Modelling & Simulation
Pseudo-random Numbers

Štefan Korečko
Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice
Slovak Republic
stefan.korecko@tuke.sk

2023

Contents

- (Pseudo)random numbers & variates
- Pseudo-random numbers generators
- Generation of random variates

- Further reading
  - [CSimTech] ***Harry Perros: Computer Simulation Techniques: The definitive introduction!***, Computer Science Department NC State University Raleigh, NC, 2009, https://repository.lib.ncsu.edu/handle/1840.2/2542

Random Numbers & Variates I

- Purpose (examples)
  - numerical analysis
    - solving of complicated integrals
  - cryptography
    - key generation & confirmation
  - Software development
    - generation of testing data
  - Simulation
    - as a source of randomness in the model

Random Numbers & Variates II

- We mean **sequences** of (pseudo-)random numbers
- True random numbers
  - generated by completely unpredictable and non-reproducible source
  - Physical phenomenon used as generators
    - radioactive source
    - thermal noise from a resistor or a semi-conductor diode
    - human computer interaction processes (i.e. mouse or keyboard use)
    - …

Random Numbers & Variates III

- Pseudo-random numbers
  - Generated by algorithms
  - Generated in a deterministic way
    - Reproducible: by using the same starting value (seed) we get the same sequence
  - Uniformly distributed in the space [0,1]
- Random variates
  - = stochastic variates
  - Pseudo-random numbers with other theoretical or empirical distribution as uniform in [0,1]

Random Numbers Generators

- Criteria for pseudo-random numbers generator output
  - uniformly distributed
  - statistically independent
  - reproducible
  - non-repeating for any desired length
- Numbers generated on demand
  - usually one by one
- Generators
  - Mid-square
  - Congruential
  - Tausworthe
  - Lagged Fibonacci
  - Mercenne Twister
  - …

Mid-square

- *(metóda stredu mocniny)*
- Oldest
- By John von Neumann
- Number generation:
  1. Take the square of previously generated number
  2. Extract the middle digits
- Not recommended
  - slow
  - (very) short period

- Period
  - = number of successively generated pseudo-random numbers after which the sequence starts repeating itself

Congruential methods I

- General formula

  $$x_{i+1} = (f(x_i, x_{i-1}, \ldots)) \bmod m$$

  - has a *full period* if the period = m
- Quadratic congruential generator

  $$x_{i+1} = (a_1 x_i^2 + a_2 x_{i-1} + c) \bmod m$$

- Linear congruential generator

  $$x_{i+1} = (a x_i + c) \bmod m$$

Congruential methods II

- Linear c. g. $x_{i+1} = (ax_i + c) \bmod m$
  - generates numbers between 0 and $m-1$
  - simple & fast
  - pseudo-random numbers statistically acceptable for computer simulation
  - Period is full (i.e. $=m$) when
    - m, c have no common divisor
    - $a-1$ is divisible by all prime factors of m
    - $a-1$ is a multiple of 4 if m is a multiple of 4
  - Optimisation: setting m to size of used register
    - mod = overflow

Congruential methods III

- Composite generators
  - Two separate generators (usually congruential) combined
  - Has good statistical properties, even if the generators used are bad
  - Example:
    1. Generate a sequence $x_1, ..., x_k$ using G1
    2. Generate an integer r, $r \in 1, \ldots, k$ using G2
    3. Return x_r
    4. Generate a new x_r using G1
       - Generate a new number by G1 and replace x_r with it
    5. Go to step 2

Lagged Fibonacci Generators

- =LFG *(Oneskorené Fibonacciho generátory)*
- Based on the Fibonacci sequence
  - $x_n = x_{n-1}+x_{n-2} \qquad x_0 = 0, x_1 = 1$
- General form
  - $x_n = (x_{n-j}\ Op\ x_{n-k}) \bmod m \qquad 0<j<k$
    - *Op* = algebraic operation (+ - * …)
- Pros
  - very good statistical properties
  - only a bit less efficient than congruential
  - can be parallelized
- Cons
  - highly sensitive on the seed
- Commonly used versions for *Op* = + ; m=2^M
  - j=5, k=17, M=31
  - j=24, k=55, M=31

Mersenne Twister

- A variation on a Two-tap generalised feedback shift register (LFG with Xor as *Op*)
- Period length is a Mersenne prime
- generates a sequence of bits.
  - sequence is grouped into blocks (32-bit)
  - these blocks are considered to be random
- Pros
  - very good statistical properties
  - very high max. period 2^{19937}-1
- Cons
  - complex to implement
  - sensitive to poor initialization

Statistical Tests for Generators

- To check the output of a pseudo-random number generator statistically
- Belong to statistical hypothesis testing

- To test the randomness of a sequence of bits
  - Frequency test
  - Serial test
  - Autocorrelation test

- To test the randomness of numbers in [0,1]
  - Runs test
  - Chi-squared test for goodness of fit

Statistical Hypothesis Testing I

- Tests validity of a hypothesis
  - assertion about (measures of) a distribution of some random variables
  - *null hypothesis, H_0*.
- *H_a - alternative hypothesis,* a negation of H_0.
- In our case:
  - H_0 = “The numbers sequence produced by the generator is random”
  - H_a = “The numbers sequence produced by the generator is **not** random”

Statistical Hypothesis Testing II

- Testing procedure
  1. Collect data *(sequence produced by the generator)*
  2. Run test
  3. Accept or reject H_0 (fail to reject or reject H_0)
- Errors
  - Type I (false negative): H_0 is rejected but is in fact true
  - Type II (false positive): H_0 is accepted but in fact is not true
    - More precisely, H_0 is failed to be rejected
- α - *the level of significance*
  - probability of type I error
  - Usually set to 0.01 – 0.05
  - c = 1- α is level of confidence

Frequency and Serial Test

- Frequency test
  - one of the most fundamental
    - if a generator fails it, it will probably fail other tests
  - checks whether there is approx. the same number of occurrences of each digit

- Serial test
  - as the frequency test but for pairs of digits

Frequency Test in Detail

1. Generate *m* pseudo-random numbers and concatenate them into a string of bits
2. Convert all “0” to “-1”.
   - The resulting sequence is $X_1 X_2 X_3 \ldots X_n$, $X_i \in \{-1,1\}$
3. Compute $S_n = X_1 + X_2 + \cdots + X_n$
4. Compute the test statistic S_{obs}
$$S_{obs} = \frac{|S_n|}{\sqrt{n}}$$
5. Compute the P-value as
$$erfc\left(\frac{S_{obs}}{\sqrt{2}}\right)$$
   erfc = complementary error function
6. If P-value ≥ α, the sequence can be considered random (H_0 accepted)

Autocorrelation, Runs and Chi² Test

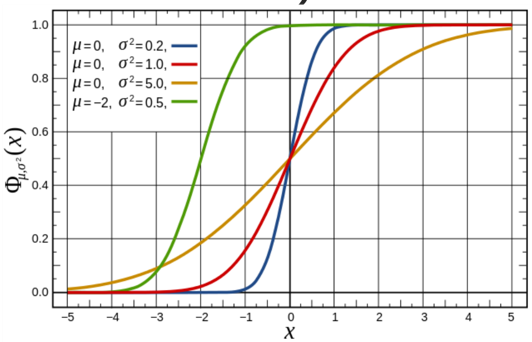
- Autocorrelation test
  - *s* = sequence of n bits created by a generator.
  - If the sequence of bits in *s* is random, then it will be different from another bit string obtained by shifting the bits of *s* by *d* positions.
- Runs test
  - to test the assumption that the pseudo-random numbers are independent of each other (mutually independent)
  - counts of ascending and descending runs should follow a certain distribution
  - belongs to the diehard tests
- Chi-squared test for goodness of fit *(Chi-kvadrát test dobrej zhody)*
  - checks whether a sequence of pseudo-random numbers in [0,1] is uniformity distributed.
  - in general, it can be used to check whether an empirical distribution follows a specific theoretical distribution

Generation of Random Variates

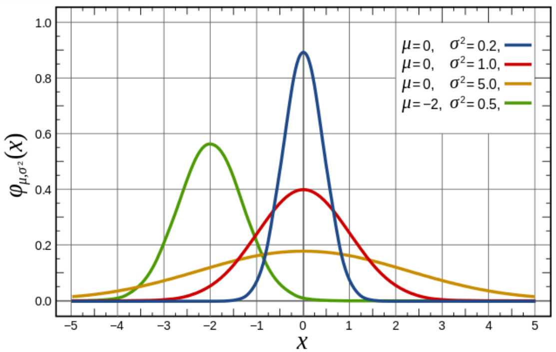
- Random variates / stochastic variates
  - Pseudo-random numbers with other theoretical or empirical distribution as uniform in [0,1]
- = sampling from xy distribution
- Methods
  - Inverse transform sampling
  - Sampling from an empirical probability distribution
  - Rejection method

Note:: cdf & pdf of Continuous Random Variable

- cdf - cumulative distribution function *(distribučná funkcia)*
  - $F_X(x) = F(x) = Pr(X \le x)$ for every real number x
  - X - real-valued random variable
  - Pr – probability
  - $Pr(a < X \le b) = F_X(b) - F_X(a)$



- pdf - probability density function *(hustota pravdepodobnosti)*
  - $f_X(x)$, $f(x)$
  - a function that describes the relative likelihood for this random variable to have a given value.
  - Probability of an exact value =0
  - $Pr(a \le X \le b) = \int_a^b f(x)\,\mathrm{d}x$



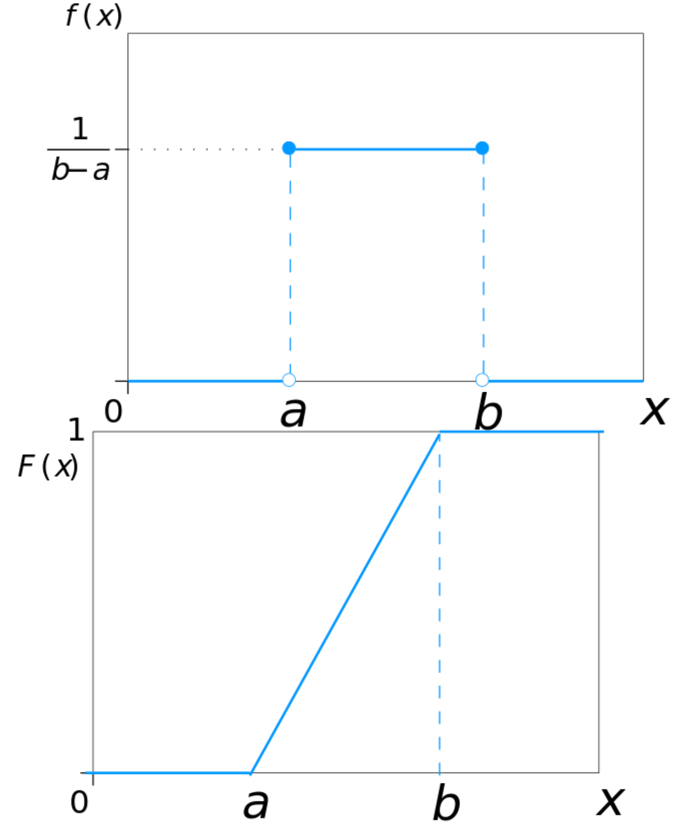
- cdf vs. pdf $F(x) = \int_{-\infty}^{x} f(u)\,\mathrm{d}u$

Inverse Transform Sampling I

- = Inverse transformation method
- Computationally efficient if the cdf can be analytically inverted
- Method:
  1. Generate a uniformly distributed random number r, $r \in [0,1]$.
  2. Compute the value *x* such that cdf $F(x) = r$.
  3. Take *x* to be the random number drawn from the distribution described by cdf F.
  - $x = F^{-1}(r)$

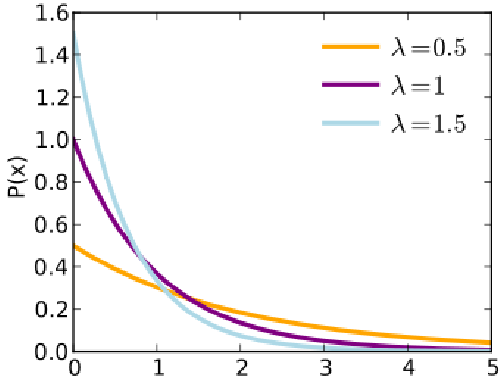
Inverse Transform Sampling II

- From a uniform distribution
  - pdf
    $$f(x)=\begin{cases}\frac{1}{b-a} & \text{for } x \in [a,b] \\ 0 & \text{otherwise}\end{cases}$$
  - cdf
    $$F(x)=\begin{cases}0 & \text{for } x \le a \\ \frac{x-a}{b-a} & \text{for } x \in [a,b] \\ 1 & \text{for } x \ge b\end{cases}$$
  - $x = a+(b-a)r$

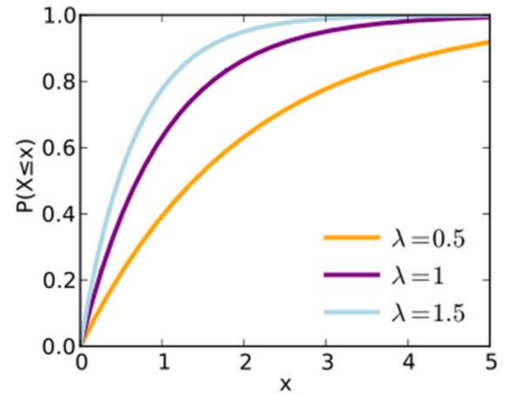


Inverse Transform Sampling III

- From an exponential distribution
  - pdf: $f_X(x)=\lambda e^{-\lambda x}$, a>0, x≥0

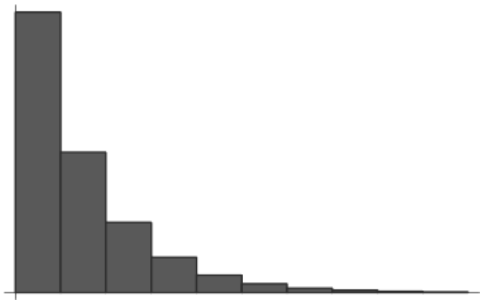


  - cdf: $F_X(x)= 1-e^{-\lambda x}$ iff x≥0; 0 otherwise
  - $x= -(1/\lambda)\, log\, r$



Inverse Transform Sampling IV

- From a geometric distribution
  - a discrete distribution
    - variable $n = 0,1,2,\ldots$
  - discrete analog of the exponential distribution
  - pdf: $p(n)=p(1-p)^n = pq^n;\ p+q=1,\ 0<p<1$
  - cdf $F(n)=1-q^{n+1}$
  - $n=(\log r / \log q) -1$

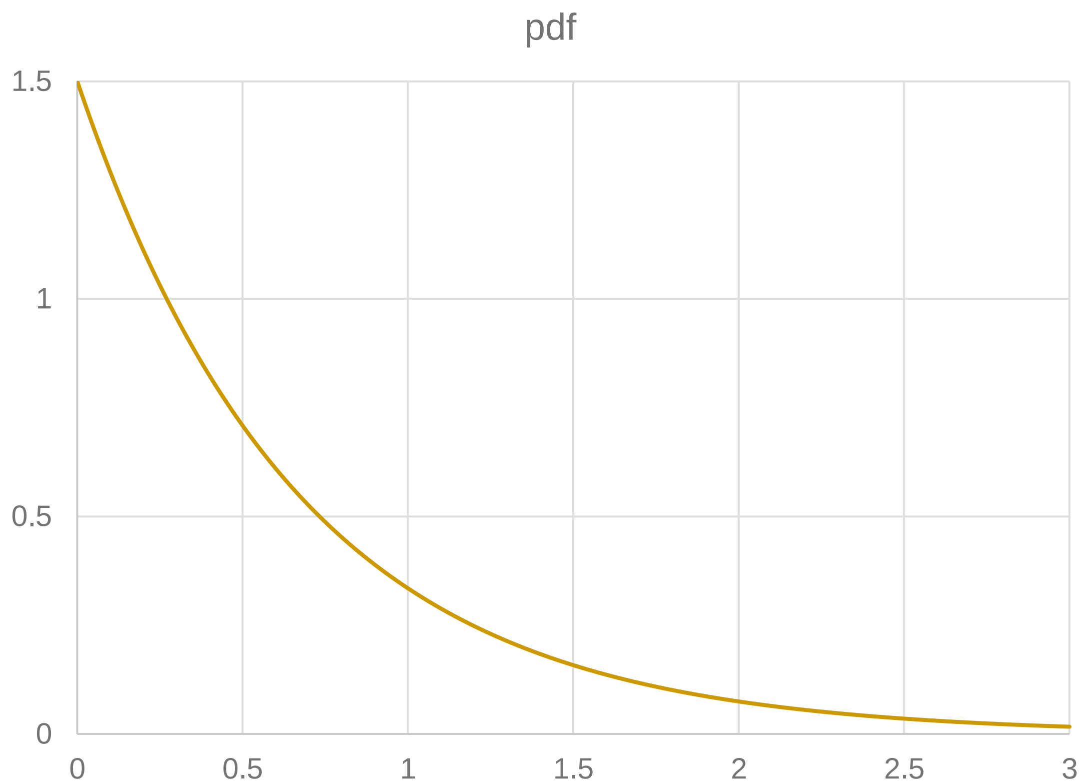


Rejection Method

- Prerequisites
  - $f(x)$ (pdf) is bounded
  - x has a finite range, i.e. $a \leq x \leq b$
- Algorithm
  1. Normalize the range of $f(x)$ by a scale factor c so that $cf(x) \leq 1$ and $a \leq x \leq b$.
  2. Define x as a linear function of r, i.e. $x = a + (b\text{-}a)r$, where r is a random number.
  3. Generate pairs of pseudo-random numbers (r_1, r_2).
  4. Accept the pair and use $x = a + (b\text{-}a)r_1$ as a random variate whenever the pair satisfies $r_2 \leq cf(a + (b\text{-}a)r_1)$.

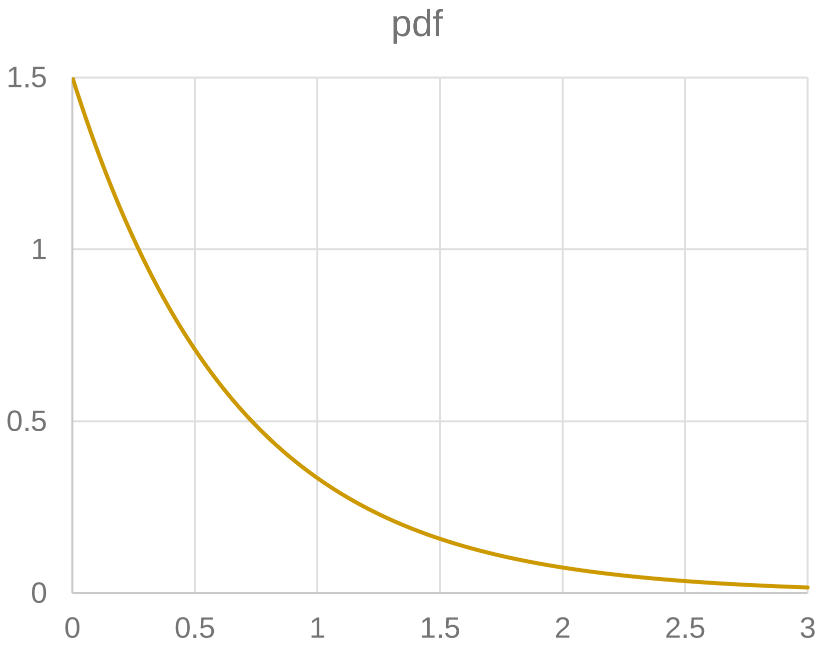
Rejection Method::Example

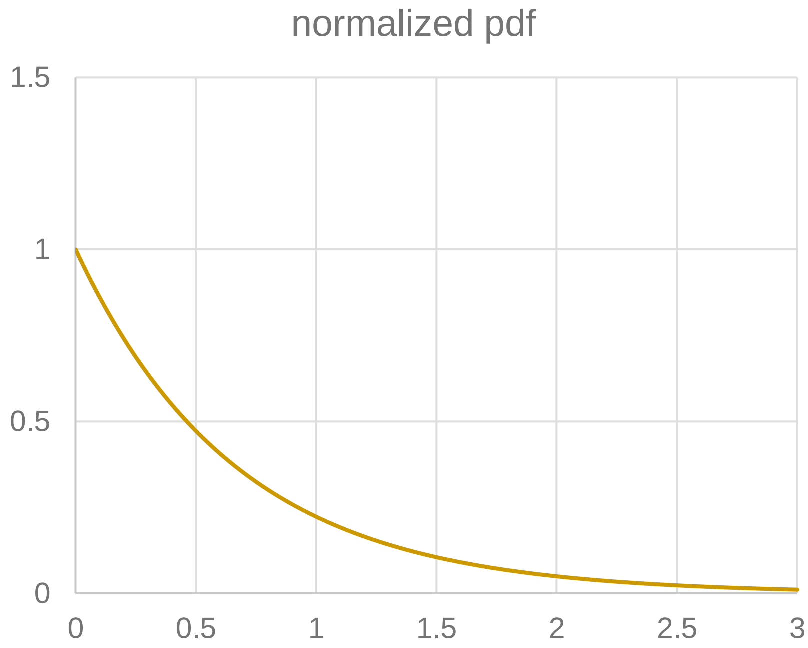
- Task: generate a random variate x, $0 \leq x \leq 2$ from exponential distribution with $\lambda = 1.5$
- pdf:
  $f_X(x) = \lambda e^{-\lambda x}$
  $= 1.5\, e^{-(1.5)x}$
- $a=0,\ b=2$



Rejection Method::Example

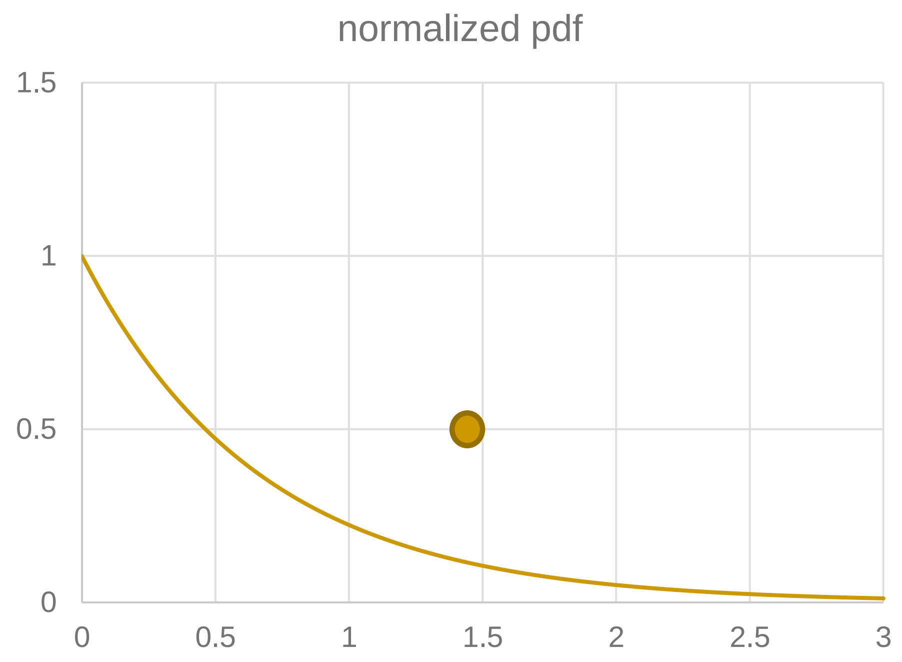
- Task: generate a random variate x, $0 \le x \le 2$ from exponential distribution with $\lambda = 1.5$
- Normalisation needed: c= $2/3$





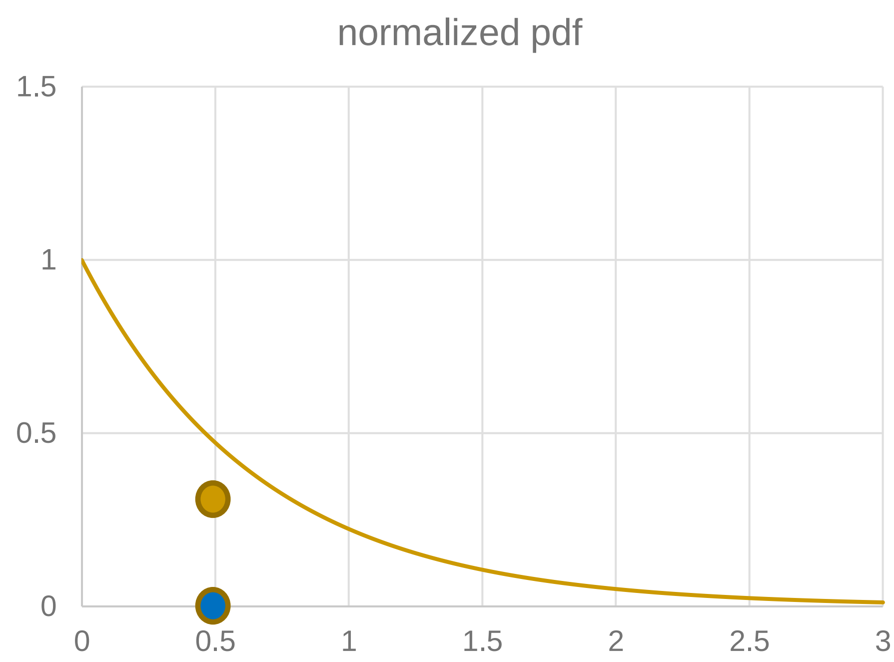
Rejection Method::Example

- Task: generate a random variate x, $0 \leq x \leq 2$ from exponential distribution with $\lambda = 1.5$
- Generated pair:
  - $r_1 = 0.7$
  - $r_2 = 0.5$
- $cf(a + (b-a)r_1)$
  $=(2/3)f(0+(2-0)0.7)$
  $=(2/3)\ f(1.4)$
  $=(2/3)\ 0.183684642$
  $=0.122456428$
- $0.5 > 0.122456428$
- *The pair is* ***rejected***



Rejection Method::Example

- Task: generate a random variate x, $0 \leq x \leq 2$ from exponential distribution with $\lambda = 1.5$
- Generated pair:
  - $r_1 = 0.25$
  - $r_2 = 0.3$



- $cf(a + (b\text{-}a)r_1)$
  $=(2/3)f(0+(2\text{-}0)0.25)$
  $=(2/3)\ f(0.5)$
  $=(2/3)\ 0.708549829$
  $=0.472366553$
- $0.3 \leq 0.472366553$
- *The pair is* ***accepted, x=*** $a + (b\text{-}a)r_1$ ***=0.5***

Monte Carlo Integration

- Using rejection method
- Task: compute $\int_a^b f(x)dx$
- Prerequisites: $f(x)$ is bounded at [a,b]

Monte Carlo Integration

- Algorithm
  - for a simplified case: $f(x) \geq 0$ at $[a,b]$
  - max – maximum of $f(x)$ at $[a,b]$
  - N – natural number, $N>0$

1. Let i=0, n=0
2. Generate a pair of pseudo-random numbers
   - uniformly distributed
   - r_1-from a to b
   - r_2-from 0 to max
3. if $r_2 < f(r_1)$ then $n=n+1$
4. $i=i+1$
5. If $i<N$, go to step 2
6. If $i=N$, return (n/N)(b-a)max

Monte Carlo Integr.::Example

- Compute $\int_0^2 2x\,dx$
  - Can be computed analytically: $\int_0^2 2x\,dx = [x^2]_0^4 = 2^2 - 0^2 = 4$
  - *a=0, b=2, max=4*