Probability and Random Processes

ECS 315

Asst. Prof. Dr. Prapun Suksompong

prapun@siit.tu.ac.th

Working with Randomness using MATLAB

Office Hours:

BKD, 4th floor of Sirindhralai building

Monday	9:30-10:30
Monday	14:00-16:00
Thursday	16:00-17:00

`rand` function: a preview

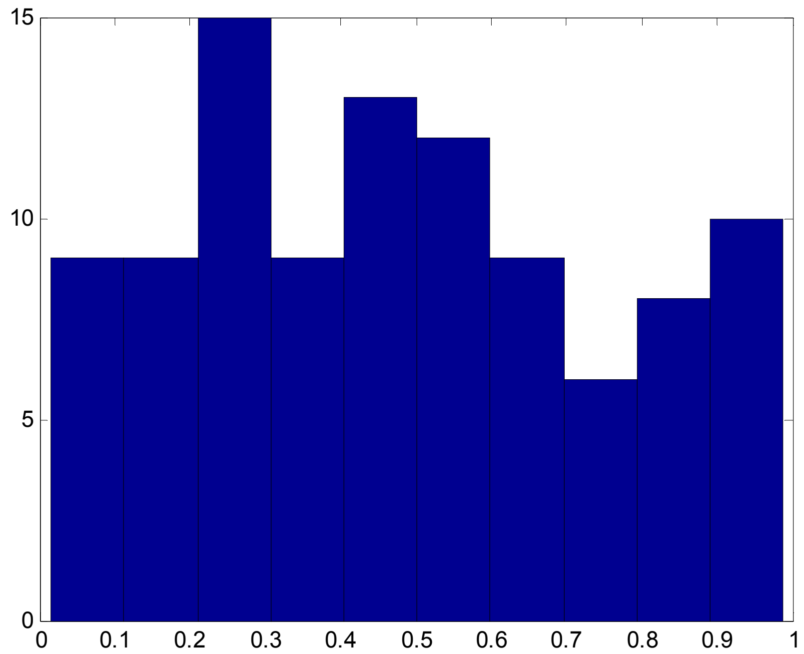
- Generate an array of uniformly distributed pseudorandom numbers.
  - The pseudorandom values are drawn from the **standard uniform distribution** on the open **interval (0,1)**.
- `rand` returns a scalar.
- `rand(m,n)` or `rand([m,n])` returns an *m*-by-*n* matrix.
  - `rand(n)` returns an *n*-by-*n* matrix

```
>> rand
ans =
    0.8147
>> rand(10,2)
ans =
    0.9058    0.9706
    0.1270    0.9572
    0.9134    0.4854
    0.6324    0.8003
    0.0975    0.1419
    0.2785    0.4218
    0.5469    0.9157
    0.9575    0.7922
    0.9649    0.9595
    0.1576    0.6557
```

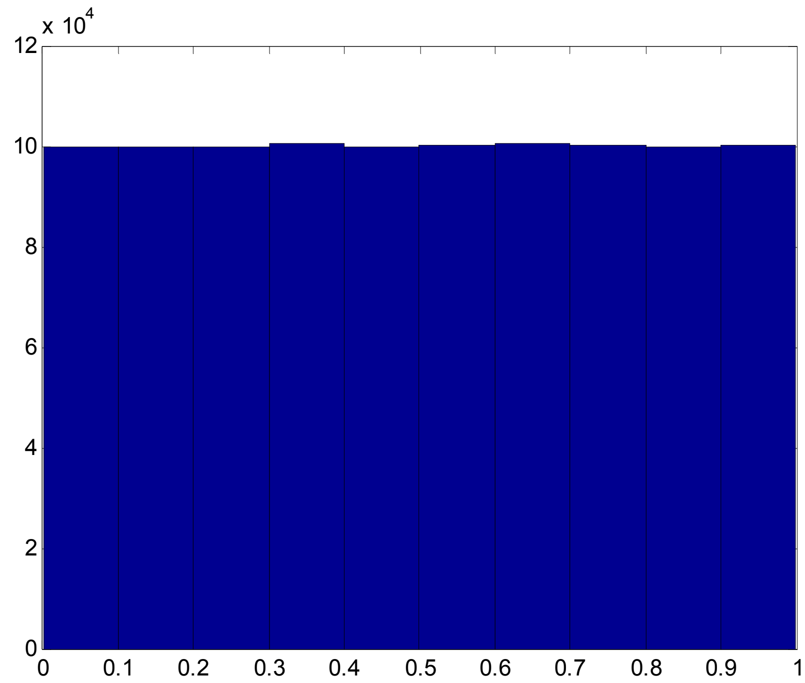
`rand` function: Histogram

- The generation is **unbiased** in the sense that "any number in the range is **as likely to occur** as another number."
- Histogram is flat over the interval (0,1).

`hist(rand(1,100))`



`hist(rand(1,1e6))`

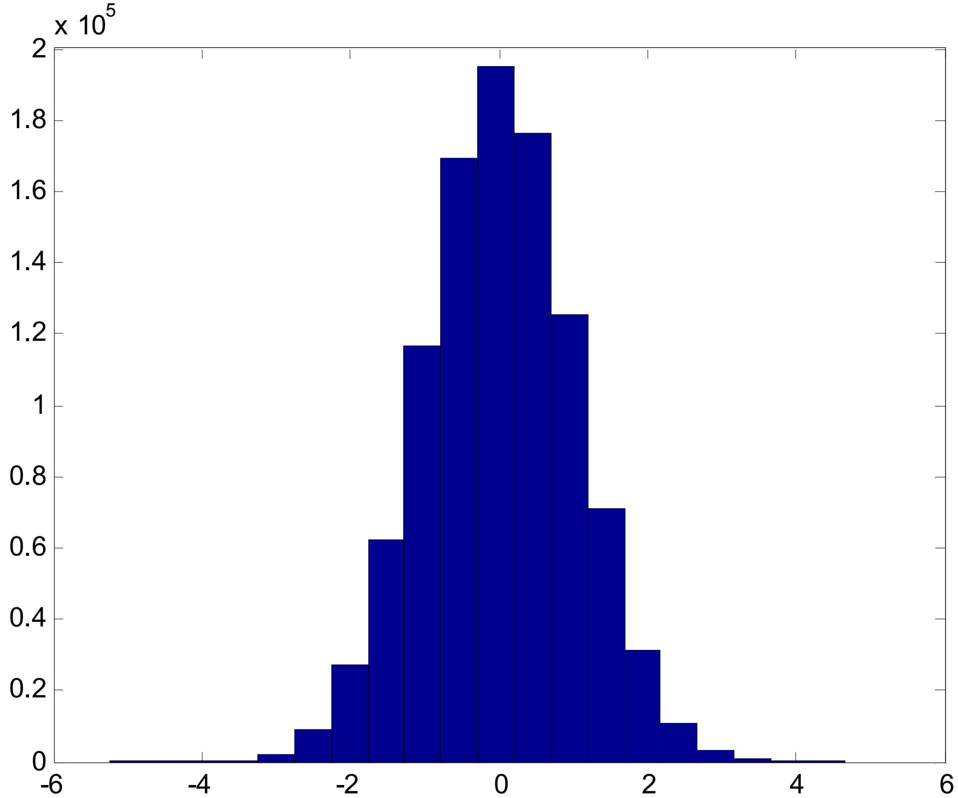


Roughly the same height

`randn` function: a preview

- Generate an array of normally distributed pseudorandom numbers

`hist(randn(1,1e6),20)`



We have already seen the `rand` and `randn` functions.

`randi` function

- Generate uniformly distributed pseudorandom **integers**
- `randi(imax)` returns a scalar value between 1 and `imax`.
- `randi(imax,m,n)` and `randi(imax,[m,n])` return an *m*-by-*n* matrix containing pseudorandom integer values drawn from the discrete uniform distribution on the interval [1,`imax`].
  - `randi(imax)` is the same as `randi(imax,1)`.
- `randi([imin,imax],...)` returns an array containing integer values drawn from the discrete uniform distribution on the interval [`imin`,`imax`].

`randi` function: examples

Coin Tosses:

```
>> randi([0,1])
ans =        T, H
      0
>> randi([0,1],10,2)
ans =
      1       0
      1       0
      1       0
      1       1
      1       1
      0       0
      1       1
      0       0
      1       0
      0       0
```

Dice Rolls

```
>> randi([1,6])
ans =
      5
>> randi([1,6],10,2)
ans =
      5       1
      2       1
      3       3
      3       6
      4       3
      5       4
      5       2
      2       5
      5       2
      4       4
```

`randi` function: examples

Coin Tosses:

```
>> S = ['T','H']
S =
TH
>> S(randi([1,2]))
ans =
H
>> S(randi([1,2],10,2))
ans =
TT
HH
HT
TT
HT
TT
TH
HT
HH
HT
```

An interesting number

- Here is an interesting number:

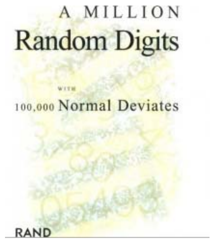
  0.814723686393179

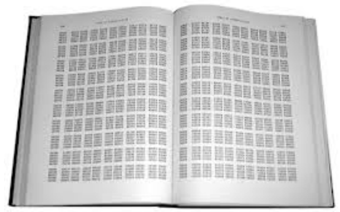
- This is the first number produced by the MATLAB random number generator with its default settings.
- Start up a fresh MATLAB, set `format long`, type `rand`, and it's the number you get.
  - Verified in MATLAB 2013a,b

> 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

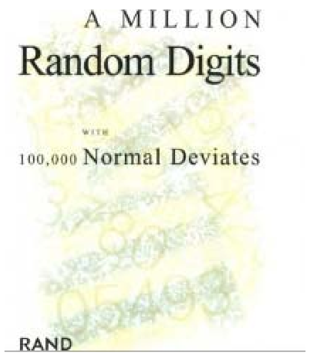


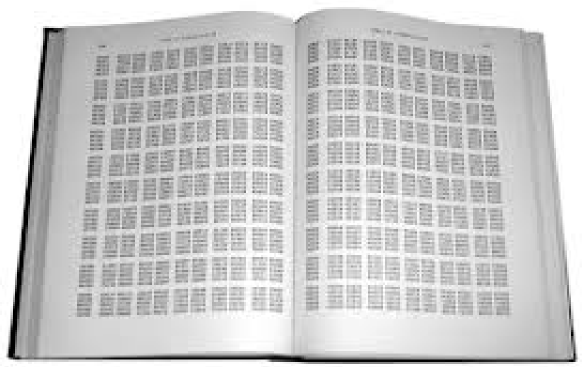


- Random numbers were originally either manually or mechanically generated, by using such techniques as spinning wheels, or dice rolling, or card shuffling.
- The modern approach is to use a **computer** to successively generate **pseudorandom numbers**.
  - Although they are **deterministically generated**, 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.

A Million Random Digits with 100,000 Normal Deviates

- Published in 1955 by the RAND Corporation.
- Production started in 1947 by an electronic simulation of a roulette wheel attached to a computer.
- Became a standard reference.
- In addition to being available in book form, one could also order the digits on a series of **punched cards**.
- The book was reissued in 2001 (ISBN 0-8330-3047-7)
  - It has generated many humorous user reviews on Amazon.com.
- http://www.rand.org/pubs/monograph_reports/MR1418.html





The most helpful favorable review	The most helpful critical review
1,333 of 1,358 people found the following review helpful ★★★★☆ **almost perfect** Such a terrific reference work! But with so many terrific random digits, it's a shame they didn't sort them, to make it easier to find the one you're looking for. Published on October 26, 2006 by a curious reader › See more 5 star, 4 star reviews	416 of 424 people found the following review helpful ★★★☆☆ **Wait for the audiobook version** While the printed version is good, I would have expected the publisher to have an audiobook version as well. A perfect companion for one's Ipod. Published on October 19, 2006 by R. Rosini › See more 3 star, 2 star, 1 star reviews

Vs.

314 of 330 people found the following review helpful

★☆☆☆☆ **Quite the opposite of random when viewed globally**, January 14, 2008

By **D. C. Froemke** (Portland OR) - See all my reviews

REAL NAME

This review is from: A Million Random Digits with 100,000 Normal Deviates (Paperback)

At first, I was overjoyed when I received my copy of this book. However, when an enemy in my department showed me HER copy, I found that they were the OPPOSITE of random - they were IDENTICAL.

It is very frustrating, let alone dangerous for my agents in the field; do not rely on this book for generating codes!

Its list of deviates is very nice for someone in my profession, however.

2,619 of 2,705 people found the following review helpful

★☆☆☆☆ **Sloppy.**, July 27, 2005

By **Brian McGroarty** (United States) - See all my reviews

REAL NAME

This review is from: A Million Random Digits with 100,000 Normal Deviates (Paperback)

The book is a promising reference concept, but the execution is somewhat sloppy. Whatever generator they used was not fully tested. The bulk of each page seems random enough. However at the lower left and lower right of alternate pages, the number is found to increment directly.

330 of 345 people found the following review helpful

★★★☆☆ **A serious reference work?**, October 16, 2006

By **BJ** (Watford, England) - See all my reviews

This review is from: A Million Random Digits with 100,000 Normal Deviates (Paperback)

For a supposedly serious reference work the omission of an index is a major impediment. I hope this will be corrected in the next edition.

(Truly) random digits from physical mechanisms

- Employ physical mechanisms rather than formal algorithms to provide random digits.
- **Hotbits project**: www.fourmilab.ch/hotbits
  - Use radioactive decay.
  - Run by Autodesk founder John Walker
  - If you connect to this site, you can listen to the Geiger counter ticks.
- **Random.org** site: www.random.org
  - Sample atmospheric noise by using a radio tuned between stations.
  - The site also provides a general discussion of random numbers.
- **Lavarnd** site: www.lavarnd.org
  - Take digital photographs of a webcam with its lens cap on.
  - Rely on thermal "noise"
- **Lavarand** site: lavarand.sgi.com
  - Take digital photographs of the patterns made by colored lava lamps.

`rng` (random number generator)

- 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

`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 rand changed the state of the generator, so the second result is different.

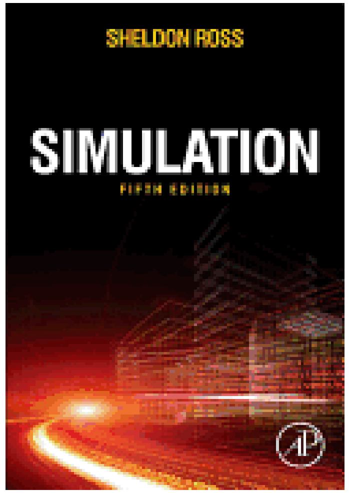
Restore the saved generator settings

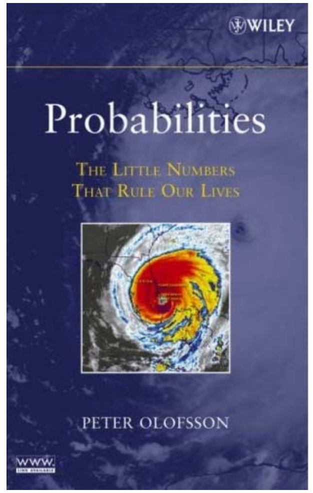
Save the current generator settings in s

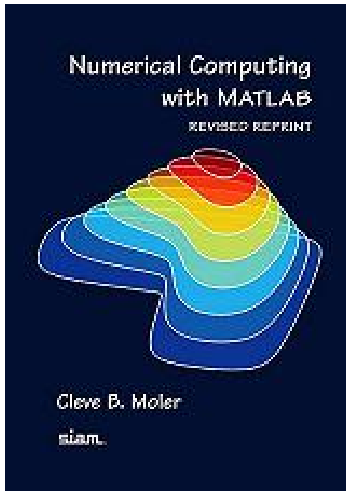
```
>> rand

ans =

   0.913375856139019

>> rand

ans =

   0.632359246225410

>> rng(s)
>> rand

ans =

   0.913375856139019

>> rand

ans =

   0.632359246225410
```

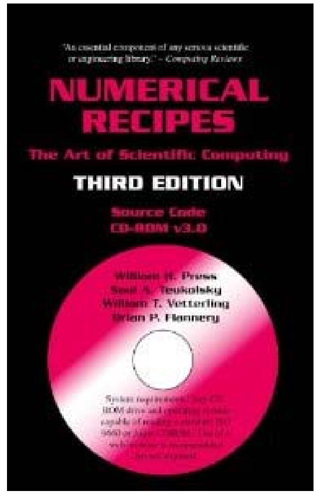
References

- Chapter 3 (Random Numbers) in Sheldon M. Ross. “Simulation.” Academic Press, 2012, 5th Edition
- Chapter 9 (p. 245-252) in Peter Olofsson, “Probabilities The Little Numbers That Rule Our Lives”, Wiley, 2006
- Chapter 9 (Random Numbers) in Cleve Moler. “Numerical Computing with MATLAB.” SIAM, 2004
- Chapter 7 (Random Numbers) (Sections 7.0-7.1.1) in W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. “Numerical Recipes: The Art of Scientific Computing.” Cambridge, 2007, 3rd Edition.









References

- Park, S.K., and K.W. Miller. "Random Number Generators: Good Ones Are Hard to Find." Communications of the ACM, 31(10):1192–1201. 1998.
- C. Moler, Random thoughts, "10^435 years is a very long time", MATLAB News and Notes, Fall, 1995

Using Excel for Statistical Analysis

- In addition to its spreadsheet functions, Excel provides a number of standard statistical and graphing procedures.
- Excel is <u>not recommended</u> for statistical analysis, beyond very basic descriptive statistics and getting a feel for your data.
- Microsoft attempted to implement the **Wichmann-Hill (1982) RNG** in Excel 2003 and failed; it did not just produce numbers between zero and unity, it would also produce negative numbers.
  - Microsoft issued a patch for Excel 2003 and Excel 2007 that incorrectly fixed the problem
  - In 2008, McCullough and Heiser showed that whatever RNG it is that Microsoft has implemented in these versions of Excel, it is not the Wichmann-Hill RNG.
  - Microsoft has failed twice to implement the dozen lines of code that define the Wichmann-Hill RNG.

[http://dl.acm.org/citation.cfm?id=1377404]

Ex. Generating a Sequence of Coin Tosses

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

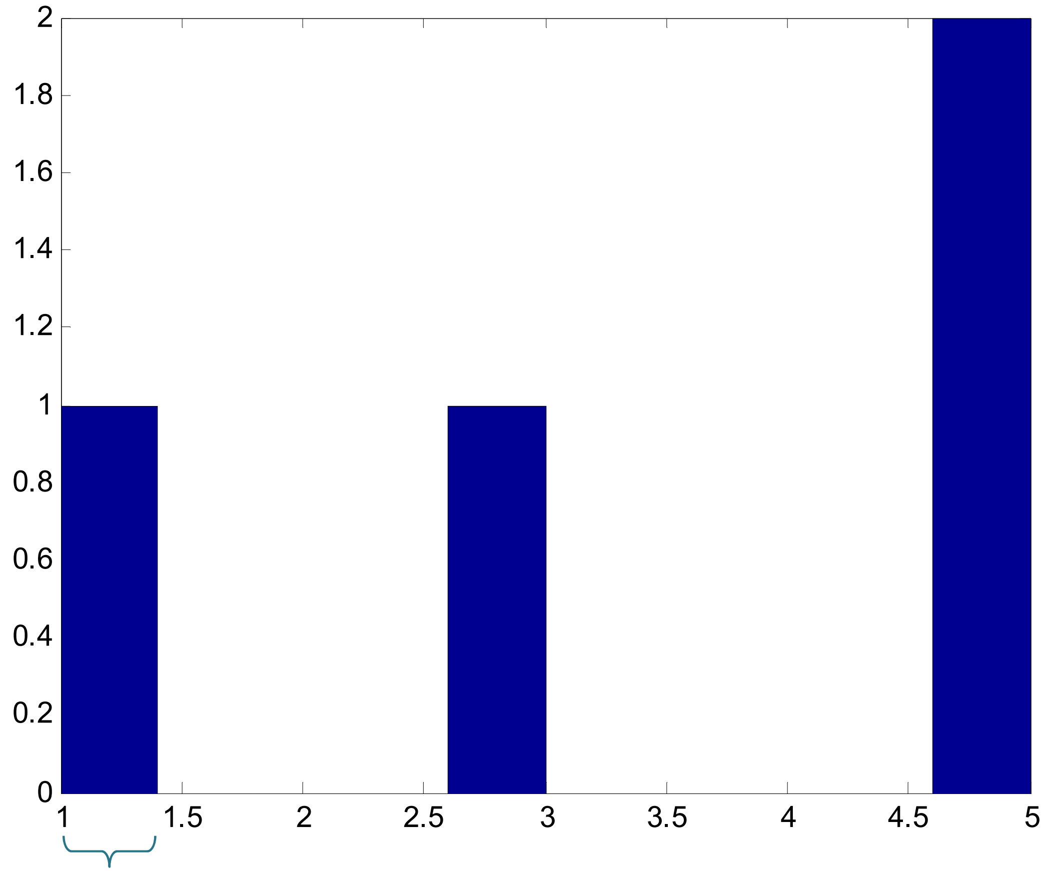
```
>> rand(1,120) < 0.5

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
```

`hist` function

- Create histogram plot
- `hist(data)` creates a histogram bar plot of data.
  - Elements in data are sorted into **10 equally spaced bins** along the x-axis between the minimum and maximum values of data.
  - Bins are displayed as rectangles such that the height of each rectangle indicates the number of elements in the bin.
  - If data is a vector, then one histogram is created.
  - If data is a matrix, then a histogram is created separately for each column.
    - Each histogram plot is displayed on the same figure with a different color.
- `hist(data,nbins)` sorts data into the number of bins specified by `nbins`.
- `hist(data,xcenters)`
  - The values in `xcenters` specify the centers for each bin on the x-axis.

`hist` function: Example

```
>> x = [1 3 5 5]

x =

     1     3     5     5

>> hist(x)
```



max → 5, min → 1

The width of each bin is $\frac{5-1}{10} = 0.4$

`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 = histc(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.
  - May 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
```

Ex. Generating a Sequence of 120 Coin Tosses

- Use 1 to represent Heads; 0 to represent Tails
- `rand(20,6) < 0.5`

Arrange the results in a 20×6 matrix.

```
>> R = rand(20,6)

R =

    0.2437    0.7482    0.6843    0.0755    0.9556    0.6781
    0.5280    0.2255    0.0868    0.4723    0.7578    0.5995
    0.6258    0.8178    0.4238    0.0544    0.1567    0.6480
    0.9217    0.8759    0.5567    0.1089    0.0768    0.9280
    0.9567    0.5547    0.3050    0.9184    0.6059    0.0736
    0.1819    0.7864    0.7504    0.6152    0.2151    0.0087
    0.2457    0.0350    0.1003    0.8668    0.3664    0.1003
    0.7068    0.4516    0.9663    0.9523    0.5152    0.2931
    0.0079    0.7121    0.0443    0.7069    0.1921    0.9299
    0.8284    0.9728    0.3609    0.1039    0.8575    0.2264
    0.6901    0.8881    0.4265    0.1593    0.3253    0.5615
    0.7637    0.9732    0.0066    0.8978    0.9331    0.5803
    0.0687    0.0411    0.0010    0.0305    0.8903    0.4463
    0.7968    0.0050    0.3274    0.6414    0.8414    0.5587
    0.2772    0.3529    0.0068    0.7676    0.0142    0.1904
    0.8423    0.1184    0.4260    0.7013    0.6582    0.8589
    0.6880    0.2206    0.5056    0.3545    0.3622    0.0476
    0.7799    0.0241    0.5777    0.0402    0.6921    0.7563
    0.7576    0.1367    0.3186    0.4661    0.9730    0.4540
    0.7705    0.2432    0.1296    0.7774    0.0123    0.7229
```

```
>> R < 0.5

ans =

     1     0     0     1     0     0
     0     1     1     1     0     0
     0     0     1     1     1     0
     0     0     0     1     1     0
     0     0     1     0     0     1
     1     0     0     0     1     1
     1     1     1     0     1     1
     0     1     0     0     0     1
     1     0     1     0     1     0
     0     0     1     1     0     1
     0     0     1     1     1     0
     0     0     1     0     0     0
     1     1     1     1     0     1
     0     1     1     0     0     0
     1     1     1     0     1     1
     0     1     1     0     0     0
     0     1     0     1     1     1
     0     1     0     1     0     0
     0     1     1     1     0     1
     0     1     1     0     1     0
```

`randi` function: Example

LLN_cointoss.m

```
close all; clear all;
N = 1e3; % Number of trials (number of times that the coin is tossed)
s = (rand(1,N) < 0.5);  % Generate a sequence of N Coin Tosses.
                        % The results are saved in a row vector s.
NH = cumsum(s); % Count the number of heads
plot(NH./(1:N)) % Plot the relative frequencies
```

Same as

`randi([0,1],1,N);`