Bloom Filters

Anna Karlin
Most slides by Shreya Jayaraman, Luxi Wang, Alex Tsun

Bloom Filters: Motivation

- Large universe of possible data items.
- Hash table is stored on disk or in network, so any lookup is expensive.
- Many (if not most) of the lookups return “Not found”.

Altogether, this is bad. You’re wasting **a lot of time and space** doing lookups for items that aren’t even present.

Examples:

- Google Chrome: wants to warn you if you’re trying to access a malicious URL. Keep hash table of malicious URLs.
- Network routers: want to track source IP addresses of certain packets, .e.g., blocked IP addresses.

Bloom Filters: Motivation

- Probabilistic data structure.
- Close cousins of hash tables.
- Ridiculously space efficient
- To get that, make occasional errors, specifically false positives.

Typical implementation: only 8 bits per element!

Bloom Filters

- Stores information about a set of elements.
- Supports two operations:
  1. **add(x)** - adds x to bloom filter
  2. **contains(x)** - returns true if x in bloom filter, otherwise returns false
     a. If return false, **definitely** not in bloom filter.
     b. If return true, **possibly** in the structure (some false positives).

Bloom Filters: Example

bloom filter t with m = 5 that uses k = 3 hash functions

function INITIALIZE(k,m)
    for $i = 1, \ldots, k$: **do**
        t_i = new bit vector of m 0's

Index →	0	1	2	3	4
t_1	0	0	0	0	0
t_2	0	0	0	0	0
t_3	0	0	0	0	0

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

function ADD(x)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

add("thisisavirus.com")

h_1("thisisavirus.com") → 2
h_2("thisisavirus.com") → 1
h_3("thisisavirus.com") → 4

Index →	0	1	2	3	4
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("thisisavirus.com")

h_1("thisisavirus.com") → 2

h_2("thisisavirus.com") → 1

h_3("thisisavirus.com") → 4

function CONTAINS(x)
  return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True True True

Index →	0	1	2	3	4
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("thisisavirus.com")

h_1("thisisavirus.com") → 2

h_2("thisisavirus.com") → 1

h_3("thisisavirus.com") → 4

function CONTAINS(x)
 return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True True True

Since all conditions satisfied, returns True (correctly)

Index →	0	1	2	3	4
t_1	0	0	1	0	0
t_2	0	1	0	0	0
t_3	0	0	0	0	1

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("verynormalsite.com")

h_1("verynormalsite.com") → 2

h_2("verynormalsite.com") → 0

h_3("verynormalsite.com") → 4

function CONTAINS(x)
return $t_1[h_1(x)] == 1 \land t_2[h_2(x)] == 1 \land \cdots \land t_k[h_k(x)] == 1$

True True True

Since all conditions satisfied, returns True (incorrectly)

Index →	0	1	2	3	4
t_1	0	1	**1**	0	0
t_2	**1**	1	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Summary

- An empty bloom filter is an empty k x m bit array with all values initialized to zeros
  - k = number of hash functions
  - m = size of each array in the bloom filter
- add(x) runs in O(k) time
- contains(x) runs in O(k) time
- requires O(km) space (in bits!)
- Probability of false positives from collisions can be reduced by increasing the size of the bloom filter

Bloom Filters: Application

- Google Chrome has a database of malicious URLs, but it takes a long time to query.
- Want an in-browser structure, so needs to be efficient and be space-efficient
- Want it so that can check if a URL is in structure:
  - If return False, then definitely not in the structure (don’t need to do expensive database lookup, website is safe)
  - If return True, the URL may or may not be in the structure. Have to perform expensive lookup in this rare case.

False positive probability

Comparison with Hash tables - Space

- Google storing 5 million URLs, each URL 40 bytes.
- Bloom filter with k=8 and m = 10,000,000.

Hash Table

Bloom Filter

Comparison with Hash tables - Time

- Say avg user visits 100,000 URLs in a year, of which 2,000 are malicious.
- 0.5 seconds to do lookup in the database, 1ms for lookup in Bloom filter.
- Suppose the false positive rate is 2%

Hash Table

Bloom Filter

Bloom Filters: Many Applications

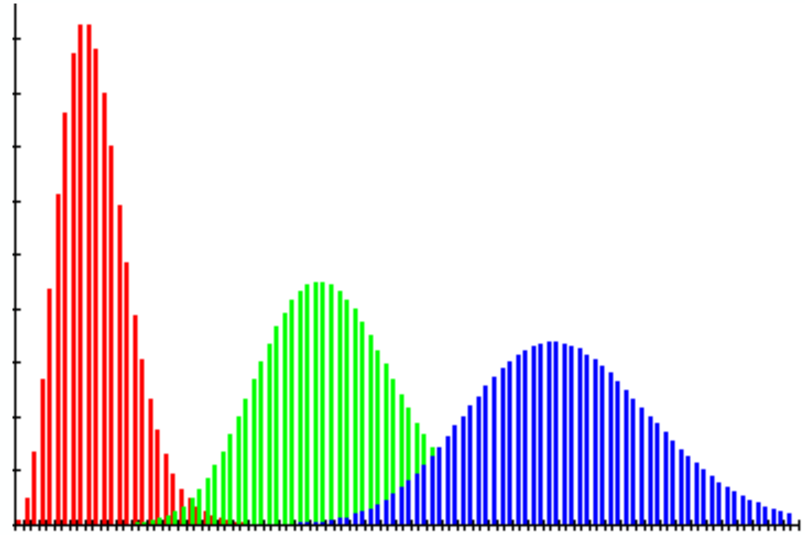
- Any scenario where space and efficiency are important.
- Used a lot in networking
- In distributed systems when want to check consistency of data across different locations, might send a Bloom filter rather than the full set of data being stored.
- Google BigTable uses Bloom filters to reduce the disk lookups for non-existent rows and columns
- Internet routers often use Bloom filters to track blocked IP addresses.
- And on and on…

Bloom Filters typical example…

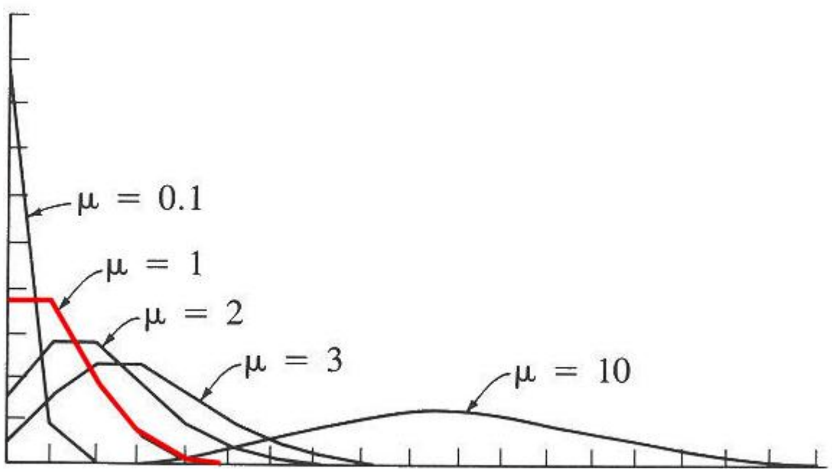
of randomized algorithms and randomized data structures.

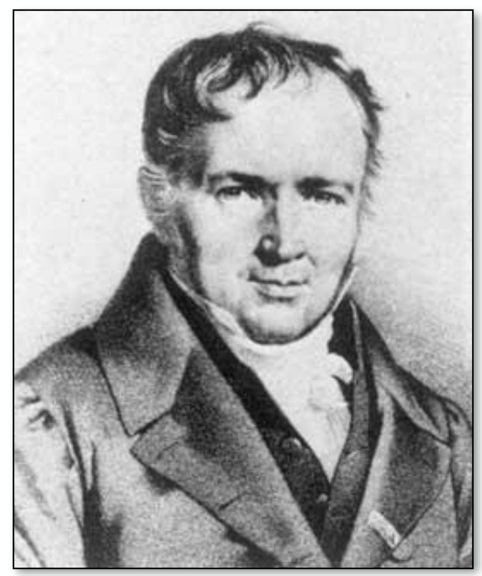
- Simple
- Fast
- Efficient
- Elegant
- Useful!

- You’ll be implementing Bloom filters on pset 4. Enjoy!



a zoo of (discrete) random variables





A discrete random variable X equally likely to take any (integer) value between integers a and b, inclusive, is *uniform*.

Notation:

Probability mass function:

Mean:

Variance:

A discrete random variable *X* equally likely to take any (integer) value between integers *a* and *b*, inclusive, is *uniform*.

Notation: $X \sim \text{Unif}(a,b)$

Probability: $$P(X = i) = \frac{1}{b - a + 1}$$

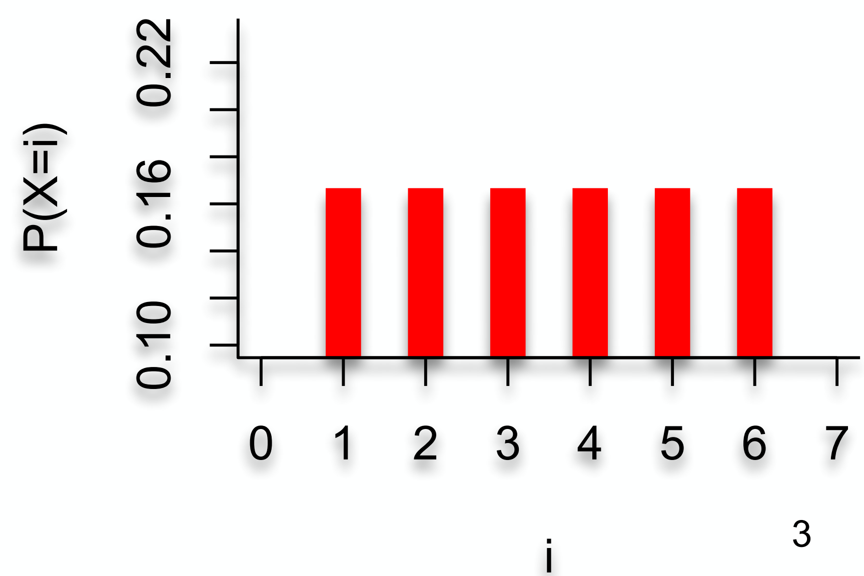
Mean, Variance: $$E[X] = \frac{a+b}{2}, \text{Var}[X] = \frac{(b-a)(b-a+2)}{12}$$

Example: value shown on one roll of a fair die is Unif(1,6):

$P(X=i) = 1/6$

$E[X] = 7/2$

$\text{Var}[X] = 35/12$



An experiment results in "Success" or "Failure"

X is an *indicator random variable* (1 = success, 0 = failure)

$P(X=1) = p$ and $P(X=0) = 1-p$

X is called a *Bernoulli* random variable: $X \sim Ber(p)$

Mean:

Variance:

Bernoulli random variables

An experiment results in "Success" or "Failure"

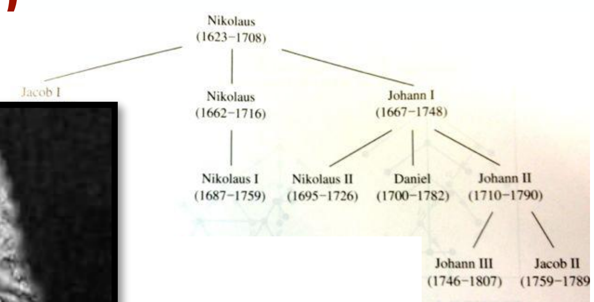
X is an *indicator random variable* (1 = success, 0 = failure)

$P(X=1) = p$ and $P(X=0) = 1-p$

X is called a *Bernoulli* random variable: $X \sim Ber(p)$

$E[X] = E[X^2] = p$

$Var(X) = E[X^2] – (E[X])^2 = p – p^2 = p(1-p)$



Examples:

coin flip

random binary digit

whether a disk drive crashed

Jacob (aka James, Jacques)
Bernoulli, 1654 – 1705

Consider n independent random variables $Y_i \sim Ber(p)$

$X = \Sigma_i Y_i$ is the number of successes in n trials

X is a *Binomial* random variable: $X \sim Bin(n,p)$

Examples

# of heads in n coin flips

# of 1's in a randomly generated length n bit string

# of disk drive crashes in a 1000 computer cluster

bit errors in file written to disk

of typos in a book

# of elements in particular bucket of large hash table

# of server crashes per day in giant data center

Consider n independent random variables $Y_i \sim \text{Ber}(p)$

$X = \Sigma_i Y_i$ is the number of successes in n trials

X is a *Binomial* random variable: $X \sim \text{Bin}(n,p)$

Probability mass function:

Mean:

Variance:

mean, variance of the binomial (II)

If $Y_1, Y_2, \ldots, Y_n \sim \text{Ber}(p)$ and independent,

then $X = \sum_{i=1}^{n} Y_i \sim \text{Bin}(n, p)$.

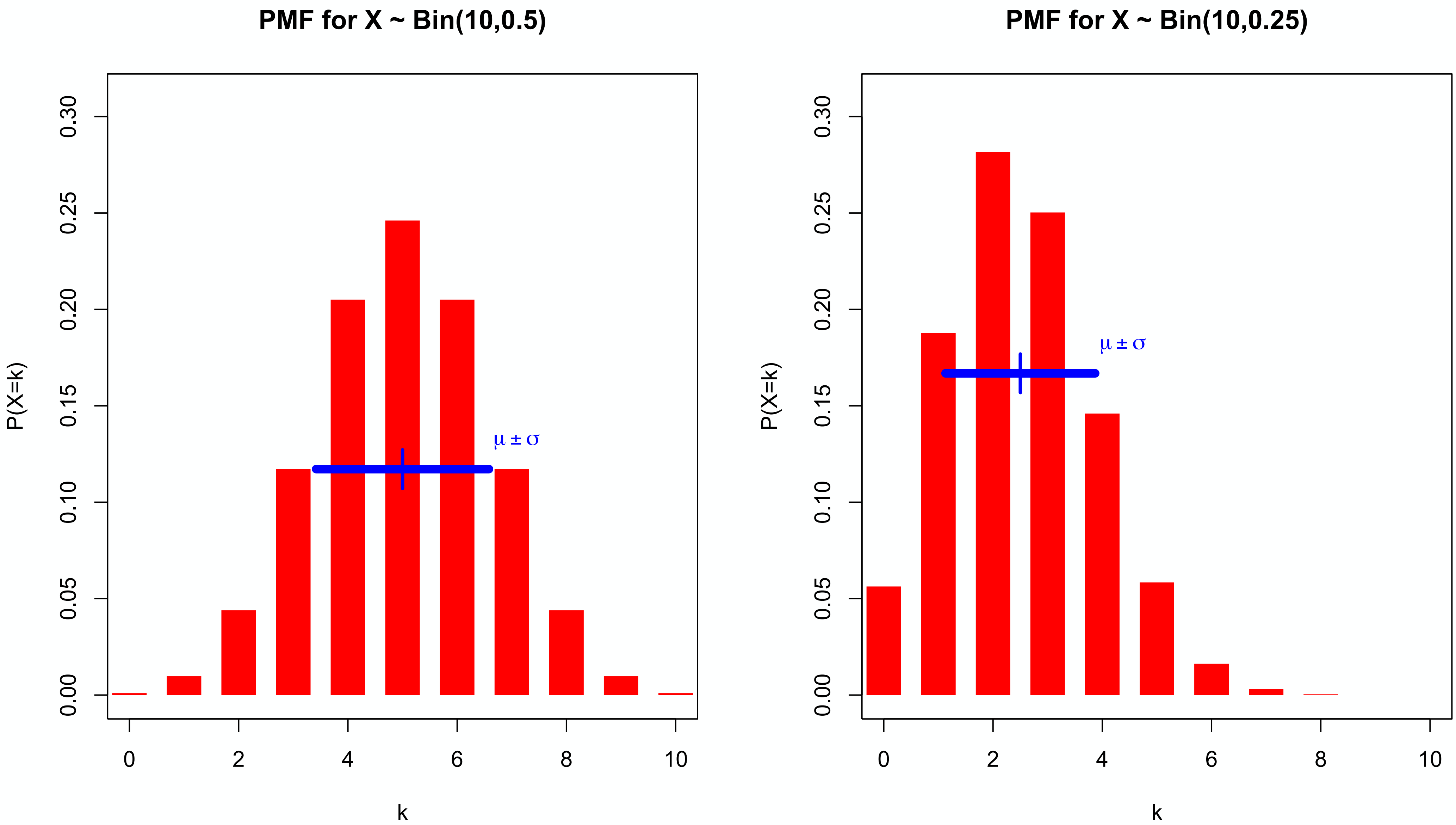
$$E[X] = np$$

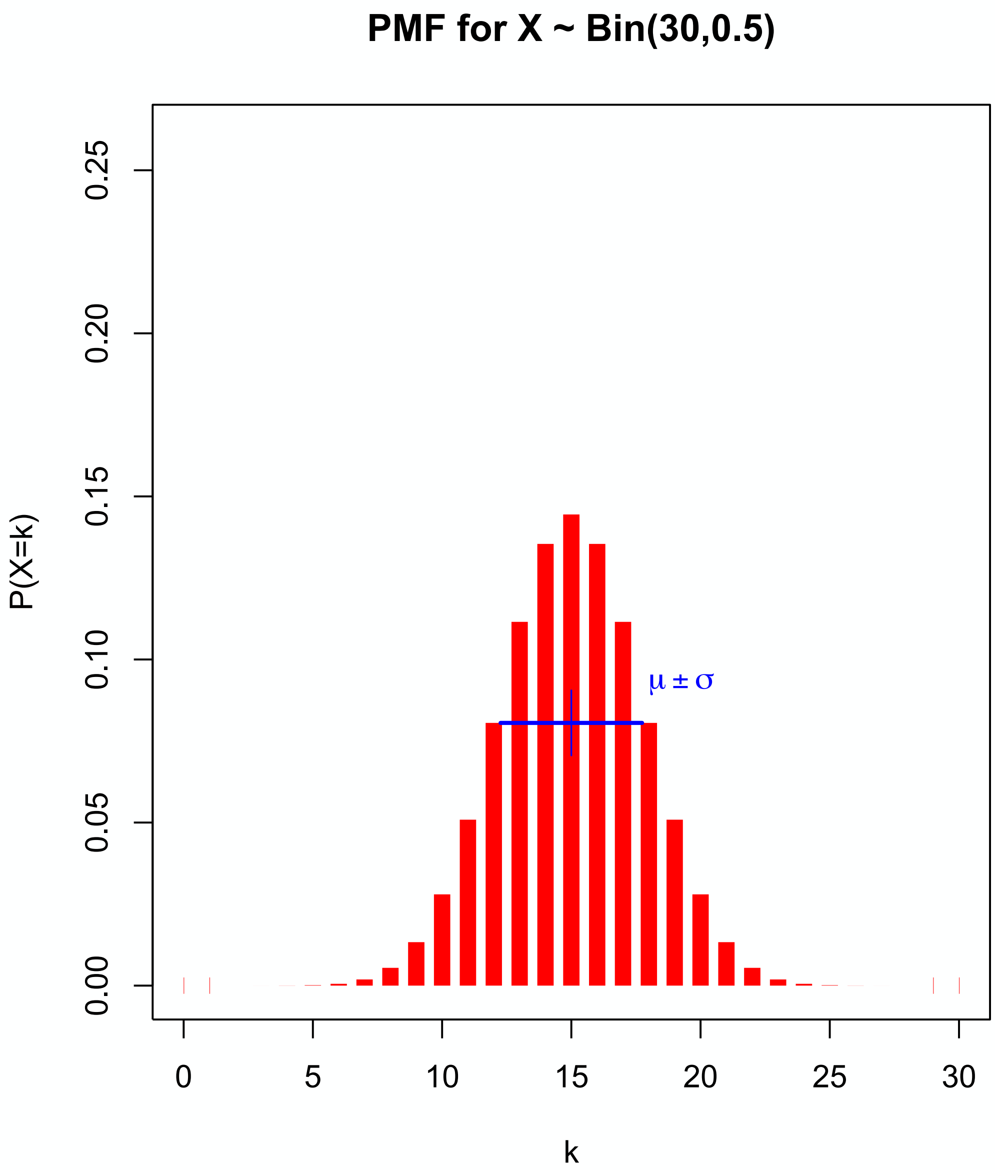
$$E[X] = E\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} E\left[Y_i\right] = nE[Y_1] = np$$

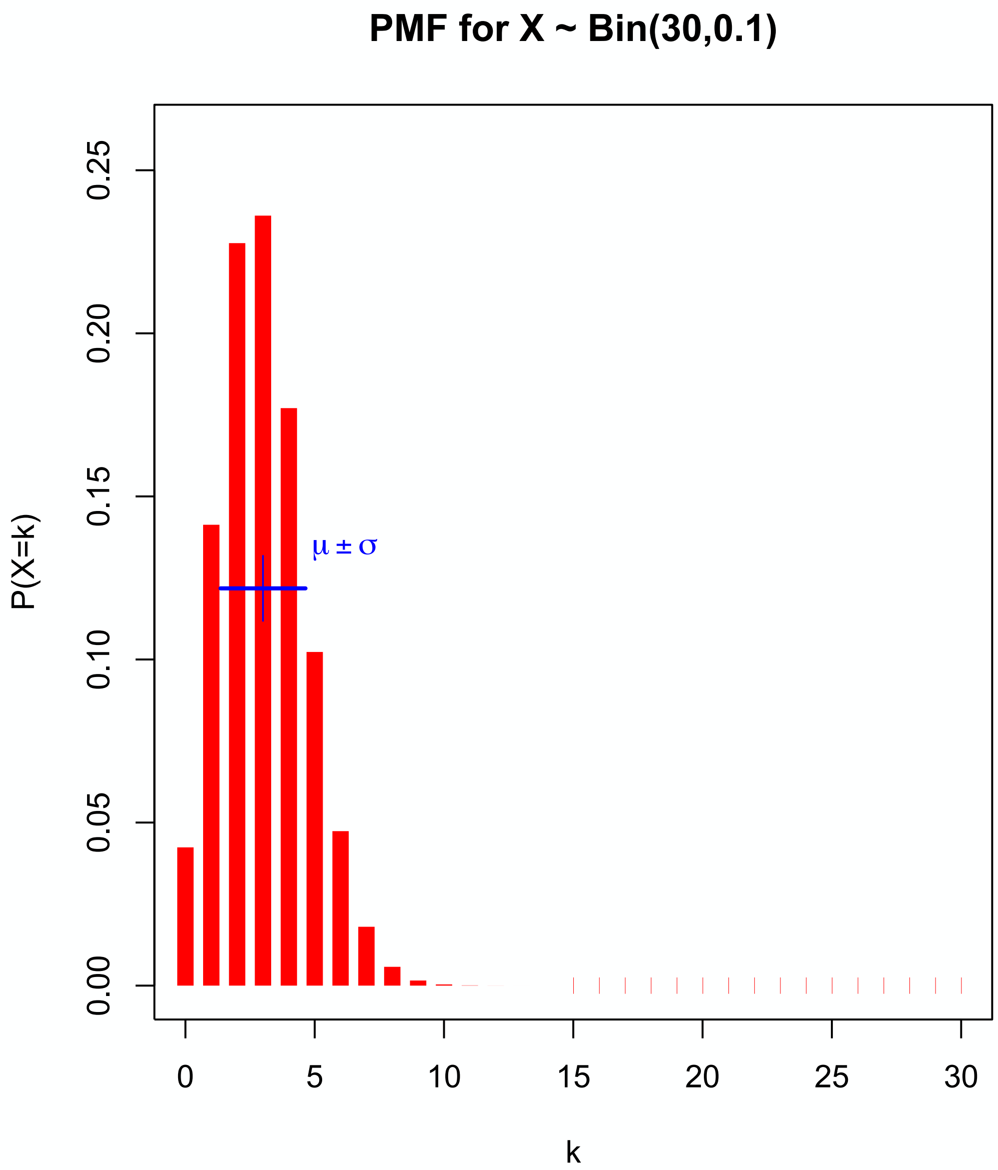
$$\text{Var}[X] = np(1-p)$$

$$\text{Var}[X] = \text{Var}\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} \text{Var}\left[Y_i\right] = n\text{Var}[Y_1] = np(1-p)$$

binomial pmfs




PMF for X ~ Bin(30,0.5)
P(X=k)
0.00
0.05
0.10
0.15
0.20
0.25
μ ± σ
0
5
10
15
20
25
30
k


PMF for X ~ Bin(30,0.1)
P(X=k)
0.00
0.05
0.10
0.15
0.20
0.25
μ ± σ
0
5
10
15
20
25
30
k

Sending a bit string over the network

n = 4 bits sent, each corrupted with probability 0.1

X = # of corrupted bits, X ~ Bin(4, 0.1)

In real networks, large bit strings (length $n \approx 10^4$)

Corruption probability is very small: $p \approx 10^{-6}$

$X \sim \text{Bin}(10^4, 10^{-6})$ is unwieldy to compute

Extreme n and p values arise in many cases

bit errors in file written to disk

of typos in a book

# of elements in particular bucket of large hash table

# of server crashes per day in giant data center

In a series $X_1, X_2, \ldots$ of Bernoulli trials with success probability p, let Y be the index of the first success, i.e.,

$$X_1 = X_2 = \ldots = X_{Y-1} = 0 \;\&\; X_Y = 1$$

Then Y is a *geometric* random variable with parameter p.

Examples:

Number of coin flips until first head

Number of blind guesses on SAT until I get one right

Number of darts thrown until you hit a bullseye

Number of random probes into hash table until empty slot

Number of wild guesses at a password until you hit it

Probability mass function:

Mean:

Variance:

In a series $X_1, X_2, \ldots$ of Bernoulli trials with success probability p, let Y be the index of the first success, i.e.,

$$X_1 = X_2 = \ldots = X_{Y-1} = 0 \ \& \ X_Y = 1$$

Then Y is a *geometric* random variable with parameter p.

Examples:

Number of coin flips until first head

Number of blind guesses on SAT until I get one right

Number of darts thrown until you hit a bullseye

Number of random probes into hash table until empty slot

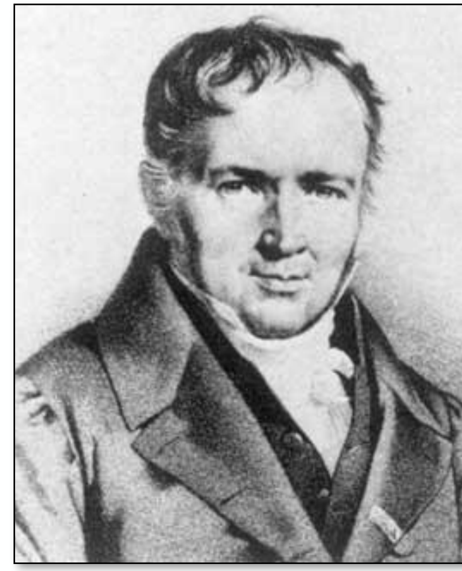
Number of wild guesses at a password until you hit it

$P(Y=k) = (1-p)^{k-1}p;$

Mean $1/p$; Variance $(1-p)/p^2$

Poisson motivation

Poisson random variables



Siméon Poisson, 1781-1840

Suppose "events" happen, independently, at an *average* rate of λ per unit time. Let X be the *actual* number of events happening in a given time unit. Then X is a *Poisson* r.v. *with parameter* λ (denoted X ~ Poi(λ)) and has distribution (PMF):

$$P(X = i) = e^{-\lambda} \frac{\lambda^i}{i!}$$

Examples:

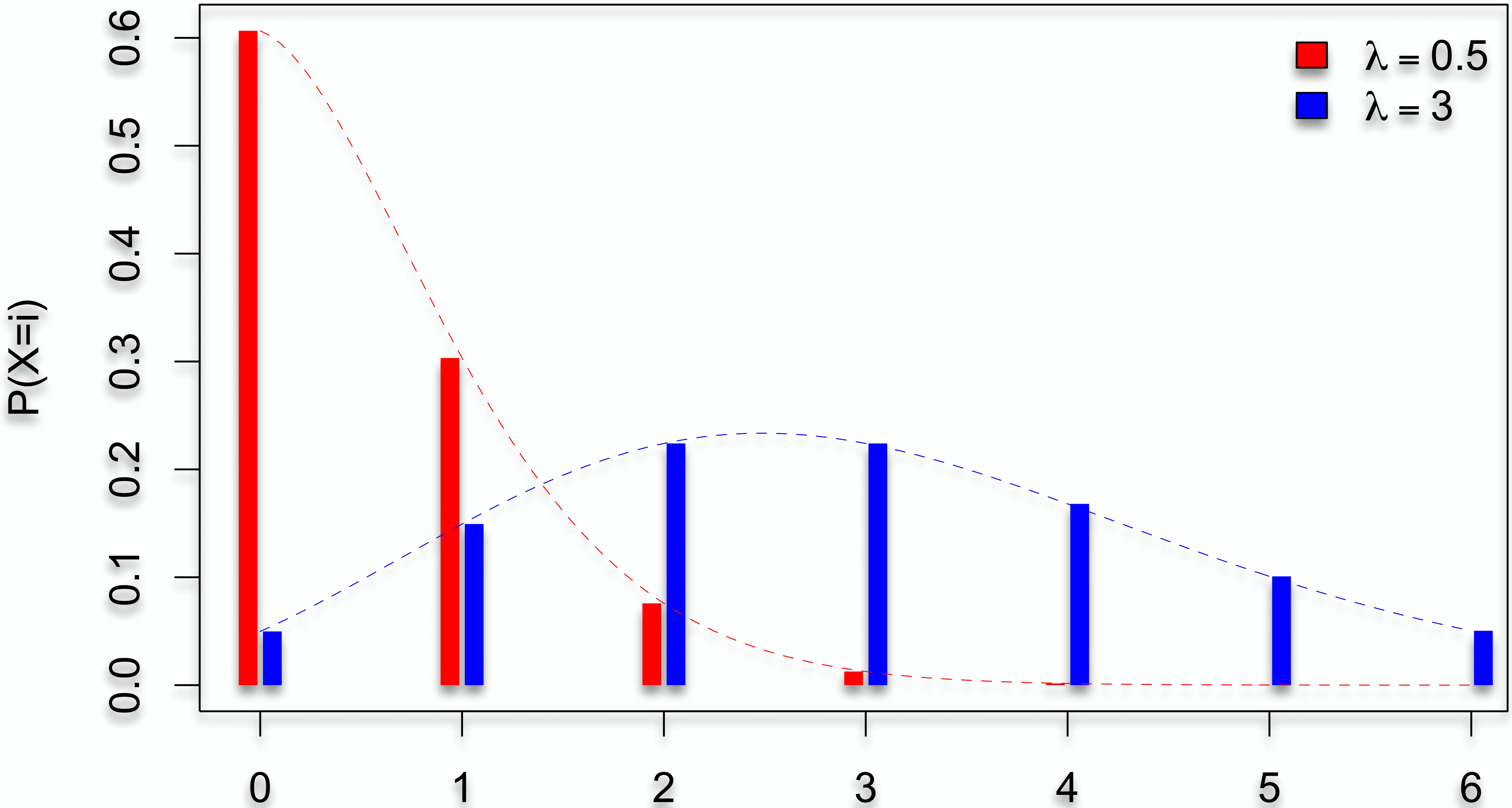
# of alpha particles emitted by a lump of radium in 1 sec.

# of traffic accidents in Seattle in one year

# of babies born in a day at UW Med center

# of visitors to my web page today

$$P(X = i) = e^{-\lambda} \frac{\lambda^i}{i!}$$



X is a Poisson r.v. with parameter λ if it has PMF:

$$P(X = i) = e^{-\lambda} \frac{\lambda^i}{i!}$$

Is it a valid distribution? Recall Taylor series:

$$e^\lambda = \frac{\lambda^0}{0!} + \frac{\lambda^1}{1!} + \cdots = \sum_{0 \le i} \frac{\lambda^i}{i!}$$

So

$$\sum_{0 \le i} P(X = i) = \sum_{0 \le i} e^{-\lambda} \frac{\lambda^i}{i!} = e^{-\lambda} \sum_{0 \le i} \frac{\lambda^i}{i!} = e^{-\lambda} e^{\lambda} = 1$$

expected value of poisson r.v.s

$$\begin{aligned}
E[X] &= \sum_{0 \le i} i \cdot e^{-\lambda} \frac{\lambda^i}{i!} \\
&= \sum_{1 \le i} i \cdot e^{-\lambda} \frac{\lambda^i}{i!} \\
&= \lambda e^{-\lambda} \sum_{1 \le i} \frac{\lambda^{i-1}}{(i-1)!} \\
&= \lambda e^{-\lambda} \sum_{0 \le j} \frac{\lambda^j}{j!} \\
&= \lambda e^{-\lambda} e^{\lambda} \\
&= \lambda
\end{aligned}$$

i = 0 term is zero

j = i-1

As expected, given definition in terms of "average rate λ"

(Var[X] = λ, too; proof similar)

binomial random variable is poisson in the limit

Poisson approximates binomial when n is large, p is small, and $\lambda = np$ is "moderate"

Different interpretations of "moderate," e.g.

$n > 20$ and $p < 0.05$

$n > 100$ and $p < 0.1$

Formally, Binomial is Poisson in the limit as $n \to \infty$ (equivalently, $p \to 0$) while holding $np = \lambda$

X ~ Binomial(n,p)

$$\begin{aligned}
P(X=i) &= \binom{n}{i} p^i (1-p)^{n-i} \\
&= \frac{n!}{i!(n-i)!}\left(\frac{\lambda}{n}\right)^i \left(1-\frac{\lambda}{n}\right)^{n-i}, \text{ where } \lambda = pn \\
&= \frac{n(n-1)\cdots(n-i+1)}{n^i}\, \frac{\lambda^i}{i!}\, \frac{(1-\lambda/n)^n}{(1-\lambda/n)^i} \\
&= \underbrace{\frac{n(n-1)\cdots(n-i+1)}{(n-\lambda)^i}}\, \frac{\lambda^i}{i!}\, \underbrace{(1-\lambda/n)^n} \\
&\approx \quad 1 \quad \cdot \frac{\lambda^i}{i!} \cdot \quad e^{-\lambda}
\end{aligned}$$

I.e., Binomial ≈ Poisson for large n, small p, moderate i, λ.

Handy: Poisson has only 1 parameter–the expected # of successes

Consider sending bit string over a network

Send bit string of length $n = 10^4$

Probability of (independent) bit corruption is $p = 10^{-6}$

$X \sim Poi(\lambda = 10^4 \bullet 10^{-6} = 0.01)$

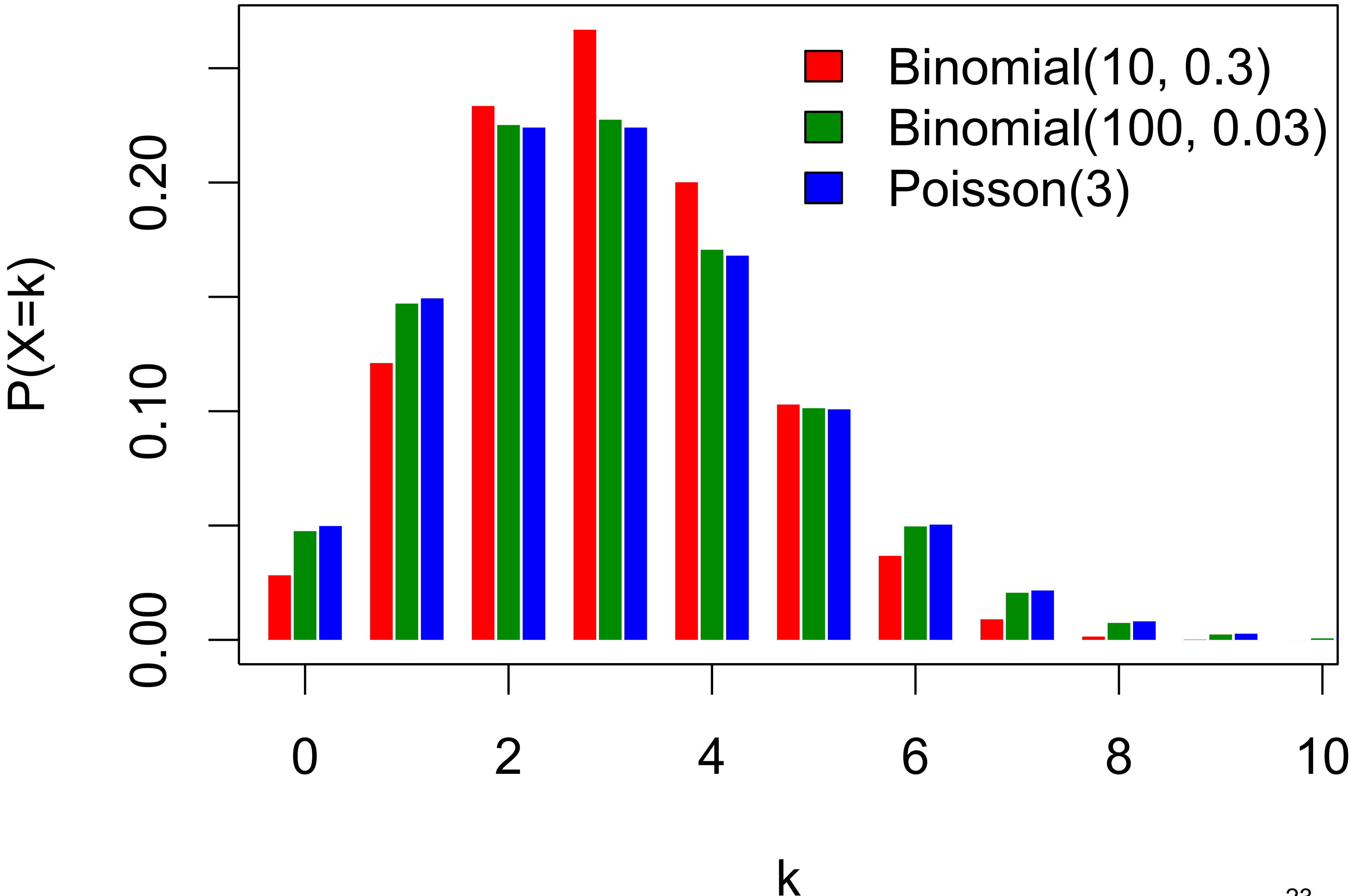
What is probability that message arrives uncorrupted?

$$P(X = 0) = e^{-\lambda} \frac{\lambda^0}{0!} = e^{-0.01} \frac{0.01^0}{0!} \approx 0.990049\underline{834}$$

Using $Y \sim Bin(10^4, 10^{-6})$:

$P(Y=0) \approx 0.9900498\underline{29}$

I.e., Poisson approximation (here) is accurate to ~5 parts per billion


Binomial(10, 0.3)
Binomial(100, 0.03)
Poisson(3)
P(X=k)
0.00
0.10
0.20
0
2
4
6
8
10
k

Recall: if $Y \sim \text{Bin}(n,p)$, then:

$E[Y] = pn$

$\text{Var}[Y] = np(1-p)$

And if $X \sim \text{Poi}(\lambda)$ where $\lambda = np$ ($n \to \infty$, $p \to 0$) then

$E[X] = \lambda = np = E[Y]$

$\text{Var}[X] = \lambda \approx \lambda(1-\lambda/n) = np(1-p) = \text{Var}[Y]$

Important Examples:

Uniform(a,b): $P(X=i)=\frac{1}{b-a+1}$ $\mu=\frac{a+b}{2}, \sigma^2=\frac{(b-a)(b-a+2)}{12}$

Bernoulli(p): $P(X = 1) = p$, $P(X = 0) = 1\text{-}p$ $\mu = p$, $\sigma^2 = p(1\text{-}p)$

Binomial(n,p) $P(X=i)=\binom{n}{i}p^i(1-p)^{n-i}$ $\mu = np$, $\sigma^2 = np(1\text{-}p)$

Poisson(λ): $P(X=i)=e^{-\lambda}\frac{\lambda^i}{i!}$ $\mu = \lambda$, $\sigma^2 = \lambda$

$Bin(n,p) \approx Poi(\lambda)$ where $\lambda = np$ fixed, $n \to \infty$ (and so $p = \lambda/n \to 0$)

Geometric(p) $P(X = k) = (1\text{-}p)^{k\text{-}1}p$ $\mu = 1/p$, $\sigma^2 = (1\text{-}p)/p^2$