# Computational Methods in Stochastics

Lecture II

# Sampling from Different Distributions aka Stochastic Simulation

## Logarithmic binning

# Motivation: Stochastic Simulation

We want to understand the dynamics of a stochastic system.

More precisely, we want to determine how an initial distribution of some (continuous) quantity evolves in time.

So, all models **in this context** are just distributions. All model parameters $\theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$ are parameters characterising these distributions, for example, mean and variance.

We have a random variable $X$ with probability density function (PDF) $f(x)$ (or $f_X(x)$). We wish to evaluate $\mathrm{E}(g(X))$ for some function $g(\cdot)$: $\mathrm{E}\big(g(X)\big) = \int_X g(x) f(x) dx$.

Here, think of $f(x)dx$ as the **measure** you are working with: you measure everything with respect to this probability measure.

# Motivation: Stochastic Simulation

We resort to *Monte Carlo integration:*

We *simulate* realisations of $x_1, \ldots, x_n$ of $X$ and form realisations of the random variable $g(X)$ as $g(x_1), \ldots, g(x_n)$. Then, provided that the variance of $g(X)$ is finite, the law of large numbers assures that we can approximate the integral by

$$\mathrm{E}\big(g(X)\big) \simeq \frac{1}{n}\sum_{i=1}^{n} g(x_i). \quad (x_i \text{ are drawn from } f)$$

Even if we cannot simulate realisations of $X$, but can simulate realisations of $Y$ that has PDF $h(\cdot)$, then

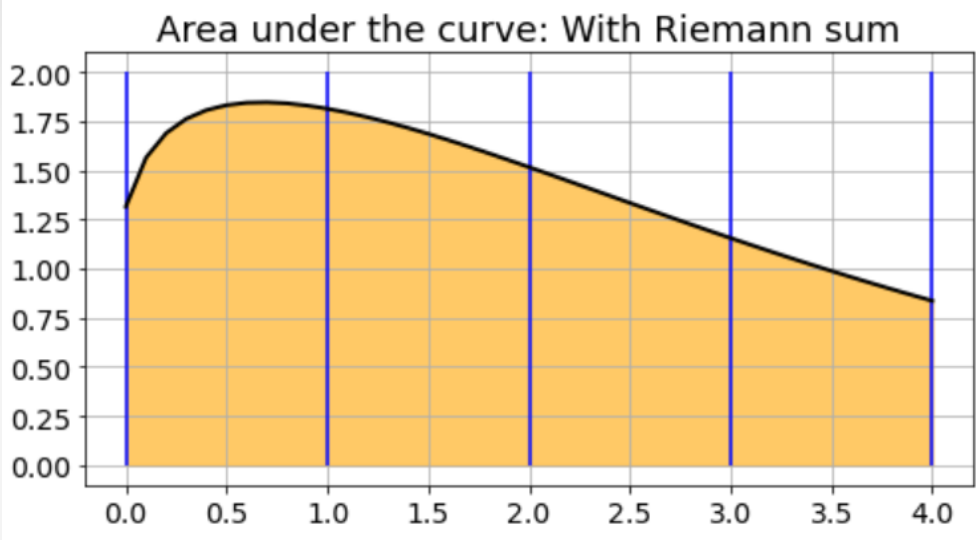
$$E\big(g(X)\big) = \int_X g(x)f(x)dx = \int_X \frac{g(x)f(x)}{h(x)}h(x)dx$$

$$\Rightarrow E(g(X)) \simeq \frac{1}{n}\sum_{i=1}^{n} \frac{g(y_i)f(y_i)}{h(y_i)}.$$

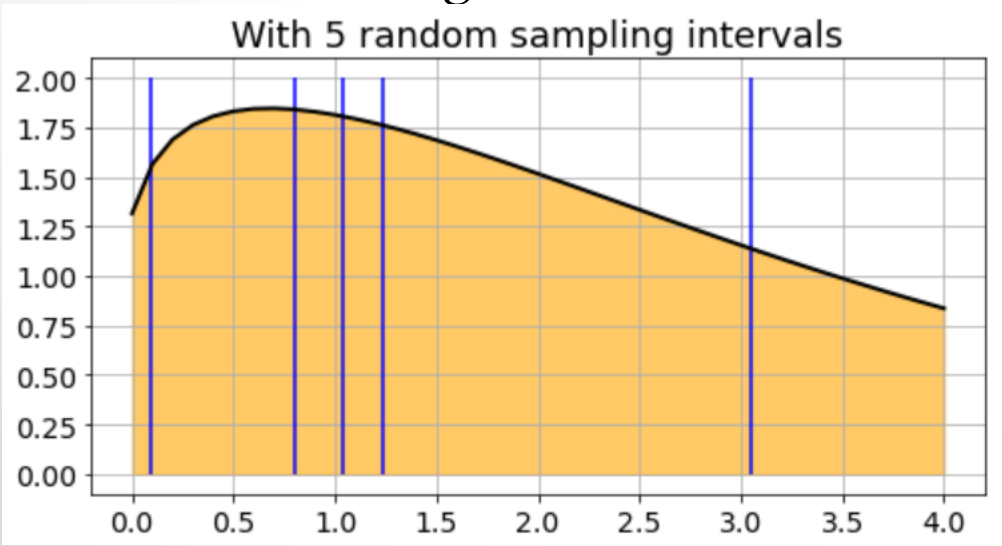
This is i*mportance sampling*.

# Motivation: Stochastic Simulation

*Basic numerical integration:*




*Monte Carlo integration:*




In (non-random) numerical integration one sums up, for instance values at mid-points of intervals.

In Monte Carlo integration one takes randomly samples of the area/volume under/inside the curve. MC integration is more effective in dimension 3 or greater.

# Motivation: Stochastic Simulation

Once more to make this clear: We have PDF $f(\cdot)$ that is hard to simulate. So, we simulate $h(\cdot)$ that is easier and use the realisations of $Y$ (samples drawn from that distribution) and compute the values of $f(y)$ and $h(y)$ in the process of computing the expectation - we change the measure.

$$E\big(g(X)\big) = \int_X g(x)f(x)dx = \int_X \frac{g(x)f(x)}{h(x)}h(x)dx$$

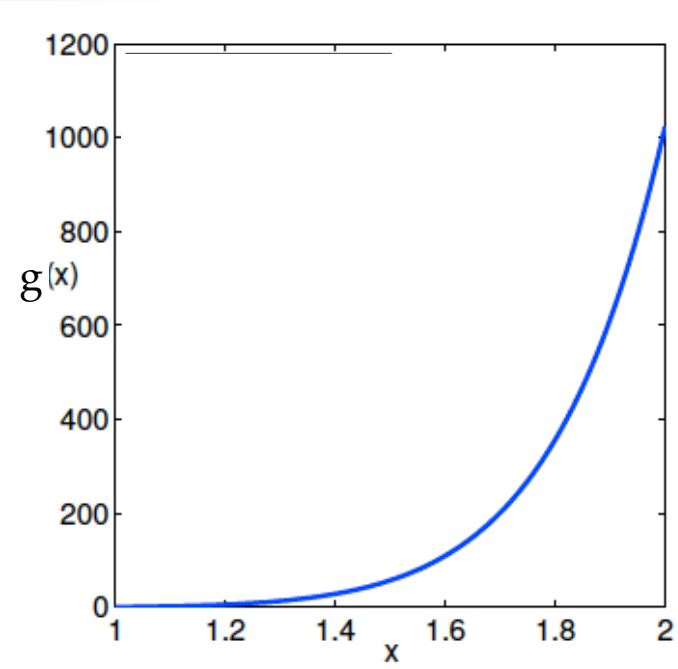
$$\Rightarrow E(g(X)) \backsimeq \frac{1}{n}\sum_{i=1}^{n}\frac{g(y_i)f(y_i)}{h(y_i)}. \qquad (y_i \text{ drawn from } h)$$

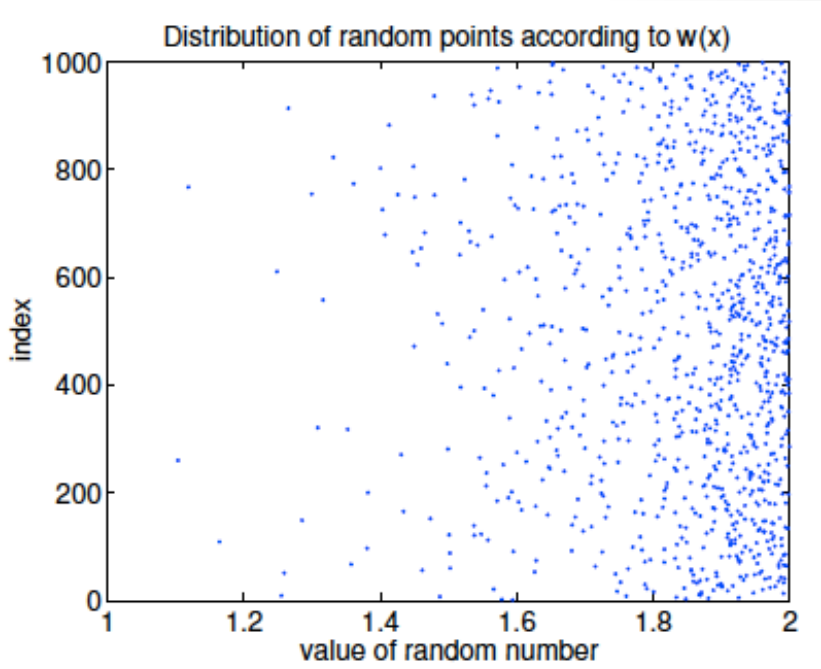
This may also be written as $E(g(X)) \backsimeq \frac{1}{n}\sum_{i=1}^{n} g(y_i)\, w(y_i),$

where $w(y_i) = \frac{f(y_i)}{h(y_i)}$ is a *weight function*; $\int_X w(x)dx = 1.$

# Motivation: Stochastic Simulation

$h(x)$ should be as similar to $f(x)$ as possible to make the integration computationally effective: this way sampling is done primarily in the regions that contribute most to $E\big(g(X)\big) \rightarrow$ weighted integration.



# Transformation Methods

**A prerequisite: Change of variable**

The mathematical foundation for transformations of (random) variates reads as follows:

The PDF of an arbitrary differentiable invertible transformation $Y = g(X)$ is

$$f_Y(y) = f_X\left(g^{-1}(y)\right)\left|\frac{d}{dy}g^{-1}(y)\right|.$$

The last term is the Jacobian of the transformation (in dimension 2 or greater). (For an example in 2D, see the Box-Müller method, page 16.)

"Equal measures": $f_Y(y)dy = f_X\left(g^{-1}(y)\right)d[g^{-1}(y)]$.

# Transformation Methods

*"Proof"*. The event $\{Y \leq y\}$ is the same as the event $\{X \leq g^{-1}(y)\}$

$$\Rightarrow F_Y(y) = \Pr\{Y \leq y\} = \Pr\{X \leq g^{-1}(y)\} = F_X(g^{-1}(y))$$

$$y = g(x) \Leftrightarrow x = g^{-1}(y)$$

$$\frac{dg^{-1}}{dy} = \frac{dx}{dy} = \frac{1}{g'(x)}$$

$$f_Y(y) = \frac{dF_Y(y)}{dy} = \frac{dF_X(g^{-1}(y))}{dy} = \frac{dF_X(\mathrm{x})}{dx}\frac{dx}{dy} = \frac{1}{g'(x)} f_X(x)$$

$$= \frac{dg^{-1}(y)}{dy} f_X\left(g^{-1}(y)\right) \blacksquare$$

The differentiation $\frac{dg^{-1}(y)}{dy}$ generalises to a Jacobian in multiple dimensions.

# Transformation Methods

**The inverse distribution method**

When the inverse transformation can be determined, the *inverse distribution method* can – should, for efficiency, - be used:

1. Sample $y$ from a uniform distribution $y \in [0,1]$ (notation Y $\sim \mathcal{U}_{[0,1]}$, or $Y \backsim U(0,1)$).

2. Compute $x = F^{-1}(y)$, where $F(x) = \Pr\{X \leq x\}$. (If $x \in [a,b]$, then $F(x) = \int_a^x p(x')dx'$, or the sum $F(x) = \sum_{a \leq x_i \leq x} p(x_i)$ in the discrete case.) $X$ follows the distribution $F(x)$ as desired.

**Proposition.** If $Y \sim U(0,1)$ and $F(\cdot)$ is a valid invertible cumulative distribution function (CDF), then $X = F^{-1}(Y)$ has CDF $F(\cdot)$.

*Proof.*

$$P(X \leq x) = P(F^{-1}(Y) \leq x) = P\big(Y \leq F(x)\big) = F_Y\big(F(x)\big) = F(x).$$

# Transformation Methods

**Example.** *Uniform random variates*. Given $U \sim U(0,1)$, simulate $V \sim U(a,b)$, where $a < b$.

Now, $F^{-1}(u) = v = a + (b-a)u$. $\left(F(v) = \frac{v-a}{b-a}, \quad a \le v \le b.\right)$

1. Sample u ∈ [0,1].
2. Compute $v$ from $v = a + (b-a)u$.

**Example.** *Exponential random variates*. Given $U \sim U(0,1)$, simulate X $\sim Exp(\lambda)$.

PDF $f(x) = \lambda e^{-\lambda x}, x \ge 0. \Rightarrow$ CDF $F(x) = 1 - e^{-\lambda x}, x \ge 0.$

$$\Rightarrow F^{-1}(u) = x = -\frac{1}{\lambda}\log\big(1 - F(x)\big) = -\frac{1}{\lambda}\log(1-u)\,.$$

1. Sample u ∈ [0,1].
2. Compute $x$ from $\mathrm{x} = -\frac{1}{\lambda}\log(1-u)$.

# Transformation Methods

**Example.** *Variates from the Lorentzian distribution.*

$$\text{PDF} \qquad p(x) = \frac{1}{\pi}\frac{1}{1+x^2} \qquad (-\infty < x < \infty)$$

1. Sample $y \in [0,1]$.
2. Compute $x$ from

$$y = F(x) = \int_{-\infty}^{x} \frac{1}{\pi}\frac{1}{1+x'^2}dx' = \frac{1}{2} + \frac{1}{\pi}\arctan(x)$$

$$\Rightarrow x = F^{-1}(y) = \tan\left[\pi(y - \frac{1}{2})\right]$$

Learn to do this "without thinking", that is, learn the "notation" and derive $F^{-1}$ using it. It's easy to get mixed up, if you think too much! Assignments →

# Transformation Methods

When playing around with distributions it is often useful to identify factors contributing to *scaling* and *location*. In the prototypical case, if $Y$ has PDF $f(y)$ and CDF $F(y)$, and $X = aY + b$ (affine relation), then $X$ has CDF

$$F_X(x) = P(X \leq x) = P(aY + b \leq x) = P\left(Y \leq \frac{x-b}{a}\right) = F\left(\frac{x-b}{a}\right)$$

and PDF

$$f_X(x) = \frac{1}{a} f\left(\frac{x-b}{a}\right).$$

Here, $a$ is the *scale parameter* and $b$ is the *location parameter*.

# Transformation Methods

**Example.** *Gamma random variates.* Simulate $X \sim Ga(n, \lambda)$ random variates for integer $n$.

Exact inverse $F^{-1}(\cdot)$ does not exist, but use the property that if $Y_i \sim Exp(\lambda)$, and the $Y_i$ are independent, then

$$X = \sum_{i=1}^{n} Y_i \backsim Ga(n, \lambda).$$

In practise, to simulate gamma random variates, simulate exponential random variates and sum them up. ($Ga(1, \lambda) =$ Exp($\lambda$), see Lecture 1, p. 39.) Note: sum of memoryless processes → Gamma.

$n$ is the *shape parameter* and $\lambda$ is the *scale parameter*.

Often library algorithms generate $Ga(n, 1)$, after which one needs to rescale: if $Y \sim Ga(n, 1)$, then $X = Y/\lambda \sim Ga(n, \lambda)$.

# Transformation Methods

**Normal random variates**

Obviously, efficient simulation of Gaussian random quantities is of great importance. However, the inverse transformation cannot be applied.

We need a technique for simulating $Z \sim N(0,1)$ random variables, because $X = \mu + \sigma Z \sim N(\mu, \sigma^2)$.

*CLT-based method*

This approximate method makes use of the central limit theorem.

For example, $Z = \sum_{i=1}^{12} U_i - 6$ , where $U_i \backsim U(0,1), i = 1, 2, \ldots, 12$

are independent, is approximately normal with $E(Z) = 0$ and $\mathrm{Var}(Z) = 1$.

# Transformation Methods

**Normal random variates**

This example case for the CLT-based method has support on [-6, 6] and is poorly behaved in the extreme tails. For $Z \sim N(0,1)$, $P(|Z| > 6) \approx 2\times10^{-9}$, so the truncation is not a major problem in many applications. The method is, however, slow: 12 uniform random numbers to get one Gaussian.

Remember: In CLT $Z_n = \frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}}$; then as $n \to \infty$ $Z_n \sim N(0,1)$.

# Transformation Methods

**Normal random variates**

*The Box-Muller method* - this is the standard method

- fast and efficient way to generate normal random variates

1. Simulate $\Theta \sim U(0,2\pi)$ and $R^2 \sim Exp(1/2)$ independently.

2. Compute two independent standard normal random variables

$X = R\cos\Theta$ and $Y = R\sin\Theta$.

*Proof.* $f_{X,Y}(x,y) = \frac{1}{2\pi}\exp(-(x^2+y^2)/2)$

If $X = R\cos\Theta$ and $Y = R\sin\Theta$,

$$f_{R,\Theta}(r,\theta) = f_{X,Y}(x,y)\left|\frac{\partial(x,y)}{\partial(r,\theta)}\right| = \frac{1}{2\pi}e^{-r^2/2}\begin{vmatrix}\cos\theta & -r\sin\theta\\ \sin\theta & r\cos\theta\end{vmatrix}$$

$$= \frac{1}{2\pi}re^{-r^2/2}$$

$\nwarrow \left|\frac{\partial(x,y)}{\partial(r,\theta)}\right| = \begin{vmatrix}\partial x/\partial r & \partial x/\partial\theta\\ \partial y/\partial r & \partial y/\partial\theta\end{vmatrix}$ (Jacobian)

# Transformation Methods

*The Box-Muller algorithm:*

1. Generate $u_1, u_2 \sim U(0,1)$ independently.
2. Compute $r = \sqrt{-2 \ln u_1}$ and
   $\theta = 2\pi u_2$
3. Compute $x = r \cos(\theta)$ and
   $y = r \sin(\theta)$
4. Multiply $x$ and $y$ by $\sigma_X$ and $\sigma_Y$ to get the desired variances $\sigma_X^2$ and $\sigma_Y^2$. Add the desired $\mu$.
5. Go to 1.

You can use this stuff as an aid when doing Assignment 2.1.

# Lookup Methods

The lookup method is **the discrete version of the inverse transformation method**. It is used in simulating a discrete random quantity $X$ with outcome space $S = \{0, 1, 2, \dots\}$ (ordered by increasing probabilities).

The probability mass function (PMF) of $X$: $p_k = \Pr\{X = k\}, k = 0, 1, 2, \dots$

Define $q_k = \Pr\{X \leq k\} = \sum_{i=0}^{k} p_i$.

To generate realisations of $X$, first simulate $u \sim U(0, 1)$, then compute $X = \min\{k | q_k \geq u\}$.

$\Rightarrow q_{k-1} < u \leq q_k$, so

$$\Pr\{X = k\} = \Pr\{u \in (q_{k-1}, q_k]\} = q_k - q_{k-1} = p_k.$$

(This method is used e.g. for simulating discrete Markov models.)

# Rejection Samplers

Rejection samples are used in generating random numbers that follow an arbitrary distribution $p(x)$ that is not analytically invertible.

***Basic method***

We want to simulate from $f(x)$ with finite support on $[a, b]$. If we can determine $m$ such that $f(x) \leq m, \forall x \in [a, b]$, we can simulate $X \sim U(a, b)$ and $Y \sim U(0, m)$ and accept $X$, if $Y < f(X)$, otherwise reject and try again. The accepted $X$ values have PDF $f(x)$.

(For a proof, see e.g. Proposition 4.4. in Stochastic Modelling for Systems Biology.)

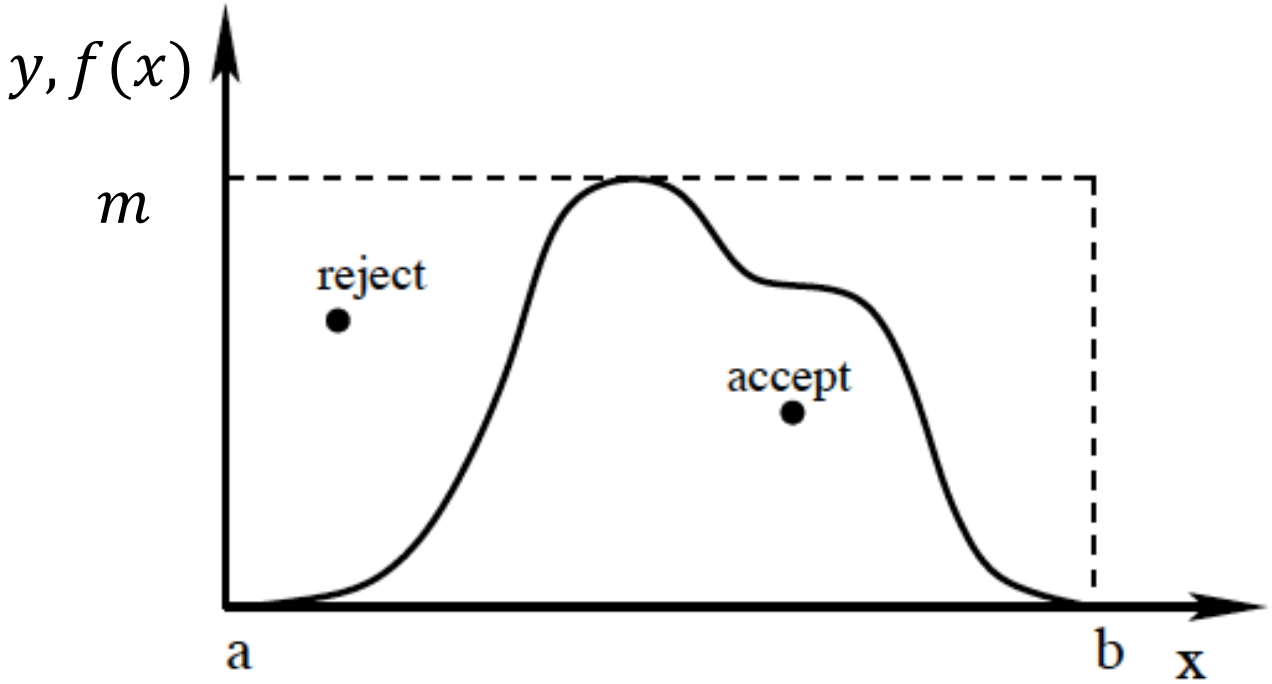
# Rejection Samplers

The algorithm:

1. Generate $x$ in the support of $X$: $x \in [a, b]$.

2. Generate $y \sim U(0,1)$: if $y \leq f(x)/m$ , accept $x$. (This means: "accept $x$ with probability $f(x)/m$".)

3. Go to 1.



# Rejection Samplers

The acceptance probability for this method is

$$\Pr\{\text{Accept}\} = \Pr\{(X,Y) \in A\}$$
$$= \int_a^b \Pr\{(X,Y) \in A | X = x\} \times \frac{1}{b-a} dx = \int_a^b \frac{f(x)}{m} \times \frac{1}{b-a} dx$$
$$= \frac{1}{m(b-a)} \int_a^b f(x) dx = \frac{1}{m(b-a)}.$$

If Pr{Accept} is very low, one can use the *envelope method*.

# Rejection Samplers

***The envelope method***

This method also extends application of the rejection method to distributions with infinite support. Decent computational efficiency can be obtained choosing the enveloping region carefully.

To simulate $X$ with PDF $f(\cdot)$ we choose a PDF $h(\cdot)$ such that

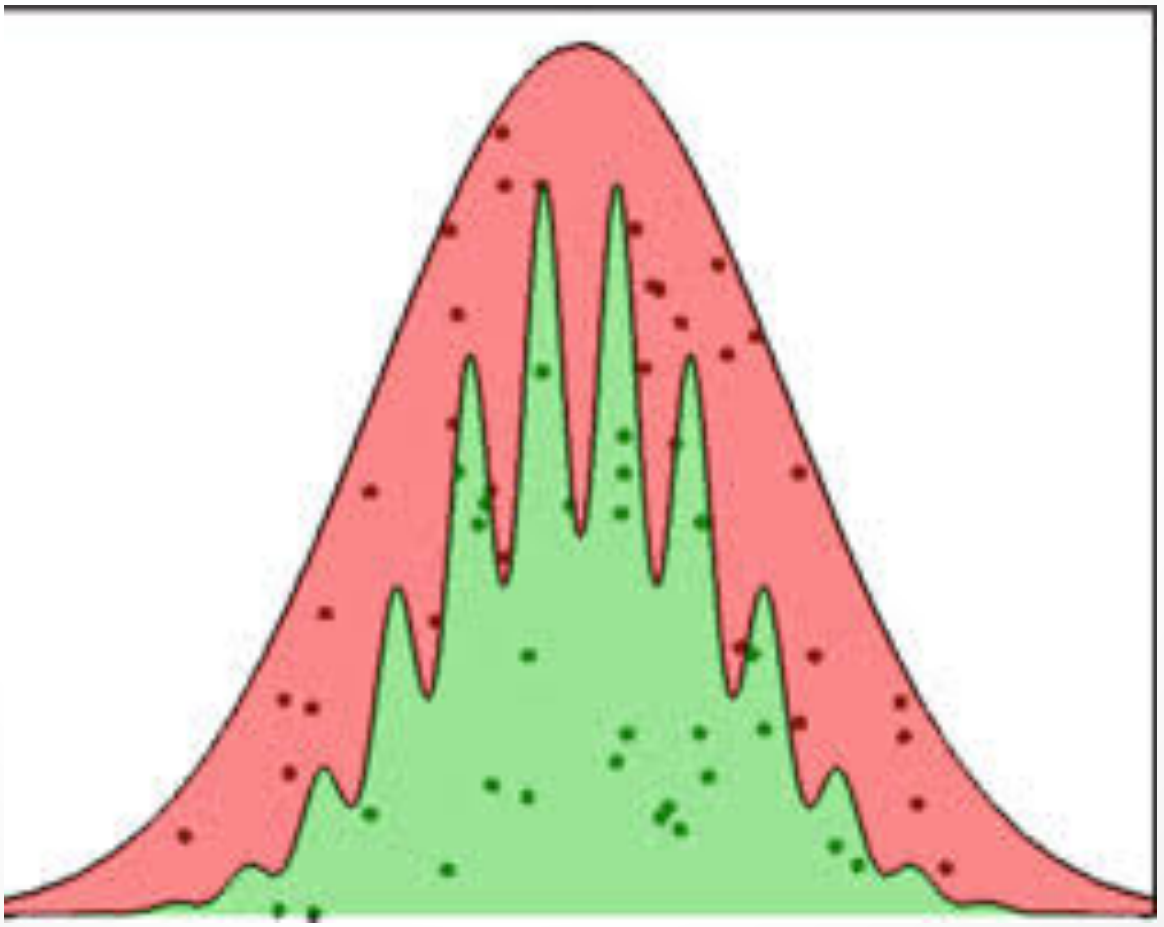
$$f(x) \leq ah(x), \forall x$$

and for which we can simulate values of $Y$ with the same support as $X$.

$a$ is an upper bound for $f(x)/h(x)$; $a \geq 1$, since both $f(x)$ and $h(x)$ integrate to 1.

# Rejection Samplers

*The envelope method.*



# Rejection Samplers

*The envelope algorithm.*

1. Draw $Y = y$ from $h(\cdot)$ and then $U = u \sim U\big(0, ah(y)\big)$.

2. Accept $y$ as a simulated value of $X$ if $u < f(y)$ (with probability $f(y)/[ah(y)]$)*, otherwise reject.

3. Go to 1.

(* "with probability $f(y)/[ah(y)]$" means:
accept $y$ if $z < f(y)/[ah(y)]$, where $z \sim U[0,1]$)

This procedure distributes points uniformly over a region covering $f(x)$ and keeps ones under $f(x)$.

The overall acceptance probability:

$$\Pr\{U < f(Y)\} = \int_{-\infty}^{\infty} \Pr\{U < f(Y)\} \,|Y = y\} h(y) dy = \int_{-\infty}^{\infty} \frac{f(y)}{a} dy$$

$$= \frac{1}{a}. \quad \rightarrow a \text{ should be as close to 1 as possible.}$$

(In practise, $a$ should be smaller than 10.)

# Importance Resampling

This stochastic simulation method sort of spawns from importance sampling and the envelope rejection method.

The advantage over the envelope rejection method is that one does not have to "guess" a good envelope and bounding constant.

Instead, any proposal distribution $h(\cdot)$ having the same support as $f(\cdot)$ can be used. In practise, the method works the better the more similar $h(\cdot)$ is to $f(\cdot)$.

Unlike importance sampling, the importance resampling is approximate: the samples are only approximately from $f(\cdot)$. The approximation improves with increasing number of generated samples.

# Importance Resampling

Here, we rewrite the expectation of an arbitrary function in the importance sampling

$$E(g(X)) \simeq \frac{1}{n}\sum_{i=1}^{n}\frac{g(y_i)f(y_i)}{h(y_i)}$$

as

$$E(g(X)) \simeq \frac{1}{n}\sum_{i=1}^{n} w_i g(y_i),$$

where $w_i = f(y_i)/h(y_i)$.

The procedure:

1. generate samples from the proposal $h(\cdot)$
2. *resample* from the sample using the weights $w_i$.

→ the new sample is distributed *approximately* according to $f(\cdot)$.

# Importance Resampling

**The algorithm:**

1. Sample $y_1, y_2, \ldots, y_n \sim h(\cdot)$.
2. Compute the weights $w_k = f(y_k)/h(y_k)$, $k = 1, 2, \ldots, n$.
3. Compute the sum of the weights $w_0 = \sum_{j=1}^{n} w_j$.
4. Compute the normalised weights $w'_k = w_k/w_0$, $k = 1, 2, \ldots, n$.
5. Sample $n$ times *with replacement* from the set $\{y_1, y_2, \ldots, y_n\}$ using the probabilities $\{w'_1, w'_2, \ldots, w'_n\}$ (using e.g. the lookup method) to generate a new sample $\{x_1, x_2, \ldots, x_n\}$.
6. Return the new sample $\{x_1, x_2, \ldots, x_n\}$ as an approximate sample from $f(\cdot)$.

**Note:** ”Sample with replacement” means that if you pick for example $y_3$, you ”mentally” put it back to the set, so you may pick it again.

# Importance Resampling

So, samples from $h(\cdot)$ are used as if they were samples from $f(\cdot)$, only they are re-weighted by $w_i$. So, generate samples from the proposal $h(\cdot)$ and resample from the sample using the weights $w_i$. One more effort to explain the resample part: Once you have the list $\{y_1, y_2, \ldots, y_n\}$, you resample $m$ times from this list using the weights $\{w_1, w_2, \ldots, w_m\}$ as probabilities to get a new list $\{x_1, x_2, \ldots, x_n\}$ and you use this as an *approximate* sample from the original $f(\cdot)$. Because there is replacement, you might have more than one instances of the same $y_i$. What's the point? To generate several approximate samples from the original – that is, different instances. To see the "sense" in this method, one might think of using the limited sample as if there were a number of them.

One should try to find $h(\cdot)$ as closely reminiscent to $f(\cdot)$ as possible.

# Importance Resampling

After all this fancy talk: Importance resampling just uses the already generated ensemble of samples and takes “new” batches of samples from it. This is done when the number of available samples is limited. This is all you need to remember of it in addition to the fact that such a trick exists; look into it and use it when you need it.

# Binning of Distributions

In order to analyse/identify distributions and make them more "presentable" one often needs to bin the data. The data is assorted to intervals along the abscissa (the first axis).

**Linear binning:** The range $[a, b]$ in the abscissa is divided into $L$ intervals of equal width $\Delta = (b - a)/L$. The data element $x_j$ belongs to the $i$th bin, if $x_j \in [a + (i-1)\Delta, a + i\Delta)$. After assorting the data there will be $N_i$ elements in the $i$th bin. One point represents the data in each bin. The binned data:

$$\check{x}_i = a + \frac{\Delta}{2} + (i-1)\Delta \quad ;$$

$$y_i = \frac{N_i}{\Delta}, \text{where } N_i \text{ is the number of } x_j \in [a + (i-1)\Delta, a + i\Delta)$$

# Binning of Distributions

**Logarithmic binning:** The range $R$ in the abscissa is divided into intervals of width $\Delta_i = \exp(i/r) - \exp[(i-1)/r]$, where $i \in \{-R, -(R-1), \ldots, R-1, R\}$. $R$ and $r$ define the range and resolution, respectively. The data element $x_j$ belongs to the $i$th bin, if $x_j \in [\exp\left[\frac{(i-1)}{r}\right], \exp(i/r))$. After assorting the data there will be $N_i$ elements in the $i$th bin. One point represents the data in each bin. The values of these points are calculated as

$$\breve{x}_i = \frac{\exp[(i-1)/r] + \exp(i/r)}{2};$$

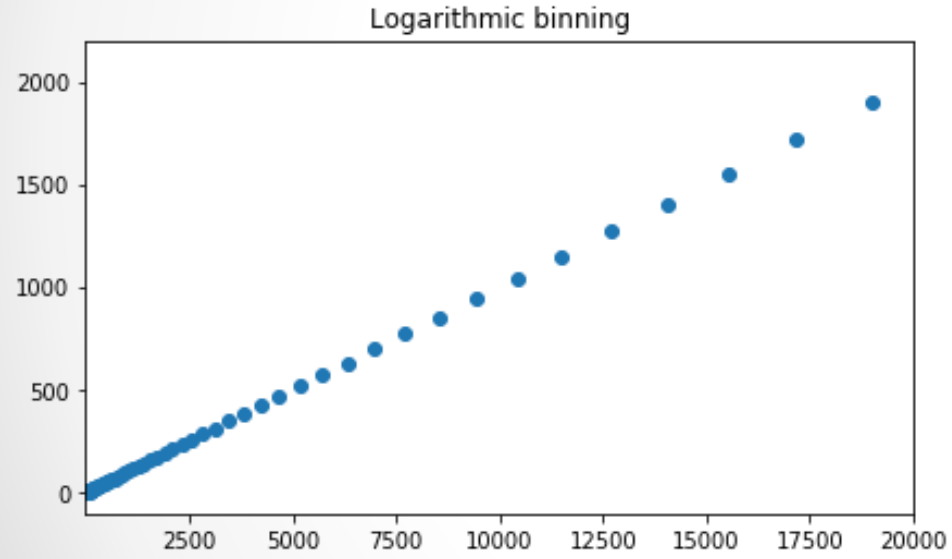
$y_i = \frac{N_i}{\Delta_i}$, where $N_i$ is the number of $x_j \in [\exp\left[\frac{(i-1)}{r}\right], \exp(i/r))$.

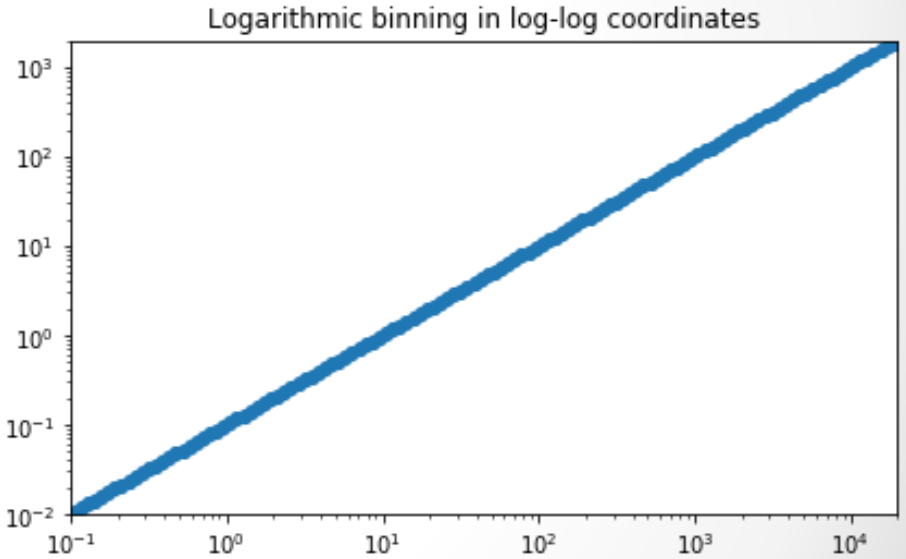
(You generate and plot points $(\breve{x}_i, y_i)$ to represent the data.)

For the layman: The bins are really small for small $i$ and grow exponentially with $i$. Plot in log-log coordinates.

# Binning of Distributions

Here are $\Delta_i$ vs interval midpoints in linear (left) and logarithmic (right) coordinates.

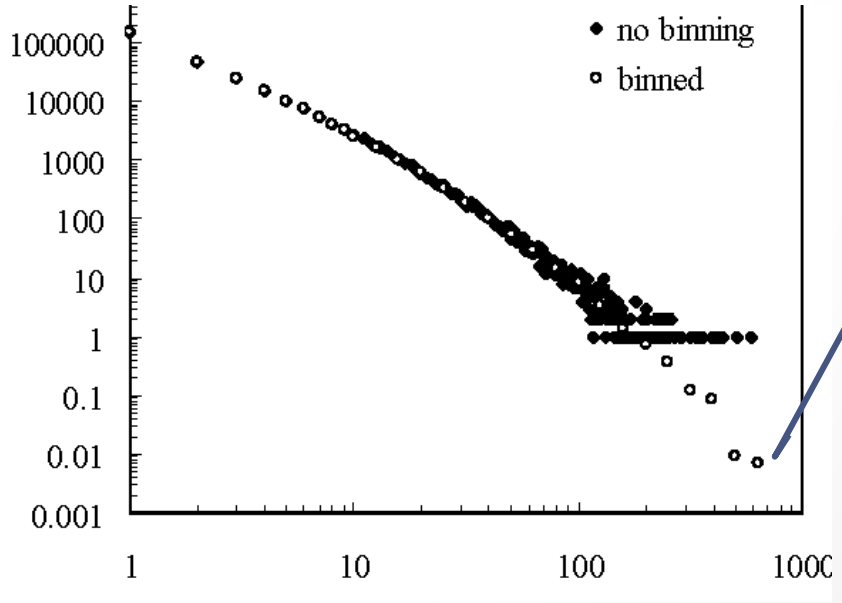


# Binning of Distributions

Log-binning allows you to see the functional dependence of a fat-tailed distribution (density) more precisely at small $x$. It also extends the visibility of the functional dependence to larger $x$ values. (This last bit holds for real data, not the one generated in the assignment.)

To my knowledge, there is no direct way to do log-binning in Python (see a log-bin query).




So, you will implement your own log-binning in an assignment.

# Word of advise about axes

When trying to understand a given data, plot in linear, semilog, and log-log axes to see what works.
Expecting exponential growth or decay: use semilog.
Expecting power-law or log-normal and/or if you are dealing with a random process you expect to exhibit multiplicative stochasticity: use log-log.