CSE 312

Foundations of Computing II

Lecture 27: Random Sampling

Random sampling

- We have seen a number of probability distributions
  - some defined by intuitive natural random processes
  - other useful distributions that seem more exotic
- We want to generate **samples** from these distributions
  - For simulating natural processes
  - For our randomized algorithms to behave well
    - Polling
    - Bloom filters
    - MinHash
    - PageRank
    - Differential Privacy
    - … (Cryptography)

Random Number Generation

- In Python, like many programming languages, there is a single core random generator `random()` for distribution $Unif(0,1)$
  - Actually generates 53-bit precision floats in $[0.0, 1.0)$
  - Based on Mersenne Twister mod $2^{19937} - 1$

    24th Mersenne Prime

    23rd Mersenne Prime

  - Only *pseudorandom*
    - Actually deterministic except for any randomness in initial "seed"
    - Good enough for many applications
    - (but not good enough for cryptography)

Assume that this really is $Unif(0,1)$ … What about other distributions?

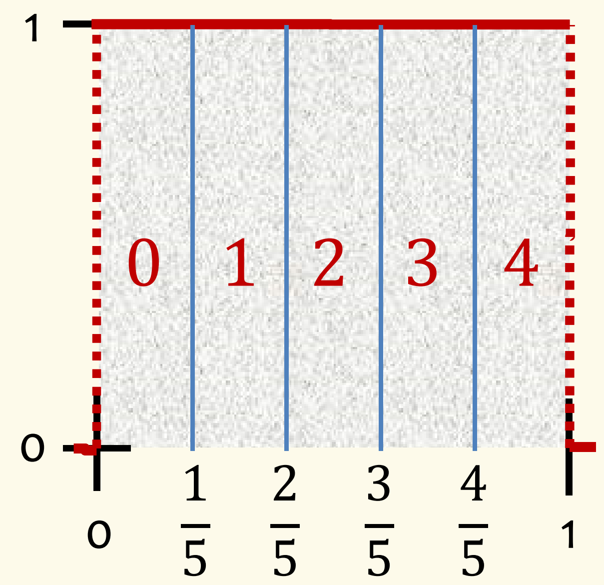
Uniformly random integer X in $\{0, \ldots, n-1\}$?

randrange(n)

$$X \leftarrow \lfloor nU \rfloor$$

Actually **floor(n*random())**
with a small correction …

$U \sim \mathrm{Unif}(0,1)$



Generating $Z \sim Geometric(1/2)$?

From the original definition…

Repeat fair coin flip until heads:

```
z=1
while(randrange(2)!=1)
   z=z+1
```

Drawbacks of this method?

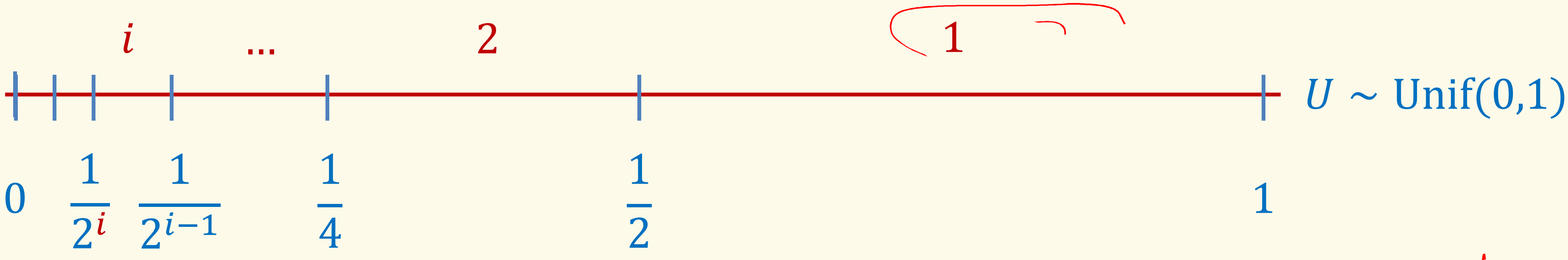
- Time taken depends on the value of Z
- If value of Z is i then it takes i calls `randrange(2)`
  - That implies i calls to `random()`
  - Such calls aren't cheap
- May allow timing attacks by others observing time taken

Is there a better way?

Generating $Z \sim Geometric(1/2)$?

From a single call to **`random()`**:

- All we need is $P(Z = i) = 1/2^i$



$$Z = \lceil \log_2(1/U) \rceil$$

base

```
z=ceiling(log(1/random(),2))
```

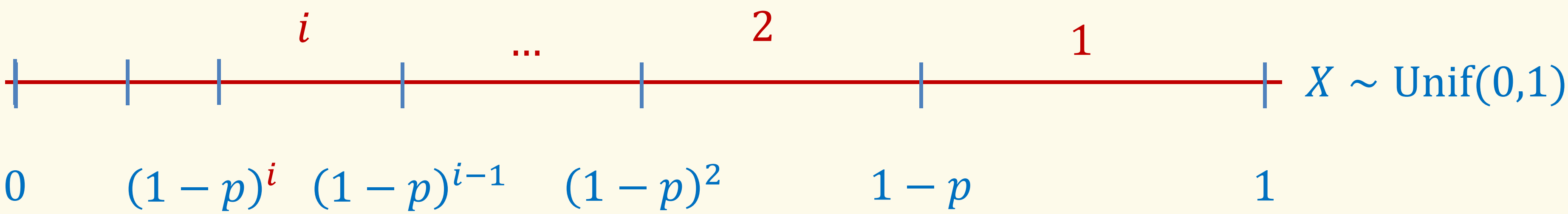
or better

```
z=ceiling(log2(1/random()))
```

Generating $Z \sim Geometric(p)$?

From a single call to `random()`:

- All we need is $P(Z = i) = (1-p)^{i-1}\, p = (1-p)^{i-1} - (1-p)^{i}$



$$Z = \lceil \log_{1-p} U \rceil = \lceil \log_{1/(1-p)} (1/U) \rceil$$

```
z=ceiling(log(1/random(),1/(1-p)))
```

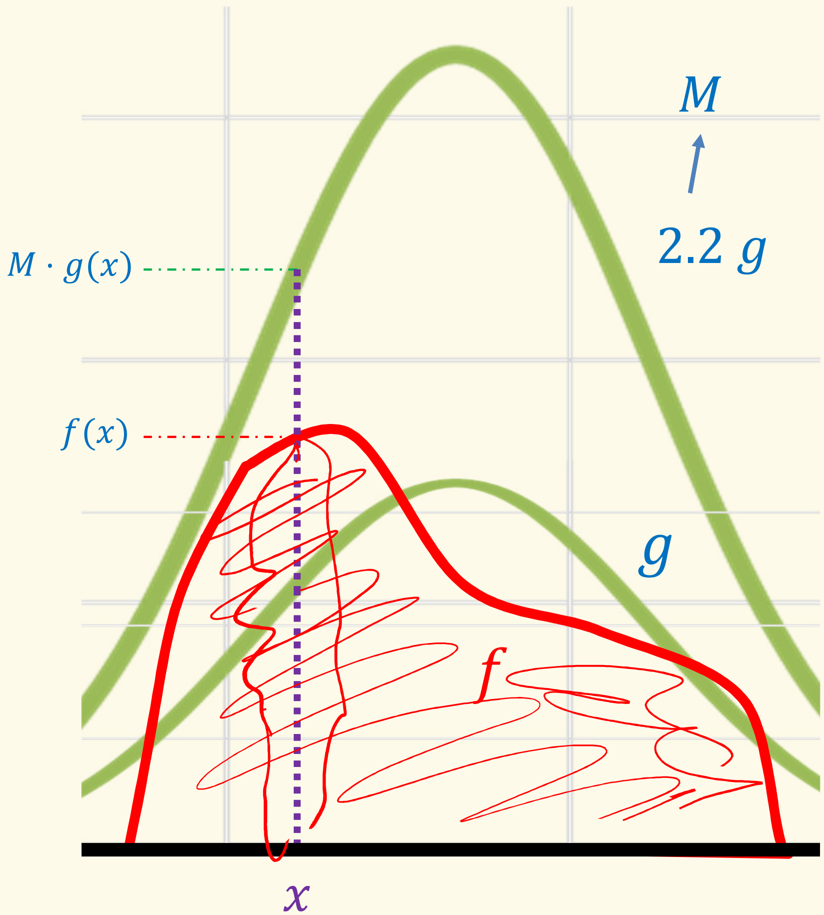
Agenda

- Sampling basics
- Rejection sampling
- Reservoir sampling

Rejection Sampling

- Method for sampling from distribution X with pdf function f that you can evaluate given that you:
  - Can sample from "candidate" simpler distribution Y with pdf function g you can evaluate such that $g(x) > 0$ whenever $f(x) > 0$
    - More precisely, for some M, we have $f(x) \leq M \cdot g(x)$ for every x
  - Can sample from $\mathrm{Unif}(0,1)$

Rejection Sampling



$U' \sim \mathrm{Unif}(0, Mg(x))$

If $U' \le f(x)$ then

1. Sample x according to candidate distribution Y with pdf g
2. Choose $U \sim \mathrm{Unif}(0,1)$
3. If $U \le \frac{f(x)}{M \cdot g(x)}$ return x

   else go to step 1 and repeat

   (reject the sample)

Claim: x is distributed according to pdf f

Idea: Equivalent to generating a uniform point under $M \cdot g$ curve and accepting if point is under f curve

Rejection Sampling

$E_{x\sim Y}[h(x)] = \int_{-\infty}^{\infty} h(x)\, g(x)\, dx$

What is the probability we get a sample in a round?

$$P\left(U \le \frac{f(x)}{M\cdot g(x)}\right)$$

$[0,1]$

$$= \mathbb{E}_{x\sim Y}\left[P_U\left(U \le \frac{f(x)}{M\cdot g(x)}\right)\right]$$ by LTP

$\int_{-\infty}^{\infty} f(x)\, dx = 1$

Unif(0,1) distribution

f is a pdf

$$= \mathbb{E}_{x\sim Y}\left[\frac{f(x)}{M\cdot g(x)}\right] = \int_{-\infty}^{\infty} \frac{f(x)}{M\cdot \cancel{g(x)}} \cancel{g(x)}\, dx = \int_{-\infty}^{\infty} \frac{f(x)}{M} dx = \frac{1}{M}$$

$\mathbb{E}[\#\text{ rounds}] = M$

Rejection Sampling – Examples of candidate distributions

- On a bounded domain
  - $\mathrm{Unif}(a, b)$ as scaled and shifted version of $\mathrm{Unif}(0,1)$
- On non-negative reals
  - e.g. Y distributed as continuous extension of geometric distribution
    - $\mathrm{Geometric}(p) - \mathrm{Unif}(0,1)$

Agenda

- Sampling basics
- Rejection sampling
- Reservoir sampling

Reservoir Sampling Algorithms

Goal: Choose a uniform random sample S of k
items from a stream $x_1, x_2, x_3, \dots, x_n$
where n is not known in advance

Not enough space to store stream: Only k elements plus $O(1)$

Easy if n is known in advance but we usually don't know it.

Useful for many applications

Basic Reservoir Sampling Algorithm

Goal: At each step $i \geq k$ array S is a uniformly random sample of $x_1, x_2, x_3, \dots, x_i$

Initialize: $S[1], \dots, S[k] \leftarrow x_1, \dots, x_k$

Step i: Choose j uniformly from $\{1, \dots, i\}$
If $j \leq k$ then $S[j] \leftarrow x_i$

Why is this correct?

Basic Reservoir Sampling Algorithm

Initialize: $S[1], \dots, S[k] \leftarrow x_1, \dots, x_k$

Step i: Choose j uniformly from $\{1, \dots, i\}$

If $j \le k$ then $S[j] \leftarrow x_i$

Write for $S^{(i)}$ for S after step i

Claim: For each $i \ge k$ and every $\ell \le i$, $P\left(x_\ell \in S^{(i)}\right) = k/i$

Proof: Base case ($i = k$): trivial

IH: Assume for every $\ell \le i - 1$, $P\left(x_\ell \in S^{(i-1)}\right) = k/(i-1)$.

IS: Case $\ell = i$: By definition $P\left(x_\ell \in S^{(i)}\right) = P(j \le k) = k/i$.

Case $\ell < i$: $P\left(x_\ell \in S^{(i)}\right) = P\left(x_\ell \in S^{(i-1)},\ j \ne \ell\right)$

IH

$$= \underbrace{P\left(x_\ell \in S^{(i-1)}\right)}_{k/(i-1) \text{ by IH}} P(j \ne \ell) = \frac{k}{\cancel{i-1}} \cdot \frac{\cancel{i-1}}{i} = k/i \quad \blacksquare$$

Reservoir Sampling

Only drawback of basic algorithm:

- Need to call `random()` n times, one per element of the stream
- Each call is expensive

It turns out that we can do it with only 3 calls per update to S!

Reservoir Sampling: Towards a more clever algorithm

4th idea: We only need to replace a random elt of $S^{(i-1)}$

- **1st idea:** an alternative algorithm:
  - For each i independently, choose a $u_i \sim \text{Unif}(0,1)$.
  - At each step, keep the x_j with the k smallest u_j values

This seems worse; how can it help!

- **2nd idea:** Let $w =$ largest u_j for $j \in S^{(i-1)}$
  - $x_i \in S^{(i)}$ iff $u_i \leq w$

Equivalent to $u_i < w$.

- **3rd idea:** Conditioned on "$w =$ largest u_j for $j \in S^{(i-1)}$ and $u_i \leq w$"
  - The u_j for $j \in S^{(i)}$ are independent samples from $[0, w]$

Reservoir Sampling: Towards a more clever algorithm

4th idea: We only need to replace a random elt of $S^{(i-1)}$

Modified Algorithm:

- Initialize $S = \{x_1, \dots, x_k\}$; set $w = \max\{u_1, \dots u_k\}$ for $u_j \sim \mathrm{Unif}(0,1)$
- Step i: Choose $u_i \sim \mathrm{Unif}(0,1)$
  if $u_i \leq w$ then replace a random elt of S with x_i
  set $w = \max\{u_1', \dots u_k'\}$ for $u_j' \sim \mathrm{Unif}(0, w)$

- **2nd idea:** Let $w =$ largest u_j for $j \in S^{(i-1)}$
  - $x_i \in S^{(i)}$ iff $u_i \leq w$

- **3rd idea:** Conditioned on "$w =$ largest u_j for $j \in S^{(i-1)}$ and $u_i \leq w$"
  - The u_j for $j \in S^{(i)}$ are independent samples from $[0, w]$

Better Reservoir Sampling: Two final ideas

Modified Algorithm:

- Initialize $S = \{x_1, \ldots, x_k\}$; set $w = \max\{u_1, \ldots u_k\}$ for $u_j \sim \mathrm{Unif}(0,1)$
- Step i: Choose $u_i \sim \mathrm{Unif}(0,1)$

  if $u_i \leq w$ then replace a random elt of S with x_i

  set $w = \max\{u_1{}', \ldots u_k{}'\}$ for $u_j' \sim \mathrm{Unif}(0, w)$

For fixed w, the indicator r.v. that Step i involves replacement is $\mathrm{Bernoulli}(w)$ so distribution of # of steps until next replacement is $\mathrm{Geometric}(w)$.

$\max\{u_1', \ldots, u_k'\}$ for $u_j' \in \mathrm{Unif}(0, w)$ is distributed as $w \cdot u^{1/k}$ for $u \sim \mathrm{Unif}(0,1)$

Better Reservoir Sampling: Final Algorithm

Optimal Reservoir Sampling:

- Initialize $S = \{x_1, \ldots, x_k\}$; set $i = k$; $w = u^{1/k}$ for $u \sim \mathrm{Unif}(0,1)$
- Loop
  - Choose $j \sim \mathrm{Geometric}(w)$ using single call to **`random()`**
  - Set $i = i + j$ skipping any elements in between
  - Replace a random elt of S with x_i
  - Set $w = w \cdot u^{1/k}$ for $u \sim \mathrm{Unif}(0,1)$

This is typical of randomized algorithms

- Ideas for good algorithms touch on many different topics we've covered in CSE 312

- Even very good algorithms can be improved with more insight

- This isn't even all there is to reservoir sampling …
  - Non-uniform sampling based on weights for elements