# Computer Applications for Engineers

## ET 601

**Asst. Prof. Dr. Prapun Suksompong**

prapun@siit.tu.ac.th

## Random Variables (Con't)

**Office Hours: (BKD 3601-7)**

| | |
|---|---|
| Wednesday | 9:30-11:30 |
| Wednesday | 16:00-17:00 |
| Thursday | 14:40-16:00 |

# Families of Discrete RVs

| $X \sim$ | Support $S_X$ | $p_X(x) =$ |
|---|---|---|
| Uniform $\mathcal{U}(S)$ | $S$ | $\begin{cases} \frac{1}{\lvert S\rvert}, & x \in S, \\ 0, & \text{otherwise.} \end{cases}$ |
| Bernoulli $\mathcal{B}(1,p)$ | $\{0,1\}$ | $\begin{cases} 1-p, & x=0, \\ p, & x=1, \\ 0, & \text{otherwise.} \end{cases}$ |
| Binomial $\mathcal{B}(n,p)$ | $\{0,1,\ldots,n\}$ | $\begin{cases} \binom{n}{x} p^x (1-p)^{n-x}, & x \in \{0,1,2,\ldots,n\}, \\ 0, & \text{otherwise.} \end{cases}$ |
| Geometric $\mathcal{G}_0(p)$ | $\mathbb{N} \cup \{0\}$ | $\begin{cases} p(1-p)^x, & x=0,1,2,\ldots \\ 0, & \text{otherwise.} \end{cases}$ |
| Geometric $\mathcal{G}_1(p)$ | $\mathbb{N}$ | $\begin{cases} p(1-p)^{x-1}, & x=1,2,\ldots \\ 0, & \text{otherwise.} \end{cases}$ |
| Poisson $\mathcal{P}(\alpha)$ | $\mathbb{N} \cup \{0\}$ | $\begin{cases} e^{-\alpha}\frac{\alpha^x}{x!}, & x=0,1,2,\ldots \\ 0, & \text{otherwise} \end{cases}$ |

$p \in (0,1), n \in \mathbb{N}, \alpha > 0$

# Families of Continuous RVs

| $X \sim$ | Support $S_X$ | $f_X(x) =$ |
|---|---|---|
| Uniform $\mathcal{U}(a,b)$ | $(a,b)$ | $\begin{cases} \frac{1}{b-a}, & a < x < b, \\ 0, & \text{otherwise.} \end{cases}$ |
| Normal (Gaussian) $\mathcal{N}(m,\sigma^2)$ | $\mathbb{N}$ | $\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-m}{\sigma}\right)^2}$ |
| Exponential $\mathcal{E}(\lambda)$ | $(0,\infty)$ | $\begin{cases} \lambda e^{-\lambda x}, & x > 0, \\ 0, & x \leq 0 \end{cases}$ |

$$\lambda, \sigma > 0$$

# Bernoulli Trial

- A Bernoulli trial involves performing an experiment once and noting whether a particular event $A$ occurs.
  - The outcome of the Bernoulli trial is said to be
    - a "success" if $A$ occurs and
    - a "failure" otherwise.
  - Success probability $= p$
- We may view the outcome of a single Bernoulli trial as the outcome of a toss of an unfair coin for which the probability of heads (success) is $p = P(A)$ and the probability of tails (failure) is $1-p$.

# Bernoulli Trials

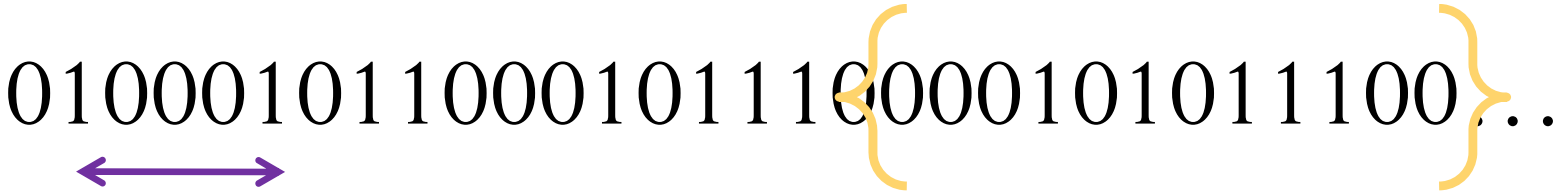
- Repeat a Bernoulli trial multiple times
- Assumptions:
  - The trials are independent. (The outcome from one trial has no effect on the outcome to be obtained from any other trials.)
  - The probability of a success $p$ in each trial is constant.
- An outcome of the complete experiment is a sequence of successes and failures which can be denoted by a **sequence of ones and zeroes**.

# Recall: Sequence of Coin Tosses

- Use 1 to represent Heads; 0 to represent Tails
- `rand(1,120) < 0.5`
- `randi([0 1],1,120)`

```
>> rand(1,120) < 0.5

ans =

  Columns 1 through 13

     1     0     0     0     1     1     1     0     1     0     0     0     0

  Columns 14 through 26

     0     1     0     0     0     0     1     0     1     0     1     0     1

  Columns 27 through 39

     0     1     1     1     0     1     0     1     1     0     1     1     0

  Columns 40 through 52

     1     0     1     0     0     0     0     1     0     1     1     1     1

  Columns 53 through 65

     1     1     0     0     0     1     0     0     1     1     0     1     1

  Columns 66 through 78

     0     1     0     0     1     0     0     1     1     1     1     1     0

  Columns 79 through 91

     0     0     0     1     1     0     1     0     1     1     0     1     1

  Columns 92 through 104

     1     0     0     1     1     0     0     0     1     0     1     0     0

  Columns 105 through 117

     0     0     1     0     1     1     0     1     1     1     1     1     0

  Columns 118 through 120

     0     0     1
```

# Bernoulli Trials




The number of trials until the next 1 is a geometric$_1$ random variable.

The number of 0 until the next 1 is a geometric$_0$ random variable.

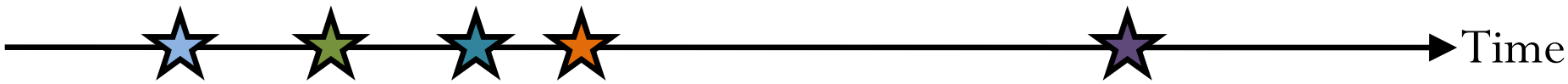
The number of 1s in $n$ trials is a binomial random variable with parameter $(n,p)$

In the limit, as $n \to \infty$ and $p \to 0$ while $np = \alpha$,

The number of 1s is a Poisson random variable with parameter $\alpha = np$.

# Poisson Process

- We start by picturing a Poisson Process as a random arrangement of "marks" (denoted by × or ★) on the time axis.
- These marks usually indicate the arrival times or occurrences of event/phenomenon of interest.
- In the language of "queueing theory," the marks denote arrival times.



# Poisson Process: Examples

- Sequence of times at which lightning strikes occur or mail carriers get bitten within some region
- Emission of particles from a radioactive source
- Occurrence of
  - serious earthquakes
  - traffic accidents
  - power outages

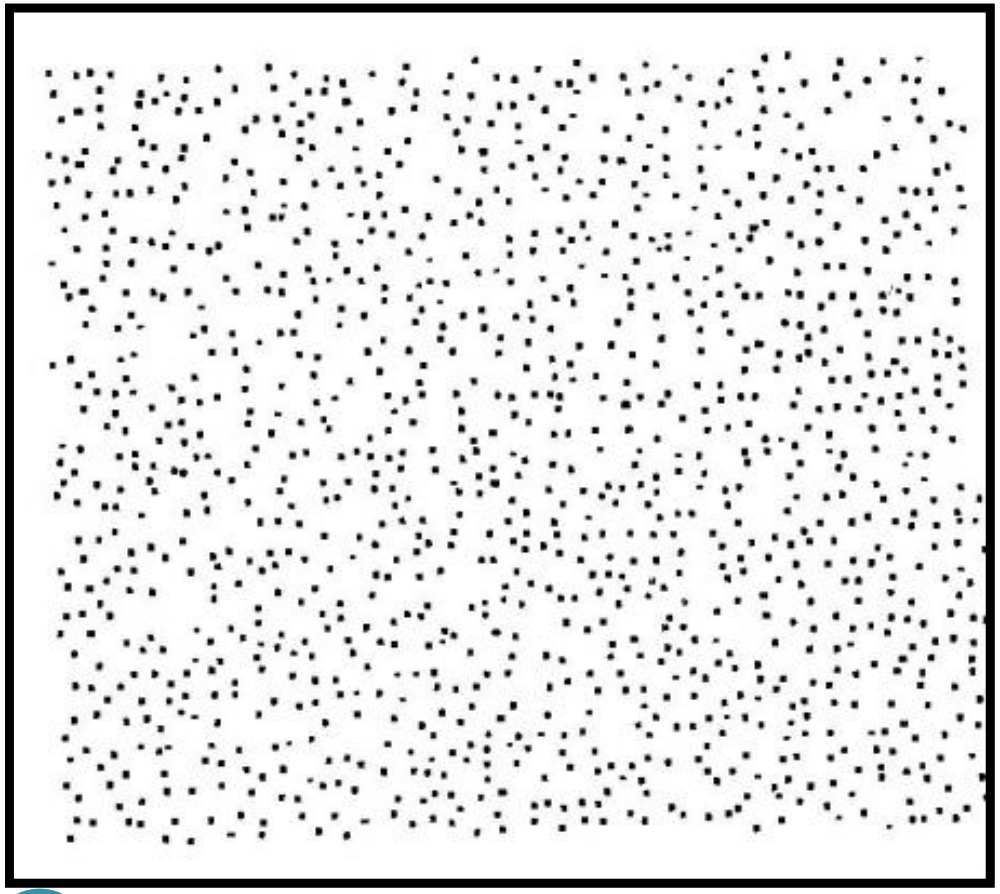
  in a certain area.
- Arrivals of
  - telephone calls at a switchboard or at an automatic phone-switching system
  - urgent calls to an emergency center
  - (filed) claims at an insurance company
  - incoming spikes (action potential) to a neuron in human brain
- Page view requests to a website

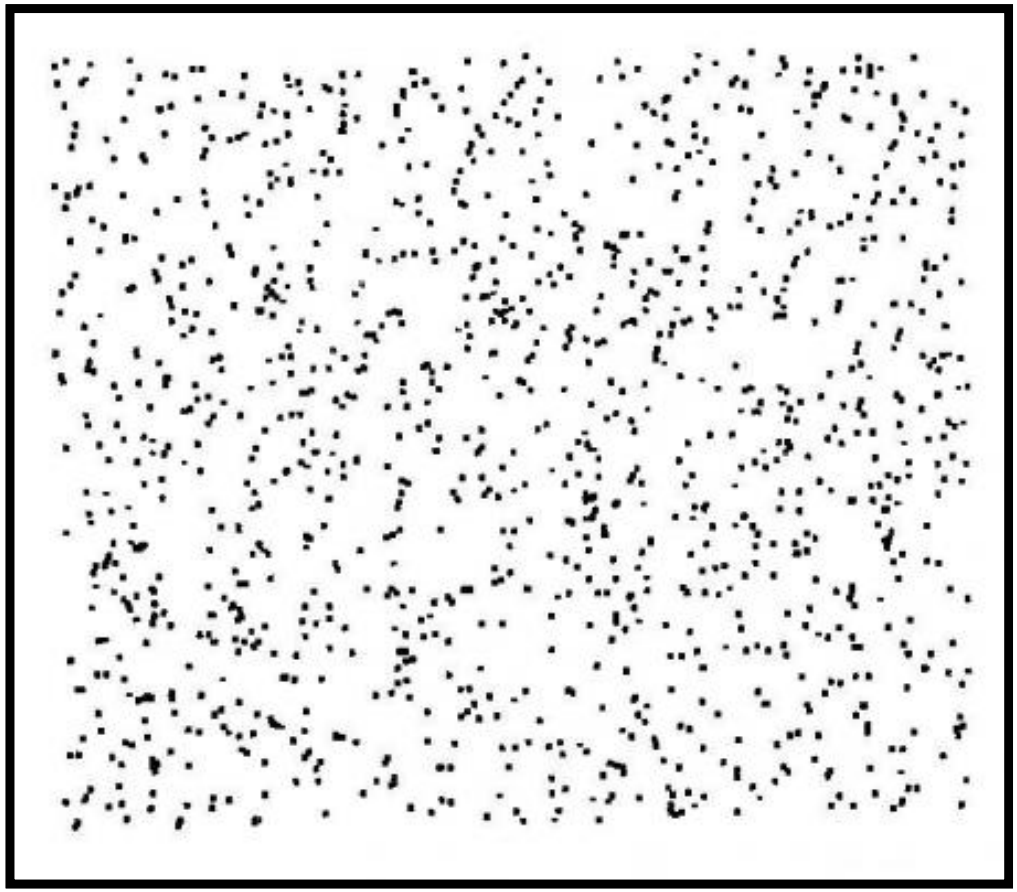
# Homogeneous Poisson Process

- We focus on one kind of Poisson process called homogeneous Poisson process.
  - From now on, when we say "Poisson process", what we mean is "homogeneous Poisson process".
- The first property that you should remember for this process is that there is only one parameter for Poisson process.
  - This parameter is the **rate** or **intensity** of arrivals (the average number of arrivals per unit time.)
  - We use $\lambda$ to denote this parameter.
- How can $\lambda$, which is the only parameter, controls Poisson process?
  - The key idea is that the Poisson process is as random/unstructured as a process can be.

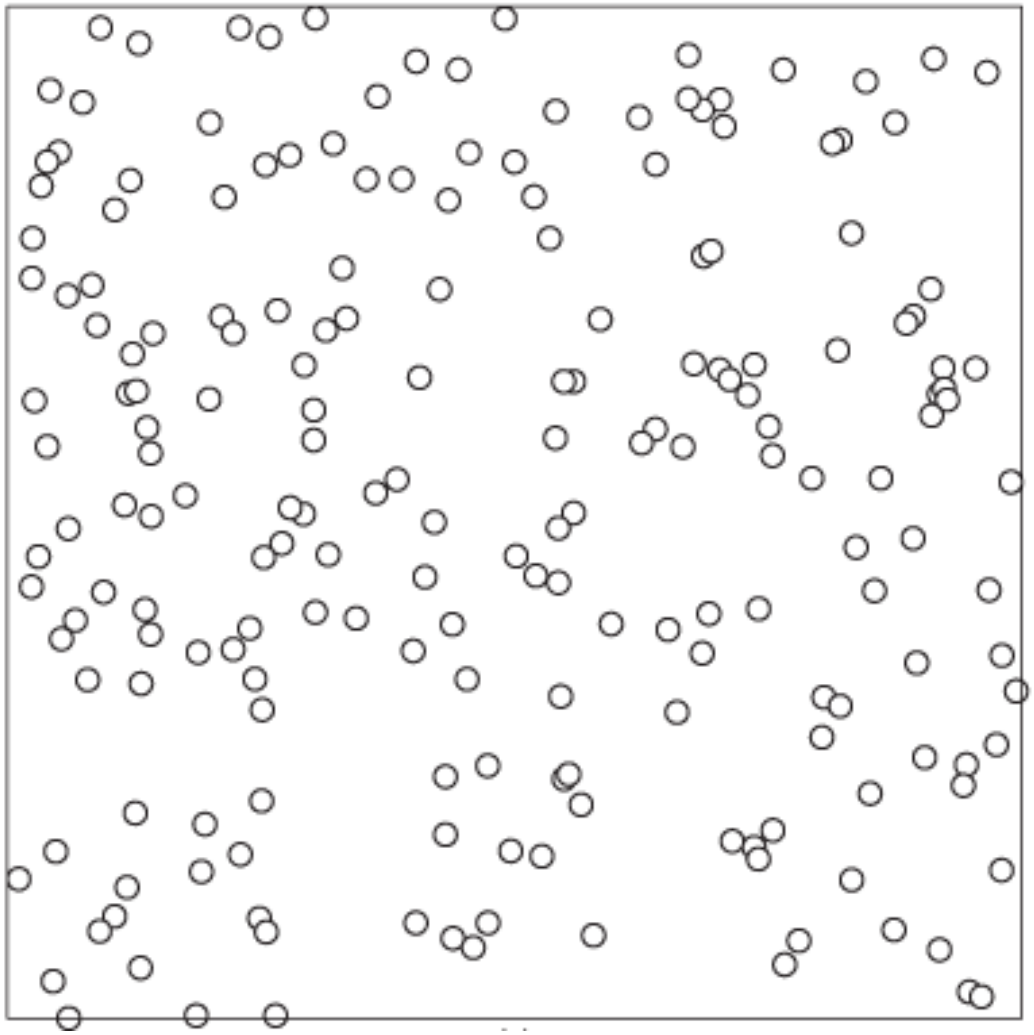
# Poisson Process?

One of these is a realization of a two-dimensional Poisson point process and the other contains correlations between the points. One therefore has a real pattern to it, and one is a realization of a completely unstructured random process.





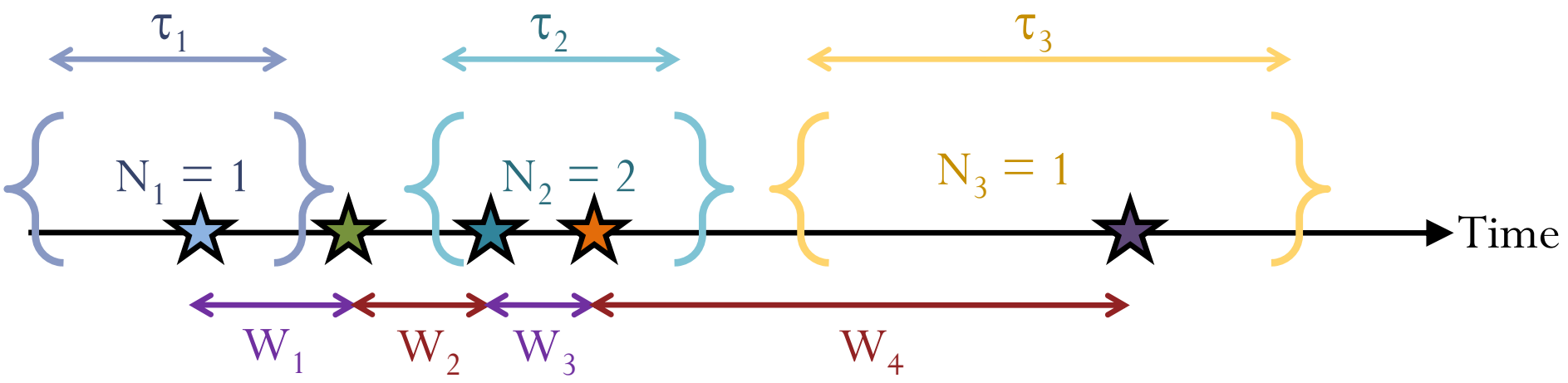
# Poisson Process



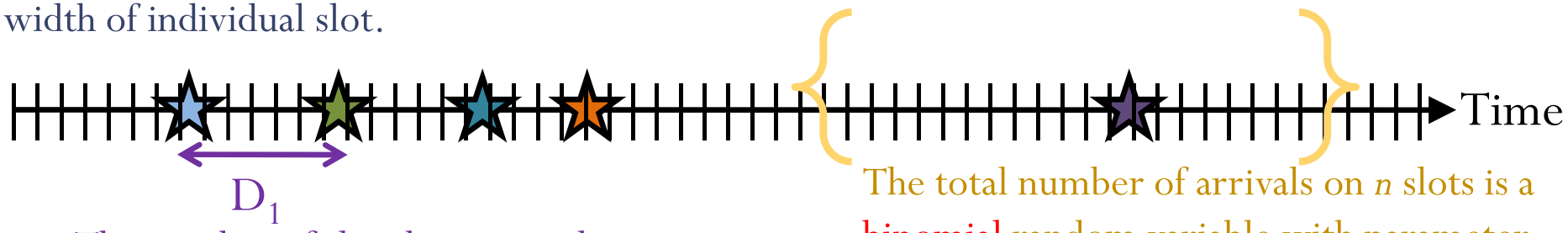
All the structure that is visually apparent is imposed by our own sensory apparatus, which has evolved to be so good at discerning patterns that it finds them when they're not even there!

# Poisson Process: Small Slot Analysis

(discrete time approximation)




In the limit, there is at most one arrival in any slot. The numbers of arrivals on the slots are i.i.d. Bernoulli random variables with probability $p_1$ of exactly one arrivals $= \lambda\delta$ where $\delta$ is the width of individual slot.




The number of slots between adjacent arrivals is a geometric random variable.

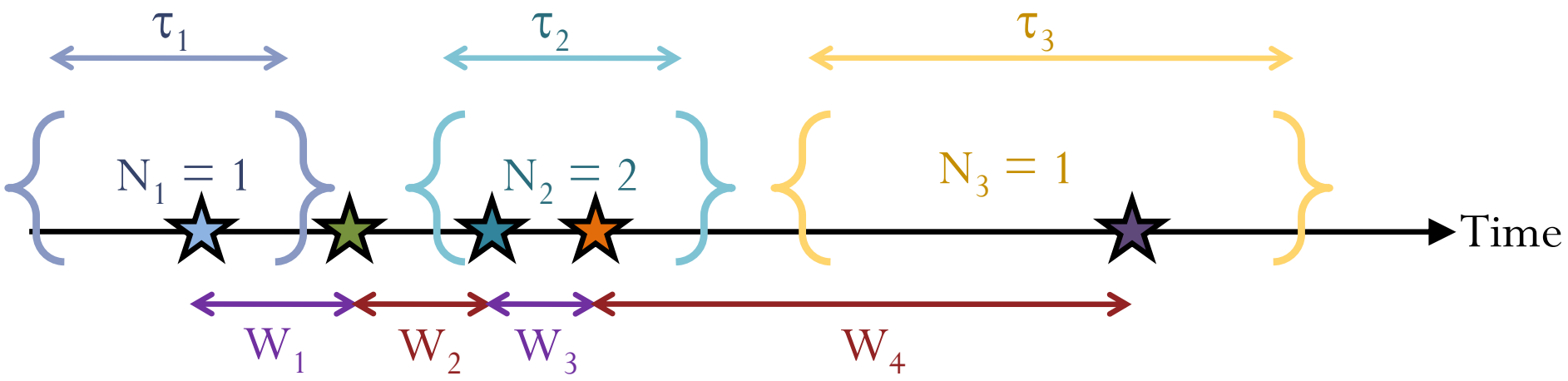
The total number of arrivals on *n* slots is a binomial random variable with parameter $(n,p_1)$

In the limit, as the slot length gets smaller, geometric ⟶ exponential

binomial ⟶ Poisson

# Poisson Process

The number of arrivals $N_1$, $N_2$ and $N_3$ during non-overlapping time intervals are independent Poisson random variables with mean = $\lambda \times$ the length of the corresponding interval.




The lengths of time between adjacent arrivals $W_1, W_2, W_3 \ldots$ are i.i.d. exponential random variables with mean $1/\lambda$.

# Computer Applications for Engineers
## ET 601

**Asst. Prof. Dr. Prapun Suksompong**

prapun@siit.tu.ac.th

## Entropy

**Office Hours: (BKD 3601-7)**

| | |
|---|---|
| Wednesday | 9:30-11:30 |
| Wednesday | 16:00-17:00 |
| Thursday | 14:40-16:00 |

# Entropy

$$H(X) \equiv -\sum_x p_X(x) \log_2 p_X(x) = \mathbb{E}\left[-\log_2 p_X(X)\right]$$

- Quantify/measure
  - amount of randomness (uncertainty, ambiguity) the RV has
  - The number of bits (in average) that are needed to describe a realization of the random variable (provided that optimal compression is used).
- Convention: $0 \log 0 = 0$.
  - Reason: $\lim_{x \to 0} x \log x = 0$

```
>> syms x; limit(x*log(x),x,0)

ans =

0
```

- In MATLAB, first construct a row vector `pX` for the pmf of `X`. Then, find `-pX*((log2(pX))')`.

# Differential Entropy

- The formula discussed earlier is for discrete RV.
- For continuous RV, we consider the **differential entropy**:

$$h(X) = -\mathbb{E}\left[\log_2 f_X(X)\right] = -\int f_X(x)\log_2 f_X(x)dx$$

# Computer Applications for Engineers
## ET 601

**Asst. Prof. Dr. Prapun Suksompong**

prapun@siit.tu.ac.th

## Generating Random Variables

**Office Hours: (BKD 3601-7)**

| | |
|---|---|
| Wednesday | 9:30-11:30 |
| Wednesday | 16:00-17:00 |
| Thursday | 14:40-16:00 |

# An interesting number

- Here is an interesting number:

  0.814723686393179

- This is the first number produced by the MATLAB random number generator with its default settings.
- Start up a fresh MATLAB, set `format long`, type `rand`, and it's the number you get.
  - Verified in MATLAB 2013a

> It may seem perverse to use a computer, that most **precise** and **deterministic** of all machines conceived by the human mind, to produce "random" numbers. More than perverse, it may seem to be a conceptual impossibility. Any program, after all, will produce output that is entirely predictable, hence not truly "random."

[Numerical Recipes, Ch 7]

# Pseudorandom Number

- Random numbers were originally either manually or mechanically generated, by using such techniques as spinning wheels, or dice rolling, or card shuffling.
- The modern approach is to use a **computer** to successively generate **pseudorandom numbers**.
  - Although they are **deterministically generated**, they approximate **independent uniform (0, 1) random variables**.
  - So, "random" numbers in MATLAB are not unpredictable. They are generated by a deterministic algorithm.
    - The algorithm is designed to be sufficiently complicated so that its output appears to be random to someone who does not know the algorithm, and can pass various statistical tests of randomness.
- Our assumption
  - Assume that we have a good pseudorandom number generators.
  - Example: the `rand` command in MATLAB.

# `rng`

- The sequence of numbers produced by `rand` is determined by the internal settings of the uniform random number generator that underlies `rand`, `randi`,and `randn`.
- You can control that shared random number generator using `rng`.
  - This can be useful for controlling the repeatability of your results.
- http://www.mathworks.com/support/2013b/matlab/8.2/demos/controlling-random-number-generation.html

# Multiplicative Congruential Generator (MCG)

- Multiplicative (Linear) Congruential Generator (MCG)
- One of the most common approaches
- Also known as
  - prime modulus multiplicative linear congruential generator (PMMLCG)
  - **Lehmer** generator (because it is invented by Lehmer.)
- Start with the **seed**: $x_0$
- Recursion: $x_n = ax_{n-1} \mod m$
- Normalization: $x_n/m$.
- Multiplier *a* and modulus *m* are some chosen positive integers.
  - *m* should be chosen to be a large prime number.

# Example

- $a = 3, m = 7, x_0 = 1.$

| x | ax | z |
|---|---|---|
| 1 | 3 | 0.1429 |
| 3 | 9 | 0.4286 |
| 2 | 6 | 0.2857 |
| 6 | 18 | 0.8571 |
| 4 | 12 | 0.5714 |
| 5 | 15 | 0.7143 |
| 1 | 3 | 0.1429 |
| 3 | 9 | 0.4286 |
| 2 | 6 | 0.2857 |
| 6 | 18 | 0.8571 |

# Example

- $a = 3, m = 7, x_0 = 2.$

| x | ax | z |
|---|---|---|
| 2 | 6 | 0.2857 |
| 6 | 18 | 0.8571 |
| 4 | 12 | 0.5714 |
| 5 | 15 | 0.7143 |
| 1 | 3 | 0.1429 |
| 3 | 9 | 0.4286 |
| 2 | 6 | 0.2857 |
| 6 | 18 | 0.8571 |
| 4 | 12 | 0.5714 |
| 5 | 15 | 0.7143 |

# "Minimal Standard Generator"

- $m = 2^{31} - 1 = 2147483647$ and $a = 7^5 = 16807$
- Recommended in a 1988 paper by Park and Miller
  - S. K. Park and K. W. Miller, *Random number generators:* ***Good ones are hard to find***, Communications of the ACM, 31 (1988), pp. 1192–1201.
- **Used in early (version 4) implementations of MATLAB**.
  - In 1995, version 5 of MATLAB introduced a completely different kind of random number generator based on the work of George **Marsaglia**.
  - In 2007, version 7.4 of MATLAB uses an algorithm known as the **Mersenne Twister**, developed by M. **Matsumoto** and T. **Nishimura**.

# rng default/shuffle

- Every time you start MATLAB, the generator resets itself to the same state.
- You can reset the generator to the startup state at any time in a MATLAB session (without closing and restarting MATLAB) by
  - `rng('default')`
  - `rng default`
- To avoid repeating the same results when MATLAB restarts:
  - Execute the command
    - `rng('shuffle')`
    - `rng shuffle`
    - It reseeds the generator using a different seed based on the current time.

```
>> rng default
>> rand

ans =

   0.814723686393179

>> rand

ans =

   0.905791937075619

>> rand

ans =

   0.126986816293506

>> rng default
>> rand

ans =

   0.814723686393179
```

# rng: Save and Restore the Generator Settings

```
>> rng default
>> rand

ans =

   0.814723686393179

>> rand

ans =

   0.905791937075619

>> rand

ans =

   0.126986816293506

>> s = rng

s =

     Type: 'twister'
     Seed: 0
    State: [625x1 uint32]
```

The first call to `randi` changed the state of the generator, so the second result is different.

Restore the saved generator settings

Save the current generator settings in `s`

```
>> rand

ans =

   0.913375856139019

>> rand

ans =

   0.632359246225410

>> rng(s)
>> rand

ans =

   0.913375856139019

>> rand

ans =

   0.632359246225410
```

# Generation of arbitrary RVs

- There are many built-in families of RV in MATLAB that can be generated by **`random`**.
- Popular families are provided.
  - Binomial, Exponential, Geometric, Normal, Poisson, Uniform
  - Beta, Birnbaum-Saunders, Burr Type XII, Chi-Square, Extreme Value, F, Gamma, Generalized Extreme Value, Generalized Pareto, Hypergeometric, Inverse Gaussian, Logistic, Loglogistic, Lognormal, Nakagami, Negative Binomial, Noncentral F, Noncentral t, Noncentral Chi-Square, Rayleigh, Rician, Student's t, t Location-Scale, Weibull
- `doc random`
- How to generate RVs whose families are not built-in?

# Bernoulli Trials

- We want to generalize the old technique.
- `rand(1,120) < 0.5`
- `randi([0 1],1,120)`
- To generate a RV $X$ whose pmf is given by

$$p_X(x) = \begin{cases} 1/2, & x = 0, \\ 1/2, & x = 1, \\ 0, & \text{otherwise.} \end{cases}$$

  - First use `rand` create a uniform RV $U$.
  - Then, set $X = \begin{cases} 1, & U < 1/2, \\ 0, & U \geq 1/2. \end{cases}$

- To generate a RV $X$ whose pmf is given by

$$p_X(x) = \begin{cases} p, & x = 1, \\ 1-p, & x = 0, \\ 0, & \text{otherwise.} \end{cases}$$

  - First use `rand` create a uniform RV $U$.
  - Then, set $X = \begin{cases} 1, & U < p, \\ 0, & U \geq p. \end{cases}$

# Example

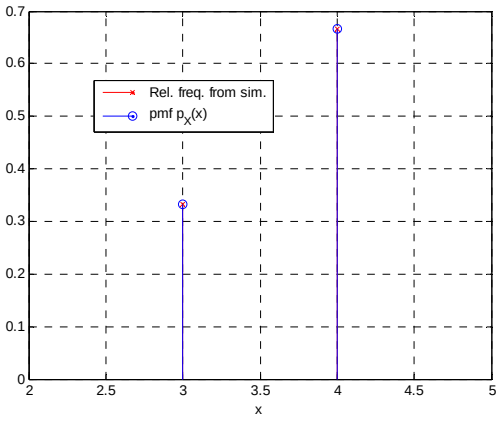
- To generate a RV $X$ whose pmf is given by

$$p_X(x) = \begin{cases} 1/3, & x = 3, \\ 2/3, & x = 4, \\ 0, & \text{otherwise}, \end{cases}$$

- First use `rand` to create a uniform RV $U$.
- Then, set $V = \begin{cases} 1, & U < 2/3, \\ 0, & U \geq 1/3. \end{cases}$
- Finally, map $V \in \{0,1\}$ to $X \in \{3,4\}$.
- Of course, we can combine all the steps into just one line:

```
X = S_X([rand(1,n) < 2/3]+1);
```

```
clear all; close all;

S_X = [3 4];
p_X = [1/3 2/3];

n = 1e6;

U = rand(1,n);
V = [U < 2/3];
X = S_X(V+1);

hist(X)
rf = hist(X,S_X)/n;
stem(S_X,rf,'rx')
hold on
stem(S_X,p_X,'bo')
xlim([min(S_X)-1,max(S_X)+1])
legend('Rel. freq. from sim.'...
    ,'pmf p_X(x)')
xlabel('x')
grid on
```



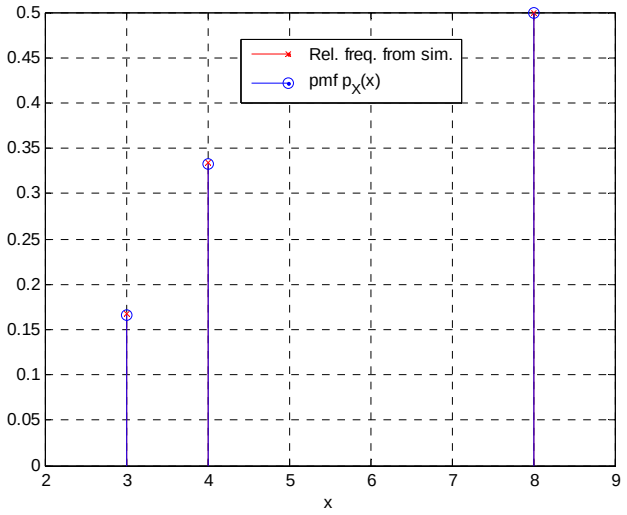
# Example

- To generate a RV $X$ whose pmf is given by

$$p_X(x) = \begin{cases} 1/6, & x = 3, \\ 1/3, & x = 4, \\ 1/2, & x = 8, \\ 0, & \text{otherwise.} \end{cases}$$

- First use `rand` to create a uniform RV $U$.
- Then, set $V = \begin{cases} 1, & 0 \le U < 1/6, \\ 2, & 1/6 \le U < 1/2, \\ 3, & 1/6 \le U < 1, \end{cases}$
- Finally, map $V \in \{1,2,3\}$ to $X \in \{3,4,8\}$.

```
clear all; close all;

S_X = [3 4 8];
p_X = [1/6 1/3 1/2];

n = 1e6;

F_X = cumsum(p_X);
U = rand(1,n);
[dum,V] = histc(U,[0 F_X/F_X(end)]);
X = S_X(V);

hist(X)
rf = hist(X,S_X)/n;
stem(S_X,rf,'rx')
hold on
stem(S_X,p_X,'bo')
xlim([min(S_X)-1,max(S_X)+1])
legend('Rel. freq. from sim.'...
    ,'pmf p_X(x)')
xlabel('x')
grid on
```



# `histc` vs `hist`

- `N = hist(U,centers)`
  - Bins' centers are defined by the vector `centers`.
    - The first bin includes data between -inf and the first center and the last bin includes data between the last bin and inf.
  - `N(k)` count the number of entries of vector `U` whose values falls inside the *k*th bin.
- `N = hist(U,edges)`
  - Bins' edges are defined by the vector `edges`.
  - `N(k)` count the value `U(i)` if `edges(k)` ≤ `U(i)` < `edges(k+1)`.
  - The last (additional) bin will count any values of `U` that match `edges(end)`.
  - Values outside the values in `edges` are not counted.
  - Use `-inf` and `inf` in `edges`.
- `[N,BIN_IND] = histc(U,EDGES)` also returns vector `BIN_IND` indicating the bin index that each entry in `U` sorts into.

# Example: histc

```
>> p_X = [1/6 1/3 1/2];
>> F_X = cumsum(p_X)

F_X =

    0.1667    0.5000    1.0000

>> U = rand(1,5)

U =

    0.2426    0.9179    0.9409    0.1026    0.8897

>> [dum,V] = histc(U,[0 F_X])

dum =

     1     1     3     0


V =

     2     3     3     1     3
```

# Loops

- Loops are MATLAB constructs that permit us to execute a sequence of statements more than once.
- There are two basic forms of loop constructs:
  - **while loops** and
  - **for loops**.
- The major difference between these two types of loops is in how the repetition is controlled.
  - The code in a **while loop** is repeated an indefinite number of times until some user-specified condition is satisfied.
  - By contrast, the code in a **for loop** is repeated a specified number of times, and the number of repetitions is known before the loops starts.

# `while` loop

- General form:

```
while expr
      body
end
```

- The statements in the `body` are repeatedly executed as long as the expression `expr` remains true.

```
clear all; close all;

S_X = [3 4 8];
p_X = [1/6 1/3 1/2];

n = 1e6;

U = rand(1,n);
V = zeros(size(U)); %Preallocation
for k = 1:n
    F = p_X(1);
    m = 1;
    while (U(k) > F)
        m = m+1;
        F = F+p_X(m);
    end
    V(k) = m;
end
X = S_X(V);

hist(X)
rf = hist(X,S_X)/n;
stem(S_X,rf,'rx')
hold on
stem(S_X,p_X,'bo')
xlim([min(S_X)-1,max(S_X)+1])
legend('Rel. freq. from sim.'...
    ,'pmf p_X(x)')
xlabel('x')
grid on
```

# datasample

```
clear all; close all;

S_X = [3 4 8];
p_X = [1/6 1/3 1/2];

n = 1e6;

X = datasample(S_X,n,'Weights',p_X);

hist(X)
rf = hist(X,S_X)/n;
stem(S_X,rf,'rx')
hold on
stem(S_X,p_X,'bo')
xlim([min(S_X)-1,max(S_X)+1])
legend('Rel. freq. from sim.'...
    ,'pmf p_X(x)')
xlabel('x')
grid on
```