RANDOM VARIABLES GENERATION

- Goal: Generate random variables
  - Discrete random variables
  - Continuous random variables (univariate)

Discrete Variables
- Finite set of values
- Example: population
- 0, 1, 2, 3, …

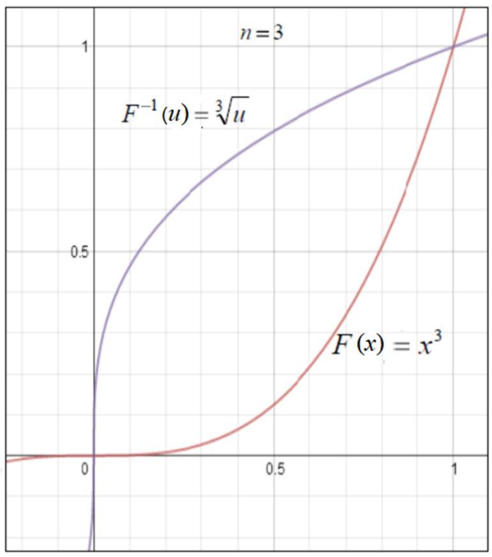
Continuous Variables
- Continuous distribution function
- Example: temperature
- Univariate
  - One predicting variable (x)
- Multivariate
  - Multiple predicting variables ($x_1, x_2 \ldots$)

1. INVERSE TRANSFORM METHOD

- Prerequisites
  - The uniform (0,1) random variable U
  - The continuous, cumulative distribution function F of the targeted random variable X
- Inverse Transform Method
  - Definition:
    - $X = F^{-1}(U)$
    - Note: The superscript "-1" is not an exponent! It indicates the inverse!
  - Transformation:
    - $F(F^{-1}(U)) = U = F(X)$
    - $F^{-1}(U)$ is defined to be that value of X such that $F(X) = U$

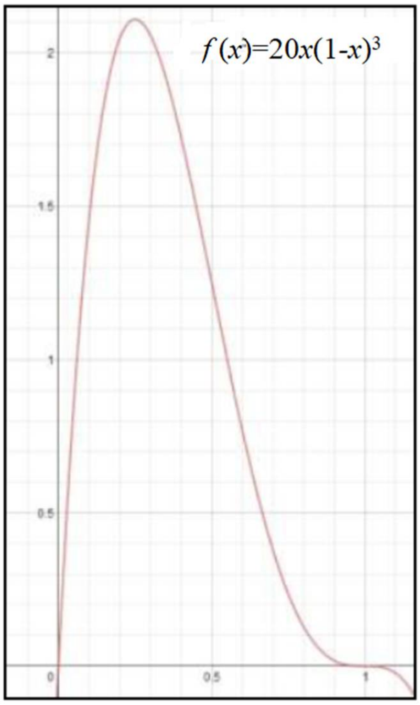
Inverse Transform Method:
1. Find a formula for the function F^{-1}
2. Generate a uniform random number U
3. Return the random number $X = F^{-1}(U)$

- Example
  - Generate a random variable X with cumulative distribution function $F(x) = x^n;\ 0 < x < 1$
- 1. Find a formula for the function F^{-1}
  - $F(x) = x^n = u$
    $\leftrightarrow\ x = \sqrt[n]{u} = F^{-1}(u)$
- 2. Formulate the algorithm for generating the random variable X
  - Generate a random number U and
  - Return $X = \sqrt[n]{U}$
- Limitation of the Inverse Transform Method
  - Cumulative distribution function needs to be invertible
  - Thus the inverse transform method is not suited for the normal distribution!



2. ACCEPTANCE-REJECTION METHOD

- Example
  - Give the rejection algorithm that generates X having $f(x) = 20x(1-x)^3, 0 < x < 1$
  - Use $g(x) = 1;\ \ 0 < x < 1$



- 1. Generate a rejection procedure

  Determine the smallest c such that $\frac{f(x)}{g(x)} \leq c$

  - 1.1 Determine maximum of $\frac{f(x)}{g(x)} = 20x(1-x)^3$

    1.1.1 Differentiation yields $\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = 20[(1-x)^3 - 3x * (1-x)^2]$

    1.1.2 Setting this equal to 0 shows that maximum for $x = \frac{1}{4}$

  - 1.2 Determine smallest c

    1.2.1 $\frac{f(x)}{g(x)} \leq 20 * \frac{1}{4} * \left(1 - \frac{1}{4}\right)^3 = \frac{135}{64} = smallest\ c$

    1.2.2 $\frac{f(x)}{c * g(x)} = \frac{20x(1-x)^3}{\frac{135}{64} * 1} = \frac{256}{27} x(1-x)^3$

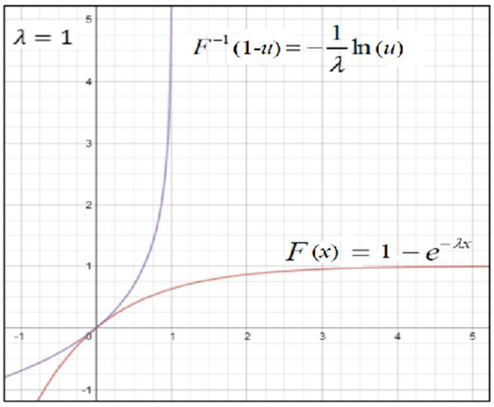
- 2. Formulate the rejection algorithm
  - 1. Generate random numbers U_1 and U_2
  - 2. If $U_2 \leq \frac{256}{27} U_1 (1-U_1)^3$ stop and set $X = U_1$; otherwise go to step 1

- Average number of iterations = $c = \frac{135}{64} = 2.11$

3. GENERATING IMPORTANT DISTRIBUTIONS

3.1. EXPONENTIAL DISTRIBUTION

- Generate an exponential random variable
  - Exponential distribution: $f(x) = \lambda e^{-\lambda x};\ F(x) = 1 - e^{-\lambda x}$



- Inverse Transform Method
- 1. Find a formula for the function F^{-1}
  - $F(x) = \quad 1 - e^{-\lambda x} = u$

    $\leftrightarrow \quad 1 - u = e^{-\lambda x}$

    $\leftrightarrow \quad \ln(1-u) = -\lambda x$

    $\leftrightarrow \quad -\frac{1}{\lambda} * \ln(1-u) = x \ = F^{-1}(u)$

    $\leftrightarrow \quad -\frac{1}{\lambda} * \ln(u) = x \ = F^{-1}(1-u)$

  since u is uniform(0,1), (1- u) is also uniform(0,1) and has the same distribution

- 2. Formulate the algorithm for generating the random variable X
  - 1. Generate a random number U
  - 2. Return $X = -\frac{1}{\lambda}\ln(U)$

3.2. NORMAL DISTRIBUTION

- Give the rejection algorithm that generates a sequence of normal random variables X_i with mean μ and variance σ
  - Approach: Generate a sequence of half-normal distributed random variables and determine each variable's sign randomly

- Half-Normal distribution: $f(x) = \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}};\ 0 < x < \infty$
  - Optimize the algorithm and assess its efficiency
- Acceptance-Rejection Method
- Set $g(x) = e^{-x};\ 0 < x < \infty$
- 1. Generate a rejection procedure
  - 1. Determine the smallest c such that $c \geq \frac{f(x)}{g(x)}$
    - 1.1 Determine maximum of $\frac{f(x)}{g(x)}$

      1.1.1 $\frac{f(x)}{g(x)} = \frac{\frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}}{e^{-x}} = \sqrt{\frac{2}{\pi}} * e^{x-\frac{x^2}{2}}$ → has its maximum if $x - \frac{x^2}{2}$ has its maximum

      1.1.1 Differentiation yields $\frac{d}{dx}\left(x - \frac{x^2}{2}\right) = 1 - x$

      1.1.2 Setting this equal to 0 shows that maximum for $x = 1$
    - 1.2 Determine smallest c

      1.2.1 $\max\left(\frac{f(x)}{g(x)}\right) = \sqrt{\frac{2}{\pi}} * e^{1-\frac{1^2}{2}} = \sqrt{\frac{2}{\pi}} * e^{\frac{1}{2}} = \sqrt{\frac{2e}{\pi}} = smallest\ c$

      1.2.2 $\frac{f(x)}{c \cdot g(x)} = \sqrt{\frac{2}{\pi}} * e^{x-\frac{x^2}{2}} * \sqrt{\frac{\pi}{2e}} = e^{x-\frac{x^2}{2}-\frac{1}{2}} = e^{-\frac{x^2-2x+1}{2}} = e^{-\frac{(x-1)^2}{2}}$
- 2. Formulate the rejection algorithm
  - 0. $i = 1$
  - 1. Generate an exponential random variable with rate 1 called Y_1
  - 2. Generate a random number U_1
  - 3. If $U_1 \leq e^{-\frac{(Y_1-1)^2}{2}}$ stop and set $Y_2 = Y_1$
    - Otherwise go to step 1
  - 4. Generate a random number U_2 and set

    $$X_i = \mu + \sigma * \begin{cases} Y_2; U_2 \leq 0.5 \\ -Y_2; U_2 > 0.5 \end{cases}$$
  - 5. Set $i = i + 1$ and go to step 1
- 3. Optimization
  - Transformations
    - $U \leq e^{-\frac{(Y_1-1)^2}{2}} \leftrightarrow -\ln(U) \geq \frac{(Y_1-1)^2}{2} \leftrightarrow * \ Y_2 \geq \frac{(Y_1-1)^2}{2} \leftrightarrow Y_2 - \frac{(Y_1-1)^2}{2} \geq 0$
      - * because $-\ln(U)$ is exponential with rate 1 (see exponential distribution)
    - Since Y_1 and Y_2 are exponentials with rate 1, $Y_3 = Y_2 - \frac{(Y_1-1)^2}{2}$ is also an exponential with rate 1
  - Optimized Algorithm
    - 0. $i = 1$
    - 1. Generate an exponential random variables with rate 1 called Y_1
    - 2. Generate an exponential random variables with rate 1 called Y_2
    - 3. If $Y_2 - \frac{(Y_1-1)^2}{2} \geq 0$ stop and set $Y_3 = Y_2 - \frac{(Y_1-1)^2}{2}$
      - Otherwise go to step 1
    - 4. Generate a random number U and set

      $$X_i = \mu + \sigma * \begin{cases} Y_1; U \leq 0.5 \\ -Y_1; U > 0.5 \end{cases}$$
    - 5. Set $i = i + 1$ and set $Y_1 = Y_3$ and go to step 2

- 4. Efficiency assessment
  - Average number of required exponential random variables = 1.64
    - Average number of iterations (step 1 and 2) = $2c = 2.64$
    - Usage of exponential Y: Average number of iterations = $2.64 - 1 = 1.64$
  - Average number of required squares (step 3): $c = 1.32$

3.2. POISSON PROCESS

(Homogeneous) Poisson Process	Nonhomogeneous Poisson Process
- Events are as likely to occur in all intervals of equal size - The rate λ, which represents the expected number of events, is constant	- Events occur randomly in time - Expected arrival rates vary with time - The rate λ, which represents the expected number of events, is not constant - The intensity function $\lambda(t)$ represents the expected number of events around the time t

3.2.1.Homogenous Poisson Processes

- Generate the first T time units of a Poisson process having rate λ
  - Approach: Use the exponential distribution to generate event times (so-called interarrival times) and stop when their sum exceeds T
- Inverse Transform Method
- 1. Find a formula for the function F^{-1}
  - We already did that (see exponential distribution)
  - $x = F^{-1}(u) = -\frac{1}{\lambda}\ln(u)$
- 2. Formulate the algorithm for generating the Poisson process
  - 0. $t = 0$; $I = 0$
  - 1. Generate U
  - 2. $t = t + (-\frac{1}{\lambda}\ln(U)) = t - \frac{1}{\lambda}\ln(U)$
    - If $t > T$, stop!
  - 3. $I = I + 1$; $S(I) = t$
  - 4. Go to step 1
- The solution is the sequence of the event times $S(1)$ to $S(I)$

Legend:
- T = first T time units
- t = time
- I = number of events that has occurred by time t (→ the final I values represent the number of events that occurred by time T)
- $S(1), \dots, S(I)$ = event times in increasing order (→ $S(I)$ represents the most recent event time)

3.2.1.Nonhomogenous Poisson Processes

- Generate the first T time units of a nonhomogeneous Poisson process with intensity function $\lambda(t)$
  - Option 1: *Thinning* (also called *random sampling*)
    - Not all simulated evens are counted
    - Events are only counted randomly, which “thins” the (homogeneous) Poisson process
  - Option 2: Successive event times
- Inverse Transform Method
- Option 1: Thinning
  - 1. Simulate a Poisson process
  - 2. Randomly count its events → Remaining events will be nonhomogeneous

- Formulate the algorithm for generating the Poisson process
  - 0. $t = 0,\ I = 0$
  - 1. Generate a random number U_1
  - 2. Set $t = t - \frac{1}{\lambda}\ln(U_1)$ and stop if $t > T$

    } simulation of a Poisson process
  - 3. Generate another random number U_2
  - 4. If $U_2 \leq \frac{\lambda(t)}{\lambda}$, set $I = I + 1,\ S(I) = t$

    } randomly counting events
  - 5. Go to step 1

Legend:
- T = first T time units
- t = time
- I = number of events that has occurred by time t
- $S(I)$ = most recent event time
- $\lambda(t)$ = intensity function = expected number of events around t; $\lambda(t) \leq \lambda$

- Efficiency
  - Rule: The more events are counted, the more efficient the thinning approach
  - Thinning is most efficient, if $\lambda(t) \approx \lambda$, because then almost all events are counted
  - Improvement :
    - 1. Break up the interval into subintervals
    - 2. Perform the thinning approach over each subinterval

4. APPLICATIONS

4.1. APPLICATION TO GENERATION OF EXPONENTIAL VARIABLES

A casualty insurance company has 1,000 policyholders, each of whom will independently present a claim in the next month with probability .05. The amount of the claims made are independent exponential random variables with mean $800. Use simulation to estimate the probability that the sum of these claims exceeds $50,000.

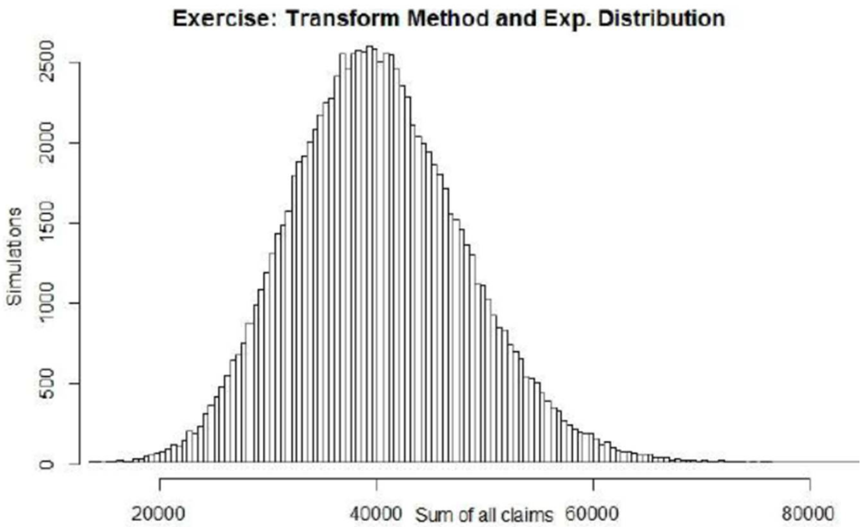
- Why do we have to use simulation?
  - The expected sum of claims will be normally distributed with mean $40,000 (= $1{,}000 * 0.05 *$ $800)
  - However, we do not know the mean and standard deviation
- Algorithm for generating an exponential random variable X
  - 1. Generate a random number U
  - 2. Return $X = -\frac{1}{\lambda}\ln(U)$

```
### Simulation of an exponential distribution + inverse transform method
### Author: Martin Kretzer ### Date:   11 April 2012

### 0. Functions:
### Performs simulations and returns a list of all resulting values
getSimulationsVector<-function( numberOfSimulations, numberOfCustomers, mean, probability )
{
   lambda<-1/mean
   simulationsVector<-c()
   for( i in 1:numberOfSimulations )
   {
      claimsPerSimulationVector<-c()
      #uniformVars<-runif( numberOfCustomers )
      for( j in 1:numberOfCustomers )
      {
         uniformVarProbability<-runif(1)
         if( uniformVarProbability<= probability )
         {
             uniformVarAlgorithm<-runif(1)
             claim<--(1/lambda)*log( uniformVarAlgorithm)
```

```
            claimsPerSimulationVector<-c( claimsPerSimulationVector, claim )
           }
         }
        simulationsVector<-c(simulationsVector, sum(claimsPerSimulationVector) )
    }
return(simulationsVector)
};

### Probability that all values are above the variable limit
getProbability<-function( simulationsVector, limit )
{
  eventcount<-0
  for( I in 1:length( simulationsVector) )
 {
    if( simulationsVector[[i]] >= limit )
    {
      eventcount<-eventcount+ 1
     }
   }
   probability<-eventcount/ length( simulationsVector)
return(probability)
};

###############
###1. Set constant variables:
numberOfSimulations<-100000;
numberOfCustomers<-1000mean<-800;
limit<-50000;
probability<-0.05;

### 2. Execute and output variables:
simulationsVector<-getSimulationsVector( numberOfSimulations, numberOfCustomers, mean,
probability );
simulationsVector; #R maximum outputs 10000 values

### 3. Execute and output variables:
hist(simulationsVector, 250, xlab="Sum of all claims",ylab="Simulations",main="Exercise:
Transform Method and Exp. Distribution");

### 4. Calculate and output probability:
probability<-getProbability( simulationsVector, limit );
probability;
```

The output:

- Executed simulations:100,000
- Estimated probability that the sum of claims exceeds $50,000:
  p =0.10917



4.1. APPLICATION TO REJECTION-ACCEPTANCE METHOD FOR NORMAL VARIABLES

Write a program that efficiently generates normal random variables using the acceptance-rejection method with (real) mean 10 and (real) standard deviation 3. What is your estimated mean and what is your estimated standard deviation?

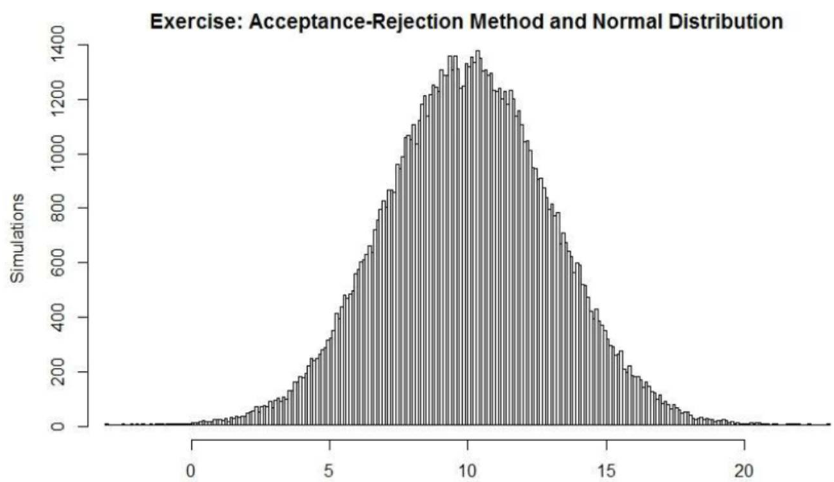
- (Optimized) Algorithm for generating a normal distribution
  - 0. $i = 1$
  - 1. Generate an exponential random variables with rate 1 called Y_1
  - 2. Generate an exponential random variables with rate 1 called Y_2
  - 3. If $Y_2 - \frac{(Y_1-1)^2}{2} \geq 0$ stop and set $Y_3 = Y_2 - \frac{(Y_1-1)^2}{2}$
    - Otherwise go to step 1
  - 4. Generate a random number U and set
    $$X_i = \mu + \sigma * \begin{cases} Y_1; U \leq 0.5 \\ -Y_1; U > 0.5 \end{cases}$$
  - 5. Set $i = i + 1$ and set $Y_1 = Y_3$ and go to step 2

```
### 0. Function:
### Performs simulations and returns a vector of all simulated values

getSimulationsVector<-function(numberOfSimulations,mean,stdDev)
{

# Define variables
simulations.vector<-rep(0,numberOfSimulations)

# Generate the exponential variable y1
y1<-runif(1)
y1<--log(y1)
for(i in 1:numberOfSimulations)
{

# Generate the independent exp. variable y2
y2<-runif(1)
y2<--log(y2)
```

```
# Acceptance-Rejection procedure
while(y2 < (y1-1)^2/2 )
{
# if rejected, generate two new independent exp. variables y1 and y2 and repeat procedure
y1<-runif(1)
y1<--log(y1)
y2<-runif(1)
y2<--log(y2)
}
y3<-y2-(y1-1)^2/2

# Generate a random number U and store the simulated variable y1 in simulations.vector
u<-runif(1)
if( u <= 0.5)
{
simulations.vector[i]<-mean + (y1*stdDev)
}
else
{
simulations.vector[i]<-mean + (-y1*stdDev)
}
y1<-y3
}
return(simulations.vector)
}

### Function End
simulations.vector<-getSimulationsVector(100000, 10, 3)
summary(simulations.vector)
sd(simulations.vector)
hist(simulations.vector, 250, xlab="",ylab="Simulations",main="Exercise: Acceptance-Rejection Method and Normal Distribution")
```

Output

Executed simulations: 100,000
Estimated mean: 10.010
Estimated standard deviation: 2.993299



Reference:

Martin Kretzer - Generating Continuous Random Variables, University of Mannheim, Business School