COMP9334
Capacity Planning for Computer Systems and Networks

Week 4B_2: Generating random numbers

Week 4B_2: Generating random numbers

- We have so far used mathematical methods to determine the performance of queues or queueing networks

- Unfortunately many queues are not analytically tractable
  - You can get upper bound of mean response time of G/G/1 but what if you want to estimate it?

- Another method to study queue performance is to use discrete event simulation which you will study in Week 5

- In order to do discrete event simulation, you need to know how to generate random numbers of any probability distribution

Random number generator in C

- In C, the function *rand()* generates random integers between 0 and RAND_MAX
- E.g. The following program generates 10 random integers:

```
#include <stdio.h>
#include <stdlib.h>

int main ()
{
  int i;

   for (i = 0; i < 10; i++)
    printf("%d\n",rand());

  return;

}
```

Let us generate 10,000 random integers using rand() and see how they are distributed

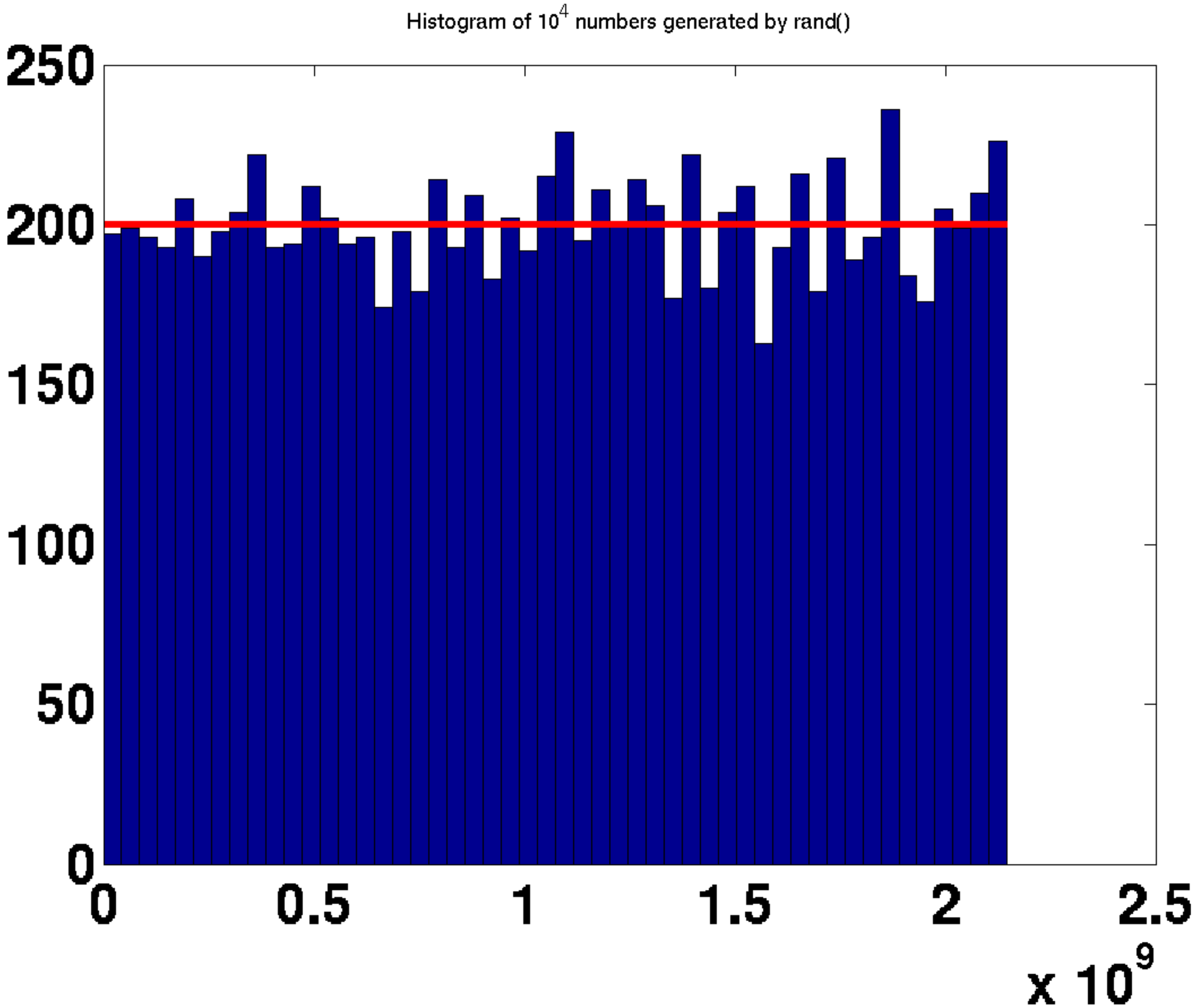
This C file “genrand1.c” is available from the course web site.

Distribution of 10000 entries from rand()

Sort into 50 bins

If the numbers are really uniformly distributed, we expect 200 numbers in each bin.

The numbers are almost uniformly distributed



LCG

- The random number generator in C is a Linear Congruential Generator (LCG)
- LCG generates a sequence of integers $\{Z_1, Z_2, Z_3, \ldots\}$ according to the recursion

  $$Z_k = a\, Z_{k-1} + c \pmod{m}$$

  where *a, c* and *m* are integers
- By choosing a, c, m, Z_1 appropriately, we can obtain a sequence of seemingly random integers
- If a = 3, c = 0, m = 5, Z_1 = 1, LCG generates the sequence 1, 3, 4, 2, 1, 3, 4, 2, …
- *Fact:* The sequence generated by LCG has a cycle of *m-1*
- We must choose *m* to be a large integer
  - For C, $m = 2^{31}$
- The proper name for the numbers generated is *pseudo-random numbers*

Seed

- LCG generates a sequence of integers $\{Z_1, Z_2, Z_3, \ldots\}$ according to the recursion

  $$Z_k = a\, Z_{k-1} + c \pmod{m}$$

  where *a, c* and *m* are integers
- The term Z_1 is call a seed
- By default, C also uses 1 as the seed and it will generate the same random sequence
- However, sometimes you need to generate different random sequences and you can change the seed by calling the function *srand()* before using *rand()*
  - Demo genrand1.c, genrand2.c and genrand3.m
  - *genrand1.c – uses the default seed*
  - *genrand2.c – sets the seed using command line argument*
  - *genrand3.c – sets the seed using current time*

Uniformly distributed random numbers between (0,1)

- With rand() in C, you can generate uniformly distributed random numbers in between 1 and 2^{31}-1(= RAND_MAX)
  - By dividing the numbers by RAND_MAX, you get randomly distributed numbers in (0,1)

- In Matlab, *rand(n,1)* generates a sequence of *n* uniformly distributed random numbers in (0,1)
  - Matlab uses the Mersenne Twister random number generator with a period of 2^{19937} - 1
    - The Python random module uses the same generator
  - If you use 10^9 random number in a second, the sequence will only repeat after 10^{5985} years

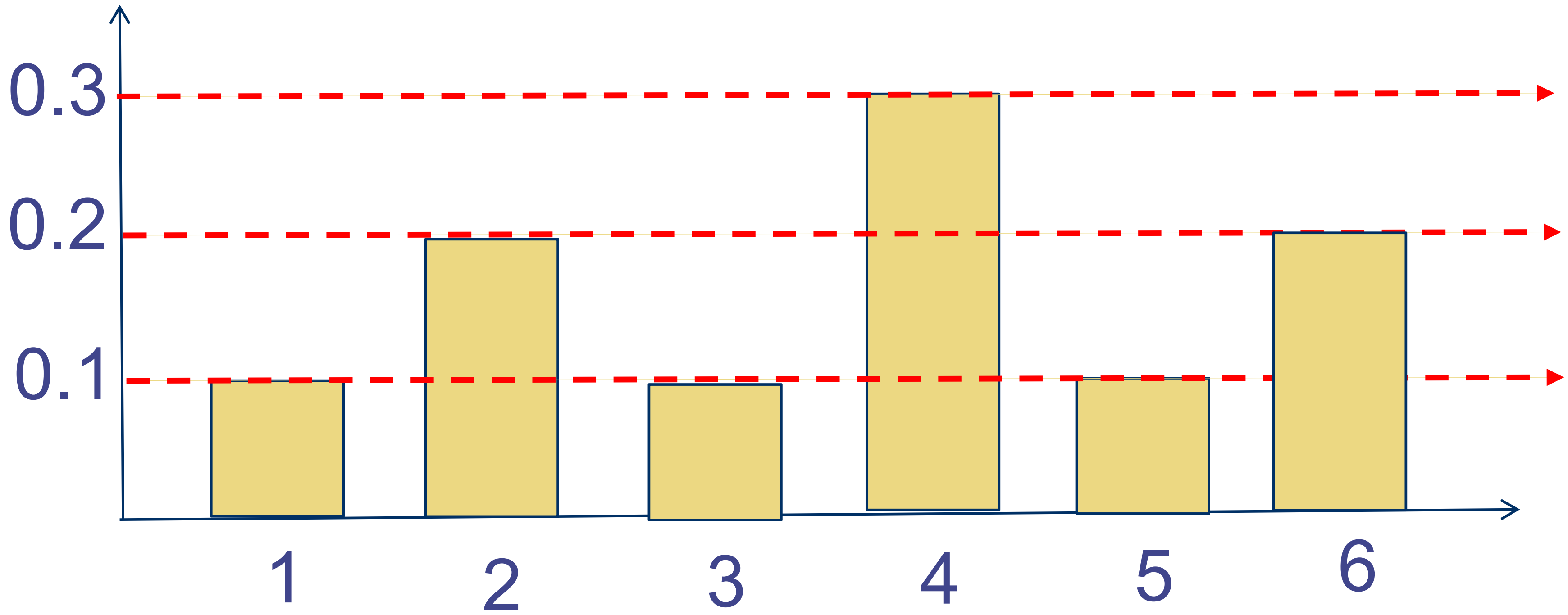
- Why are uniformly distributed random numbers important?
  - If you can generate uniformly distributed random numbers between (0,1), you can generate random numbers for any probability distribution

Fair coin distribution

- You can generate random numbers between 0 and 1
- You want to use these random numbers to imitate fair coin tossing, i.e.
  - Probability of HEAD = 0.5
  - Probability of TAIL = 0.5
- You can do this using the following algorithm
  - Generate a random number *u*
  - If $u < 0.5$, output HEAD
  - If $u \geq 0.5$, output TAIL

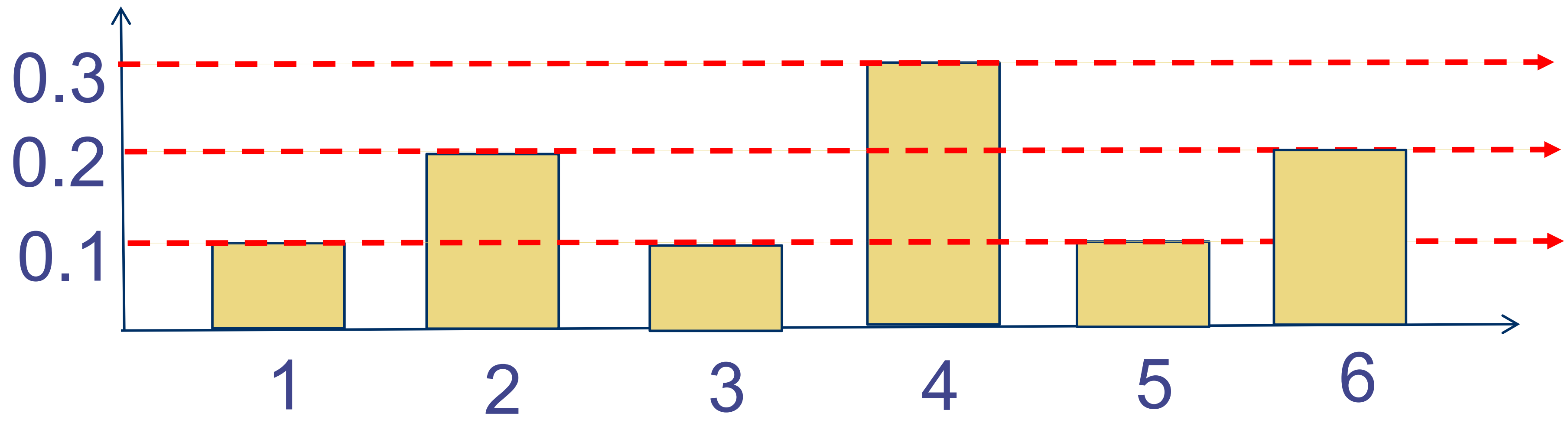
A loaded die

- You want to create a loaded die with probability mass function



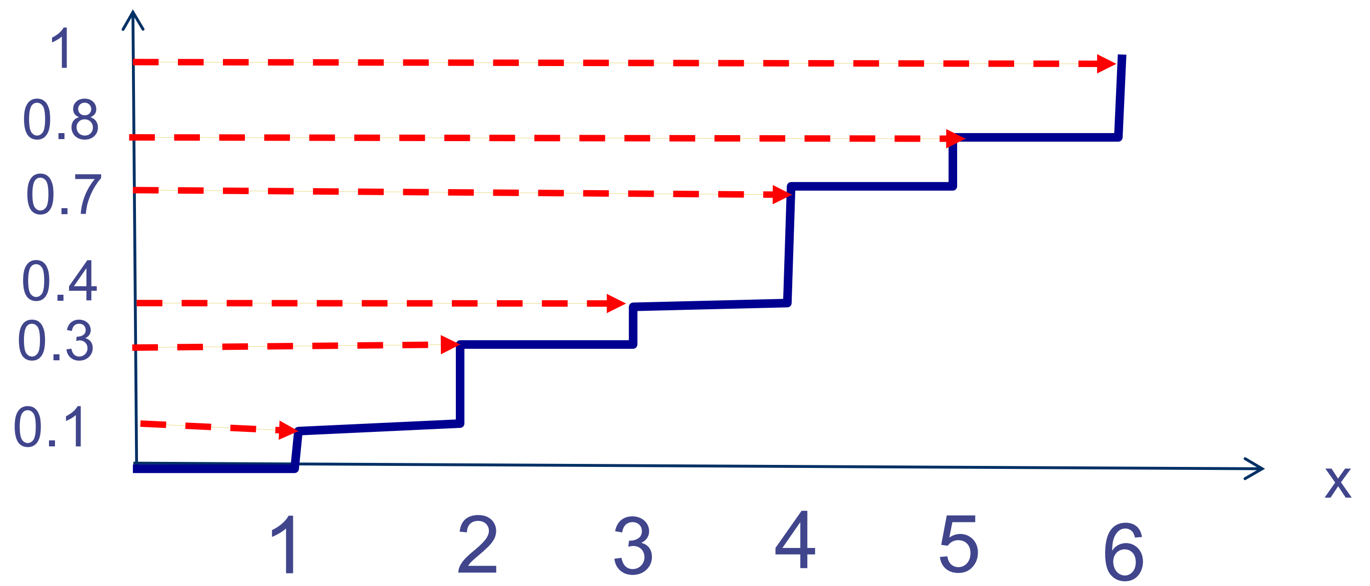
- The algorithm is:
  - Generate a random number u
  - If u < 0.1, output 1
  - If 0.1 ≤ u < 0.3, output 2
  - If 0.3 ≤ u < 0.4, output 3
  - If 0.4 ≤ u < 0.7, output 4
  - If 0.7 ≤ u < 0.8, output 5
  - If 0.8 ≤ u , output 6

Cumulative probability distribution



Probability that the dice gives a value ≤ x

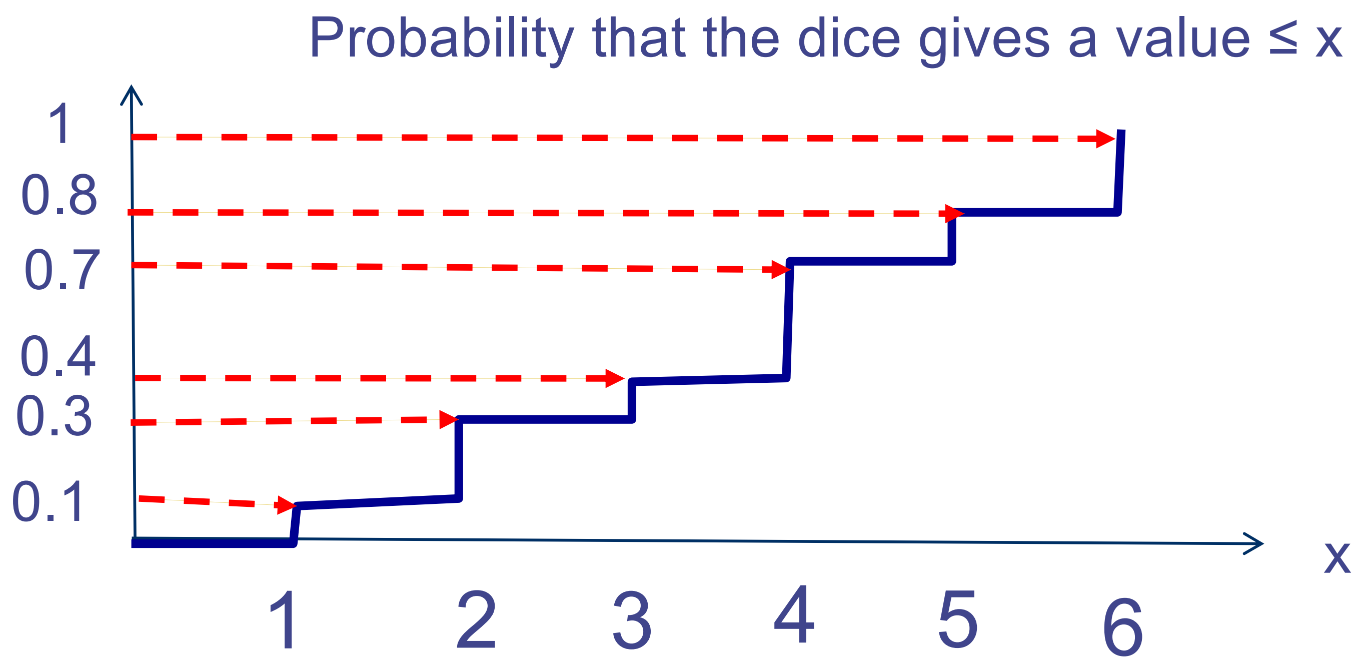
Ex: Can you work out what these levels should be



Comparing algorithm with cumulative distribution

- The algorithm is:
  - Generate a random number u
  - If $u < 0.1$, output 1
  - If $0.1 \leq u < 0.3$, output 2
  - If $0.3 \leq u < 0.4$, output 3
  - If $0.4 \leq u < 0.7$, output 4
  - If $0.7 \leq u < 0.8$, output 5
  - If $0.8 \leq u$, output 6

Ex: What do you notice about the intervals in the algorithm and the cumulative distribution?

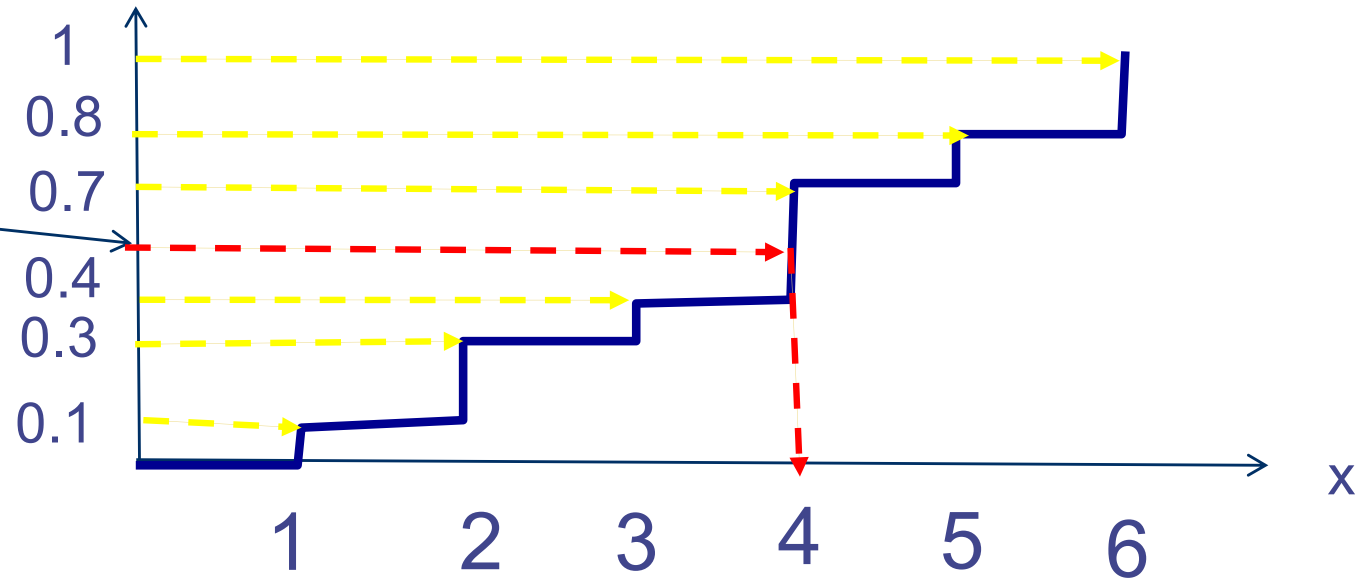


Graphical interpretation of the algorithm

- The algorithm is:
  - Generate a random number u
  - If u < 0.1, output 1
  - If 0.1 ≤ u < 0.3, output 2
  - If 0.3 ≤ u < 0.4, output 3
  - If 0.4 ≤ u < 0.7, output 4
  - If 0.7 ≤ u < 0.8, output 5
  - If 0.8 ≤ u , output 6

Probability that the dice gives a value ≤ x

Ex: Let us assume u = 0.5126, what should the algorithm output?

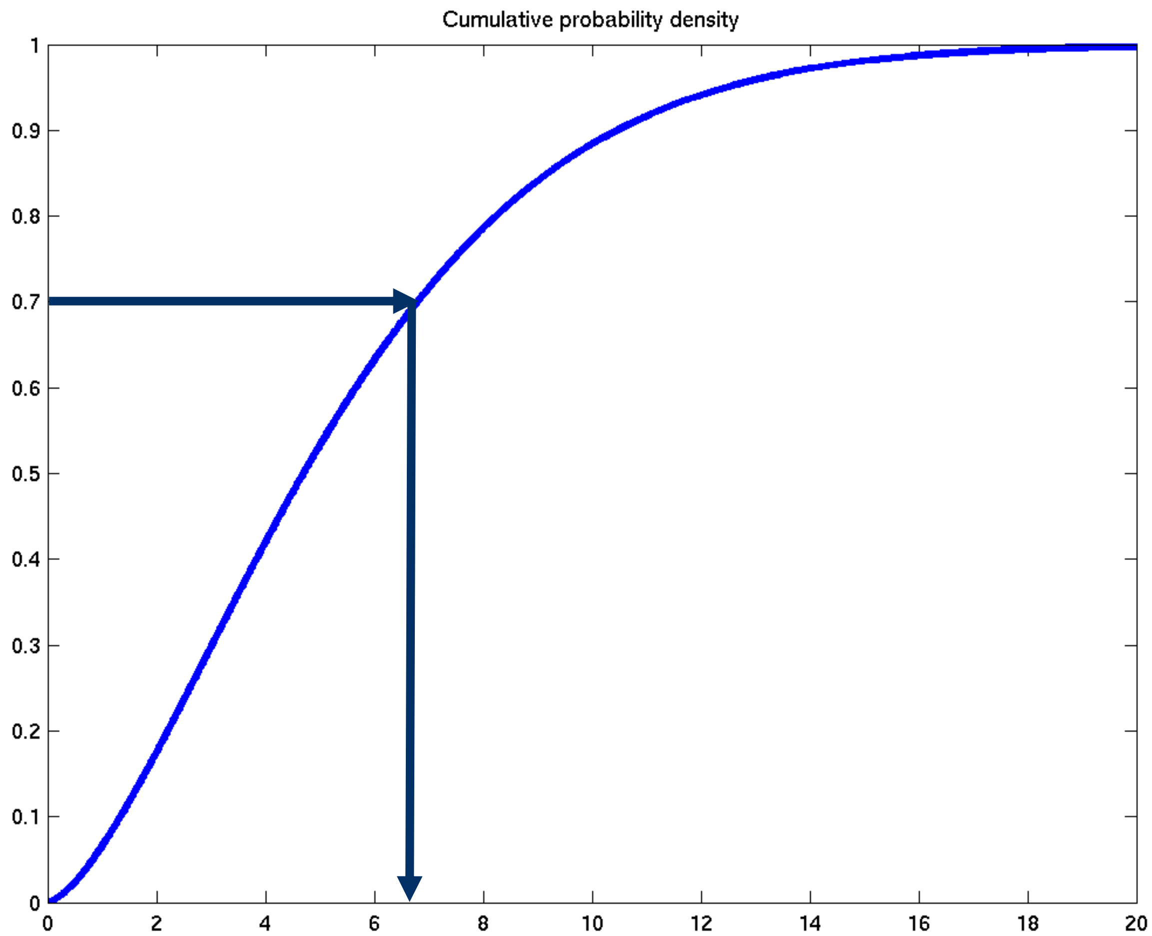


Output 4

Graphical representation of inverse transform method

- Consider the cumulative density function (CDF) $y = F(x)$, showed in the figure below

For this particular $F(x)$, if $u = 0.7$ is generated then $F^{-1}(0.7)$ is 6.8



Inverse transform method

- A method to generate random number from a particular distribution is the *inverse transform method*

- In general, if you want to generate random numbers with cumulative density function (CDF) $F(x) = Prob[X \leq x]$, you can use the following procedure:
  - Generate a number u which is uniformly distributed in (0,1)
  - Compute the number $F^{-1}(u)$

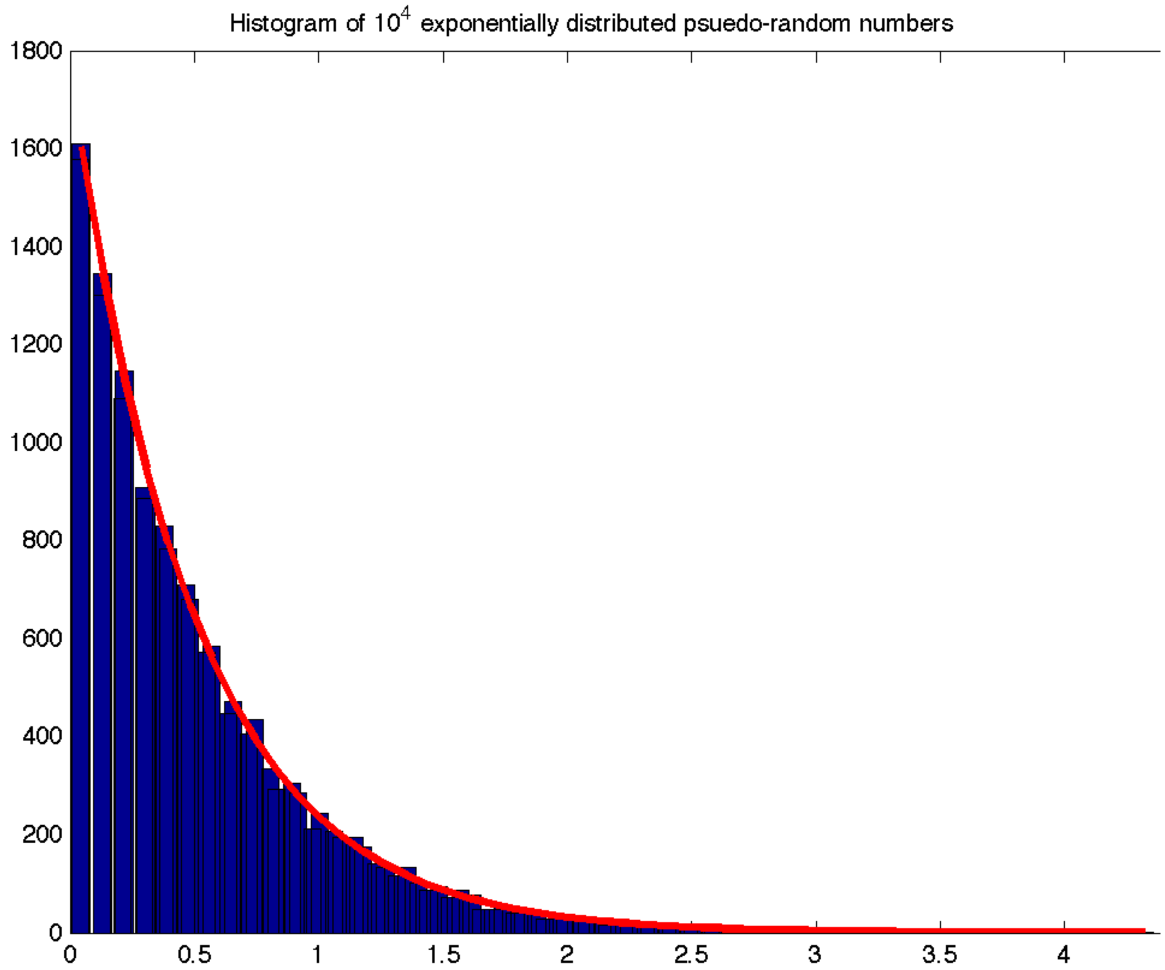
- Example: Let us apply the inverse transform method to the exponential distribution
  - *CDF is* $1 - \exp(-\lambda x)$

Generating exponential distribution

- Given a sequence $\{U_1, U_2, U_3, \ldots\}$ which is uniformly distributed in (0,1)
- The sequence $-\log(1 - U_k)/\lambda$ is exponentially distributed with rate λ

- (Matlab file hist_expon.m)

1. Generate 10,000 uniformly distributed numbers in (0,1)
2. Compute $-\log(1-u_k)/2$ where u_k are the numbers generated in Step 1
3. The plot shows
    1. The histogram of the numbers generated in Step 2 in 50 bins
    2. The red line show the expected number of exponential distributed numbers in each bin



References

- Generation of random numbers
  - Raj Jain, “The Art of Computer Systems Performance Analysis”
    - Sections 26.1 and 26.2 on LCG
    - Section 28.1 on the inverse transform methods