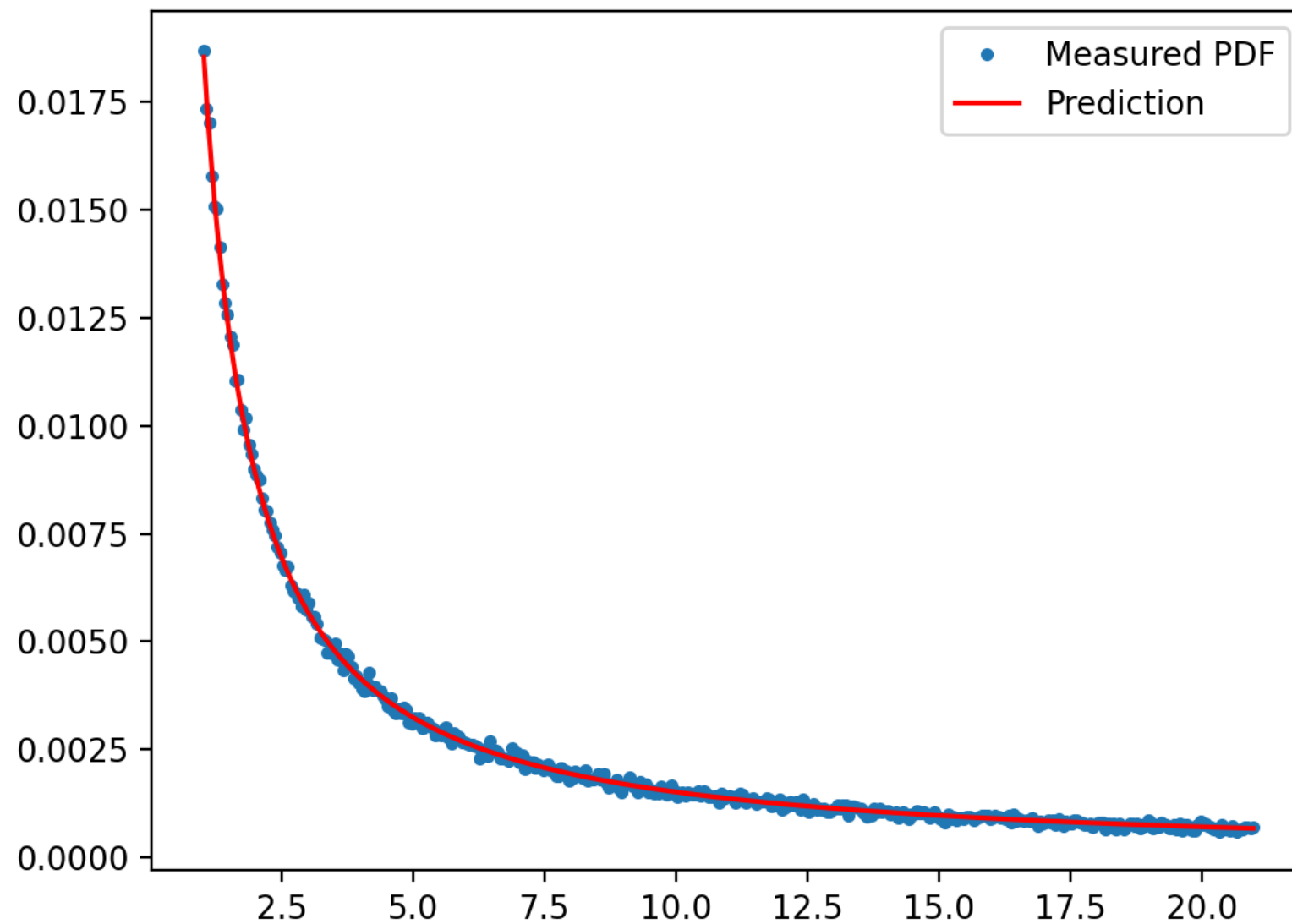
Power Law

- Power law distributions are another common case.
- $dN/ds = s^{-\alpha}$.
- How do we do this?

Power Law

- Power law distributions are another common case.
- $dN/ds = s^{-\alpha}$.
- How do we do this?
- $\int s^{-\alpha} = s^{(1-\alpha)}/(1-\alpha) = x$, $s = ((1-\alpha)x)^{1/(1-\alpha)}$. Need to normalize PDF, cancels $(1-\alpha)$, leaves $s = x^{1/(1-\alpha)}$ where x is uniform on $(0,1)$.
- NB - area unbounded towards 0 for $\alpha > 1$, towards infinity for $\alpha < 1$.

See plot_powlaw.py



Lorentzian

- if $\text{PDF} = 1/(1+x^2)$, $\text{CDF} = \int dx/(1+x^2) = \text{atan}(x)$.
- We need $\text{CDF}(-\infty) = 0$, $\text{CDF}(\infty) = 1$, but $\text{atan}(-\infty) = -\pi/2$ etc.
- So, normalized CDF is $\text{atan}(x)/\pi + 1/2$. $x = \tan(\pi(x - 1/2))$

General Technique

- More generally, if we can integrate the PDF to a CDF and invert that, we can generate deviates drawn from the PDF.
- Sometimes you can do this, sometimes you can't.
- Clever techniques can sometimes help.

Gaussian Deviates (Box-Muller)

- Can we analytically integrate a Gaussian? Sadly, no.
- Can we do so in 2-d? Happily, yes.
- So, we can convert the unit circle to *two* Gaussian numbers.
- $P(x)P(y) = \exp(-(x^2+y^2)/2) dx dy = \exp(-r^2/2) 2\pi r dr$
- $CDF = \int PDF = \exp(-r^2/2)$, $r = (-2\ln(x))^{1/2}$ where x uniform
- BM: pick x, y in $(-1, 1)$. if $x^2 + y^2 > 1$, try again. Otherwise, we want $\cos(\theta)(-2\ln(r^2))^{1/2}$, but $\cos/\sin = x/r, y/r$, so we take $[x, y](-2\ln(r^2)/r^2)^{1/2}$ to be our two deviates.

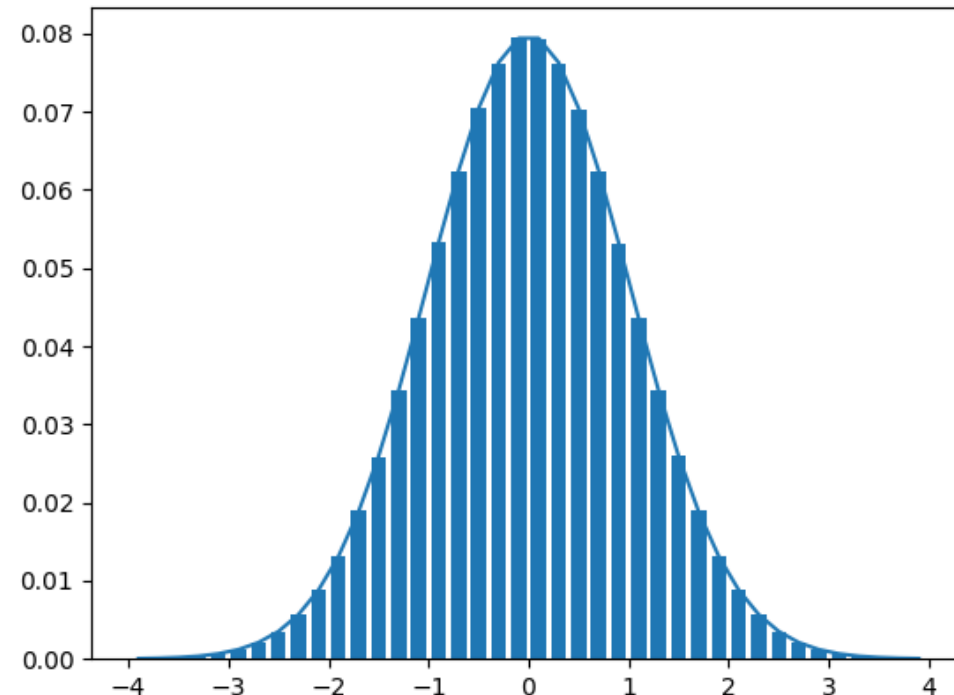
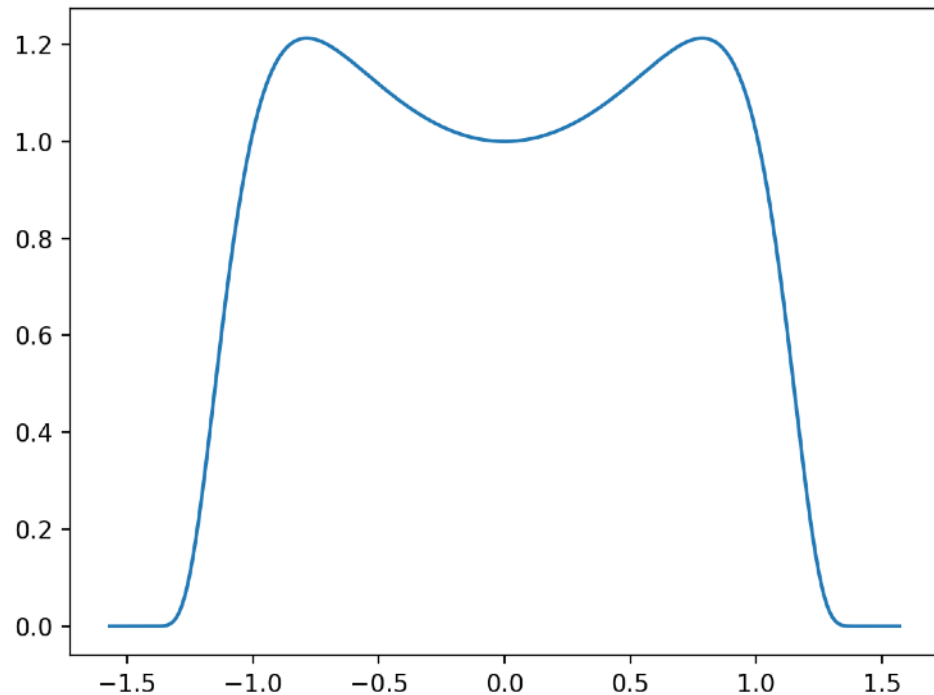
Rejection

- Say I have a distribution I can analytically draw from.
- Then if that is always larger than the distribution I care about, I can ask if a sample from the first falls within the second.
- If yes, that is a random deviate following the second.
- The closer the first distribution is to the second, the more efficient this technique is.

Transforms Revisited

- Rejection can be tricky - need distribution we can analytically sample that is always larger than desired. Easier if range is finite.
- Change of variables: $P'(y)dy = P(x)dx$, $P'(y) = P(x)dx/dy$.
- Pick say $x = \text{atan}(y)$. $dx/dy = \sec^2(y)$. Then $P'(y) = P(\text{atan}(y)) / \cos^2(y)$. But y bounded in $(-\pi/2, \pi/2)$, so any sufficiently tall box works as bounding region.
- Generate samples in box. If sample falls under $P'(y)$, return $\text{atan}(y)$.

Gaussian Deviates from atan

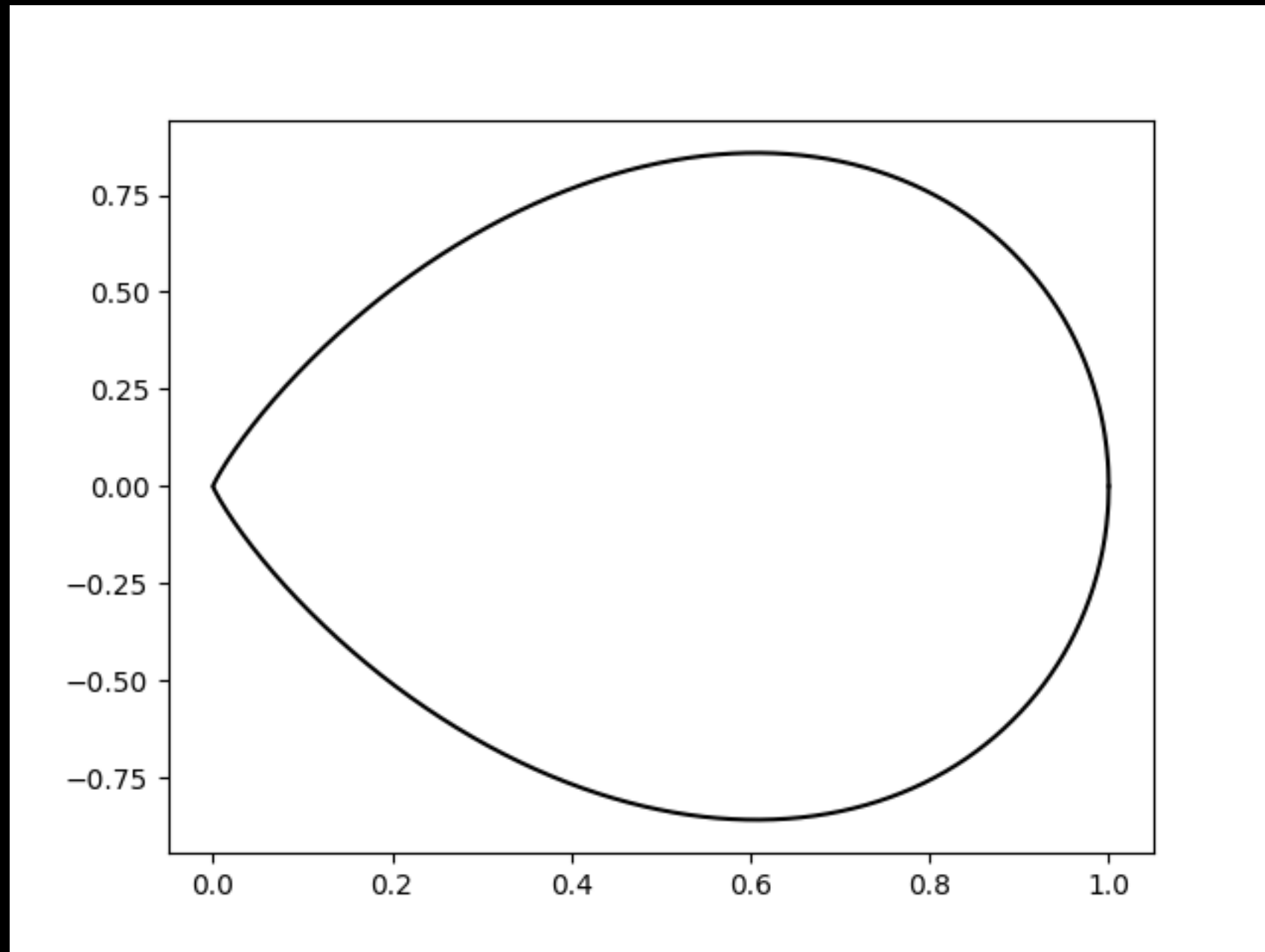


- Left: $P'(y) = \exp(-0.5 \operatorname{atan}(y)^2) / \cos(y)^2$.
- Right: distribution of deviates w/Gaussian prediction plotted.

Ratio of Uniforms

- We can map the number line to a box, allows us to quite generally sample from distributions.
- Math is:
 - take a (u,v) plane where $0 < u < \sqrt{p(v/u)}$
 - sample u,v uniformly in this region
 - return $v/u(!)$
- This works because Jacobian is constant, so this is a remapping of the full number line.
- In practice: draw a box in u,v big enough to cover the probability region. Draw a random sample, if it falls inside the probability region, return v/u .

Gaussian ROU Acceptance Region



Throw darts at the board until one lands inside the teardrop. One you do, report v/u for that point.

Squeeze

- Slowest part of ratio of uniforms usually evaluation of pdf.
- If we can write curve that are easy to check inside & outside true PDF, we can save time
- The closer interior/exterior bound (“squeeze”) the answer, the better off we are
- Any sufficiently fast curves help. The better they are, the fewer times we need to check exact PDF, but answers will still be correct for not great curves.

Brute Force

- Of course, if you have the PDF, you can tabulate it and make a sum to get the CDF.
- You can then just numerically invert the CDF to get random deviates. Accuracy will only be as good as your interpolation/inversion, but efficiency is very high and can handle arbitrary distributions.
- Can always combine with e.g. transform to get fit onto compact region