Synthetic Load Injection

J-M Vincent

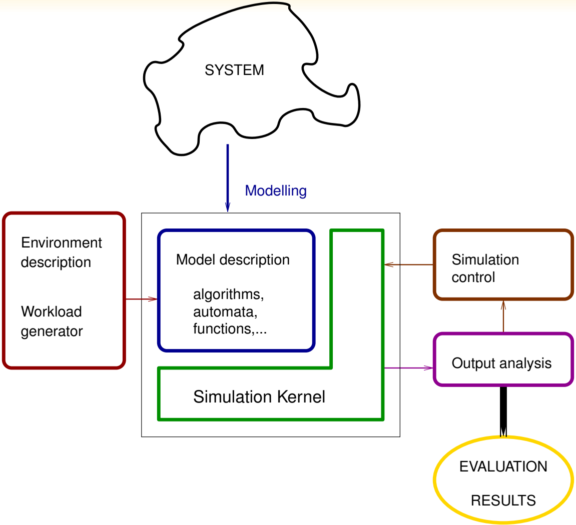
Laboratory ID-IMAG
MESCAL Project
Universities of Grenoble, France
Jean-Marc.Vincent@imag.fr

Outline

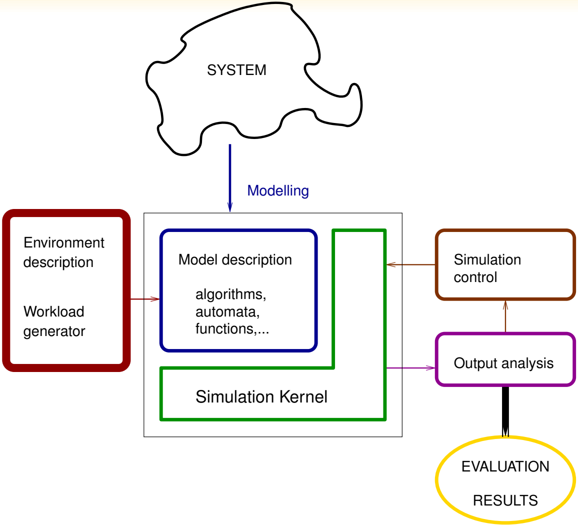
Outline

1. **Workload generation problem**
2. Generating random objects
3. Generation of complex objects
4. Quantity generation
5. Synthesis

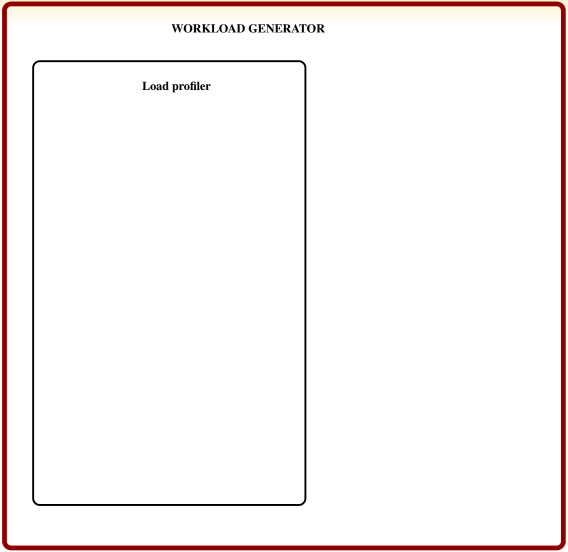
Workload generation problem



Workload generation problem



Workload generation problem (2)

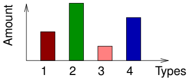


Workload generation problem (2)

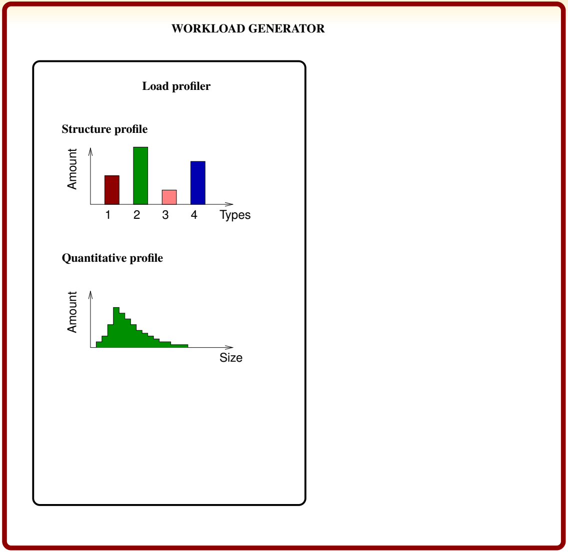
WORKLOAD GENERATOR

Load profiler

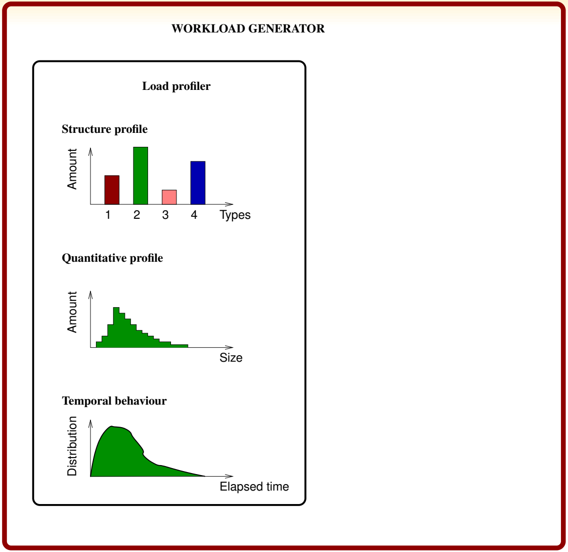
Structure profile



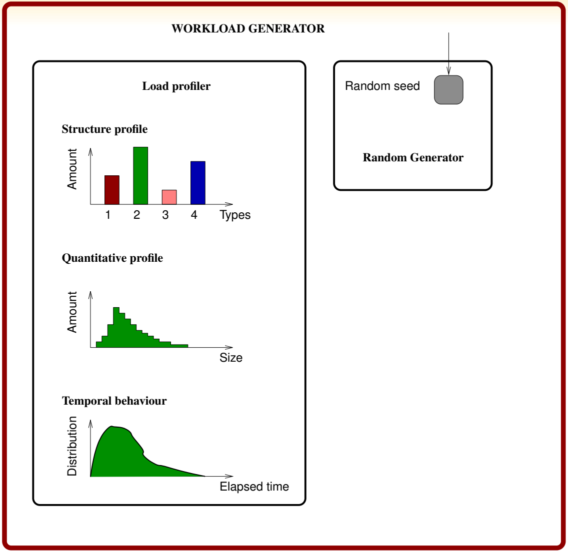
Workload generation problem (2)



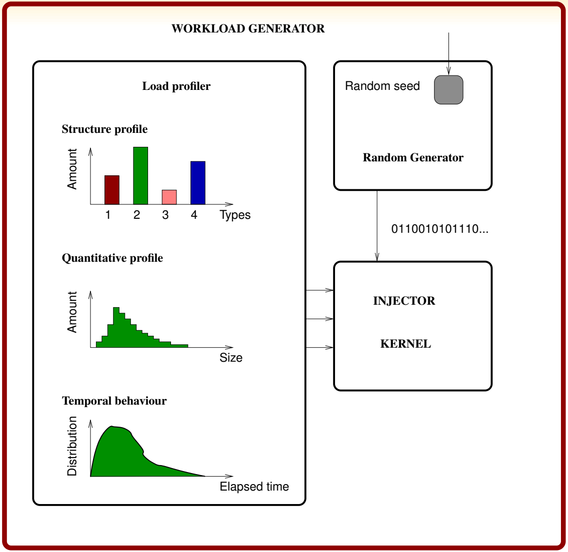
Workload generation problem (2)



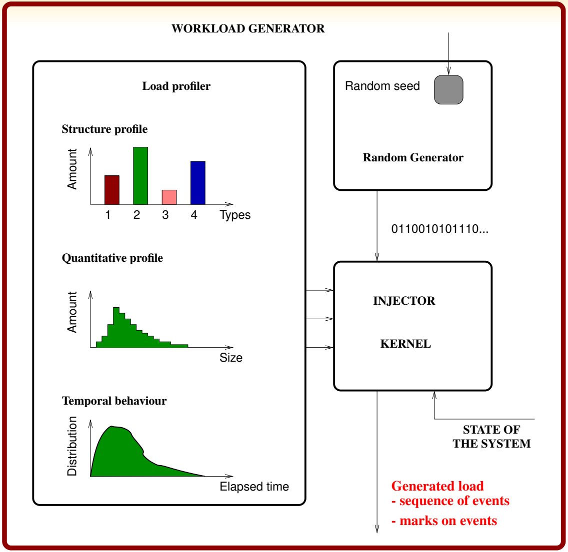
Workload generation problem (2)



Workload generation problem (2)



Workload generation problem (2)



Outline

Generating random objects

Denote by X the generated object (X is a random variable)
Distribution (proportion of observations, input of the load injector)

$$p_k = \mathbb{P}(X = k).$$

Remarks :

$$0 \leqslant p_i \leqslant 1; \quad \sum_k p_k = 1.$$

Expectation (average, mean)

$$\mathbb{E}X = \sum_k k.\mathbb{P}(X = k) = \sum_k k p_k.$$

Variance and standard deviation

$$\mathbb{V}arX = \sum_k (k - \mathbb{E}X)^2 \mathbb{P}(X = k) = \mathbb{E}X^2 - (\mathbb{E}X)^2.$$

$$\sigma(X) = \sqrt{\mathbb{V}arX}.$$

Generating random objects

Denote by X the generated object (X is a random variable)
Distribution (proportion of observations, input of the load injector)

$$p_k = \mathbb{P}(X = k).$$

Remarks :

$$0 \leqslant p_i \leqslant 1; \quad \sum_k p_k = 1.$$

Expectation (average, mean)

$$\mathbb{E}X = \sum_k k.\mathbb{P}(X = k) = \sum_k k p_k.$$

Variance and standard deviation

$$\mathbb{V}arX = \sum_k (k - \mathbb{E}X)^2 \mathbb{P}(X = k) = \mathbb{E}X^2 - (\mathbb{E}X)^2.$$

$$\sigma(X) = \sqrt{\mathbb{V}arX}.$$

Generating random objects

Denote by X the generated object (X is a random variable)
Distribution (proportion of observations, input of the load injector)

$$p_k = \mathbb{P}(X = k).$$

Remarks :

$$0 \leqslant p_i \leqslant 1; \quad \sum_k p_k = 1.$$

Expectation (average, mean)

$$\mathbb{E}X = \sum_k k.\mathbb{P}(X = k) = \sum_k k p_k.$$

Variance and standard deviation

$$\mathbb{V}arX = \sum_k (k - \mathbb{E}X)^2 \mathbb{P}(X = k) = \mathbb{E}X^2 - (\mathbb{E}X)^2.$$

$$\sigma(X) = \sqrt{\mathbb{V}arX}.$$

Generating random objects

Denote by X the generated object (X is a random variable)
Distribution (proportion of observations, input of the load injector)

$$p_k = \mathbb{P}(X = k).$$

Remarks :

$$0 \leqslant p_i \leqslant 1; \quad \sum_k p_k = 1.$$

Expectation (average, mean)

$$\mathbb{E}X = \sum_k k.\mathbb{P}(X = k) = \sum_k k p_k.$$

Variance and standard deviation

$$\mathbb{V}arX = \sum_k (k - \mathbb{E}X)^2 \mathbb{P}(X = k) = \mathbb{E}X^2 - (\mathbb{E}X)^2.$$

$$\sigma(X) = \sqrt{\mathbb{V}arX}.$$

The random function

Random bit generator (see previous lecture)

drand48 manpage

double drand48(void) (48 bits encoded in 8 bytes)
The rand48() family of functions generates pseudo-random numbers using a linear congruential algorithm working on integers 48 bits in size. The particular formula employed is r(n+1) = (a * r(n) + c) mod m where the default values are for the multiplicand a = 0xfdeece66d = 25214903917 and the addend c = 0xb = 11. The modulo is always fixed at m = 2 ** 48. r(0) is called the seed of the random number generator.

The sequence of returned values from a sequence of calls to the random function is modeled by a sequence of independent random variables uniformly distributed on the real interval $[0, 1[$.

The random function

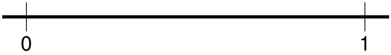
Random bit generator (see previous lecture)

drand48 manpage

double drand48(void) (48 bits encoded in 8 bytes)
The rand48() family of functions generates pseudo-random numbers using a linear congruential algorithm working on integers 48 bits in size. The particular formula employed is r(n+1) = (a * r(n) + c) mod m where the default values are for the multiplicand a = 0xfdeece66d = 25214903917 and the addend c = 0xb = 11. The modulo is always fixed at m = 2 ** 48. r(0) is called the seed of the random number generator.

The sequence of returned values from a sequence of calls to the random function is modeled by a sequence of independent random variables uniformly distributed on the real interval $[0, 1[$.

The random function



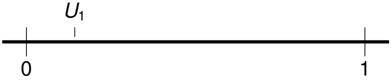
Problem

All the difficulty is to find a function (an algorithm) that transforms the $[0, 1[$ in a set with a good probability conserving.

Example : flip a coin

u= r andom()
if u $\leqslant \frac{1}{2}$ **then**
    return Head
else
    return Tail
end if

The random function



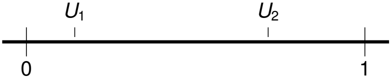
Problem

All the difficulty is to find a function (an algorithm) that transforms the [0, 1[in a set with a good probability conserving.

Example : flip a coin

u= r andom()
if u $\leqslant \frac{1}{2}$ **then**
  return Head
else
  return Tail
end if

The random function



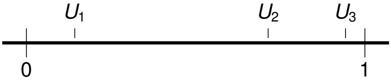
Problem

All the difficulty is to find a function (an algorithm) that transforms the $[0, 1[$ in a set with a good probability conserving.

Example : flip a coin

u= r andom()
if u $\leqslant \frac{1}{2}$ **then**
  return Head
else
  return Tail
end if

The random function



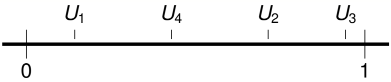
Problem

All the difficulty is to find a function (an algorithm) that transforms the $[0, 1[$ in a set with a good probability conserving.

Example : flip a coin

u= r andom()
if u $\leqslant \frac{1}{2}$ **then**
  return Head
else
  return Tail
end if

The random function



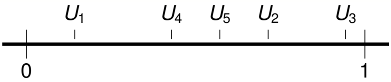
Problem

All the difficulty is to find a function (an algorithm) that transforms the $[0, 1[$ in a set with a good probability conserving.

Example : flip a coin

u= r andom()
if u $\leqslant \frac{1}{2}$ **then**
  return Head
else
  return Tail
end if

The random function



Problem

All the difficulty is to find a function (an algorithm) that transforms the $[0, 1[$ in a set with a good probability conserving.

Example : flip a coin

u= r andom()
if u $\leqslant \frac{1}{2}$ **then**
  return Head
else
  return Tail
end if

The random function

Problem

All the difficulty is to find a function (an algorithm) that transforms the $[0, 1[$ in a set with a good probability conserving.

Example : flip a coin

u= r andom()
if u $\leqslant \frac{1}{2}$ **then**
  return Head
else
  return Tail
end if

The random function

$$\mathbb{P}(U \in [a, b[) = (b - a)$$

Problem

All the difficulty is to find a function (an algorithm) that transforms the $[0, 1[$ in a set with a good probability conserving.

Example : flip a coin

u= r andom()
if u $\leqslant \frac{1}{2}$ **then**
  return Head
else
  return Tail
end if

The random function

$\mathbb{P}(U \in [a, b[) = (b - a)$

Problem

All the difficulty is to find a function (an algorithm) that transforms the $[0, 1[$ in a set with a good probability conserving.

Example : flip a coin

u= r andom()
if u $\leqslant \frac{1}{2}$ **then**
  return Head
else
  return Tail
end if

The random function

$\mathbb{P}(U \in [a, b[) = (b - a)$

Problem

All the difficulty is to find a function (an algorithm) that transforms the $[0, 1[$ in a set with a good probability conserving.

Example : flip a coin

u= random()
if u $\leqslant \frac{1}{2}$ **then**
  return Head
else
  return Tail
end if

Practical example : Web server

Types of request

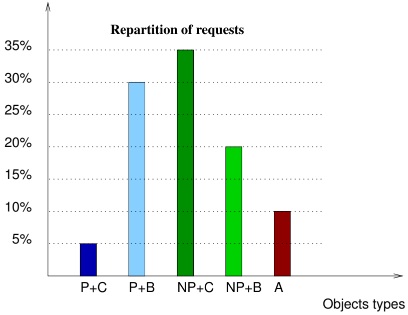
1. Professional customer, consult
2. Professional customer, purchase
3. Non professional customer, consult
4. Non professional customer, purchase
5. Adminstration

Build an algorithm that provides a set of requests according the observed distribution.

Practical example : Web server

Types of request

1. Professional customer, consult
2. Professional customer, purchase
3. Non professional customer, consult
4. Non professional customer, purchase
5. Adminstration

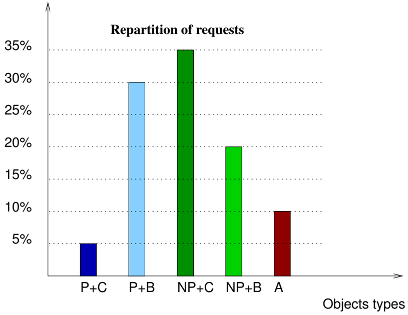


Build an algorithm that provides a set of requests according the observed distribution.

Practical example : Web server

Types of request

1. Professional customer, consult
2. Professional customer, purchase
3. Non professional customer, consult
4. Non professional customer, purchase
5. Adminstration



Build an algorithm that provides a set of requests according the observed distribution.

Tabulation method

Pre-computation

$$p_k = \frac{m_k}{m} \text{ where } \sum_k m_k = m.$$

Create a table T with size m.
Fill T such that m_k cells contains k.
Computation cost : m steps
Memory cost : m

Table construction

```
i=0
for k=1, k ⩽ K, k++ do
    for j=1, j ⩽ m_k, j++ do
        T[i]= k ; i=i+1 ;
    end for
end for
```

Generation

Generate uniformly on the set $\{0, 1, \cdots, m-1\}$
Returns the value in the table
Computation cost : $\mathcal{O}(1)$ step
Memory cost : $\mathcal{O}(m)$

Generation algorithm

```
u= r andom() ;
i= (int) floor(u*m)
return T[i]
```

Tabulation method

Pre-computation

$$p_k = \frac{m_k}{m} \text{ where } \sum_k m_k = m.$$

Create a table T with size m.
Fill T such that m_k cells contains k.
Computation cost : m steps
Memory cost : m

Table construction

i=0
for k=1, k $\leqslant$ K, k++ **do**
  for j=1, j $\leqslant$ m_k, j++ **do**
    T[i]= k ; i=i+1 ;
  end for
end for

Generation

Generate uniformly on the set $\{0, 1, \cdots, m-1\}$
Returns the value in the table
Computation cost : $\mathcal{O}(1)$ step
Memory cost : $\mathcal{O}(m)$

Generation algorithm

u= r andom() ;
i= (int) floor(u*m)
return T[i]

Tabulation method

Pre-computation

$$p_k = \frac{m_k}{m} \text{ where } \sum_k m_k = m.$$

Create a table T with size m.
Fill T such that m_k cells contains k.
Computation cost : m steps
Memory cost : m

Table construction

i=0
for k=1, k $\leqslant$ K, k++ **do**
  for j=1, j $\leqslant m_k$, j++ **do**
    T[i]= k ; i=i+1 ;
  end for
end for

Generation

Generate uniformly on the set $\{0, 1, \cdots, m-1\}$
Returns the value in the table
Computation cost : $\mathcal{O}(1)$ step
Memory cost : $\mathcal{O}(m)$

Generation algorithm

u= r andom() ;
i= (int) floor(u*m)
return T[i]

Tabulation method

Pre-computation

$$p_k = \frac{m_k}{m} \text{ where } \sum_k m_k = m.$$

Create a table T with size m.
Fill T such that m_k cells contains k.
Computation cost : m steps
Memory cost : m

Table construction

i=0
for k=1, k $\leqslant$ K, k++ **do**
  for j=1, j $\leqslant$ m_k, j++ **do**
    T[i]= k ; i=i+1 ;
  end for
end for

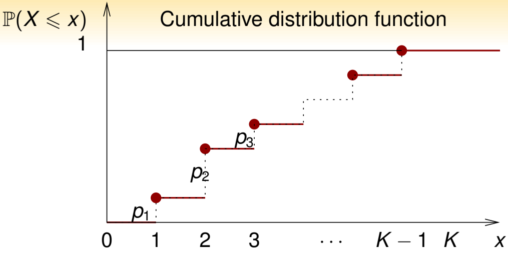
Generation

Generate uniformly on the set $\{0, 1, \cdots, m-1\}$
Returns the value in the table
Computation cost : $\mathcal{O}(1)$ step
Memory cost : $\mathcal{O}(m)$

Generation algorithm

u= r andom() ;
i= (int) floor(u*m)
return T[i]

Inverse of PDF



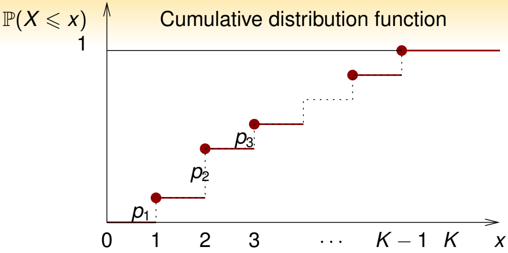
Generation

Divide $[0,1[$ in intervals with length p_k
Find the interval in which *Random* falls
Returns the index of the interval
Computation cost : $\mathcal{O}(\mathbb{E}X)$ steps
Memory cost : $\mathcal{O}(1)$

Inverse function algorithm

s=0 ; k=0 ;
u=random()
while u $>$s **do**
    k=k+1
    s=s+p_k
end while
return k

Inverse of PDF



Generation

Divide $[0, 1[$ in intervals with length p_k
Find the interval in which *Random* falls
Returns the index of the interval
Computation cost : $\mathcal{O}(\mathbb{E}X)$ steps
Memory cost : $\mathcal{O}(1)$

Inverse function algorithm

s=0 ; k=0 ;
u=random()
while u $>$s **do**
  k=k+1
  s=s+p_k
end while
return k

Searching optimization

Optimization methods

- pre-compute the pdf in a table
- rank objects by decreasing probability
- use a dichotomy algorithm
- use a tree searching algorithm (optimality = Huffmann coding tree)

Comments

- Depends on the usage of the injector (repeated use or not)
- pre-computation usually $\mathcal{O}(K)$ could be huge
-

Searching optimization

Optimization methods

- pre-compute the pdf in a table
- rank objects by decreasing probability
- use a dichotomy algorithm
- use a tree searching algorithm (optimality = Huffmann coding tree)

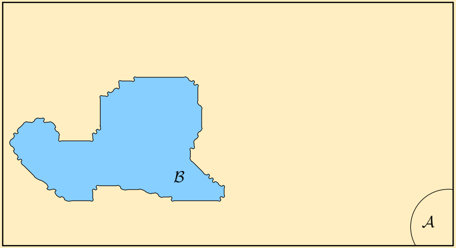
Comments

- Depends on the usage of the injector (repeated use or not)
- pre-computation usually $\mathcal{O}(K)$ could be huge
-

Rejection technique

Base of the method

Generate uniformly on $\mathcal{A}$ accept when point is in $\mathcal{B}$.



Rejection algorithm

repeat
  x = uniform-generate($\mathcal{A}$)
until x$\in \mathcal{B}$
return x

Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

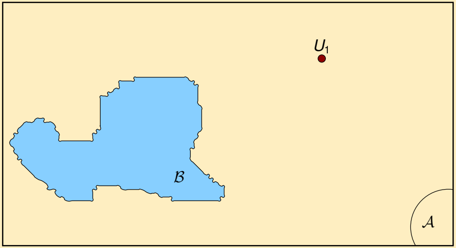
N number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

Rejection technique

Base of the method

Generate uniformly on $\mathcal{A}$ accept when point is in $\mathcal{B}$.



Rejection algorithm

repeat
  x = uniform-generate($\mathcal{A}$)
until x$\in \mathcal{B}$
return x

Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

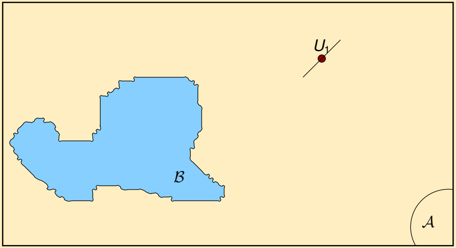
N number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

Rejection technique

Base of the method

Generate uniformly on $\mathcal{A}$ accept when point is in $\mathcal{B}$.



Rejection algorithm

repeat
    x = uniform-generate($\mathcal{A}$)
until x$\in \mathcal{B}$
return x

Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

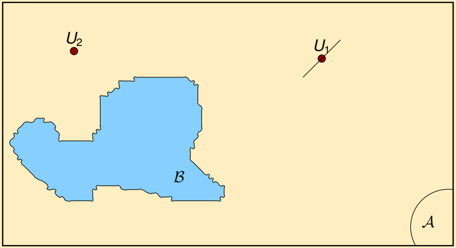
N number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

Rejection technique

Base of the method

Generate uniformly on $\mathcal{A}$ accept when point is in $\mathcal{B}$.



Rejection algorithm

repeat
  x = uniform-generate($\mathcal{A}$)
until x$\in \mathcal{B}$
return x

Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

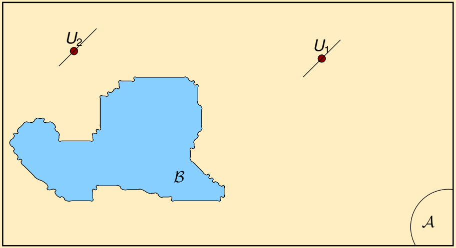
N number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

Rejection technique

Base of the method

Generate uniformly on $\mathcal{A}$ accept when point is in $\mathcal{B}$.



Rejection algorithm

repeat
  x = uniform-generate($\mathcal{A}$)
until x$\in \mathcal{B}$
return x

Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

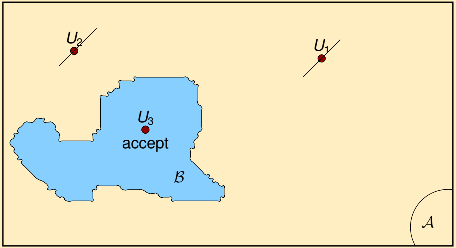
N number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

Rejection technique

Base of the method

Generate uniformly on $\mathcal{A}$ accept when point is in $\mathcal{B}$.



Rejection algorithm

repeat
x = uniform-generate($\mathcal{A}$)
until x$\in \mathcal{B}$
return x

Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

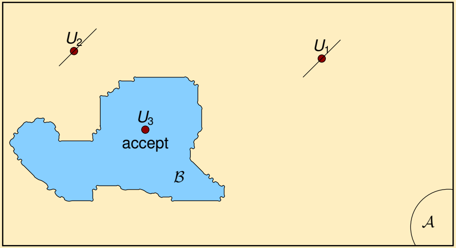
N number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

Rejection technique

Base of the method

Generate uniformly on $\mathcal{A}$ accept when point is in $\mathcal{B}$.



Rejection algorithm

repeat
  x = uniform-generate($\mathcal{A}$)
until x$\in \mathcal{B}$
return x

Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

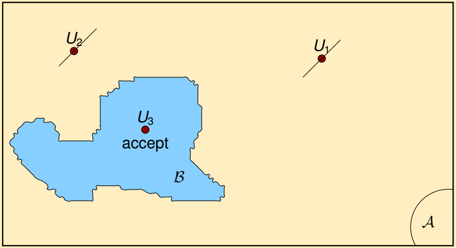
N number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

Rejection technique

Base of the method

Generate uniformly on $\mathcal{A}$ accept when point is in $\mathcal{B}$.



Rejection algorithm

repeat
  x = uniform-generate($\mathcal{A}$)
until x$\in \mathcal{B}$
return x

Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

N number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

Rejection technique

Rejection adaptation

K objects

$$h \geqslant \max_k p_k$$

Generate uniformly on the surface $K \times h$
Accept if the point is under the distribution

Rejection algorithm

repeat
  k= alea(K)
until Random . h $\leqslant p_k$
return k

alea(K) generate uniformly a number in $\{1, \cdots, K\}$

Complexity

Acceptance probability $p_a = \frac{1}{hK}$
N number of iterations $\mathbb{E}N = \frac{1}{p_a} = hK$.
Minimal complexity for $h^* = \max_k p_k$.
Uniform distribution $\Rightarrow$ no rejection
Interest : distribution near the uniform distribution

Rejection technique

Rejection adaptation

K objects

$$h \geqslant \max_k p_k$$

Generate uniformly on the surface $K \times h$
Accept if the point is under the distribution

Rejection algorithm

repeat
k= alea(K)
until Random . h $\leqslant p_k$
return k

alea(K) generate uniformly a number in $\{1, \cdots, K\}$

Complexity

Acceptance probability $p_a = \frac{1}{hK}$
N number of iterations $\mathbb{E}N = \frac{1}{p_a} = hK$.
Minimal complexity for $h^* = \max_k p_k$.
Uniform distribution $\Rightarrow$ no rejection
Interest : distribution near the uniform distribution

Rejection technique

Rejection adaptation

K objects

$$h \geqslant \max_k p_k$$

Generate uniformly on the surface $K \times h$

Accept if the point is under the distribution

Rejection algorithm

repeat
  k= alea(K)
until Random . h $\leqslant p_k$
return k

alea(K) generate uniformly a number in $\{1, \cdots, K\}$

Complexity

Acceptance probability $p_a = \frac{1}{hK}$
N number of iterations $\mathbb{E}N = \frac{1}{p_a} = hK$.
Minimal complexity for $h^* = \max_k p_k$.
Uniform distribution $\Rightarrow$ no rejection
Interest : distribution near the uniform distribution

Aliasing technique

Combine uniform and alias value when rejection

Initialization

K objects

- list L=$\emptyset$,U=$\emptyset$;
- **for** k=1 ; k$\leqslant$ K ; k++ **do**
  - P[k]=p_k
  - **if** P[k] $\geqslant \frac{1}{K}$ **then**
    - U=U+$\{$k$\}$;
  - **else**
    - L=L+$\{$k$\}$;
  - **end if**
- **end for**

Alias and threshold tables

- **while** L $\neq \emptyset$ **do**
  - Extract $k \in L$
  - Extract $i \in U$
  - S[k]=P[k]
  - A[k]=i
  - P[i] = P[i] - ($\frac{1}{K}$-P[k])
  - **if** P[i] $\geqslant \frac{1}{K}$ **then**
    - U=U+$\{$i$\}$;
  - **else**
    - L=L+$\{$i$\}$;
  - **end if**
- **end while**

Aliasing technique

Combine uniform and alias value when rejection

Initialization

K objects

```
list L=∅,U=∅ ;
for k=1 ; k⩽ K ; k++ do
    P[k]=p_k
    if P[k] ⩾ 1/K then
        U=U+{k} ;
    else
        L=L+{k} ;
    end if
end for
```

Alias and threshold tables

```
while L ≠ ∅ do
    Extract k ∈ L
    Extract i ∈ U
    S[k]=P[k]
    A[k]=i
    P[i] = P[i] - (1/K-P[k])
    if P[i] ⩾ 1/K then
        U=U+{i} ;
    else
        L=L+{i} ;
    end if
end while
```

Aliasing technique : generation

Generation

k=alea(K)
if Random . $\frac{1}{K} \leqslant$ S[k] **then**
  return k
else
  return A[k]
end if

Complexity

Computation time :
- $\mathcal{O}(K)$ for pre-computation
- $\mathcal{O}(1)$ for generation

Memory :
- threshold $\mathcal{O}(K)$ (real numbers as probability)
- alias $\mathcal{O}(K)$ (integers indexes in a tables)

Aliasing technique : generation

Generation

k=alea(K)
if Random . $\frac{1}{K} \leqslant$ S[k] **then**
  return k
else
  return A[k]
end if

Complexity

Computation time :
- $\mathcal{O}(K)$ for pre-computation
- $\mathcal{O}(1)$ for generation

Memory :
- threshold $\mathcal{O}(K)$ (real numbers as probability)
- alias $\mathcal{O}(K)$ (integers indexes in a tables)

Outline

Generation of complex objects

Structured workload

- task graph
- sequence of pages
- route to destination
- ...

Structured environment

- Interconnection graph
- memory configuration
- repartition of sites on an area...
- ...

Generate uniformly a set of k positions among n possibilities

Generation of complex objects

Structured workload

- task graph
- sequence of pages
- route to destination
- ...

Structured environment

- Interconnection graph
- memory configuration
- repartition of sites on an area...
- ...

Generate uniformly a set of k positions among n possibilities

Generation of complex objects

Structured workload

- task graph
- sequence of pages
- route to destination
- ...

Structured environment

- Interconnection graph
- memory configuration
- repartition of sites on an area...
- ...

Generate uniformly a set of k positions among n possibilities

Route generation

Given a feed-forward communication network, generate uniformly a route between two nodes

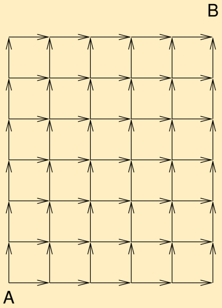
Manhattan topology

General topology

Route generation

Given a feed-forward communication network, generate uniformly a route between two nodes

Manhattan topology

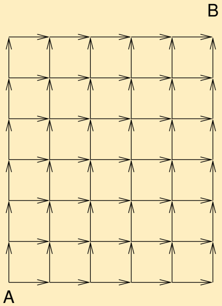


General topology

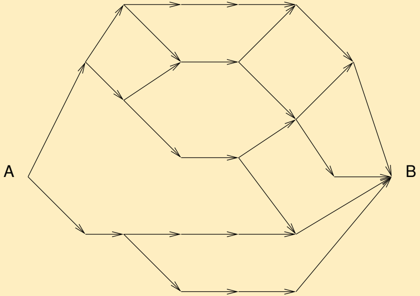
Route generation

Given a feed-forward communication network, generate uniformly a route between two nodes

Manhattan topology



General topology



Permutation generation

Given a size N of an array generate a uniform permutation of its elements.

Based on position

for i=1 ; i⩽ N-1 ; i++ **do**
  j=alea(N-i)
  {Generate uniformly on $\{0, 1, \cdots, N-i\}$ }
  Exchange(i,i+j)
end for

Based on value

Generate_Permutation(N-1)
j=alea(N)
{Generate uniformly on $\{1, \cdots, N\}$ }
for i=N ; i>j ; j- - **do**
  Exchange(i,i-i)
end for
T[j]=N

Permutation generation

Given a size N of an array generate a uniform permutation of its elements.

Based on position

for i=1 ; i$\leqslant$ N-1 ; i++ **do**
  j=alea(N-i)
  {Generate uniformly on $\{0, 1, \cdots, N - i\}$ }
  Exchange(i,i+j)
end for

Based on value

Generate_Permutation(N-1)
j=alea(N)
{Generate uniformly on $\{1, \cdots, N\}$ }
for i=N ; i>j ; j- - **do**
  Exchange(i,i-i)
end for
T[j]=N

Permutation generation

Given a size N of an array generate a uniform permutation of its elements.

Based on position

for i=1 ; i$\leqslant$ N-1 ; i++ **do**
  j=alea(N-i)
  {Generate uniformly on $\{0, 1, \cdots, N - i\}$ }
  Exchange(i,i+j)
end for

Based on value

Generate_Permutation(N-1)
j=alea(N)
{Generate uniformly on $\{1, \cdots, N\}$ }
for i=N ; i>j ; j- - **do**
  Exchange(i,i-i)
end for
T[j]=N

Binary tree generation

Given a size N generate a binary tree uniformly on the set of trees with N nodes

Uniform node decomposition

Recursive algorithm

```
tree Generate_tree(integer N)
if N=0 then
   return empty_tree
else
   q=alea(0,N-1)
   TL=Generate_tree(q)
   TR=Generate_tree(N-1-q)
   T=Join(TL,TR)
   return T
end if
```

Non uniform

Binary tree generation

Given a size *N* generate a binary tree uniformly on the set of trees with *N* nodes

Uniform node decomposition

Recursive algorithm

```
tree Generate_tree(integer N)
if N=0 then
    return empty_tree
else
    q=alea(0,N-1)
    TL=Generate_tree(q)
    TR=Generate_tree(N-1-q)
    T=Join(TL,TR)
    return T
end if
```

Non uniform

Binary tree generation

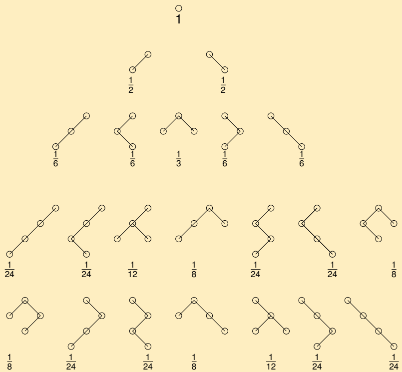
Given a size N generate a binary tree uniformly on the set of trees with N nodes

Uniform node decomposition

Recursive algorithm

```
tree Generate_tree(integer N)
if N=0 then
   return empty_tree
else
   q=alea(0,N-1)
   TL=Generate_tree(q)
   TR=Generate_tree(N-1-q)
   T=Join(TL,TR)
   return T
end if
```

Non uniform



Uniform binary tree generation

Catalan's numbers

Recursion equation

$$C_0 = C_1 = 1;$$

$$C_N = \sum_{q=0}^{N-1} C_q C_{N-1-q}.$$

Then

$$1 = \sum_{q=0}^{N-1} \frac{C_q C_{N-1-q}}{C_N} = \sum_{q=0}^{N-1} p_{N,q}.$$

$$C_N = \frac{1}{N+1} \binom{2N}{N}$$

Uniform generation

```
tree Generate_tree(integer N)
if N=0 then
    return empty_tree
else
    q=Generate(p_{N,0}, ··· , p_{N,N-1})
    TL=Generate_tree(q)
    TR=Generate_tree(N-1-q)
    T=Join(TL,TR)
    return T
end if
```

Pre-computation of the $p_{N,q}$

Uniform binary tree generation

Catalan's numbers

Recursion equation

$$C_0 = C_1 = 1;$$

$$C_N = \sum_{q=0}^{N-1} C_q C_{N-1-q}.$$

Then

$$1 = \sum_{q=0}^{N-1} \frac{C_q C_{N-1-q}}{C_N} = \sum_{q=0}^{N-1} p_{N,q}.$$

$$C_N = \frac{1}{N+1} \binom{2N}{N}$$

Uniform generation

```
tree Generate_tree(integer N)
if N=0 then
    return empty_tree
else
    q=Generate(p_{N,0}, ··· , p_{N,N-1})
    TL=Generate_tree(q)
    TR=Generate_tree(N-1-q)
    T=Join(TL,TR)
    return T
end if
```

Pre-computation of the $p_{N,q}$

Outline

Generation of length, duration,...

The workload is defined by :
- type
- structure
- amount of work
- time distribution

- service duration
- communication time
- size of messages
- ...

Generation of continuous variates

From a probability density, generate samples of variates in a continuous state space.

Generation of length, duration,...

The workload is defined by :
- type
- structure
- amount of work
- time distribution

- service duration
- communication time
- size of messages
- ...

Generation of continuous variates

From a probability density, generate samples of variates in a continuous state space.

Generation of length, duration,...

The workload is defined by :
- type
- structure
- amount of work
- time distribution

- service duration
- communication time
- size of messages
- ...

Generation of continuous variates

From a probability density, generate samples of variates in a continuous state space.

Generating random quantities

Denote by X the object size (X is a real valued random variable)
Distribution density

$$f(x)dx = \mathbb{P}(X \in [x, x + dx[).$$

Remarks :

$$0 \leqslant f(x); \quad \int f(x)dx = 1.$$

Expectation (average, mean)

$$\mathbb{E}X = \int xf(x)dx.$$

Variance and standard deviation

$$\mathbb{V}arX = \int (x - \mathbb{E}X)^2 f(x)dx = \mathbb{E}X^2 - (\mathbb{E}X)^2.$$

$$\sigma(X) = \sqrt{\mathbb{V}arX}.$$

Generating random quantities

Denote by X the object size (X is a real valued random variable)
Distribution density

$$f(x)dx = \mathbb{P}(X \in [x, x + dx[).$$

Remarks :

$$0 \leqslant f(x); \quad \int f(x)dx = 1.$$

Expectation (average, mean)

$$\mathbb{E}X = \int xf(x)dx.$$

Variance and standard deviation

$$\mathbb{V}arX = \int (x - \mathbb{E}X)^2 f(x)dx = \mathbb{E}X^2 - (\mathbb{E}X)^2.$$

$$\sigma(X) = \sqrt{\mathbb{V}arX}.$$

Generating random quantities

Denote by X the object size (X is a real valued random variable)
Distribution density

$$f(x)dx = \mathbb{P}(X \in [x, x + dx[).$$

Remarks :

$$0 \leqslant f(x); \quad \int f(x)dx = 1.$$

Expectation (average, mean)

$$\mathbb{E}X = \int xf(x)dx.$$

Variance and standard deviation

$$\mathbb{V}arX = \int (x - \mathbb{E}X)^2 f(x)dx = \mathbb{E}X^2 - (\mathbb{E}X)^2.$$

$$\sigma(X) = \sqrt{\mathbb{V}arX}.$$

Generating random quantities

Denote by X the object size (X is a real valued random variable)
Distribution density

$$f(x)dx = \mathbb{P}(X \in [x, x + dx[).$$

Remarks :

$$0 \leqslant f(x); \quad \int f(x)dx = 1.$$

Expectation (average, mean)

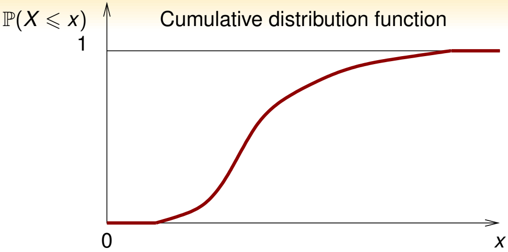
$$\mathbb{E}X = \int xf(x)dx.$$

Variance and standard deviation

$$\mathbb{V}arX = \int (x - \mathbb{E}X)^2 f(x)dx = \mathbb{E}X^2 - (\mathbb{E}X)^2.$$

$$\sigma(X) = \sqrt{\mathbb{V}arX}.$$

Inverse of CDF



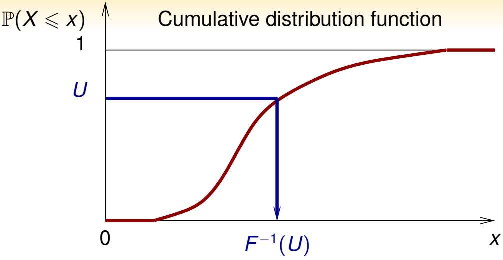
Let $X = F^{-1}(U)$

$$\mathbb{P}(X \leqslant x) = \mathbb{P}(F^{-1}(U) \leqslant x) = \mathbb{P}(U \leqslant F(x)) = F(x).$$

Classic distribution

- Uniform on $[a, b]$: $F^{-1}(u) = a + (b - a).u$
- Exponential, rate λ : $F^{-1}(u) = \frac{1}{\lambda} \log(1 - u)$
- Pareto, Weibul,...

Inverse of CDF



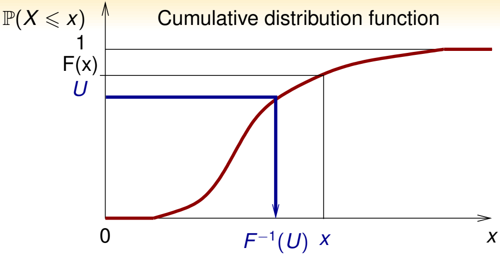
Let $X = F^{-1}(U)$

$$\mathbb{P}(X \leqslant x) = \mathbb{P}(F^{-1}(U) \leqslant x) = \mathbb{P}(U \leqslant F(x)) = F(x).$$

Classic distribution

- Uniform on $[a, b]$: $F^{-1}(u) = a + (b - a).u$
- Exponential, rate λ : $F^{-1}(u) = \frac{1}{\lambda}\log(1 - u)$
- Pareto, Weibul,...

Inverse of CDF



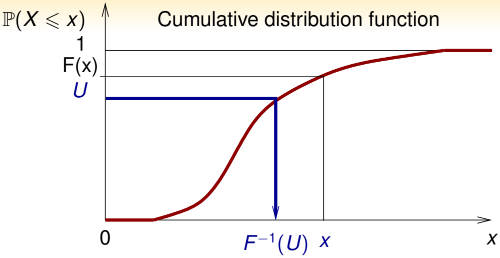
Let $X = F^{-1}(U)$

$$\mathbb{P}(X \leqslant x) = \mathbb{P}(F^{-1}(U) \leqslant x) = \mathbb{P}(U \leqslant F(x)) = F(x).$$

Classic distribution

- Uniform on $[a, b]$: $F^{-1}(u) = a + (b - a).u$
- Exponential, rate λ : $F^{-1}(u) = \frac{1}{\lambda}\log(1 - u)$
- Pareto, Weibul,...

Inverse of CDF



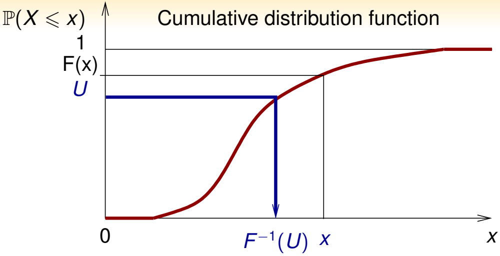
Let $X = F^{-1}(U)$

$$\mathbb{P}(X \leqslant x) = \mathbb{P}(F^{-1}(U) \leqslant x) = \mathbb{P}(U \leqslant F(x)) = F(x).$$

Classic distribution

- Uniform on $[a, b]$: $F^{-1}(u) = a + (b - a).u$
- Exponential, rate λ : $F^{-1}(u) = \frac{1}{\lambda}\log(1 - u)$
- Pareto, Weibul,...

Inverse of CDF



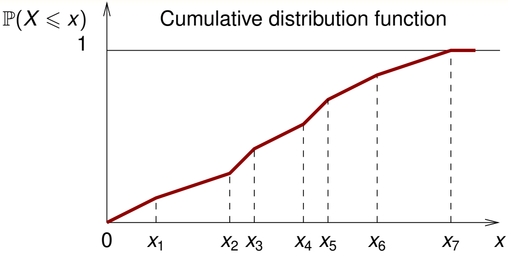
Let $X = F^{-1}(U)$

$$\mathbb{P}(X \leqslant x) = \mathbb{P}(F^{-1}(U) \leqslant x) = \mathbb{P}(U \leqslant F(x)) = F(x).$$

Classic distribution

- Uniform on $[a, b]$: $F^{-1}(u) = a + (b - a).u$
- Exponential, rate λ : $F^{-1}(u) = \frac{1}{\lambda}\log(1 - u)$
- Pareto, Weibul,...

Inverse of CDF : empirical data

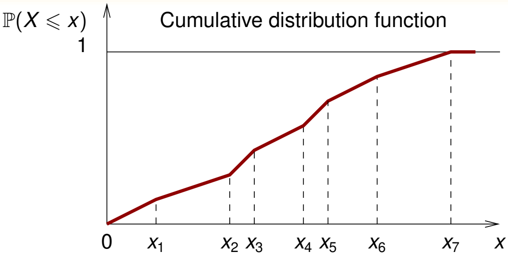


Set of observed values (sorted) $x_1, \cdots, x_N$, x_0 fixed by hand

```
j=alea(1,N)
x=x_{j-1} + (x_j - x_{j-1}).random
return x
```

Linear interpolation Extensions : fit with middle of intervals, polynomial interpolation

Inverse of CDF : empirical data



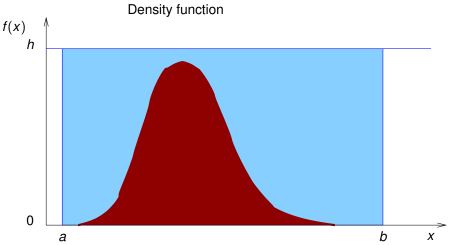
Set of observed values (sorted) $x_1, \cdots, x_N$, x_0 fixed by hand

j=alea(1,N)
x=$x_{j-1} + (x_j - x_{j-1})$.random
return x

Linear interpolation Extensions : fit with middle of intervals, polynomial interpolation

Rejection

Bounded density on a bounded interval



The rejection algorithm

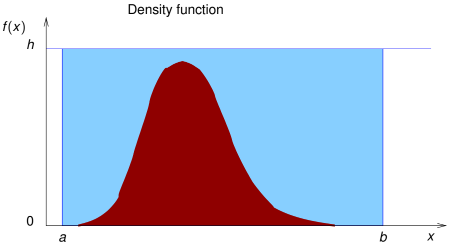
repeat
x= Uniform(a,b)
y= Uniform(0,h)
until y $\leqslant$ f(x)
return x

Complexity

Acceptance probability $p_a = \frac{1}{h.(b-a)}$
Mean number of iterations :
$\mathbb{E}N = h.(b-a)$
Optimality : $h^* = \max f(x)$

Rejection

Bounded density on a bounded interval



The rejection algorithm

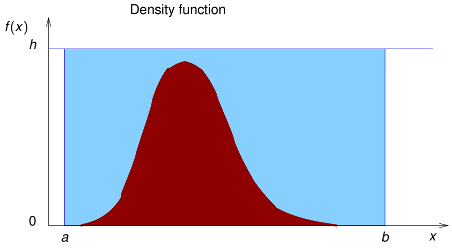
repeat
  x= Uniform(a,b)
  y= Uniform(0,h)
until y $\leqslant$ f(x)
return x

Complexity

Acceptance probability $p_a = \frac{1}{h.(b-a)}$
Mean number of iterations :
$\mathbb{E}N = h.(b-a)$
Optimality : $h^* = \max f(x)$

Rejection

Bounded density on a bounded interval



The rejection algorithm

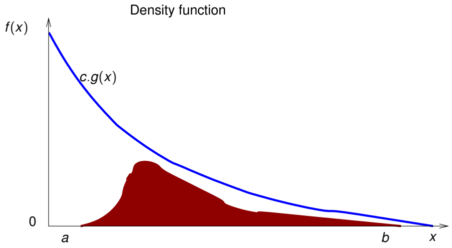
repeat
  x= Uniform(a,b)
  y= Uniform(0,h)
until y $\leqslant$ f(x)
return x

Complexity

Acceptance probability $p_a = \frac{1}{h.(b-a)}$
Mean number of iterations :
$\mathbb{E}N = h.(b-a)$
Optimality : $h^* = \max f(x)$

Rejection : unbounded case

$f(x) \leqslant c.g(x)$ and there is a generator for g density



The rejection algorithm

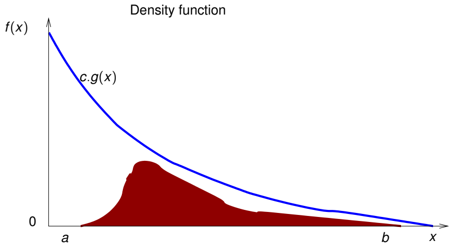
repeat
  x= Generate according g
  y= Uniform(0,c.g(x))
until y $\leqslant$ f(x)
return x

Complexity

Acceptance probability $p_a = \frac{1}{c}$
Mean number of iterations :
$\mathbb{E}N = c$

Rejection : unbounded case

$f(x) \leqslant c.g(x)$ and there is a generator for g density



The rejection algorithm

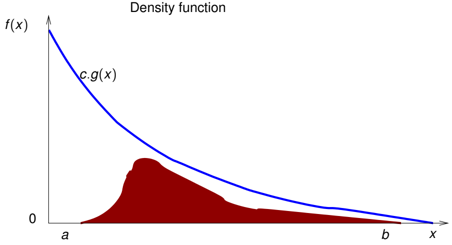
repeat
  x= Generate according g
  y= Uniform(0,c.g(x))
until y $\leqslant$ f(x)
return x

Complexity

Acceptance probability $p_a = \frac{1}{c}$
Mean number of iterations :
$\mathbb{E}N = c$

Rejection : unbounded case

$f(x) \leqslant c.g(x)$ and there is a generator for g density



The rejection algorithm

repeat
  x= Generate according g
  y= Uniform(0,c.g(x))
until y $\leqslant$ f(x)
return x

Complexity

Acceptance probability $p_a = \frac{1}{c}$
Mean number of iterations :
$\mathbb{E}N = c$

Outline

1. Workload generation problem
2. Generating random objects
3. Generation of complex objects
4. Quantity generation
5. **Synthesis**

Synthesis

Characterization of the load :

1. Types
2. Structure
3. Quantificaton

Statistical description :

- empirical data : histograms, samples
- models (law of probability, classical distributions : uniform, exponential, erlang,...)
- bootstrapping

Validation of the workload generator : samples + statistical tests

Synthesis

Characterization of the load :

1. Types
2. Structure
3. Quantificaton

Statistical description :

- empirical data : histograms, samples
- models (law of probability, classical distributions : uniform, exponential, erlang,...)
- bootstrapping

Validation of the workload generator : samples + statistical tests

Synthesis

Characterization of the load :

1. Types
2. Structure
3. Quantificaton

Statistical description :

- empirical data : histograms, samples
- models (law of probability, classical distributions : uniform, exponential, erlang,...)
- bootstrapping

Validation of the workload generator : samples + statistical tests